20th International Conference on Database Theory

ICDT 2017, March 21–24, 2017, Venice, Italy

Edited by

Michael Benedikt

Giorgio Orsi

LIPIcs – Vol. 68 – ICDT 2017 www.dagstuhl.de/lipics

Editors

Michael Benedikt
Department of Computer Science
University of Oxford, UK
michael.benedikt@cs.ox.ac.uk

Giorgio Orsi
Meltwater, UK
giorgio.orsi@meltwater.com

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 [Systems] Query Processing, Relational Databases, 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 [Mathematical Logic] Computational Logic, Model Theory, G.2.2 [Graph Theory] Hypergraphs

ISBN 978-3-95977-024-8

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/978-3-95977-024-8.

Publication date
March, 2017

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.2017.0

ISBN 978-3-95977-024-8 **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

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 20th International Conference on Database Theory (ICDT 2017) was held in Venice Italy, March 21-24, 2017. Originally biennial, the ICDT conference has been held annually and jointly with the conference on Extending Database Technology (EDBT) since 2009.

The proceedings of ICDT 2017 includes a paper by Carsten Lutz (University of Bremen), based on his keynote address, an overview of the keynote by Tova Milo (Tel Aviv University), as well as an overview of the invited lecture of Daniel Marx (Hungarian Academy of Sciences), and 19 research papers that were selected by the Program Committee.

Out of the 19 accepted papers, the Program Committee selected the paper *How many variables are needed to express an existential positive query?* by Simone Bova and Hubie Chen for the ICDT 2017 Best Paper Award. Furthermore, the Program Committee selected the paper *k-Regret Minimizing Set: Efficient Algorithms and Hardness* by Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan for the ICDT 2017 Best Newcomer Award.

We want to acknowledge Andrei Voronkov for his prompt support with the EasyChair system, which has made ICDT's transition to a two-cycle submission process work very smoothly. We thank the previous program chair and current ICDT Council Chair Wim Martens for his guidance on the mechanics of running the program committee, and last year's proceedings chair Thomas Zeume for his help in producing the proceedings. We also thank the EDBT/ICDT general chair Salvatore Orlando for enormous assistance with issues related to the conference organization.

Our final thanks is to the members of the ICDT program committee for their work. We are particularly grateful to the following PC members for the enormous effort and care they put in, beyond what we can reasonably expect: Meghyn Bienvenue, Gianluigi Greco, Egor Kostylev, Paweł Parys, Gabriele Puppis, Pierre Senellart, and Jef Wijsen.

We experimented with a different system for running the review and discussion this year, and we appreciate the patience and flexibility of the PC in trying something new.

Michael Benedikt and Giorgio Orsi
March 2017

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Conference Chair

Vim Martens (University of Bayreuth, Germany)

Program Chair

Michael Benedikt (University of Oxford, UK)

Proceedings Chair

Giorgio Orsi (Meltwater, UK)

Program Committee

Marcelo Arenas (Pontificia Universidad Católica, Chile)
Meghyn Bienvenu (CNRS, France)
Gianluigi Greco (University of Calabria, Italy)
Andre Hernich (University of Liverpool, UK)
Egor Kostylev (University of Oxford, UK)
Ashwin Machanavajjhala (Duke University, US)
Yakov Nekrich (University of Waterloo, Canada)
Frank Neven (Hasselt University, Belgium)
Hung Ngo (LogicBlox Inc., US)
Magdalena Ortiz (TU Wien, Austria)
Pawel Parys (University of Warsaw, Poland)
Jeff Phillips (University of Utah, US)
Andreas Pieris (TU Wien, Austria)
Gabriele Puppis (CNRS, France)
Lucian Popa (IBM Almaden Research Center, US)
Pierre Senellart (Télécom ParisTech, France)
Lidia Tendera (University of Opole, Poland)
Srikanta Tithapura (Iowa State University, US)
Jef Wijsen (University of Mons, Belgium)
Frank Wolter (University of Liverpool, UK)

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi

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

List of Authors

Foto N. Afrati
Antoine Amarilli
Yael Amsterdamer
Pablo Barceló
Christoph Berkholz
Pierre Bourhis
Simone Bova
Douglas Burdick
Wei Cao
Hubie Chen
Shaleen Deep
Ronald Fagin
Cristina Feier
Dominik D. Freydenberger
Minos Garofalakis
Tomasz Gogacz
Shachar Itzhaky
Manas R. Joglekar
Jens Keppeler
Benny Kimelfeld
Phokion G. Kolaitis
Tomer Kotek
Paraschos Koutris
Antti Kuusisto
Willis Lang
Jian Li
Ester Livshits
Markus Lohrey
Carsten Lutz
Sebastian Maneth
Dániel Marx
Andrew McGregor
Tova Milo
Mikaël Monet
Jeffrey Naughton
Liat Peterfreund
Lucian Popa
Christopher M. Ré
Carl Philipp Reh
Noam Rinetzky
Miguel Romero
Mooly Sagiv
Semih Salihoglu
Vasilis Samoladas
Nicole Schweikardt
Thomas Schwentick
Luc Segoufin
Pierre Senellart
Bruhathi Sundarmurthy
Orr Tamir
Wang-Chiew Tan
Val Tannen
Szymon Toruńczyk
Jeffrey D. Ullman
Helmut Veith
Alexandre Vigny
Nils Vortmeier
Hoa T. Vu
Haitao Wang
Kangning Wang
Ruosong Wang
Raymond Chi-Wing Wong
Thomas Zeume
Wei Zhan
Florian Zuleger

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Rewritability in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics*†

Cristina Feier[1], Antti Kuusisto[2], and Carsten Lutz[3]

1 **Fachbereich Informatik, University of Bremen, Bremen, Germany**
`feier@uni-bremen.de`
2 **Fachbereich Informatik, University of Bremen, Bremen, Germany**
`kuusisto@uni-bremen.de`
3 **Fachbereich Informatik, University of Bremen, Bremen, Germany**
`lutz@uni-bremen.de`

Abstract

We study rewritability of monadic disjunctive Datalog programs, (the complements of) MMSNP sentences, and ontology-mediated queries (OMQs) based on expressive description logics of the $\mathcal{ALC}$ family and on conjunctive queries. We show that rewritability into FO and into monadic Datalog (MDLog) are decidable, and that rewritability into Datalog is decidable when the original query satisfies a certain condition related to equality. We establish 2NExpTime-completeness for all studied problems except rewritability into MDLog for which there remains a gap between 2NExpTime and 3ExpTime. We also analyze the shape of rewritings, which in the MMSNP case correspond to obstructions, and give a new construction of canonical Datalog programs that is more elementary than existing ones and also applies to non-Boolean queries.

1998 ACM Subject Classification F.4.1 Mathematical Logic, I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods, I.1.2 Algorithms, F.2.2 Non-numerical Algorithms and Problems, H.2.4 Systems, H.2.3 Languages

Keywords and phrases FO-Rewritability, MDDLog, MMSNP, DL, Ontology Mediated Queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.1

Category Invited Talk

1 Introduction

In data access with ontologies, the premier aim is to answer queries over incomplete and heterogeneous data while taking advantage of the domain knowledge provided by an ontology [17, 10]. Since traditional database systems are often unaware of ontologies, it is common to rewrite the emerging ontology-mediated queries (OMQs) into more standard database query languages. For example, the DL-Lite family of description logics (DLs) was designed specifically so that any OMQ $Q = (\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is a DL-Lite ontology, Σ a data signature, and q a conjunctive query, can be rewritten into an equivalent first-order (FO) query that can then be executed using a standard SQL database system [18, 2]. In more expressive ontology languages, it is not guaranteed that for every OMQ there is an equivalent FO query. For example, this is the case for DLs of the $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ families [38, 23], and for DLs of the expressive $\mathcal{ALC}$ family. In many members of the $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ families, however,

* A long version of the paper is available at `http://arxiv.org/abs/1701.02231`.
† The authors were funded by ERC grant 647289.

rewritability into monadic Datalog (MDLog) is guaranteed, thus enabling the use of Datalog engines for query answering. In $\mathcal{ALC}$ and above, not even Datalog-rewritability is generally ensured. Since ontologies emerging from practical applications tend to be structurally simple, though, there is reason to hope that (FO-, MDLog-, and Datalog-) rewritings do exist in many practically relevant cases even when the ontology is formulated in an expressive language. This has in fact been experimentally confirmed for FO-rewritability in the $\mathcal{EL}$ family of DLs [27], and it has led to the implementation of rewriting tools that, although incomplete, are able to compute rewritings in many practical cases [37, 28, 41].

Fundamental problems that emerge from this situation are to understand the exact limits of rewritability and to provide (complete) algorithms that decide the rewritability of a given OMQ and that compute a rewriting when it exists. These problems have been adressed in [9, 27, 8] for DLs from the $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ families. For DLs from the $\mathcal{ALC}$ family, first results were obtained in [11] where a connection between OMQs and constraint satisfaction problems (CSPs) was established that was then used to transfer decidability results from CSPs to OMQs. In fact, rewritability is an important topic in CSP (where it would be called definability) as it constitutes a central tool for analyzing the complexity of CSPs [24, 30, 22, 21]. In particular, rewritability of (the complement of) CSPs into FO and into Datalog is NP-complete [30, 4, 19], and rewritability into MDLog is NP-hard and in ExpTime [19]. In [11], these results were used to show that FO- and Datalog-rewritability of OMQs $(\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is formulated in $\mathcal{ALC}$ or a moderate extension thereof and q is an atomic query (AQ) of the form $A(x)$ is decidable and, in fact, NExpTime-complete. For MDLog-rewritability, one can show NExpTime-hardness and containment in 2ExpTime.

The aim of this paper is to study the above questions for OMQs where the ontology is formulated in an expressive DL from the $\mathcal{ALC}$ family and where the actual query is a conjunctive query (CQ) or a union of conjunctive queries (UCQ). As observed in [11], transitioning in OMQs from AQs to UCQs corresponds to the transition from CSP to its logical generalization MMSNP introduced by Feder and Vardi [24] and studied, for example, in [34, 32, 33, 12]. More precisely, while the OMQ language $(\mathcal{ALC}, \text{AQ})$ that consists of all OMQs $(\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is formulated in $\mathcal{ALC}$ and q is an AQ has the same expressive power as the complement of CSP (with multiple templates and a single constant), the OMQ language $(\mathcal{ALC}, \text{UCQ})$ has the same expressive power as the complement of MMSNP (with free variables)—which in turn is a notational variant of monadic disjunctive Datalog (MDDLog). It should be noted, however, that while all these formalisms are equivalent in expressive power, they differ significantly in succinctness [11]; in particular, the best known translation of OMQs into MMSNP/MDDLog involves a double exponential blowup. In contrast to the CSP case, FO-, MDLog-, and Datalog-rewritability of (the complement of) MMSNP sentences was not known to be decidable. In this paper, we establish decidability of FO- and MDLog-rewritability in $(\mathcal{ALC}, \text{UCQ})$ and related OMQ languages, in MDDLog, and the complement of MMSNP. We show that FO-rewritability is 2NExpTime-complete in all three cases, and that MDLog-rewritability is in 3ExpTime; a 2NExpTime lower bound was established in [15]. Let us discuss our results on FO-rewritability from three different perspectives. From the OMQ perspective, the transition from AQs to UCQs results in an increase of complexity from NExpTime to 2NExpTime. From the monadic Datalog perspective, adding disjunction (transitioning from monadic Datalog to MDDLog) results in a moderate increase of complexity from 2ExpTime [6] to 2NExpTime. And from the CSP perspective, the transition from CSPs to MMSNP results in a rather dramatic complexity jump from NP to 2NExpTime.

For Datalog-rewritability, we obtain only partial results. In particular, we show that Datalog-rewritability is decidable and 2NExpTime-complete for MDDLog programs that, in a certain technical sense made precise in the paper, *have equality*. For the general case, we only obtain a potentially incomplete procedure. It is well possible that the procedure is in fact complete, but proving this remains an open issue for now. These results also apply to analogously defined classes of MMSNP sentences and OMQs that have equality.

While we mainly focus on deciding whether a rewriting exists rather than actually computing it, we also analyze the shape that rewritings can take. Since the shape turns out to be rather restricted, this is important information for algorithms (complete or incomplete) that seek to compute rewritings. In the CSP/MMSNP world, this corresponds to analyzing obstruction sets for MMSNP, in the style of CSP obstructions [35, 16, 3] and not to be confused with colored forbidden patterns sometimes used to characterize MMSNP [34]. More precisely, we show that an OMQ $(\mathcal{T}, \Sigma, q)$ from $(\mathcal{ALC}, \text{UCQ})$ is FO-rewritable if and only if it is rewritable into a UCQ in which each CQ has treewidth $(1, \max\{2, n_q\})$, n_q the size of q;[1] similarly, the complement of an MMSNP sentence φ is FO-definable if and only if it admits a finite set of finite obstructions of treewidth $(1, k)$ where k is the diameter of φ (the maximum size of a negated conjunction in its body, in Feder and Vardi's terminology). We also show that $(\mathcal{T}, \Sigma, q)$ is MDLog-rewritable if and only if it is rewritable into an MDLog program of diameter $(1, \max\{2, n_q\})$; similarly, the complement of an MMSNP sentence φ is MDLog-definable if and only if it admits a (potentially infinite) set of finite obstructions of treewidth $(1, k)$ where k is the diameter of φ. For the case of rewriting into unrestricted Datalog, we give a new and direct construction of canonical Datalog-rewritings. It has been observed in [24] that for every CSP and all ℓ, k, it is possible to construct a canonical Datalog program Π of width ℓ and diameter k in the sense that if any such program is a rewriting of the CSP, then so is Π; moreover, even when there is no (ℓ, k)-Datalog rewriting, then Π is the best possible approximation of such a rewriting. The existence of canonical Datalog-rewritings for (the complement of) MMSNP sentences was already known from [13]. However, the construction given there is quite complex, proceeding via an infinite template that is obtained by applying an intricate construction due to Cherlin, Shelah, and Shi [20], which makes them rather hard to analyze. In contrast, our construction is elementary and essentially parallels the CSP case; it also applies to MMSNP formulas with free variables, where the canonical program takes a rather special form that involves *parameters*, similar in spirit to the parameters to least fixed-point operators in FO(LFP) [5].

Our main technical tool is the translation of an MMSNP sentence into a generalized CSP, i.e. a CSP in which there are multiple templates, exhibited by Feder and Vardi in [24]. The translation is not equivalence preserving and involves a double exponential blowup, but was designed so as to preserve complexity up to polynomial time reductions. Here, we are not so much interested in the complexity aspect, but rather in the semantic relationship between the original MMSNP sentence and the constructed CSP. It turns out that the translation does not quite preserve rewritability. In particular, when the original MMSNP sentence has a rewriting, then the natural way of constructing from it a rewriting for the CSP is sound only on instances of high girth. However, FO- and MDLog-rewritings that are sound on high girth (and unconditionally complete) can be converted into rewritings that are unconditionally sound (and complete). The same is true for Datalog-rewritings when the MMSNP sentence has equality, but it remains open whether it is true in the general case.

[1] What we mean here is that q has a tree decomposition in which every bag has at most $\max\{2, n_q\}$ elements and in which neighboring bag overlap in at most one element.

The structure of this paper is as follows. In Section 2, we give some preliminaries. In Section 3, we summarize the main properties of Feder and Vardi's translation of MMSNP into CSP. This is used in Section 4 to show that FO- and MDLog-rewritability of Boolean MDDLog programs and of the complement of MMSNP sentences is decidable, also establishing the announced complexity results. In Section 5, we analyze the shape of FO- and MDLog-rewritings and of obstructions for MMSNP sentences. In Section 6, we study Datalog-rewritability of MDDLog programs that have equality and construct canonical Datalog programs. Section 7 lifts the results from the Boolean case to the general case. Section 8 introduces OMQs and further lifts our results to this setting. We conclude in Section 9.

A long version of the paper is available at: `http://arxiv.org/abs/1701.02231`.

2 Preliminaries

A *schema* is a finite collection $\mathbf{S} = (S_1, \dots, S_k)$ of relation symbols with associated arity. An $\mathbf{S}$-*fact* is an expression of the form $S(a_1, \dots, a_n)$ where $S \in \mathbf{S}$ is an n-ary relation symbol, and $a_1, \dots, a_n$ are elements of some fixed, countably infinite set const of *constants*. For an n-ary relation symbol S, $\mathsf{pos}(S)$ is $\{1, \dots, n\}$. An $\mathbf{S}$-*instance* I is a finite set of $\mathbf{S}$-facts. The *active domain* $\mathsf{dom}(I)$ of I is the set of all constants that occur in a fact in I. For an instance I and a schema $\mathbf{S}$, we write $I|_{\mathbf{S}}$ to denote the restriction of I to the relations in $\mathbf{S}$.

A *tree decomposition* of an instance I is a pair $(T, (B_v)_{v \in V})$, where $T = (V, E)$ is an undirected tree and $(B_v)_{v \in V}$ is a family of subsets of $\mathsf{dom}(I)$ such that: for all $a \in \mathsf{dom}(I)$, $\{v \in V \mid a \in B_v\}$ is nonempty and connected in T; for every fact $R(a_1, \dots a_r)$ in I, there is a $v \in V$ such that $a_1, \dots, a_r \in B_v$. Unlike in the traditional setup [25], we are interested in two parameters of tree decompositions instead of only one. We call $(T, (B_v)_{v \in V})$ an (ℓ, k)-*tree decomposition* if for all $v, v' \in V$, $|B_v \cap B_{v'}| \leq \ell$ and $|B_v| \leq k$. An instance I *has treewidth* (ℓ, k) if it admits an (ℓ, k)-tree decomposition.

An instance I has a *cycle* of length n if it contains distinct facts $R_0(\mathbf{a}_0), \dots, R_{n-1}(\mathbf{a}_{n-1})$, $\mathbf{a}_i = a_{i,1} \cdots a_{i,m_i}$, and there exist $p_i, p_i' \in \mathsf{pos}(R_i)$, $0 \leq i < n$ such that: $p_i \neq p_i'$ for $1 \leq i \leq n$, and $a_{i,p_i'} = a_{i \oplus 1, p_{i \oplus 1}}$ for $0 \leq i < n$, where $\oplus$ denotes addition modulo n. The *girth* of I is the length of its shortest cycle and ∞ if it has no cycle (in which case we say that I is a *tree*).

A *constraint satisfaction problem (CSP)* is defined by an instance T over a schema $\mathbf{S}_E$, called *template*. The problem associated with T, denoted $\mathrm{CSP}(T)$, is to decide whether an input instance I over $\mathbf{S}_E$ admits a homomorphism to T, denoted $I \to T$. We use $\mathrm{coCSP}(T)$ to denote the complement problem, that is, deciding whether $I \not\to T$. A *generalized CSP* is defined by a set of templates S over the same schema $\mathbf{S}_E$ and asks for a homomorphism from the input I to at least one templates $T \in S$, denoted $I \to S$.

An *MMSNP sentence* θ over schema $\mathbf{S}_E$ has the form $\exists X_1 \cdots \exists X_n \forall x_1 \cdots \forall x_m \varphi$ with $X_1, \dots, X_n$ monadic second-order variables, $x_1, \dots, x_m$ first-order variables, and φ a conjunction of quantifier-free formulas of the form $\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta_1 \vee \cdots \vee \beta_m$ with $n, m \geq 0$, where each α_i takes the form $X_i(x_j)$ or $R(\mathbf{x})$ with $R \in \mathbf{S}_E$, and each β_i takes the form $X_i(x_j)$. The *diameter* of θ is the maximum number of variables in some implication in φ.

A *conjunctive query (CQ)* takes the form $\exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ where φ is a conjunction of relational atoms and $\mathbf{x}$, $\mathbf{y}$ denote tuples of variables; the equality relation may be used. Whenever convenient, we will confuse a CQ $\exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ with the set of atoms in φ. A *union of conjunctive queries (UCQ)* is a disjunction of CQs with the same free variables.

A *disjunctive Datalog rule* ρ has the form $S_1(\mathbf{x}_1) \vee \cdots \vee S_m(\mathbf{x}_m) \leftarrow R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$ where $n > 0$ and $m \geq 0$. When $m \leq 1$, the rule is a *Datalog rule*. We refer to $S_1(\mathbf{x}_1) \vee \cdots \vee S_m(\mathbf{x}_m)$ as the *head* of ρ, and to $R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$ as the *body*. Every variable that occurs in the head of a rule ρ is required to also occur in the body of ρ. A *(disjunctive)*

Datalog ((D)DLog) program Π is a finite set of (disjunctive) Datalog rules with a selected *goal relation* goal that does not occur in rule bodies and appears only in non-disjunctive *goal rules* $\mathsf{goal}(\mathbf{x}) \leftarrow R_1(\mathbf{x}_1) \wedge \cdots \wedge R_n(\mathbf{x}_n)$. The *arity of* Π is the arity of the goal relation; Π is *Boolean* if it has arity zero. Relation symbols that occur in the head of at least one rule of Π are *intensional (IDB) relations*, and all remaining relation symbols in Π are *extensional (EDB) relations*. relation. A (D)DLog program is called *monadic* or an *M(D)DLog program* if all its IDB relations with the possible exception of goal have arity at most one. The *size* of a DDLog program Π is the number of symbols needed to write it (where relation symbols and variables names count one), its *width* is the maximum arity of non-goal IDB relations used in it, and its *diameter* is the maximum number of variables that occur in a rule in Π. An (ℓ, k)*-DLog program* is a DLog program of width ℓ and diameter k. Sometimes we omit k and speak of ℓ*-DLog programs*.

For Π an n-ary MDDLog program over schema $\mathbf{S}_E$, an $\mathbf{S}_E$-instance I, and $a_1, \ldots, a_n \in \mathsf{dom}(I)$, we write $I \models \Pi(a_1, \ldots, a_n)$ if $\Pi \cup I \models \mathsf{goal}(a_1, \ldots a_n)$ where variables in all rules of Π are universally quantified and thus Π is a set of first-order (FO) sentences. A query q over $\mathbf{S}_E$ of arity n is:

- *sound for* Π if for all $\mathbf{S}_E$-instances I and $\mathbf{a} \in \mathsf{dom}(I)$, $I \models q(\mathbf{a})$ implies $I \models \Pi(\mathbf{a})$;
- *complete for* Π if for all $\mathbf{S}_E$-instances I and $\mathbf{a} \in \mathsf{dom}(I)$, $I \models \Pi(\mathbf{a})$ implies $I \models q(\mathbf{a})$;
- a *rewriting of* Π if it is sound for Π and complete for Π.

To additionally specify the syntactic shape of q, we speak of a UCQ-rewriting, an MDLog-rewriting, and so on. An *FO-rewriting* takes the form of an FO-query that uses only relations from the EDB schema and possibly equality, but neither constants nor function symbols. We say that Π is Q*-rewritable* if there is a Q-rewriting of Π, for $Q \in \{\text{FO, UCQ, MDLog}\}$.

It was shown in [11] that the complement of an MMSNP sentence can be translated into an equivalent Boolean MDDLog program in polynomial time and vice versa; moreover, the transformations preserve diameter and all other parameters relevant for this paper. Thus, we will not explicitly distinguish between Boolean MDDLog and (the complement of) MMSNP.

3 From MDDLog via Simple MDDLog to CSPs

Feder and Vardi show how to translate an MMSNP sentence into a generalized CSP that has the same complexity up to polynomial time reductions [24]. The generalized CSP has a different schema to the original MMSNP sentence and is thus not equivalent to it. We use this translation to reduce rewritability problems for MDDLog to corresponding problems for CSPs. In this section, we sum up the results obtained in [24] that are relevant for our reduction and refer to the long version for more details.

To capture the relation between the schema of an MDDLog program and the generalized CSP constructed from it, we introduce the notion of an aggregation schema. Let $\mathbf{S}_E$ be a schema. A schema $\mathbf{S}'_E$ is a k*-aggregation schema* for $\mathbf{S}_E$ if its relations have the form $R_{q(\mathbf{x})}$ where $q(\mathbf{x})$ is a CQ over $\mathbf{S}_E$ without quantified variables and the arity of $R_{q(\mathbf{x})}$ is identical to the number of variables in $\mathbf{x}$, which is at most k. For I an $\mathbf{S}_E$-instance, its *corresponding* $\mathbf{S}'_E$*-instance* I' consists of all facts $R_{q(\mathbf{x})}(\mathbf{a})$ such that $I \models q(\mathbf{a})$. Conversely, for I' an $\mathbf{S}'_E$-instance, the *corresponding* $\mathbf{S}_E$*-instance* I consists of all facts $S(\mathbf{b})$ such that $R_{q(\mathbf{x})}(\mathbf{a}) \in I'$ and $S(\mathbf{b})$ is a conjunct of $q(\mathbf{a})$.

▶ **Example 1.** Let $\mathbf{S}_E = \{r\}$, r a binary relation, $q(\mathbf{x}) = r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1)$ where $\mathbf{x} = (x_1, x_2, x_3)$, and $\mathbf{S}'_E = \{R_{q(\mathbf{x})}\}$. If I' is the $\mathbf{S}'_E$-instance $\{R_q(a, a', c'), R_q(b, b', a'), R_q(c, c', b')\}$, then its corresponding $\mathbf{S}_E$-instance I is $\{r(a, a'), r(a', c'), r(c', a), r(b, b'),$

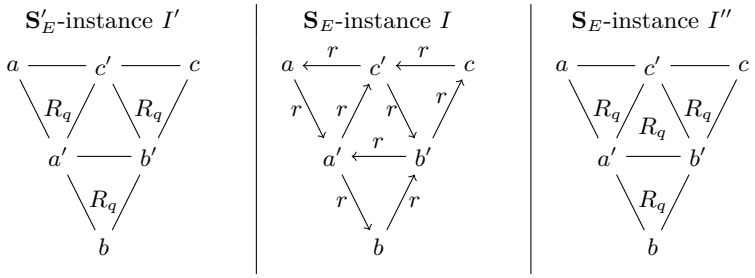


Figure 1 Translating an $\mathbf{S}'_E$-instance into an $\mathbf{S}_E$-instance and vice versa.

$r(b', a'), r(a', b), r(c, c'), r(c', b), r(b', c)\}$. Note that the $\mathbf{S}'_E$-instance I'' that corresponds to I is a strict superset of I': it contains additional facts such as $R_q(c', b', a')$, see Figure 1.

The translation in [24] consists of two steps. The first step is to transform the given Boolean MDDLog program Π into a Boolean MDDLog program Π_S over a suitable aggregation schema $\mathbf{S}'_E$. Π_S is of a restricted syntactic form, called *simple*, which means that each of its rules contains at most one EDB atom, that this atom contains all variables of the rule body, each variable exactly once, and that rules without an EDB atom contain at most a single variable. To achieve this, Π is first saturated by adding all rules that can be obtained from a rule in Π by identifying variables; then Π is rewritten in an equivalence-preserving way so that all rule bodies are biconnected, introducing fresh unary and nullary IDBs as needed. Finally, for each rule body the conjunction $q(\mathbf{x})$ of its EDB atoms is replaced with a single EDB atom $R_{q(\mathbf{x})}(\mathbf{x})$, additionally taking care of interactions between the new EDB relations that arise, e.g. when we have two relations $R_{q(\mathbf{x})}$ and $R_{p(\mathbf{x})}$ such that $q(\mathbf{x})$ is contained in $p(\mathbf{x})$ (in the sense of query containment).

▶ **Theorem 2** ([24]). *Given a Boolean MDDLog program Π over EDB schema $\mathbf{S}_E$ of diameter k and size n, one can construct a simple Boolean MDDLog program Π_S over a k-aggregation schema $\mathbf{S}'_E$ for $\mathbf{S}_E$ such that*

1. *If I is an $\mathbf{S}_E$-instance and I' the corresponding $\mathbf{S}'_E$-instance, then $I \models \Pi$ iff $I' \models \Pi_S$;*
2. *If I' is an $\mathbf{S}'_E$-instance and I the corresponding $\mathbf{S}_E$-instance, then*
   (a) *$I' \models \Pi_S$ implies $I \models \Pi$;*
   (b) *$I \models \Pi$ implies $I' \models \Pi_S$ if the girth of I' exceeds k.*

The size of Π_s and the cardinality of $\mathbf{S}'_E$ are bounded by $2^{p(k \cdot \log n)}$, p a polynomial. The construction takes time polynomial in the size of Π_S.

Note that Π_S is equivalent to Π only on instances whose girth exceeds k, the maximal arity of a relation symbol in $\mathbf{S}'_E$.

In the second step, the simple MDDLog program Π_S is translated into a generalized CSP whose complement is equivalent to Π_S. Informally, one introduces one template for every 0-type (set of nullary IDBs), each template contains one constant for every 1-type (set of at most unary IDBs) that is compatible with the 0-type and interprets the EDB relations in a maximal way so that all rules in Π are satisfied (when interpreting the IDBs true as suggested by the 1-types).

▶ **Theorem 3** ([24]). *Let Π be a simple Boolean MDDLog program over EDB schema $\mathbf{S}_E$ and with IDB schema $\mathbf{S}_I$, m the maximum arity of relations in $\mathbf{S}_E$. Then there exists a set of templates S_Π over $\mathbf{S}_E$ such that*

1. *Π is equivalent to coCSP(S_Π);*

2. $|S_\Pi| \leq 2^{|\mathbf{S}_I|}$ *and* $|T| \leq |\mathbf{S}_E| \cdot 2^{m|\mathbf{S}_I|}$ *for each* $T \in S_\Pi$*;*

The construction takes time polynomial in $\sum_{T \in S_\Pi} |T|$.

4 FO- and MDLog-Rewritability of Boolean MDDLog Programs

We exploit the translation described in the previous section to lift the decidability of FO-rewritability and of MDLog-rewritability from coCSPs to Boolean MDDLog, and thus also to MMSNP. In the case of FO, we obtain tight 2NExpTime complexity bounds. For MDLog, the exact complexity remains open (as in the CSP case), between 2NExpTime and 3ExpTime.

We start with observing that FO-rewritability and MDLog-rewritability are more closely related than one might think at first glance. In fact, every MDLog-rewriting can be viewed as an infinitary UCQ-rewriting and, by Rossman's homomorphism preservation theorem [39], FO-rewritability of a Boolean MDDLog program coincides with (finitary) UCQ-rewritability. The latter is true also in the non-Boolean case.

▶ **Proposition 4.** *An MDDLog program* Π *is FO-rewritable iff it is UCQ-rewritable.*

For utilizing the translation of Boolean MDDLog programs to generalized CSPs in the intended way, the interesting aspect is to deal with the translation of a Boolean MDDLog program Π into a simple program Π_S stated in Theorem 2, since it is not equivalence preserving. The following lemma relates rewritings of Π to rewritings of Π_S.

▶ **Lemma 5.** *Let* Π *be a Boolean MDDLog program of diameter* k*,* Π_S *as in Theorem 2, and* $\mathcal{Q} \in \{\mathit{UCQ}, \mathit{MDLog}, \mathit{DLog}\}$*. Then*

1. *every* $\mathcal{Q}$*-rewriting of* Π_S *can effectively be converted into a* $\mathcal{Q}$*-rewriting of* Π*;*
2. *every* $\mathcal{Q}$*-rewriting of* Π *can effectively be converted into a* $\mathcal{Q}$*-rewriting of* Π_S *that is (i) sound on instances of girth exceeding* k *and (ii) complete.*

Proof (Sketch). We only give the constructions for the case $\mathcal{Q} = \text{UCQ}$ and refer to the long version for the other cases and for correctness proofs. For Point 1, let q_{Π_S} be a UCQ-rewriting of Π_S. Then we obtain a UCQ-rewriting of Π by replacing every atom $R_{q(\mathbf{x})}(\mathbf{y})$ with $q[\mathbf{y}/\mathbf{x}]$, that is, with the result of replacing the variables $\mathbf{x}$ in $q(\mathbf{x})$ with the variables $\mathbf{y}$.

For Point 2, let q_Π be a UCQ-rewriting of Π. We obtain a UCQ-rewriting of Π_S by taking the UCQ that consists of all CQs which can be obtained as follows:

1. choose a CQ $q(\mathbf{x})$ from q_Π, identify variables in q to obtain a CQ $q'(\mathbf{x}')$, and choose a partition $q_1(\mathbf{x}_1), \ldots, q_n(\mathbf{x}_n)$ of $q'(\mathbf{x}')$;
2. for each $i \in \{1, \ldots, n\}$, choose a relation $R_{p(\mathbf{z})}$ from the EDB schema of Π_S and a vector $\mathbf{y}$ of $|\mathbf{z}|$ variables (repeated occurrences allowed) that are either from $\mathbf{x}_i$ or do not occur in $\mathbf{x}'$ such that $q_i(\mathbf{x}_i) \subseteq p[\mathbf{y}/\mathbf{z}]$; then replace $q_i(\mathbf{x}_i)$ in $q'(\mathbf{x}')$ with the single atom $R_{p(\mathbf{z})}(\mathbf{y})$. ◀

Point 2 of Lemma 5 only yields a rewriting of Π_S on $\mathbf{S}'_E$-instances of high girth. We next show that for CSPs, the existence of a $\mathcal{Q}$-rewriting on instances of high girth, $\mathcal{Q} \in \{\text{UCQ}, \text{MDLog}\}$, implies the existence of a $\mathcal{Q}$-rewriting that works on instances of unrestricted girth. Whether the same is true for $\mathcal{Q} = \text{Datalog}$ remains as an open problem.

▶ **Lemma 6.** *Let* S *be a set of templates over schema* $\mathbf{S}_E$*,* $g \geq 0$*, and* $\mathcal{Q} \in \{\mathit{UCQ}, \mathit{MDLog}\}$*. If coCSP(*$S$*) is* $\mathcal{Q}$*-rewritable on instances of girth exceeding* g*, then it is* $\mathcal{Q}$*-rewritable.*

The proof of Lemma 6 uses a well-known combinatorial lemma that goes back to Erdös and was adapted to CSPs by Feder and Vardi. Putting together Theorem 2 and 3, Proposition 4, and Lemmas 5 and 6, we obtain the following reductions of rewritability of Boolean MDDLog programs to CSP rewritability.

▶ **Proposition 7.** *Every Boolean MDDLog program* Π *can be converted into a set of templates* S_Π *such that*

1. Π *is* $\mathcal{Q}$*-rewritable iff* S_Π *is* $\mathcal{Q}$*-rewritable for every* $\mathcal{Q} \in \{FO, UCQ, MDLog\}$*;*
2. *every* $\mathcal{Q}$*-rewriting of* Π *can be effectively translated into a* $\mathcal{Q}$*-rewriting of* S_Π *and vice versa, for every* $\mathcal{Q} \in \{UCQ, MDLog\}$*.*
3. $|S_\Pi| \leq 2^{2^{p(n)}}$ *and* $|T| \leq 2^{2^{p(n)}}$ *for each* $T \in S_\Pi$*,* n *the size of* Π *and* p *a polynomial.*

The construction takes time polynomial in $\sum_{T \in S_\Pi} |T|$*.*

FO-rewritability of CSPs is NP-complete [30] and it was observed in [11] that the upper bound lifts to generalized CSPs. MDLog-rewritability of coCSPs is NP-hard and in ExpTime [19]. We show in the long version that this upper bound lifts to generalized coCSPs. Together with Proposition 7 and the lower bounds from [15], we obtain the following:

▶ **Theorem 8.** *For Boolean MDDLog programs and the complement of MMSNP sentences,*

1. *FO-rewritability (equivalently: UCQ-rewritability) is 2*NExpTime*-complete;*
2. *MDLog-rewritability is in 3*ExpTime *(and 2*NExpTime*-hard).*

5 Shape of Rewritings and Obstructions for MMSNP sentences

An important first step towards the design of practical algorithms that compute rewritings (when they exist) is to analyze the shape of the rewritings. In the case of CSPs, both UCQ- and MDLog-rewritings are known to be of a rather restricted shape, far from exploiting the full expressive power of the target languages: any FO-rewritable CSP has a UCQ-rewriting that consists of tree-shaped CQs and any MDLog-rewritable CSP has an MDLog-rewriting in which each rule has at mosrt a single EDB atom. In this section, we establish corresponding results for Boolean MDDLog.

Exploiting the results concerning the shape of UCQ- and MDLog- rewritings for CSPs and the constructions from the proof of Point 2 of Lemma 5, one can show the following.

▶ **Theorem 9.** *Let* Π *be a Boolean MDDLog program of diameter* k*. Then*

1. *if* Π *is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth* $(1, k)$*;*
2. *if* Π *is MDLog-rewritable, then it has an MDLog-rewriting of diameter* k*.*

In a sense, the concrete bound k in Points 1 and 2 of Theorem 9 is quite remarkable. Point 2 says, for example, that when eliminating disjunction from a Boolean MDDLog program, it never pays off to increase the diameter.

For CSPs, FO- and MDLog-rewritability is closely related to the theory of obstructions: an *obstruction set* $\mathcal{O}$ for a CSP template T over schema $\mathbf{S}_E$ is a set of instances over the same schema, called *obstructions*, such that for any $\mathbf{S}_E$-instance I, we have $I \not\to T$ iff $O \to I$ for some $O \in \mathcal{O}$. A lot is known about CSP obstructions. For example, T is FO-rewritable if and only if it has a finite obstruction set [3] if and only if it has a finite obstruction set that consists of finite trees [36], and T is MDLog-rewritable if and only if it has a (potentially infinite) obstruction set that consists of finite trees [24].

Here we establish similar results for the case of MMSNP. Obstruction sets for MMSNP are defined in the obvious way: an *obstruction set* $\mathcal{O}$ for an MMSNP sentence θ over schema $\mathbf{S}_E$ is a set of instances over the same schema such that for any $\mathbf{S}_E$-instance I, we have $I \not\models \theta$ iff $O \to I$ for some $O \in \mathcal{O}$. The following result, which essentially is a consequence of Point 1 of Theorem 9, characterizes FO-rewritability of MMSNP sentences in terms of obstruction sets.

▶ **Corollary 10.** *For every MMSNP sentence θ, the following are equivalent:*
1. *θ is FO-rewritable;*
2. *θ has a finite obstruction set;*
3. *θ has a finite set of finite obstructions of treewidth $(1, k)$.*

We now turn to MDLog-rewritability.

▶ **Proposition 11.** *Let θ be an MMSNP sentence of diameter k. Then $\neg\theta$ is MDLog-rewritable iff θ has a set of obstructions (equivalently: finite obstructions) that are of treewidth $(1, k)$.*

We remark that the results in [13] almost give Proposition 11, but do not seem to deliver a concrete bound on the parameter k of the treewidth of obstruction sets.

6 Datalog-Rewritability of Boolean MDDLog Programs and Canonical Datalog Programs

We study the Datalog-rewritability of Boolean MDDLog programs. In contrast to the case of FO- or MDLog-rewritings, we obtain a procedure that is sound, but whose completeness remains an open problem. We can show, however, that the procedure is complete for MDDLog programs that have equality, a condition that is defined in detail below. We also give a new and direct construction of canonical Datalog-rewritings of Boolean MDDLog programs (equivalently: the complements of MMSNP sentences), bypassing the construction of infinite templates which involves the application of a non-trivial construction due to Cherlin, Shelah, and Shi [13, 20].

6.1 Datalog-Rewritability of Boolean MDDLog Programs

We say that an MDDLog program Π *has equality* if its EDB schema includes the distinguished binary relation eq, Π contains the rules $P(x) \wedge \mathsf{eq}(x, y) \to P(y)$ and $P(y) \wedge \mathsf{eq}(x, y) \to P(x)$ for each IDB relation P, and these are the only rules that mention eq. For an MDDLog program Π that does not have equality, we use $\Pi^{=}$ to denote the extension of Π with the fresh EDB relation eq and the above rules. If Π has equality, then $\Pi^{=}$ simply denotes Π. Clearly, a DLog-rewriting of $\Pi^{=}$ can be converted into a DLog-rewriting of Π by dropping all rules that use the relation eq. This gives the following lemma.

▶ **Lemma 12.** *For any MDDLog program Π, DLog-rewritability of $\Pi^{=}$ implies DLog-rewitability of Π.*

It remains an interesting open question whether the converse of Lemma 12 holds.

A CSP template T *has equality* if its schema includes the distinguished binary relation eq and T interprets eq as the relation $\{(a, a) \mid a \in \mathsf{dom}(T)\}$. It can be verified that when an MDDLog program that has equality is converted into a generalized CSP based on a set of templates S_Π according to Theorems 2 and 3 (using the concrete constructions in the long version), then all templates in S_Π have equality. The interesting aspect of having equality is that it allows us to establish a counterpart of Lemma 6 also for Datalog-rewritability.

▶ **Lemma 13.** *Let S be a set of templates over schema $\mathbf{S}_E$ that have equality, and let $g \geq 0$. If coCSP(S) is DLog-rewritable on instances of girth exceeding g, then it is DLog-rewritable.*

Proof (Sketch). With every $\mathbf{S}_E$-instance I and $g \geq 0$, we associate an $\mathbf{S}_E$-instance I^g of girth exceeding g such that for any template T over $\mathbf{S}_E$ that has equality, $I^g \to T$ iff $I \to T$. In fact, I^g is obtained from I by duplicating domain elements, and introducing chains of

equality atoms. Then, for every DLog rewriting Γ of coCSP(S) on instances of girth exceeding g, it is possible to construct a DLog program Γ' such that $I \models \Gamma'$ iff $I^g \models \Gamma$. Intuitively, when executed over I, Γ' mimics the execution of Γ over I^g. Clearly, Γ' is then a DLog-rewriting of coCSP(S) on unrestricted instances. ◀

DLog-rewritability of CSPs is NP-complete [4, 19] and it was observed in [11] that this result lifts to generalized CSPs. It thus follows from Theorems 2 and 3 and Lemma 13 that DLog-rewritability of Boolean MDDLog programs that have equality is decidable in 2NExpTime. It is straightforward to verify that the 2NExpTime lower bound for DLog-rewritability of MDDLog programs from [15] applies also to programs that have equality.

▶ **Theorem 14.** *For Boolean MDDLog programs that have equality, DLog-rewritability is 2*NExpTime*-complete.*

MDDLog programs obtained from OMQs typically do not have equality. Due to Lemma 12, though, we obtain a sound but possibly incomplete algorithm for deciding DLog-rewritability of an unrestricted MDDLog program Π by first replacing it with $\Pi^=$ and then deciding DLog-rewritability as per Theorem 14. We speculate that this algorithm is actually complete. Note that for CSPs, it is known that adding equality preseves DLog-rewritability [31], and completeness of our algorithm is equivalent to an analogous result holding for MDDLog.

6.2 Canonical Datalog-Rewritings

For constructing actual DLog-rewritings instead of only deciding their existence, *canonical Datalog programs* play an important role. Feder and Vardi show that for every CSP template T and all $\ell, k > 0$, once can construct an (ℓ, k)-Datalog program that is canonical for T in the sense that if there is any (ℓ, k)-Datalog program which is equivalent to the complement of T, then the canonical one is [24]. In this section, we show that there are similarly simple canonical Datalog programs for Boolean MDDLog. Note that the existence of canonical Datalog programs for MMSNP (and thus for Boolean MDDLog) is already known from [13]. However, the construction given there is rather complex, proceeding via an infinite template and exploiting that it is ω-categorial. This makes it hard to analyze the exact structure and size of the resulting canonical programs. Here, we define canonical Datalog programs for Boolean MDDLog programs in a more elementary way.

Let $0 \leq \ell < k$, and let Π be a Boolean MDDLog program over EDB schema $\mathbf{S}_E$ and with IDB relations from $\mathbf{S}_I$. We first convert Π into a DDLog program Π' that is equivalent to Π on instances of treewidth (ℓ, k). Unlike Π, the new program Π' is no longer monadic. We start with a preliminary. With every DDLog rule $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ where $q(\mathbf{x})$ is of treewidth (ℓ, k) and every (ℓ, k)-tree decomposition $(T, (B_v)_{v \in V})$ of $q(\mathbf{x})$, we associate a set of *rewritten rules* constructed as follows. Choose a root v_0 of T, thus inducing a direction on the undirected tree T. We write $v \prec v'$ if v' is a successor of v in T and use $\mathbf{x}_{v'}$ to denote $B_v \cap B_{v'}$. For all $v \in V \setminus \{v_0\}$ such that $|\mathbf{x}_v| = m$, introduce a fresh m-ary IDB relation Q_v; note that $m \leq \ell$. Now, the set of rewritten rules contains one rule for each $v \in V$. For $v \neq v_0$, the rule is

$$p_v(\mathbf{y}_v) \vee Q_v(\mathbf{x}_v) \leftarrow q(\mathbf{x})|_{B_v} \wedge \bigwedge_{v \prec v'} Q_{v'}(\mathbf{x}_{v'})$$

where $p_v(\mathbf{y}_v)$ is the sub-disjunction of $p(\mathbf{y})$ that contains all disjuncts $P(\mathbf{z})$ with $\mathbf{z} \subseteq B_v$. For v_0, we include the same rule, but use only $p_v(\mathbf{y}_v)$ as the head. The set of rewritten rules associated with $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ is obtained by taking the union of the rewritten rules associated with $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ and any $(T, (B_v)_{v \in V})$.

The DDLog program Π' is constructed from Π as follows:

1. first extend Π with all rules that can be obtained from a rule in Π by identifying variables;
2. then delete all rules with $q(\mathbf{x})$ not of treewidth (ℓ, k) and replace every rule $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ with $q(\mathbf{x})$ of treewidth (ℓ, k) with the rewritten rules associated with it.

It can be verified that Π' satisfies the following conditions:

(i) Π' is sound for Π, that is, for all $\mathbf{S}_E$-instances I, $I \models \Pi'$ implies $I \models \Pi$;

(ii) Π' is complete for Π on $\mathbf{S}_E$-instances of treewidth (ℓ, k), that is, for all such instances I, $I \models \Pi$ implies $I \models \Pi'$.

Let $\mathbf{S}'_I$ denote the additional IDB relations in Π'. We now construct the canonical (ℓ, k)-DLog program Γ^c for Π. Fix constants $a_1, \dots, a_\ell$. For $\ell' \leq \ell$, we use $\mathfrak{I}_{\ell'}$ to denote the set of all $\mathbf{S}_I \cup \mathbf{S}'_I$-instances with domain $\mathbf{a}_{\ell'} := a_1, \dots, a_{\ell'}$. The program uses ℓ'-ary IDB relations P_M, for all $\ell' \leq \ell$ and all $M \subseteq \mathfrak{I}_{\ell'}$. It contains all rules $q(\mathbf{x}) \to P_M(\mathbf{y})$, $M \subseteq \mathfrak{I}_{\ell'}$, that satisfy the following conditions:

1. $q(\mathbf{x})$ contains at most k variables;
2. for every extension J of the $\mathbf{S}_E$-instance $I_q|_{\mathbf{S}_E}$ with $\mathbf{S}_I \cup \mathbf{S}'_I$-facts such that
   (a) J satisfies all rules of Π' and does not contain $\mathsf{goal}()$ and
   (b) for each $P_N(\mathbf{z}) \in q$, $N \subseteq \mathfrak{I}_{\ell''}$, there is an $L \in N$ such that $L[\mathbf{z}/\mathbf{a}_{\ell''}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{z}$ there is an $L \in M$ such that $L[\mathbf{y}/\mathbf{a}_{\ell'}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{y}$

where $L[\mathbf{x}/\mathbf{a}]$ denotes the result of replacing the constants in $\mathbf{a}$ with the variables in $\mathbf{x}$ (possibly resulting in identifications) and where $J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{x}$ denotes the simultaneous restriction of J to schema $\mathbf{S}_I \cup \mathbf{S}'_I$ and constants $\mathbf{x}$.[2] We also include all rules of the form $P_\emptyset(\mathbf{x}) \to \mathsf{goal}()$, $P_\emptyset$ of any arity from 0 to ℓ.

The following theorem says that the canonical program is indeed canonical in the desired sense. For two Boolean DLog programs Π_1, Π_2 over the same EDB schema $\mathbf{S}_E$, we write $\Pi_1 \subseteq \Pi_2$ if for every $\mathbf{S}_E$-instance I, $I \models \Pi_1$ implies $I \models \Pi_2$.

▶ **Theorem 15.** *Let Π be a Boolean MDDLog program, $0 \leq \ell \leq k$, and Γ^c the canonical (ℓ, k)-DLog program for Π. Then*

1. *$\Gamma \subseteq \Gamma^c$ for every (ℓ, k)-DLog program Γ that is sound for Π;*
2. *Π is (ℓ, k)-DLog-rewritable iff Γ^c is a DLog-rewriting of Π.*

Note that by Point 2 of Theorem 15, the canonical (ℓ, k)-DLog program for an MDDLog program Π is interesting even if Π is not rewritable into an (ℓ, k)-DLog program as it is the strongest sound (ℓ, k)-DLog approximation of Π.

7 Non-Boolean MDDLog Programs

We lift the results about the complexity of rewritability, about canonical DLog programs, and about the shape of rewritings and obstructions from the case of Boolean MDDLog programs to the non-Boolean case. For all of this, a certain extension of (ℓ, k)-Datalog programs with parameters plays a central role. We thus begin by introducing these extended programs.

7.1 Deciding Rewritability

An *(ℓ, k)-Datalog program with n parameters* is an n-ary $(\ell + n, k + n)$-Datalog program in which all IDBs have arity at least n and where in every rule, all IDB atoms agree on the

[2] We could additionally demand that M is minimal so that Condition 2 is satisfied, but this is not strictly required.

variables used in the last n positions (both in rule bodies and heads and including the goal IDB). The last n positions of IDBs are called *parameter positions*. To visually separate the parameter positions from the non-distinguished positions, we use "|" as a delimiter, writing e.g. $P(x_1, x_2 \,|\, y_1, y_1, y_2) \leftarrow Q(y_1 \,|\, y_1, y_1, y_2) \wedge R(x_1, y_1, y_2, x_2)$ where P, Q are IDB, R is EDB, and there are three parameter positions. Note that, by definition, all variable positions in goal atoms are parameter positions.

▶ **Example 16.** The following is an MDLog program with one parameter that returns all constants which are on an R-cycle, R a binary EDB relation:

$$P(y \,|\, x) \leftarrow R(x \,|\, y); \qquad P(z \,|\, x) \leftarrow P(y \,|\, x) \wedge R(y, z); \qquad \mathsf{goal}(x) \leftarrow P(x \,|\, x)$$

Parameters in Datalog programs play a similar role as parameters to least fixed-point operators in FO(LFP), see for example [5] and references therein. The program in Example 16 is not definable in MDLog without parameters, which shows that adding parameters increases expressive power. But although (ℓ, k)-DLog programs with n parameters are $(\ell + n, k + n)$-DLog programs, one should think of them as a mild generalization of (ℓ, k)-programs.

To lift decidability and complexity results from the Boolean case to the non-Boolean case, we show that rewritability of an n-ary MDDLog program into (ℓ, k)-DLog with n parameters can be Turing reduced to rewritability of Boolean MDDLog programs into (ℓ, k)-DLog (without parameters). Note that the case $\ell = 0$ is about UCQ-rewritability (and thus about FO-rewritability) since 0-DLog programs (with and without parameters) are an alternative presentation of UCQs.

The reduction proceeds in two steps: first, rewritability of an n-ary MDDLog program into (ℓ, k)-DLog with n parameters is Turing reduced to rewritability of Boolean MDDLog programs with constants into (ℓ, k)-DLog with constants; then (ℓ, k)-DLog rewritability of a Boolean MDDLog program with constants is reduced to (ℓ, r)-DLog rewritability of a Boolean MDDLog program with constants, where r is the maximum number of occurrences of variables in a rule body of the program with constants. We can now lift the complexity results from Theorems 8 and 14 to the non-Boolean case.

▶ **Theorem 17.** *In MDDLog,*

1. *FO-rewritability (equivalently: UCQ-rewritability) is 2*NExpTime*-complete;*
2. *rewritability into MDLog with parameters is in 3*ExpTime *(and 2*NExpTime*-hard);*
3. *DLog-rewritability is 2*NExpTime*-complete for programs that have equality.*

In view of Point 2, we remark that for non-Boolean MDDLog programs Π, MDLog with parameters is in a sense a more natural target for rewriting than MDLog without parameters.

▶ **Example 18.** The following MDDLog program is rewritable into the MDLog program with parameters from Example 16, but not into an MDLog program without parameters:

$$P_0(x) \vee P_1(y) \leftarrow R(x, y); \quad \mathsf{goal}(x) \leftarrow P_0(x;) \quad P_1(y) \leftarrow P_1(x) \wedge R(x, y); \quad \mathsf{goal}(x) \leftarrow P_1(x)$$

MDLog with parameters also enjoys similarly nice properties as standard MDLog. For example, containment is decidable. This follows from [40, 14] where generalizations of MDLog with parameters are studied, the actual parameters being represented by constants. We also remark that Theorem 17 remains true even when we admit constants in MDDLog programs and that, as another consequence of our reductions, rewritability of MDDLog programs into DLog programs with parameters is decidable if and only if DLog-rewritability of Boolean MDDLog programs is decidable.

7.2 Canonical Datalog-Rewritings

We now turn our attention to canonical DLog-rewritings for non-Boolean MDDLog programs. For any n-ary MDDLog program Π, and $\ell < k$, we construct a *canonical* (ℓ, k)*-DLog program with* n *parameters*. The construction is a refinement of the one from the Boolean case.

We start with some preliminaries. An *n-marked instance* is an instance I endowed with n (not necessarily distinct) distinguished elements $\mathbf{c} = c_1, \ldots, c_n$. An (ℓ, k)*-tree decomposition with* n *parameters* of an n-marked instance $(I, \mathbf{c})$ is an $(\ell + m, k + m)$-tree decomposition of I, m the number of distinct constants in $\mathbf{c}$, in which every bag B_v contains all constants from $\mathbf{c}$. An n-marked instance *has treewidth* (ℓ, k) *with* n *parameters* if it admits an (ℓ, k)-tree decomposition with n parameters.

We first convert Π into a DDLog program Π' that is equivalent to Π on instances of bounded treewidth. The construction is identical to the Boolean case (first variable identification, then rewriting) except that, in the rewriting step,

1. we use treewidth $(\ell + n, k + n)$ in place of treewidth (ℓ, k); consequently, the arity of the freshly introduced IDB relations may also be up to $\ell + n$;
2. for goal rules, all head variables occur in the root bag of the tree decomposition (they can then be treated in the same way as a Boolean goal rule despite the n-ary head relation).

It can be verified that Π' is sound for Π. It is complete for Π only on n-marked instances of treewidth (ℓ, k) with n parameters: for all such instances $(I, \mathbf{c})$, $I \models \Pi[\mathbf{c}]$ implies $I \models \Pi'[\mathbf{c}]$.

Let $\mathbf{S}'_I$ denote the new IDB relations in Π'. We now construct the canonical (ℓ, k)-DLog program with n parameters Γ^c. Fix constants $a_1, \ldots, a_\ell$, $b_1, \ldots, b_n$ and let $\mathfrak{I}_{\ell'+n}$ denote the set of all $\mathbf{S}_I \cup \mathbf{S}_{I'}$-instances with domain $\mathbf{a}_{\ell'+n} := a_1, \ldots, a_{\ell'}, b_1, \ldots, b_n$. The program uses $\ell' + n$-ary IDB relations P_M, for all $\ell' \leq \ell$ and all $M \subseteq \mathfrak{I}_{\ell'+n}$. It contains all rules $q(\mathbf{x}) \to P_M(\mathbf{y} \,|\, \mathbf{x}_p)$, $M \subseteq \mathfrak{I}_{\ell'+n}$, that satisfy the following conditions:

1. $q(\mathbf{x})$ contains at most $k + n$ variables;
2. in every extension J of the $\mathbf{S}_E$-instance $I_q|_{\mathbf{S}_E}$ with $\mathbf{S}_I \cup \mathbf{S}'_I$-facts such that
   - **(a)** J satisfies all rules of Π' and does not contain $\mathsf{goal}(\mathbf{x}_p)$ and
   - **(b)** for each $P_N(\mathbf{z} \,|\, \mathbf{x}_p) \in q$, $N \subseteq \mathfrak{I}_{\ell''+n}$, there is an $L \in N$ such that $L[\mathbf{z}\mathbf{x}_p/\mathbf{a}_{\ell''+n}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{z}$

   there is an $L \in M$ such that $L[\mathbf{y}\mathbf{x}_p/\mathbf{a}_{\ell'+n}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{y}$

We also include all rules of the form $P_\emptyset(\mathbf{y} \,|\, \mathbf{x}_p) \to \mathsf{goal}(\mathbf{x}_p)$. This finishes the construction. It is straightforward to verify that Γ^c is sound for Π and complete in the same sense as Π'. We obtain the following generalization of Theorem 15.

▶ **Theorem 19.** *Let Π be an n-ary MDDLog program, $0 < \ell \leq k$, and Γ^c the canonical (ℓ, k)-DLog program with n parameters associated with Π. Then*

1. *$\Gamma \subseteq \Gamma^c$ for every (ℓ, k)-DLog program Γ that is sound for Π;*
2. *Π is rewritable into (ℓ, k)-DLog with n parameters iff Γ^c is a rewriting of Π.*

7.3 Shape of Rewritings and Obstructions

We now analyze the shape of rewritings of non-Boolean MDDLog programs. An (ℓ, k)*-tree decomposition with* n *parameters* of an n-ary CQ q is an $(\ell + n, k + n)$-tree decomposition of q in which every bag B_v contains all answer variables of q. The treewidth with parameters of an n-ary CQ is now defined in the expected way.

▶ **Theorem 20.** *Let Π be an n-ary MDDLog program of diameter k. Then*

1. *if Π is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth $(1, k)$ with n parameters;*

2. *if* Π *is rewritable into MDLog with* n *parameters, then it has an MDLog-rewriting with* n *parameters of diameter* k.

As in the Boolean case, rewritings are closely related to obstructions. We define obstruction sets for MMSNP formulas with free variables and summarize the results that we obtain for them. A *set of marked obstructions* $\mathcal{O}$ for an MMSNP formula θ with n free variables over schema $\mathbf{S}_E$ is a set of n-marked instances over the same schema such that for any $\mathbf{S}_E$-instance I, we have $I \not\models \theta[\mathbf{a}]$ iff for some $(O, \mathbf{c}) \in \mathcal{O}$, there is a homomorphism h from O to I with $h(\mathbf{c}) = \mathbf{a}$. We obtain the following corollary from Point 1 of Theorem 20 in exactly the same way in which Corollary 10 is obtained from Point 1 of Theorem 9.

▶ **Corollary 21.** *For* θ *an MMSNP formula with* n *free variables, the following are equivalent:*

1. θ *is FO-rewritable;*
2. θ *has a finite marked obstruction set;*
3. θ *has a finite set of finite marked obstructions of treewidth* $(1, k)$ *with* n *parameters.*

It is interesting to note that this result can be viewed as a generalization of the characterization of obstruction sets for CSP templates with constants in terms of 'c-acyclicity' in [1]; our parameters correspond to constants in that paper. We now turn to MDLog-rewritability.

▶ **Proposition 22.** *Let* θ *be an MMSNP formula of diameter* k *with* n *free variables. Then* $\neg\theta$ *is rewritable into an MDLog program with* n *parameters iff* θ *has a set of marked obstructions (equivalently: finite marked obstructions) that are of treewidth* $(1, k)$ *with* n *parameters.*

8 Ontology-Mediated Queries

We study rewritability of ontology-mediated queries, covering several standard description logics as the ontology language. We start with introducing the relevant classes of queries.

An *ontology-mediated query (OMQ)* over a schema $\mathbf{S}_E$ is a triple $(\mathcal{T}, \mathbf{S}_E, q)$ where $\mathcal{T}$ is a TBox formulated in a description logic and q is a query over the schema $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T})$, $\mathsf{sig}(\mathcal{T})$ the set of relation symbols used in $\mathcal{T}$. The TBox can introduce symbols that are not in $\mathbf{S}_E$, which allows it to enrich the schema of the query q. As the TBox language, we use the description logic $\mathcal{ALC}$, its extension $\mathcal{ALCI}$ with inverse roles, and the further extension $\mathcal{SHI}$ of $\mathcal{ALCI}$ with transitive roles and role hierarchies. Since all these logics admit only unary and binary relations, we assume that these are the only allowed arities in schemas throughout the section. As the actual query language, we use UCQs and CQs. The OMQ languages that these choices give rise to are denoted with $(\mathcal{ALC}, \text{CQ})$, $(\mathcal{SHI}, \text{UCQ})$, and so on. In OMQs $(\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, \text{UCQ})$, we disallow superroles of transitive roles in q; it is known that allowing such roles in the query poses serious additional complications, which are outside the scope of this paper, see e.g. [7, 26]. The semantics of an OMQ is given in terms of *certain answers*. More details are provided in the long version of the paper. An OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ is *FO-rewritable* if there is an FO query $\varphi(x)$ over schema $\mathbf{S}_E$ (and possibly involving equality), called an *FO-rewriting* of Q, such that for all $\mathbf{S}_E$-instances I and $\mathbf{a} \subseteq \mathsf{dom}(I)$, we have $I \models Q(\mathbf{a})$ iff $I \models \varphi(\mathbf{a})$. Other notions of rewritability such as UCQ-rewritability are defined accordingly. Note that the TBox $\mathcal{T}$ can be inconsistent with the input instance I, that is, there could be no model of $\mathcal{T}$ and I. It can thus be a sensible alternative to work with *consistent FO-rewritability*, considering only $\mathbf{S}_E$-instances I that are consistent w.r.t. $\mathcal{T}$. As argued in the long version, our results apply to consistent FO-rewritability, too.

▶ **Theorem 23.** *In all OMQ languages between* $(\mathcal{ALC}, UCQ)$ *and* $(\mathcal{SHI}, UCQ)$*, as well as between* $(\mathcal{ALCI}, CQ)$ *and* $(\mathcal{SHI}, UCQ)$*,*

1. *FO-rewritability (equivalently: UCQ-rewritability) is 2*NExpTime*-complete; in fact, there is an algorithm which, given an OMQ* $Q = (\mathcal{T}, \mathbf{S}_E, q)$*, decides in time* $2^{2^{p(n_q \cdot \log n_\mathcal{T})}}$ *whether* Q *is FO-rewritable;*
2. *MDLog-rewritability is in 3*ExpTime *(and 2*NExpTime*-hard); in fact, there is an algorithm which, given an OMQ* $Q = (\mathcal{T}, \mathbf{S}_E, q)$*, decides in time* $2^{2^{2^{p(n_q \cdot \log n_\mathcal{T})}}}$ *whether* Q *is MDLog-rewritable*

where n_q *and* $n_\mathcal{T}$ *are the size of* q *and* $\mathcal{T}$ *and* p *is a polynomial.*

Note that the runtime for deciding FO-rewritability stated in in Theorem 23 is double exponential only in the size of the actual query q (which tends to be very small) while it is only single exponential in the size of the TBox (which can become large) and similarly for MDLog-rewritability, only one exponential higher.

The lower bounds in Theorem 23 are from [15]. We obtain the upper bounds by translating the OMQ into an equivalent MDDLog program and then applying the constructions that we have already established. As shown in [11] and refined in [15], every OMQ from the languages mentioned in Theorem 23 can be converted into an equivalent MDDLog program at the expense of a single or even double exponential blowup, depending on the OMQ language. Thus, we can decide FO- or MDLog-rewritability of an OMQ Q from $(\mathcal{SHI}, \text{UCQ})$ by translating Q into an MDDLog program Π and deciding the same problem for Π. A detailed analysis of all the relevant blowups involved in the composed reductions reveals that, when implemented with sufficent care, we actually obtain a 2NExpTime upper bound.

9 Discussion

We have clarified the decidability status and computational complexity of FO- and MDLog-rewritability in MMSNP, MDDLog, and various OMQ languages based on expressive description logics and conjunctive queries. For Datalog-rewritability, we were only able to obtain partial results, namely a sound algorithm that is complete only on a certain class of inputs and potentially incomplete in general. This raises several natural questions: is our algorithm actually complete in general? Does an analogue of Lemma 6 (that is, rewritability on high girth implies rewritability) hold for Datalog as a target language? What is the complexity of deciding Datalog-rewritability in the afore-mentioned languages? From an OMQ perspective, it would also be important to work towards more practical approaches for computing (FO-, MDLog-, and DLog-) rewritings. Given the high computational complexities involved, such approaches might have to be incomplete to be practically feasible. However, the degree/nature of incompleteness should then be characterized, and we expect the results in this paper to be helpful in such an endeavour.

Acknowledgement. We thank Libor Barto, Manuel Bodirsky, and Florent Madeleine for helpful discussions.

References

1 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23, 2011.

2 Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *JAIR*, 36:1–69, 2009.

3 Albert Atserias. On digraph coloring problems and treewidth duality. *Eur. J. Comb.*, 29(4):796–820, 2008.

4 Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016. doi:10.1093/logcom/exu070.

5 Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *Proc. of LICS*. IEEE Computer Society, 2016.

6 Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *Proc. of LICS*, pages 293–304. IEEE Computer Society, 2015.

7 Meghyn Bienvenu, Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Simkus. Query answering in the description logic $\mathcal{S}$. In *Proc. of DL2010*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL: http://ceur-ws.org/Vol-573/paper_20.pdf.

8 Meghyn Bienvenu, Peter Hansen, Carsten Lutz, and Frank Wolter. First order-rewritability and containment of conjunctive queries in horn description logics. In *Proc. of IJCAI*, 2016.

9 Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-order rewritability of atomic queries in horn description logics. In *Proc. of IJCAI*, 2013.

10 Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proc. of Reasoning Web*, volume 9203 of *LNCS*, pages 218–307. Springer, 2015.

11 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

12 Manuel Bodirsky, Hubie Chen, and Tomás Feder. On the complexity of MMSNP. *SIAM J. Discrete Math.*, 26(1):404–414, 2012.

13 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013. doi:10.1016/j.jcss.2012.05.012.

14 Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages. In *Proc. of IJCAI2015*, pages 2826–2832. AAAI Press, 2015. URL: http://ijcai.org/papers15/Abstracts/IJCAI15-400.html.

15 Pierre Bourhis and Carsten Lutz. Containment in monadic disjunctive Datalog, MMSNP, and expressive description logics. In *Proc. of KR*, 2016.

16 Andrei A. Bulatov, Andrei A. Krokhin, and Benoit Larose. Dualities for constraint satisfaction problems. In *Complexity of Constraints - An Overview of Current Research Themes*, volume 5250 of *LNCS*, pages 93–124. Springer, 2008.

17 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: The DL-Lite approach. In *Proc. of Reasoning Web 2009*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009. doi:10.1007/978-3-642-03754-2_7.

18 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

19 Hubie Chen and Benoit Larose. Asking the metaquestions in constraint tractability. *CoRR*, abs/1604.00932, 2016.

20 Gregory L. Cherlin, Saharon Shelah, and Niandong Shi. Universal graphs with forbidden subgraphs and algebraic closure. *Advances in Applied Mathematics*, 22:454–491, 1999.

21 Víctor Dalmau and Benoit Larose. Maltsev + datalog –> symmetric datalog. In *Proc. of LICS*, pages 297–306. IEEE Computer Society, 2008.

22 László Egri, Benoit Larose, and Pascal Tesson. Symmetric datalog and constraint satisfaction problems in logspace. In *Proc. of LICS*, pages 193–202. IEEE Computer Society, 2007.

23 Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI*. AAAI Press, 2012.

24 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

25 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

26 Georg Gottlob, Andreas Pieris, and Lidia Tendera. Querying the guarded fragment with transitivity. In *Proc. of ICALP2013*, volume 7966 of *LNCS*, pages 287–298. Springer, 2013. doi:10.1007/978-3-642-39212-2_27.

27 Peter Hansen, Carsten Lutz, İnanç Seylan, and Frank Wolter. Efficient query rewriting in the description logic el and beyond. In *Proc. of IJCAI*, 2015.

28 Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In *Proc. of RR*, pages 76–91, 2014.

29 Gábor Kun. Constraints, MMSNP and expander relational structures. *Combinatorica*, 33(3):335–347, 2013. doi:10.1007/s00493-013-2405-4.

30 Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:6)2007.

31 Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra universalis*, 56(3):439–466, 2007.

32 Florent R. Madelaine. Universal structures and the logic of forbidden patterns. *Logical Methods in Computer Science*, 5(2), 2009. URL: http://arxiv.org/abs/0904.2521.

33 Florent R. Madelaine. On the containment of forbidden patterns problems. In *Proc. of CP2010*, volume 6308 of *LNCS*, pages 345–359. Springer, 2010. doi:10.1007/978-3-642-15396-9_29.

34 Florent R. Madelaine and Iain A. Stewart. Constraint satisfaction, logic and forbidden patterns. *SIAM J. Comput.*, 37(1):132–163, 2007. doi:10.1137/050634840.

35 Jaroslav Nesetril. Many facets of dualities. In *Proc. of Workshop on Combinatorial Optimization*, pages 285–302. Springer, 2008.

36 Jaroslav Nešetřil and Claude Tardif. Duality theorems for finite structures (characterising gaps and good characterisations). *J. Comb. Theory, Ser. B*, 80(1):80–97, 2000.

37 Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *JAL*, 8(2):186–209, 2010.

38 Riccardo Rosati. On conjunctive query answering in $\mathcal{EL}$. In *Proc. of DL*, pages 451–458, 2007.

39 Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.

40 Sebastian Rudolph and Markus Krötzsch. Flag & check: data access with monadically defined queries. In *Proc. of PODS*, pages 151–162. ACM, 2013.

41 Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015.

Graphs, Hypergraphs, and the Complexity of Conjunctive Database Queries*

Dániel Marx

Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
dmarx@cs.bme.hu

Abstract

The complexity of evaluating conjunctive queries can depend significantly on the structure of the query. For example, it is well known that various notions of acyclicity can make the evaluation problem tractable. More generally, it seems that the complexity is connected to the "treelikeness" of the graph or hypergraph describing the query structure. In the lecture, we will review some of the notions of treelikeness that were proposed in the literature and how they are relevant for the complexity of evaluating conjunctive queries and related problems.

1998 ACM Subject Classification E.1 [Data Structures] Graphs and Networks. H.2.4 [Database Management] Systems, Query Processing

Keywords and phrases Conjunctive queries, treewidth, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.2

Category Invited Talk

* Supported by the ERC grant PARAMTIGHT: "Parameterized complexity and the search for tight complexity results", no. 280152.

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi; Article No. 2; pp. 2:1–2:1

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

The Smart Crowd – Learning from the Ones Who Know

Tova Milo

School of Computer Science, Tel Aviv University, Tel Aviv, Israel
milo@cs.tau.ac.il

Abstract

One of the foremost challenges for information technology over the last few years has been to explore, understand, and extract useful information from large amounts of data. Some particular tasks such as annotating data or matching entities have been outsourced to human workers for many years. But the last few years have seen the rise of a new research field called crowdsourcing that aims at delegating a wide range of tasks to human workers, building formal frameworks, and improving the efficiency of these processes.

In order to provide sound scientific foundations for crowdsourcing and support the development of efficient crowd sourcing processes, adequate formal models and algorithms must be defined. In particular, the models must formalize unique characteristics of crowd-based settings, such as the knowledge of the crowd and crowd-provided data; the interaction with crowd members; the inherent inaccuracies and disagreements in crowd answers; and evaluation metrics that capture the cost and effort of the crowd.

Clearly, what may be achieved with the help of the crowd depends heavily on the properties and knowledge of the given crowd. In this talk we will focus on knowledgeable crowds. We will examine the use of such crowds, and in particular domain experts, for assisting solving data management problems. Specifically we will consider three dimensions of the problem:

1. How domain experts can help in improving the *data* itself, e.g. by gathering missing data and improving the quality of existing data,
2. how they can assist in gathering *meta-data* that facilitate improved data processing, and
3. how can we find and identify the most relevant crowd for a given data management task.

1998 ACM Subject Classification H.2.4 [Database Management] Systems, Query Processing, H.2.3 [Database Management] Languages, Query Languages, H.2.8,[Database Management] Database Applications, Data Mining

Keywords and phrases Data Management, Crowdsourcing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.3

Category Invited Talk

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi; Article No. 3; pp. 3:1–3:1

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

GYM: A Multiround Distributed Join Algorithm

Foto N. Afrati[1], Manas R. Joglekar[2], Christopher M. Re[3], Semih Salihoglu[4], and Jeffrey D. Ullman[5]

1 **National Technical University of Athens, Athens, Greece**
afrati@gmail.com
2 **Stanford University, Stanford, CA, USA**
manasrj@stanford.edu
3 **Stanford University, Stanford, CA, USA**
chrismre@cs.stanford.edu
4 **University of Waterloo, Waterloo, ON, Canada**
semih.salihoglu@uwaterloo.ca
5 **Stanford University, Stanford, CA, USA**
ullman@gmail.com

Abstract

Multiround algorithms are now commonly used in distributed data processing systems, yet the extent to which algorithms can benefit from running more rounds is not well understood. This paper answers this question for several rounds for the problem of computing the equijoin of n relations. Given any query Q with width w, *intersection width* iw, input size IN, output size OUT, and a cluster of machines with $M = \Omega(\text{IN}^{\frac{1}{\epsilon}})$ memory available per machine, where $\epsilon > 1$ and $w \geq 1$ are constants, we show that:

1. Q can be computed in $O(n)$ rounds with $O(n\frac{(\text{IN}^{\mathsf{w}}+\text{OUT})^2}{M})$ communication cost with high probability.
2. Q can be computed in $O(\log(n))$ rounds with $O(n\frac{(\text{IN}^{\max(\mathsf{w},3\mathsf{iw})}+\text{OUT})^2}{M})$ communication cost with high probability.

Intersection width is a new notion we introduce for queries and generalized hypertree decompositions (GHDs) of queries that captures how connected the adjacent components of the GHDs are.

We achieve our first result by introducing a distributed and generalized version of Yannakakis's algorithm, called GYM. GYM takes as input any GHD of Q with width w and depth d, and computes Q in $O(\mathsf{d} + \log(n))$ rounds and $O(n\frac{(\text{IN}^{\mathsf{w}}+\text{OUT})^2}{M})$ communication cost. We achieve our second result by showing how to construct GHDs of Q with width $\max(\mathsf{w}, 3\mathsf{iw})$ and depth $O(\log(n))$. We describe another technique to construct GHDs with longer widths and lower depths, demonstrating other tradeoffs one can make between communication and the number of rounds.

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

Keywords and phrases Joins, Yannakakis, Bulk Synchronous Processing, GHDs

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.4

1 Introduction

The problem of evaluating joins efficiently in distributed environments has gained importance since the advent of Google's MapReduce [9] and the emergence of a series of distributed systems with relational operators, such as Pig [24], Hive [28], SparkSQL [27], and Myria [18]. These systems are conceptually based on Valiant's *bulk synchronous parallel* (BSP) computational model [29]. Briefly, there are a set of machines that do not share any memory and are

connected by a network. The computation is broken into a series of *rounds*. In each round, machines perform some local computation in parallel and communicate messages over the network. Costs of algorithms in these systems can be broken down to: (1) local computation of machines; (2) communication between the machines; and (3) the number of new rounds of computation that are started, which can have large overheads in some systems, e.g. due to reading input from disk or waiting for resources to be available in the cluster. In this paper, we focus on communication and the number of rounds, as for many data processing tasks, the computation cost is generally subsumed by the communication cost [19, 20].

This paper studies the problem of evaluating an equijoin query Q in multiple rounds of computation in a distributed cluster. We restrict ourselves to queries that are full, i.e. do not contain projections, but queries can contain self-joins. We let n be the number of relations, IN the input size, OUT the output size of Q, and $M = o(\text{IN})$ the memory available per machine in the cluster. Memory sizes of the machines intuitively capture different parallelism levels: when memory sizes are smaller, we need a larger number of machines to evaluate the join, which increases parallelism. We assume throughout the paper that $M = \Omega(\text{IN}^{\frac{1}{\epsilon}})$ for some constant $\epsilon > 1$. For practical values of input and memory sizes, ϵ is a small constant. For instance, if IN is in terabytes, then even when M is in megabytes, $\epsilon \approx 2$.

Our study of multiround join algorithms is motivated by two developments. First, it has been shown recently that there are prohibitively high lower bounds on the communication cost of any one-round algorithm for evaluating some join queries [3, 6]. For example, for the *chain query*, $C_n = R_1(A_0, A_1) \bowtie R_2(A_1, A_2) \bowtie \cdots \bowtie R_n(A_{n-1}, A_n)$, the lower bound on the communication cost of any one-round algorithm is $\geq (\frac{\text{IN}}{M})^{n/4}$. For example, if the input is one petabyte, i.e., IN=10^{15}, even when we have machines with ten gigabytes of memory, i.e., M=10^{10}, the communication cost of any one-round algorithm to evaluate the C_{16} query is 100000 petabytes. Moreover, this lower bound holds even when the query output is known to be small, e.g., OUT = $O(\text{IN})$, and the input has no skew [6], implying that designing multiround algorithms is the only way to compute such joins more efficiently.

Second, the cost of running a new round of computation has decreased from several minutes in the early systems (e.g., Hadoop [5]) to milliseconds in some recent systems (e.g., Spark [31]), making it practical to run algorithms that consist of a large number of rounds. Although multiround algorithms are becoming commonplace, how much algorithms can benefit from running more rounds is not well understood. In this paper, we answer this question for equijoin queries.

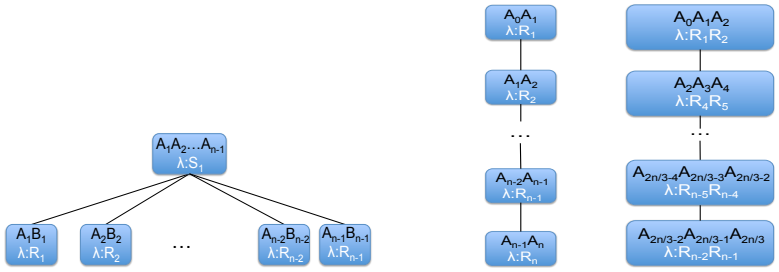
We describe a multiround algorithm, called *GYM*, for **G**eneralized **Y**annakakis in **M**apReduce (Sections 4-5), which is a distributed and generalized version of Yannakakis's algorithm for acyclic queries [30]. The performance of GYM depends on two important structural properties of the input query: *depths* and *widths* of its *generalized hypertree decompositions* (GHDs). We then present two algorithms, *Log-GTA* (Section 6) and *C-GTA* (Section 7), for constructing GHDs of queries with different depths and widths, exposing a spectrum of tradeoffs one can make between the number of rounds and communication using GYM. In the remainder of this section, we give an overview of our results.

1.1 GYM: A Multiround Join Algorithm

The width of a query, i.e., the minimum width of any of its GHDs, characterizes its degree of cyclicity, where acyclic queries are equivalent to width-1 queries. The original serial algorithm of Yannakakis takes as input a width-1 GHD of an acyclic query. GYM generalizes Yannakakis's algorithm to take as input any GHD of any query Q and evaluates Q in a distributed fashion. In this paper, we focus on bounded width queries in this paper, i.e. those whose widths are a constant.

Table 1 Example Queries S_n, C_n, and TC_n.

Query	Width	Min-Depth GHD	Intersection Width
$S_n : S(A_1, \ldots, A_{n-1}) \bowtie R_1(A_1, B_1) \bowtie \cdots \bowtie R_{n-1}(A_{n-1}, B_{n-1})$	1	1	1
$C_n : R_1(A_0, A_1) \bowtie R_2(A_1, A_2) \bowtie \cdots \bowtie R_n(A_{n-1}, A_n)$	1	$\Theta(n)$	1
$TC_n : R_1(A_0, A_1) \bowtie R_2(A_0, A_2) \bowtie R_3(A_1, A_2) \bowtie$ $R_4(A_2, A_3) \bowtie R_5(A_2, A_4) \bowtie R_6(A_3, A_4) \bowtie$ $\ldots$ $R_{n-2}(A_{\frac{2n}{3}-2}, A_{\frac{2n}{3}-1}) \bowtie R_{n-1}(A_{\frac{2n}{3}-2}, A_{\frac{2n}{3}}) \bowtie R_n(A_{\frac{2n}{3}-1}, A_{\frac{2n}{3}})$	2	$\Theta(n)$	1



(a) S_n. **(b)** C_n. **(c)** TC_n.

Figure 1 Example GHDs.

▶ **Main Result 1.** *Given a width-*w*, depth-*d *GHD of a query Q over n relations, GYM computes Q in $O(\mathsf{d} + \log(n))$ rounds with $O(n\frac{(\text{IN}^{\mathsf{w}}+\text{OUT})^2}{M})$ communication cost with high probability.*

Since every width-w query over n relations has a GHD of width w and depth at most n, an immediate corollary to our first main result is that every width-w query can be computed in $O(n)$ rounds and $O(n\frac{(\text{IN}^{\mathsf{w}}+\text{OUT})^2}{M})$ communication cost using GYM.

Table 1 lists three example queries and their widths w, minimum depths of their width-w GHDs, and intersection widths (explained momentarily). Figure 1 shows example GHDs of these queries. The labels on the vertices of the GHDs in Figure 1 are the λ and χ values, following the notation in Section 3.

▶ **Example 2.** The star query S_n is an acyclic query. As shown in Figure 1, S_n has a depth-1 and width-1 GHD. Using this GHD, GYM executes S_n in $O(\log(n))$ rounds with a communication cost of $O(n\frac{(\text{IN}+\text{OUT})^2}{M})$.

▶ **Example 3.** The chain query C_n is also an acyclic query. Figure 1 shows an example width-1 GHD of C_n with depth $n-1$. On this GHD, GYM executes C_n in $O(n)$ rounds with a communication cost of $O(n\frac{(\text{IN}+\text{OUT})^2}{M})$.

We present GYM within the context of the MapReduce system because it is the earliest and one of the simplest modern large-scale data processing systems. However, GYM can easily run on any BSP system, so our results apply to other BSP systems as well. We also note that all of the results presented in this paper hold under any amount of skew in the input data. We discuss the improvements to our results when the inputs to queries are skew-free in the longer version of our paper [1].

1.2 Log-GTA: Log-depth GHDs

For some width-w queries, any width-w GHD of the query has a depth of $\Theta(n)$. C_n and the triangle-chain query, TC_n, shown in Table 1, are examples of such queries with widths 1 and 2, respectively. Therefore, on any width-w GHD of such queries, GYM executes $\Theta(n)$ rounds. Our second main result shows how to execute such queries by GYM in exponentially fewer number of rounds but with more communication cost by proving a combinatorial lemma about GHDs, which may be of independent interest to readers:

▶ **Main Result 4.** *Given a width-*w*, intersection-width-*iw*, and depth-*d *GHD D of Q, we can construct a GHD D' of Q of depth $O(\log(n))$ and width at most* $\max(\mathsf{w}, 3\mathsf{iw})$.

Intersection width is a new notion of GHDs we introduce, that captures how connected the adjacent components of a GHD are. We present an algorithm *Log-GTA*, for ***Log**-depth **G**HD **T**ransformation **A**lgorithm*, to achieve our second main result. Using Log-GTA, we can tradeoff rounds and communication for queries with high depth GHDs as follows:

▶ **Example 5.** The TC_n query has a width of 2 and intersection width of 1. Figure 1c shows an example width-2 GHD D of TC_n which has a depth of $\frac{n}{3}$-1. One option to evaluate TC_n is to use D directly. On D, GYM will execute $\Theta(n)$ rounds and have a communication cost of $O(n\frac{(\text{IN}^2+\text{OUT})^2}{M})$. Another option is to construct a new GHD D' from D by Log-GTA, which will have a depth of $O(\log(n))$ and width of 3. On D', GYM will take $O(\log(n))$ rounds and have a communication cost of $O(n\frac{(\text{IN}^3+\text{OUT})^2}{M})$.

We end this section by discussing two interesting consequences of Log-GTA and GYM.

Log-depth Decompositions. GHDs [12] are one of several structural decomposition methods that are used to characterize the cyclicity of queries. Each decomposition method represents queries as a graph (if the input relations have arity at most 2) or a hypergraph and has a notion of "width" to measure the cyclicity of queries. Examples include *query decompositions* [8], *tree decompositions* (TDs) [26], and *hypertree decompositions* (HDs) [17]. Two previous results by Bodlaender [7] and Akatov [4] have proved the existence of log-depth TDs of hypergraphs with thrice their *treewidths* and HDs of hypergraphs with thrice their *hypertreewidths*, respectively. Our second main result proves that a similar and stronger property also holds for GHDs of hypergraphs. Interestingly, neither of these results (including ours) imply each other. However, we show in the longer version of our paper [1] that Log-GTA also recovers Bodlaender's result. That is, Log-GTA also transforms a given TD into log-depth one with thrice its treewidth. In addition, we show that a modification of Log-GTA recovers both Akatov's and Bodlaender's results and a weaker version of our second main result. We also show that using similar definitions of intersection widths for TDs and HDs, we can improve both Bodlaender's and Akatov's results.

Parallel Complexity of Bounded-width Queries. Database researchers have often thought of Yannakakis's algorithm as having a sequential nature, executing for $\Theta(n)$ steps in the PRAM model. In the PRAM literature [10, 15, 16], acyclic queries have been described as being polynomial-time sequentially solvable by Yannakakis's algorithm, but highly parallelizable by the *ACQ* algorithm [14], where parallelizability refers to being in the complexity class NC. By constructing log-depth GHDs of queries and simulating GYM in the PRAM model, we show that unlike previously thought, with simple modifications Yannakakis's algorithm can run in logarithmic rounds, implying that bounded-width queries are in the complexity class

NC, recovering a result that was prove using the ACQ algorithm [14]. We note that BSP models can also simulate PRAM and a comparison of GYM against a distributed simulation of ACQ, which we call *ACQ-MR*, is given in the longer version of our paper [1]. We also show in the longer version of our paper [1] that GYM can match ACQ-MR's performance on every query using an appropriate GHD for the query and on some queries GYM strictly outperforms ACQ-MR.

2 Related Work

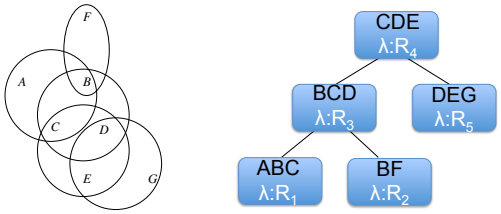
We provide a brief overview of related work here. A comprehensive coverage of related work is in the full version of this paper [1].

Shares. Shares [3] is the optimal one-round join algorithm. References [2] and [6] have shown that for every query Q, and value of M, and skew level, the Shares algorithm can be configured to incur the lowest possible communication cost among one-round algorithms that send at most M input tuples to each machine. However, as we discussed in Section 1, for some queries, these costs can be prohibitively expensive.

Other Distributed Join Algorithms. Reference [6] studies multiround distributed join algorithms in the *Massively Parallel Computing* (MPC) model. Reference [6] proves lower bounds on the number of rounds required to compute queries when $M = \frac{\text{IN}}{p^{1-\epsilon'}}$, where p is the number of machines, and ϵ' is a constant $\in [0, 1)$ called the *space exponent*. They show that when evaluating a query whose GHDs are of depth d on an arbitrary database, any algorithm with limited memory, where the limitation is defined as space exponent being a constant, will have to run $O(\log(\mathsf{d}))$ rounds. The authors show that running the Shares algorithm iteratively on sets of the input relations matches these lower bounds on a limited set of inputs, called *matching databases*, which represent skew-free inputs. On arbitrary databases, their iterative Shares algorithm can produce intermediate data of size $\text{IN}^{\Theta(n)}$ for any query irrespective of its width. By our second result, GYM evaluates these queries on any database instance in $O(\log(n))$ rounds. So when d i constant but n is unbounded, GYM runs $O(\log(n))$ rounds whereas their lower bound is $O(1)$. However, GYM keeps intermediate relation sizes bounded by $\text{IN}^{\max(w,3\mathsf{iw})} + \text{OUT}$. On matching databases, their algorithms matches these lower bounds exactly. In the longer version of our paper [1], we show that our GYM algorithm matches these lower bounds exactly using several optimizations.

Reference [22] describes worst case optimal constant-round join algorithms for several classes of conjunctive queries when the frequencies of each value in the attributes of the relations are known. The authors relate the communication cost of algorithms on a query to a structural property of the query called *edge quasi-packing number*, which can be smaller than the width of the query. In contrast, we do not assume any prior knowledge of frequencies.

Generalized Hypertree Decompositions. Structural decomposition methods, such as GHDs [13], query decompositions (QDs) [8], tree decompositions (TDs) [26], and hypertree decompositions (HDs) [17], are mathematical tools to characterize the difficulty of computational problems that can be represented as graphs or hypergraphs, such as joins or constraint satisfaction problems. GYM can use methods other than GHDs, such as QDs, and HDs, but we use GHDs because the widths of GHDs are known to be smaller than HDs and QDs, giving us stronger results in terms of communication cost.



(a) Hypergraph. **(b)** GHD.

Figure 2 Hypergraph and GHD of Example 6.

3 Preliminaries

We review GHDs, describe our model, and specify the assumptions we make in this paper.

3.1 Generalized Hypertree Decompositions

A ***hypergraph*** is a pair $H = (V(H), E(H))$, consisting of a nonempty set $V(H)$ of vertices, and a set $E(H)$ of subsets of $V(H)$, the hyperedges of H. Natural join queries can be expressed as hypergraphs, where we have a vertex for each attribute of the query, and a hyperedge for each relation.

▶ **Example 6.** Consider the query Q:

$$R_1(A, B, C) \bowtie R_2(B, F) \bowtie R_3(B, C, D) \bowtie \\ R_4(C, D, E) \bowtie R_5(D, E, G)$$

The hypergraph of Q is shown in Figure 2a.

Let H be a hypergraph. A ***generalized hypertree decomposition (GHD)*** of H is a triple $D = (T, \chi, \lambda)$, where:

- $T(V(T), E(T))$ is a tree;
- $\chi : V(T) \to 2^{V(H)}$ is a function associating a set of vertices $\chi(t) \subseteq V(H)$ to each vertex t of T;
- $\lambda : V(T) \to 2^{E(H)}$ is a function associating a set of hyperedges to each vertex t of T;

such that the following properties hold:

1. For each $e \in E(H)$, there is a vertex $t \in V(T)$ such that $e \subseteq \chi(t)$.
2. For each $v \in V(H)$, the set $\{t \in V(T) | v \in \chi(t)\}$ is connected in T.
3. For every $t \in V(T)$, $\chi(t) \subseteq \bigcup \lambda(t)$, i.e., hyperedges of $\lambda(t)$ must "cover" the vertices of $\chi(t)$.

For any $t \in V(T)$, we refer to $\chi(t)$ as the attributes of t and $\lambda(t)$ as the relations on t. A GHD of a join query Q is defined to be a GHD on the hypergraph of Q.

▶ **Example 7.** Figure 2b shows a GHD of the query from Example 6. In the figure, the attribute values on top of each vertex t are the χ assignments for t and the λ assignments are explicitly shown.

We next define several properties of GHDs and hypergraphs:

- The ***depth*** of a GHD $D = (T, \chi, \lambda)$ is the depth of the tree T.
- The ***width*** of a GHD D is $\max_{t \in V(T)}\{|\lambda(t)|\}$, i.e., the maximum number of relations assigned to any vertex t.

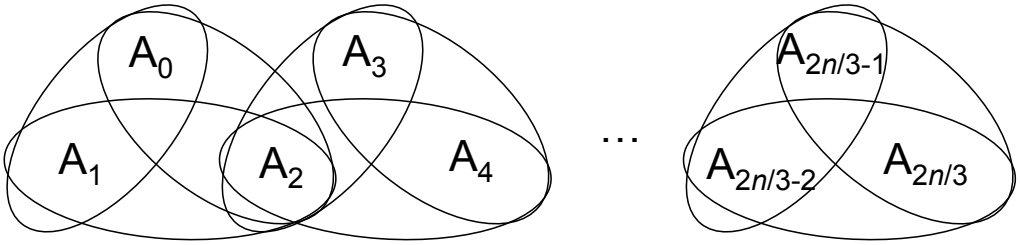


Figure 3 Hypergraph of TC_n.

- The ***generalized hypertree width***, or ***width*** for short, of a hypergraph H is the minimum width of all GHDs of H.

The width of a query captures its degree of cyclicity. In general, the larger the width of a query, the more "cyclic" it is. Acyclic queries are exactly the queries with width 1 [8]. We next define a new notion called intersection width.

- The ***intersection width*** of a GHD $D = (T, \chi, \lambda)$ is defined as follows: For any adjacent vertices $t, t' \in V(T)$, let $\mathsf{iw}(t, t')$ denote the size of the smallest set $S \subseteq E(H)$ such that $\chi(t) \cap \chi(t') \subseteq \bigcup_{s \in S} s$. In other words, $\mathsf{iw}(t, t')$ is the size of the smallest set of relations whose attributes cover the common attributes between t and t'. The intersection width iw of a GHD is the maximum $\mathsf{iw}(t, t')$ over all adjacent $t, t' \in V(T)$.

Notice that the intersection width of D is never larger than the width of D, because $\forall\ t, t' \in V(T) : \mathsf{iw}(t, t') \leq |\lambda(t)|$, since by the 3rd property of GHDs $\lambda(t)$ is one (possibly not the smallest) set of relations that covers the attributes of t, and therefore any common attribute that t shares with its neighbors. The intersection width of a GHD can be strictly smaller than the width, as the next example shows.

▶ **Example 8.** Consider the TC_n example from Table 1. TC_n is a width-2 query. The hypergraph of TC_n, shown in Figure 3, visually is a chain of triangles, where any two consecutive triangles are connected by a single attribute. Figure 1c shows a width-2 GHD of TC_n, where each node covers one of the triangles in the same order they appear in the hypergraph. The intersection width of this GHD is 1, as the common attribute between each triangle can be covered by one relation (e.g., A_2, which is the common attribute between the first two triangles, can be covered by R_2).

In the rest of this paper we restrict ourselves, for simplicity of presentation, to queries whose hypergraphs are connected. All of our results generalize to queries with disconnected hypergraphs. A GHD $D(T, \chi, \lambda)$ of a hypergraph H is called *complete* if each hyperedge $e \in E(H)$ occurs in $\lambda(t)$ of some vertex $t \in V(T)$. That is, each relation is assigned to the λ-label of some vertex $t \in V(T)$. We assume throughout the paper that the GHDs we use are rooted, i.e., one of the vertices (arbitrarily) in T is picked as a root. This ensures that there is a well defined notion of height of vertices and parent-child relationships between the vertices in T. We end this section by stating a lemma about complete GHDs of queries:

▶ **Lemma 9.** *If a query Q has a width-*w*, intersection width-*iw *GHD $D = (T, \chi, \lambda)$ of depth* d*, then Q has a complete GHD $D' = (T', \chi', \lambda')$ with depth $\leq d + 1$, width* w*, intersection width* iw*, and $|V(T')| \leq 4n$.*

We prove this lemma in the longer version of our paper [1]. Using this lemma, we will assume w.l.o.g. that the GHDs of queries that we use in our algorithms are complete and have $O(n)$ size.

3.2 MapReduce and Cost Model

Our MapReduce (MR) model is equivalent to the MR model in reference [25] except we use tuples instead of bits as our cost unit. In the tuple-based MR model, the unit of memory and communication cost is a base or intermediate tuple consisting of any set of attributes in Q. There is a set of distributed machines on a networked file system, each with memory $M = o(\text{IN})$.

Map Stage: Each machine, referred to as a *mapper*, reads a set of base or intermediate tuples over any set of attributes in the query from the networked file system. Mappers can send tuples to one or more machines, called *reducers*[1], deterministically or by hashing them on any set of their attributes. We assume that mappers have access to the same random bits and families of universal hash functions.[2] Suppose there are k reducers. Using an appropriate family of hash functions, mappers can hash tuples of an input or intermediate relation R, using any subset of the tuples' attributes, to one of the k reducers. Therefore two tuples with the same attributes that were used in the hashing will go to the same reducer. The total number of tuples received by a reducer from all mappers should not exceed memory size M. Otherwise the computation aborts. We note that we use randomization for load-balancing only. Specifically, GYM might send a machine more than M tuples, exceeding the memory capacity of the machine, resulting in the computation to abort. However, this will happen with exponentially small probability.

Reduce Stage: Each reducer locally performs any computation on the $\leq M$ tuples it receives, produces a set of output tuples, and streams the output tuples to the network file system. The local computation at a reducer cannot exceed memory size M, but the output of a reducer can exceed M as it is streamed to the file system.

The *communication cost* of each round is defined as the total number of tuples sent from all mappers to reducers plus the number of output tuples produced by the reducers. We measure the complexity of our algorithms in terms of the total communication cost and the number of rounds. In the longer version of our paper [1], we compare our model to existing models of modern distributed BSP systems.

3.3 Assumptions

We next specify three assumptions we make throughout the paper.

1. As in many MapReduce and distributed BSP models [6, 11, 21, 23, 25], we constrain M to be $o(\text{IN})$. This ensures that machines cannot store the entire input. Otherwise we can send the entire input to a single machine and incrementally evaluate the join using any binary join plan, without exceeding memory $O(M)$, in a single round and without any communication (except to write the output to the networked file system).
2. We assume $M = \Theta(\text{IN}^{\frac{1}{\epsilon}})$ for some constant $\epsilon > 1$.
3. We assume queries have constant widths, i.e., the term w is a constant ($O(1)$).

As we discuss in Section 5, the complexities of our algorithms become slightly worse when we drop assumptions (2) and (3).

---

[1] The machines we refer to as reducers are equivalent to *reduce keys* in the original description of MR [9]. These are separate groups of data on which the *reduce()* function is executed in the original system.

[2] Access to these random bits do not require any synchronization. In practice this would achieved by using a pseudorandom number generator with the same seed.

3.4 Basic Relational Operations in MR

We next state four lemmas characterizing the costs of joins, duplicate elimination, semijoins, and intersections in our model.

▶ **Lemma 10.** *Any z relations $R_1, \ldots, R_z$ can be joined in 1 round with $O(\frac{z^z(\Sigma_i|R_i|)^z}{M^{z-1}} + |R_1 \bowtie \cdots \bowtie R_z|)$ communication. When z is a constant, the join can be performed in $O(\frac{(\Sigma_i|R_i|)^z}{M^{z-1}} + |R_1 \bowtie \cdots \bowtie R_z|)$ communication.*

Proof. We perform the join as follows. We divide each R_i in $g_{r_i} = \frac{z|R_i|}{M}$ disjoint groups of size $\frac{M}{z}$ each. Then we use a total of $g_{r_1} \times \cdots \times g_{r_z}$ reducers and map a distinct set of z groups, one from each relation, to each reducer. Each reducer joins its z groups locally. Thus we use $\frac{z^z|R_1|\ldots|R_z|}{M^z}$ reducers and each reducer gets an input equal to M, and the total output size is $|R_1 \bowtie \cdots \bowtie R_z|$. Therefore, the total communication cost is $O(\frac{z^z(\Sigma_i|R_i|)^z}{M^{z-1}} + |R_1 \bowtie \cdots \bowtie R_z|)$. ◀

We note that Lemma 10 holds even if the given relations contain self-joins.

▶ **Lemma 11.** *Let S be a multiset such that each tuple $t \in S$ has at most k duplicates. We can remove the duplicates in S w.h.p. in $O(\log_M(k))$ rounds and $O(\log_M(k)|S|)$ communication.*

Proof. We cannot send the duplicates of each tuple to a separate reducer because k might be greater than M, exceeding the memory of machines. Let h be a hash function mapping the tuples of S into $|S|^2$ buckets randomly, so w.h.p. each bucket $h(i)$ gets $O(1)$ unique tuples. In the first round of duplicate elimination, we use $|S|^2k^2$ reducers indexed with two numbers, $(1,1), \ldots, (1,k^2), \ldots, (|S|^2,1), \ldots, (|S|^2,k^2)$, and each tuple t is mapped to the reducer with the first index $h(t)$ and a uniformly random second index. Therefore, w.h.p., each reducer gets $O(1)$ tuples. Reducers do not perform any computation on their tuples. Note that every duplicate of tuple t is mapped to a reducer with the first index $h(t)$. In addition, the number of non-duplicate tuples across all of the reducers with the same first index is $O(1)$, since h maps $O(1)$ unique S tuples to each $h(i)$. In the second round, for each set of reducers with the same first index we do the following in parallel: We group the reducers into groups of $\sqrt{M}$, map their tuples to the same reducer, and eliminate the duplicates across them.[3] This reduces the number of reducers that can contain duplicates to $\frac{k^2}{\sqrt{M}}$. We then repeat this procedure in parallel for each group of reducers with the same first index until all duplicates within each group are eliminated. In each round, each reducer gets $O(\sqrt{M})$ tuples and outputs $O(1)$ tuples. This computation takes $O(\log_{\sqrt{M}}(k))$ rounds. Since the communication in each round is $|S|$, this computation takes $O(\log_{\sqrt{M}}(k)|S|)$ communication.[4] ◀

▶ **Lemma 12.** *Let $B(X,M) = \frac{X^2}{M}$. Given two relations R and S, the semijoin $S \ltimes R$ can be computed w.h.p. in $O(\log_M(|R|))$ rounds with $O(\log_M(|R|)B(|R|+|S|,M))$ communication. When $M = \Omega((|R|)^{1/\epsilon})$, for some $\epsilon > 1$, the semijoin can be performed in $O(1)$ rounds and $O(B(|R|+|S|,M))$ communication.*

[3] We can also group them into $\frac{M}{c}$ for a constant c that is larger than the (O(1)) tuples any reducer has.

[4] As is standard, we use the term w.h.p. in this paper to refer to probabilities that are exponentially small in the input size IN. The probabilities mentioned in Lemma 11 are exponentially small in the size of S, which as we will see, in Yannakakis's algorithm can be smaller than IN. However when an input relation S on which we perform duplicate elimination is small, we can make this probability exponentially small in IN by simply increasing the number of our buckets to IN^2.

Proof. The first round of the semijoin $S \ltimes R$ is similar to the join. Let $g_r = \frac{2|R|}{M}$ and $g_s = \frac{2|S|}{M}$ be disjoint groups of size $\frac{M}{2}$. Each of the $g_r g_s$ reducers locally computes the semijoin of one S group and one R group it receives. Because each tuple of S is sent to g_r different reducers, there may be up to g_r duplicates of each tuple. So the size of the multiset S' with duplicates of S is $g_r|S|$. Using Lemma 11, we can eliminate these tuples w.h.p. in $O(\log_M(g_r))$ rounds and $O(\log_M(g_r)g_r|S|) = O(\frac{\log_M(|R|)|R||S|}{M})$ communication. Together with the costs of the join operation, which takes 1 round and $O(\frac{(|R|+|S|)^2}{M})$ communication, we conclude that the semijoin can be performed w.h.p. in $O(\log_M(|R|))$ rounds and $O(\log_M(|R|)B(|R|+|S|, M))$ communication cost. When $M = \Omega((|R|)^{1/\epsilon})$, the semijoin can be computed in $O(1)$ rounds and $O(B(|R|+|S|, M))$ communication. ◀

▶ **Lemma 13.** *Two relations R and S can be intersected w.h.p in 1 round with $O(|R|+|S|)$ communication.*

Proof. Suppose w.l.o.g. that $|R| > |S|$. We simply hash each tuple of R and S using all of their attributes into $|R|^2$ machines, so w.h.p. each machine gets $O(1)$ tuples from both R and S and performs a local intersection, without exceeding M, and writes the at most $|R|+|S|$ output. ◀

4 Distributed Yannakakis

We first review the serial version of Yannakakis's algorithm for acyclic queries in Section 4.1. In Section 4.2, we show how to run Yannakakis's algorithm in a distributed setting in $O(n)$ rounds and $O(n\mathrm{B}(\mathrm{IN}+\mathrm{OUT}, M))$ communication cost. In Section 4.3, we reduce the number of rounds to $O(\mathsf{d}+\log(n))$ rounds without affecting the communication cost.

4.1 Serial Yannakakis Algorithm

The serial version of Yannakakis's algorithm takes as input an acyclic query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, and constructs a width-1 GHD $D = (T, \chi, \lambda)$ of Q. Since D is a GHD with width 1, each vertex of D is assigned exactly one relation R_i. We will refer to relations that are assigned to leaf (non-leaf) vertices in T as *leaf (non-leaf) relations*. Yannakakis's algorithm first eliminates all tuples that will not contribute to the final output by a series of semijoin operations. The overall algorithm consists of two phases: (1) a **semijoin phase**; and (2) a **join phase**.

Semijoin Phase: The semijoin phase operates recursively as follows.

- Basis: If T is a single node, do nothing.
- Induction: If T has more than one node, pick a leaf t that is assigned relation R, and let S be the relation assigned to t's parent.
  1. Replace S by the semijoin of S with R, $S \ltimes R = S \bowtie \pi_{R \cap S}(R)$.
  2. Recursively process $T \setminus R$.
  3. Compute the final value of R by computing its semijoin with the value of S that results from step (2); that is, $R := R \ltimes S$.

The executions of step (1) in this recursive algorithm form the *upward* sub-phase, and the executions of step (3) form the *downward* sub-phase. In total, this version of the algorithm performs $2(n-1)$ semijoin operations.

Join Phase: The algorithm performs a series of $(n-1)$ joins, in any bottom-up order on T.

An important property of Yannakakis's algorithm is that the semijoin phase removes all of the "dangling" tuples in the input, i.e., those that will not contribute to the final output. This guarantees that the sizes of all intermediate tables during the join phase are no larger than the final output size OUT [30].

4.2 DYM-n

If we simply execute each semijoin and join operation of Yannakakis's algorithm by the algorithms in Lemmas 10 and 12, we get a distributed algorithm which we refer to as ***DYM-n***:

▶ **Theorem 14.** *DYM-n computes every acyclic query Q in $O(n)$ rounds and in $O(n\mathrm{B}(\mathrm{IN}+\mathrm{OUT}, M))$ communication cost.*

Proof. The algorithm executes a total of $2(n-1)$ pairwise semijoins and $n-1$ joins, in a total of $O(n)$ rounds. The largest input to any semijoin operation is the largest relation size, which is at most IN. By Lemma 12, the communication cost of the semijoin phase is $O(n\mathrm{B}(\mathrm{IN}, M))$. In each round of the join phase, the input and outputs are at most the final output size OUT. By Lemma 10, the cost of each round is $O(\frac{\mathrm{OUT}^2}{M}+\mathrm{OUT})$. Therefore, the total cost of both phases is $O(n(\mathrm{B}(\mathrm{IN}, M)+\mathrm{B}(\mathrm{OUT}, M)+\mathrm{OUT}))$, which is $O(n\mathrm{B}(\mathrm{IN}+\mathrm{OUT}, M))$ when $M = O(\mathrm{IN})$, as we assume in this paper (recall Section 3.3). ◀

4.3 DYM-d

DYM-d parallelizes Yannakakis's algorithm further by executing multiple semijoins and joins in parallel, reducing the number of rounds to $O(\mathsf{d}+\log(n))$, where d is the depth of the GHD $D(T, \chi, \lambda)$, without asymptotically affecting DYM-n's communication cost.

Upward Semijoin Sub-phase in $O(\mathsf{d}+\log(n))$ Rounds: During the upward semijoin sub-phase, the algorithm from Section 4.1 picks one leaf t that is assigned relation R and processes R by replacing it's parent S with $S \ltimes R$ in $O(1)$ rounds. Instead we can pick and process all leaves in parallel. Consider the set L of leaves of T. Let L_1 be the set of leaves that have no siblings, and let L_2 be the remaining leaves. We will replace step (1) of the algorithm from Section 4.1 with two steps, which will be performed in parallel.

1.1. For each R in L_1 in parallel, replace R's parent S with $S \ltimes R$.

1.2. Divide the leaves in L_2 into disjoint pairs of siblings, and up to one triple of siblings per parent, if there is an odd number of siblings with the same parent. Then in parallel perform the following computation. Suppose R_1 and R_2 form such a pair with parent S. Replace R_1 with $(S \ltimes R_1) \cap (S \ltimes R_2)$ and remove R_2. If there is a triple R_1, R_2, R_3, replace R_1 with $(S \ltimes R_1) \cap (S \ltimes R_2) \cap (S \ltimes R_3)$ (using two pairwise intersections) and remove R_2 and R_3.

▶ **Lemma 15.** *The above procedure runs in $O(\mathsf{d}+\log(n))$ rounds.*

The proof of this lemma is provided in the longer version of our paper [1]. Since we perform $O(n)$ intersection or semijoin operations in total and all of the initial and intermediate relations involved have size at most IN, by Lemmas 12 and 13, the total communication cost of the upward semijoin sub-phase is $O(n\mathrm{B}(\mathrm{IN}, M))$.

Downward Semijoin Sub-phase in $O(\mathsf{d})$ Rounds: Note that in the downward semijoin sub-phase, the semijoins of the children relations with the same parent are independent and can be done in parallel in $O(1)$ rounds. Thus we can perform the downward sub-phase in $O(\mathsf{d})$ rounds and in $O(n\mathrm{B}(\mathrm{IN}, M))$ communication.

Join Phase in $O(\mathsf{d} + log(n))$ Rounds: The join phase is similar to the upward semijoin sub-phase. The only difference is, we compute $S \bowtie R$ instead of $S \ltimes R$ for $R \in L_1$, and $(R_1 \bowtie S) \bowtie (R_2 \bowtie S)$ for pair $R_1, R_2 \in L_2$. The total number of rounds required is again $O(\mathsf{d} + \log(n))$. The total communication cost of each pairwise join is $O(n\mathrm{B}(\mathrm{OUT}, M))$, since the intermediate relations being joined are at most as large as OUT. Therefore, both the semijoin and join phases can be performed in $O(\mathsf{d}+\log(n))$ rounds with a total communication cost of $O(n\mathrm{B}(\mathrm{IN} + \mathrm{OUT}, M))$, justifying the following theorem:

▶ **Theorem 16.** *DYM-d evaluates an acyclic query Q in $O(\mathsf{d}+\log(n))$ rounds and $O(n\mathrm{B}(\mathrm{IN}+\mathrm{OUT}, M))$ communication cost, where d is the depth of a width-1 GHD $D(T, \chi, \lambda)$ of Q.*

5 GYM

Our *GYM* algorithm generalizes DYM-d from acyclic queries to any query. Consider a width-w, depth-d GHD $D(T, \chi, \lambda)$ of a query Q. By Lemma 9, we assume w.l.o.g. that D is complete. Consider "materializing" each $v \in V(T)$ by computing $IDB_v = \bowtie_{R_i \in \lambda(v)} R_i$. Now, consider the query $Q' = \bowtie_{v \in V(T)} IDB_v$. Note that Q' has the exact same output as Q. This is because Q' is also the join of all R_i, where some R_i might (unnecessarily) be joined multiple times if they are assigned to multiple vertices. However, observe that Q' is now an acyclic query and D is now a width-1 GHD for Q'. Therefore we can directly run DYM-d to compute Q'.

▶ **Theorem 17** (First Main Result)**.** *Given a width-w, depth-d GHD $D(T, \chi, \lambda)$ of a query Q over n relations, GYM executes Q in $O(\mathsf{d} + \log(n))$ rounds and $O(n\mathrm{B}(\mathrm{IN}^{\mathsf{w}} + \mathrm{OUT}, M))$ communication cost.*

Proof. For the materialization stage, for each vertex v of D, joining the w relations inside $\lambda(v)$ takes 1 round and $O(\frac{\mathrm{IN}^{\mathsf{w}}}{M^{w-1}} + |IDB_v|)$ communication cost by Lemma 10. In the worst case when the relations constitute a Cartesian product, $|IDB_v|$ is IN^w, so evaluating IDB_v takes $O(\mathrm{IN}^{\mathsf{w}})$ cost. By Lemma 9 there are at most $4n$ vertices in $V(T)$, so the materialization stage takes $O(n\mathrm{IN}^{\mathsf{w}})$ communication cost. Since the size of each IDB_v is at most IN^{w}, executing DYM-d on the IDB_v's takes $O(\mathsf{d} + \log(n))$ rounds and $O(n\mathrm{B}(\mathrm{IN}^{\mathsf{w}} + \mathrm{OUT}, M))$ communication, which dominates the cost of materialization phase, completing the proof. ◀

In the longer version of our paper [1], we present an example execution of GYM on a query. We note that if we drop our assumptions that $M = \Omega(\mathrm{IN}^{\frac{1}{c}})$ and w is a constant, the number of rounds that GYM takes on a width-w, depth-d GHD increases by a factor of $\log_M(\mathrm{IN}^{\mathsf{w}}) = \mathsf{w}\log_M(\mathrm{IN})$. This is because the semijoin operations on $O(\mathrm{IN}^{\mathsf{w}})$ size inputs will execute $O(w\log_M(\mathrm{IN}))$ rounds instead of $O(1)$. Similarly, the communication cost of GYM will increase by at most a factor of $\max\{\mathsf{w}\log_M(\mathrm{IN}), \mathsf{w}^{\mathsf{w}}\}$. The $\mathsf{w}\log_M(\mathrm{IN})$ and w^{w} factors are due to the communication cost increases in the semijoin (Lemma 12) and join operations (Lemma 10), respectively.

6 Log-GTA

We now describe our ***Log-GTA*** algorithm (for **Log**-depth **G**HD **T**ransformation **A**lgorithm) which takes as input a hypergraph H of a query Q, and its GHD $D(T, \chi, \lambda)$ with width w and intersection width iw, and constructs a GHD D^* with depth $O(\log(|V(T)|))$ and width $\leq \max(\mathsf{w}, 3\mathsf{iw})$. For simplicity, we will refer to all GHDs during Log-GTA's transformation as $D'(T', \chi', \lambda')$, i.e., $D' = D$ in the beginning and $D' = D^*$ at the end. By running GYM on D^*, we can execute Q in $O(\log(n))$ rounds with $O(n\mathrm{B}(\mathrm{IN}^{\max(\mathsf{w},3\mathsf{iw})} + \mathrm{OUT}, M))$ communication.

6.1 Extending to D′

Log-GTA associates two new labels with the vertices of T':

1. **Active/Inactive:** An 'active' vertex is one that will be modified in later iterations of Log-GTA. Log-GTA starts with all vertices active, and inactivates vertices iteratively until all of them are inactive. At any point, we refer to the subtree of T' consisting of only active vertices as ***active***(T'). We prove that active(T') is indeed a tree in Lemma 19.
2. **Height:** The height of a vertex is its minimum distance from a leaf of the tree. The height of each vertex v is assigned when v is first inactivated, and remains unchanged thereafter.

In addition, Log-GTA associates a label with each "active" edge $(u, v) \in E(active(T'))$:

- **Common-cover(u, v) (cc(u, v)):** Is a set $S \subseteq E(H)$ such that $(\chi(u) \cap \chi(v)) \subseteq \bigcup_{s \in S} s$. In query terms, cc$(u, v)$ is a set of relations whose attributes cover the common attributes between u and v. In the original $D(T, \chi, \lambda)$, for each (u, v), we set cc(u, v) to any covering subset of size at most iw. Recall from Section 3 that by definition of iw, such a subset must exist.

Unique-c-gc vertices: Consider a tree T of n vertices with a high depth, say, $\Theta(n)$. Intuitively, such high depths are caused by long chains of vertices, where vertices in the chain have only a single child. Log-GTA reduces the depth of high-depth GHDs by identifying and "branching out" such chains. At a high-level, Log-GTA finds a vertex v with a unique child c (for child), which also has unique child gc (for grandchild), and puts v, c, and gc under a new vertex s. We call vertices like v *unique-c-gc* vertices.

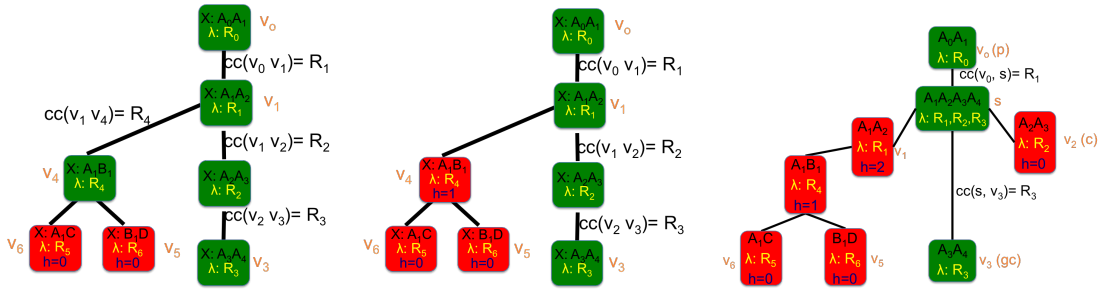
In each iteration, Log-GTA identifies a set of nonadjacent unique-c-gc vertices and leaves of active(T'), and inactivates them (while shortening the chains of unique-c-gc vertices). We next state an important lemma that will help bound the number of iterations of Log-GTA (proved in the longer version of our paper [1]):

▶ **Lemma 18.** *In a tree with N vertices, we can find two sets L' and U' such that $|L'|+|U'| > \lceil \frac{N}{4} \rceil$ and vertices in L' are leaves, and vertices in U' are (1) unique-c-gc vertices; and (2) pairwise non-adjacent.*

6.2 Two Transformation Operations

We next describe the two operations that Log-GTA performs on the nodes of active(T').

Leaf Inactivation: Takes a leaf l of active(T') and (1) sets its label to inactive; and (2) sets height(l) to $\max\{0, \max_c\{\text{height}(c)\} + 1\}$, where c is over the (inactive) children of l. $\chi(l)$ and $\lambda(l)$ remain the same. The common-cover between l and l's parent is removed.



(a) Original GHD. **(b)** Leaf Inactivation. **(c)** Unique-c-gc Inactivation.

Figure 4 Effects of leaf inactivation and unique-c-gc inactivation.

Unique-c-gc (And Child) Inactivation: Let u be a unique-c-gc vertex in active(T'). Note that u is not necessarily a unique-c-gc vertex in T'. Let u's parent be p (if one exists), u's child be c, and u's grandchild be gc. Unique-c-gc inactivation does the following:

1. Creates a new active vertex s, where $\lambda(s) = cc(p,u) \cup cc(u,c) \cup cc(c,gc)$ and $\chi(s) = (\chi(p) \cap \chi(u)) \cup (\chi(u) \cap \chi(c)) \cup (\chi(c) \cap \chi(gc))$.
2. Inactivates u and c. Similar to leaf inactivation, sets their heights to 0 if they have no inactive children, and one plus the maximum height of their inactive children otherwise.
3. Removes the edges (p, u) and (u, c) and adds an edge from s to both u and c.
4. Adds an edge from p to s with $cc(p,s) = cc(p,u)$ and s to gc with $cc(s,gc) = cc(c,gc)$.

Figure 4b shows the effect of leaf inactivation on vertex v_4 of the extended GHD in Figure 4a. In the figure, green and red indicate that the vertex is active and inactive, respectively. The attributes of each R_i are the χ values on the nodes that R_i is assigned to. Figure 4c shows the effect of Unique-c-gc Inactivation on a unique-c-gc vertex v_1 from Figure 4b. We next state a key lemma about these two operations:

▶ **Lemma 19.** *Assume that an extended GHD $D'(T', \chi', \lambda')$ of a hypergraph H with active/inactive labels on $V(T')$, and common covers on $E(T')$ initially satisfies the following five properties:*

1. *active(T') is a tree.*
2. *The subtree rooted at each inactive vertex v contains only inactive vertices.*
3. *The height of each inactive vertex v is v's correct height in T'.*
4. *$|cc(u, v)| \leq \mathsf{iw}$ between any two active vertices u and v and does indeed cover the shared attributes of u and v.*
5. *D' is a GHD of H with width at most $\max(\mathsf{w}, 3\mathsf{iw})$.*

Performing any sequence of leaf and unique-c-gc inactivations maintains these five properties.

We prove this lemma in the longer version of our paper [1]. We next state an immediate corollary to Lemma 19.

▶ **Corollary 20.** *Let $D(T, \chi, \lambda)$ be a GHD of a hypergraph H with width w, intersection width iw. Consider extending D to GHD $D'(T', \chi', \lambda')$ with active/inactive labels, common-covers, and heights as described in Section 6.1, and then applying any sequence of leaf and unique-c-gc inactivations on D'. Then the resulting D' is a GHD with width at most $\max(\mathsf{w}, 3\mathsf{iw})$ and the height of each inactive vertex v is v's actual height in T'.*

```
Input: GHD D(T, χ, λ) for hypergraph H
Extend D into D'(T', χ', λ') as described in Section 6.1.
while(there are active nodes in T')
  Select at least 1/4 of the active vertices that are either leaves L'
                              or non-adjacent unique-c-gc vertices U'
  Inactivate each l ∈ L', each u ∈ U' and the child of u
return D'
```

Figure 5 Log-GTA.

6.3 Log-GTA

Finally, we present our Log-GTA algorithm. Log-GTA takes a GHD D and extends it into D' by following the procedure in Section 6.1. Then, Log-GTA iteratively inactivates a set of active leaves L' and nonadjacent unique-c-gc vertices U' (along with the children of U'), which constitute at least $\frac{1}{4}$ fraction of the remaining active vertices in T' by Lemma 18, until all vertices are inactive. Figure 5 shows the pseudocode of Log-GTA. We next state two lemmas about Log-GTA and then prove our second main result.

▶ **Lemma 21.** *Log-GTA takes $O(\log(|V(T)|))$ iterations.*

Proof. Observe that both leaf inactivation and unique-c-gc inactivation decrease the number of active vertices in T' by 1. In each iteration the number of active vertices decreases by a factor of $\frac{1}{4}$. Therefore the algorithm terminates in $O(\log(|V(T)|)$ iterations. ◀

▶ **Lemma 22.** *The height of each inactive vertex v is at most the iteration number at which v was inactivated.*

Proof. By Corollary 20, the heights assigned to vertices are their correct heights in the final GHD returned. Moreover the height numbers start at 0 in the first iteration and increase by at most one in each iteration, because in each iteration, the height of each inactivated vertex v is set to the maximum of v's inactive children plus one. Therefore the height numbers assigned in iteration i are less than i, completing the proof. ◀

▶ **Theorem 23** (Second Main Result)**.** *Given any GHD $D(T, \chi, \lambda)$ with width* w*, intersection width* iw*, we can construct a GHD $D'(T', \chi', \lambda')$ where width $w' \leq \max(\mathsf{w}, 3\mathsf{iw})$, $\text{depth}(T') = \min\{\text{depth}(T), O(\log(|V(T)|))\}$.*

Proof. By Corollary 20 the width of D' is at most $\max(\mathsf{w}, 3\mathsf{iw})$. By Lemmas 19, 21 and 22, the height of each vertex v is v's true height in the tree and is at most the maximum iteration number, which is $O(\log(|V(T)|))$. Therefore, the depth of T' is $O(\log(|V(T)|))$. Also, the leaf and unique-c-gc inactivation operations never increase the depth of the tree, justifying that the depth of the final tree is $\min\{\text{depth}(T), O(\log(|V(T)|))\}$. ◀

Theorems 17 and 23 and Lemma 9 imply the following two results:

▶ **Corollary 24.** *Given a hypergraph H with n hyperedges, width* w*, intersection width* iw*, we can construct a $\log(n)$ depth GHD of H with width at most $\max(\mathsf{w}, 3\mathsf{iw})$.*

▶ **Theorem 25.** *Any query Q with width* w *can be executed in $O(\log(n))$ rounds and $O(n\text{B}(\text{IN}^{\max(\mathsf{w},3\mathsf{iw})} + \text{OUT}, M))$ communication.*

We note that Corollary 24 shows that, similar to Bodlaender's and Akatov's results about log-depth TDs and HDs, a similar and stronger property also holds for GHDs. In the longer version of our paper [1] we further show: (1) Log-GTA without any modifications recovers Bodlaender's result about TDs; and (2) a modification of Log-GTA recovers Bodlaender's and Akatov's results and a weaker version of our result.

Surprisingly, if we simulate GYM on PRAM using a log-depth GHD generated by Log-GTA, we show that any bounded-width query can be evaluated in logarithmic PRAM steps using polynomial number of processors, i.e., bounded-width queries are in the complexity class NC—a result that was proven by the ACQ algorithm [14]. This is interesting in itself, since we recover this positive parallel complexity result by using only a simple variant of Yannakakis's algorithm, which has been thought to be an inherently sequential algorithm.

7 C-GTA (Constant-depth GHD Transformation Algorithm)

Our C-GTA algorithm is based on the following observation. For any two adjacent nodes $t_1, t_2 \in V(T)$, we can "merge" them and replace them with a new node $t \in V(T)$ and set $\chi(t) = \chi(t_1) \cup \chi(t_2)$, $\lambda(t) = \lambda(t_1) \cup \lambda(t_2)$ and set t's neighbors to the union of neighbors of t_1 and t_2. As long as t_1 and t_2 were either neighbors, or both leaves with the same parent, T remains a valid GHD tree after this operation. C-GTA operates as follows:

1. For each node u that has an even number of leaves as children, divide u's leaves into pairs and merge each pair.
2. For each node u that has an odd number of leaves as children, divide the leaves into pairs and merge them, and merge the remaining leaf with u.
3. For each vertex u that has a unique child c, if c has an even number of leaf children, then merge u and c.

If T has L leaves and a set U of pairwise non-adjacent unique-c-gc nodes, then the above procedure removes at least $\frac{\max(L,U)}{2}$ nodes from T. We next state a combinatorial lemma to bound this quantity, which is proved in the longer version of our paper [1].

▶ **Lemma 26.** *Suppose a tree has N nodes, L of which are leaves, and U of which are unique-c-gc nodes. Then $4L + U \geq N + 2$.*

By Lemma 26, $\max(L, U)$ is at least $n/8$. Therefore, the resulting tree T' has at most $15n/16$ nodes, and width $\leq 2\mathsf{w}$. We can use this operation repeatedly to reduce the number of vertices while increasing width. We can then apply Log-GTA to get the following theorem:

▶ **Theorem 27.** *For any query Q with a width-w, intersection width-iw GHD $D = (T, \chi, \lambda)$, for any i, there exists a GHD $D' = (T', \chi', \lambda')$ with width $\leq 2^i . \max(\mathsf{w}, 3\mathsf{iw})$ and depth $\leq \log((\frac{15}{16})^i n)$.*

Thus we can further trade off communication by constructing trees of even lower depth than a single invocation of Log-GTA.

8 Conclusions and Future Work

We have shown that by using GYM as a primitive and proving different properties of depths and widths of GHDs of queries, we can trade off communication against number of rounds of computations. We believe our approach of discovering such tradeoffs using different combinatorial properties of GHDs is a promising direction for future work. An important open area is to explore other GHD construction algorithms that output GHDs with different

depths and widths. Specifically, we believe it is plausible that an algorithm can generate polynomially lower depth GHDs with twice their widths (instead of the constant depth reduction of C-GTA). We also plan to investigate the lower bounds on the communication costs of algorithms that run $O(\log(n))$ or $O(n)$ rounds.

Acknowledgements. We would like to thank the anonymous ICDT reviewers whose comments and suggestions about many parts of this paper were critical when preparing the final version of this paper.

References

1 F. Afrati, M. Joglekar, C. Ré, Salihoglu S., and J. D. Ullman. GYM: A Multiround Join Algorithm in MapReduce and Its Analysis. *CoRR*, abs/1410.4156, 2014. URL: `http://arxiv.org/abs/1410.4156`.

2 F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and Lower Bounds on the Cost of a Map-Reduce Computation. In *VLDB*, 2013.

3 F. N. Afrati and J. D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE TKDE*, 2011.

4 D. Akatov. *Exploiting Parallelism in Decomposition Methods for Constraint Satisfaction.* PhD thesis, University of Oxford, 2010.

5 Apache Hadoop. `http://hadoop.apache.org/`.

6 P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. In *PODS*, 2013.

7 H. L. Bodlaender. NC-Algorithms for Graphs with Small Treewidth. In *Graph-Theoretic Concepts in Computer Science*, 1988.

8 C. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *TCS*, 2000.

9 J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

10 A. Durand and E. Grandjean. The Complexity of Acyclic Conjunctive Queries Revisited. *CoRR*, abs/cs/0605008, 2006. URL: `http://arxiv.org/abs/cs/0605008`.

11 M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, Searching, and Simulation in the Mapreduce Framework. In *ISAAC*, 2011.

12 G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. In *J. Comput. Syst. Sci*, 2003.

13 G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Hypertree Decompositions: Structure, Algorithms, and Applications. In *WG*, 2005.

14 G. Gottlob, N. Leone, and F. Scarcello. Advanced Parallel Algorithms for Acyclic Conjunctive Queries. Technical report, Vienna University of Technology, 1998.

15 G. Gottlob, N. Leone, and F. Scarcello. On Tractable Queries and Constraints. In *DEXA*, 1999.

16 G. Gottlob, N. Leone, and F. Scarcello. The Complexity of Acyclic Conjunctive Queries. *J. ACM*, 2001.

17 G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. In *J. Comput. Syst. Sci.*, 2002.

18 D. Halperin, V. Teixeira de Almeida, L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, S. Xu, M. Balazinska, B. Howe, and D. Suciu. Demonstration of the Myria Big Data Management Service. In *SIGMOD*, 2014.

19 S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In *New Frontiers in Information and Software as Services*. Springer Berlin Heidelberg, 2011.

20 S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud. In *International Conference on Cloud Computing Technology and Science*, 2010.

21 H. Karloff, S. Suri, and S. Vassilvitskii. A Model of Computation for MapReduce. In *SODA*, 2010.

22 P. Koutris, P. Beame, and D. Suciu. Worst-Case Optimal Algorithms for Parallel Query Processing. In *ICDT*, 2016.

23 P. Koutris and D. Suciu. Parallel Evaluation of Conjunctive Queries. In *PODS*, 2011.

24 C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

25 A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round Tradeoffs for MapReduce Computations. In *ICS*, 2012.

26 N. Robertson and P. D. Seymour. Graph Minors. II. Algorithmic Aspects of Tree-width. *Journal of Algorithms*, 1986.

27 Spark SQL. `https://spark.apache.org/sql/`.

28 A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A Warehousing Solution Over a Map-Reduce Framework. *VLDB*, 2009.

29 L. G. Valiant. A Bridging Model for Parallel Computation. *CACM*, August 1990.

30 M. Yannakakis. Algorithms for Acyclic Database Schemes. In *VLDB*, 1981.

31 Zaharia, M. and Chowdhury, M. and Franklin, M. J. and Shenker, S. and Stoica, I. Spark: Cluster Computing with Working Sets. In *HotCloud*, 2010.

Top-k Querying of Unknown Values under Order Constraints

Antoine Amarilli[1], Yael Amsterdamer[2], Tova Milo[3], and Pierre Senellart[4,5]

1 **LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France**
`first.last@telecom-paristech.fr`
2 **Bar Ilan University, Ramat Gan, Israel**
`first.last@biu.ac.il`
3 **Tel Aviv University, Tel Aviv, Israel**
`last@cs.tau.ac.il`
4 **DI, École normale supérieure, PSL Research University, Paris, France; and Inria Paris, Paris, France**
`first.last@ens.fr`

Abstract

Many practical scenarios make it necessary to evaluate top-k queries over data items with partially unknown values. This paper considers a setting where the values are taken from a numerical domain, and where some *partial order constraints* are given over known and unknown values: under these constraints, we assume that all possible worlds are equally likely. Our work is the first to propose a principled scheme to derive the value distributions and expected values of unknown items in this setting, with the goal of computing estimated top-k results by interpolating the unknown values from the known ones. We study the complexity of this general task, and show tight complexity bounds, proving that the problem is intractable, but can be tractably approximated. We then consider the case of tree-shaped partial orders, where we show a constructive PTIME solution. We also compare our problem setting to other top-k definitions on uncertain data.

1998 ACM Subject Classification H.2 Database Management

Keywords and phrases uncertainty, partial order, unknown values, crowdsourcing, interpolation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.5

1 Introduction

Many data analysis tasks involve queries over ordered data, such as maximum and top-k queries, which must often be evaluated in presence of *unknown data values*. This problem occurs in many real-life scenarios: retrieving or computing exact data values is often expensive, but querying the partially unknown data may still be useful to obtain approximate results, or to decide which data values should be retrieved next. In such contexts, we can often make use of *order constraints* relating the data values, even when they are unknown: for instance, we know that object A is preferred to object B (though we do not know their exact rating).

This paper thus studies the following general problem. We consider a set of numerical values, some of which are unknown, and we assume a *partial order* on these values: we may know that $x \geqslant y$ should hold although the values x or y are unknown. Our goal is to *estimate* the unknown values, in a principled way, and to evaluate top-k queries, namely find the items with (estimated) highest values.

Without further information, one may assume that every valuation compatible with the order constraints is equally likely, i.e., build a probabilistic model where valuations are

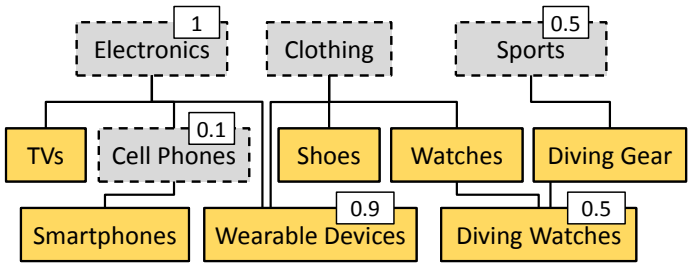


Figure 1 Sample catalog taxonomy with compatibility scores.

uniformly distributed. Indeed, uniform distributions in the absence of prior knowledge is a common assumption in probabilistic data management [1, 12, 34] for continuous distributions on data values within an interval; here we generalize to a uniform distribution over multiple unknown values. Though the distribution is uniform, the dependencies between values lead to non-trivial insights about unknown values and top-k results, as we will illustrate.

Illustrative example. We consider a specific application setting where our problem occurs. Consider a scenario where products are classified in a catalog taxonomy (Figure 1) using human input: the relevance of a product to any category is captured by a *compatibility score*. Assessing this compatibility is often left to human judgement rather than attempting to derive it from records or statistics [49, 9, 40]. Thus we assume scores are obtained via questions to domain experts or to a crowd of unqualified users[1]. Our goal is to assign the product to the top-k most compatible categories among a set of *end categories* (in yellow with a solid border), as opposed to *virtual categories* (dashed border). The virtual categories generalize the end categories, and allow us to ask broader questions to experts, but we do not try to assign products to them, e.g., they do not have a dedicated page in our online store.

Imagine now that the product to classify is a *smartwatch*, and that we want to find the top-2 end categories for it. We asked an expert for its compatibility score with some categories (both end and virtual categories), which we indicate in Figure 1. Because expert input is costly, however, we wish to choose the top-2 end categories based on the incomplete information that we have. The naïve answer is to look only at categories with known scores, and to identify Wearable Devices and Diving Watches as the best end categories.

In this scenario, however, we can impose a natural partial order over the scores, both known and unknown: any product that belongs to a specific category (e.g., Smartphones) conceptually also belongs to each of the more general categories (e.g., Cell Phones). We can thus require that if a category x is a sub-category of y, then the product's compatibility score for x should be at most its score for y. This can be explained in the instructions to the humans providing the scores, and enforced by the user interface. Beyond this constraint, we do not make any further assumption on the scores.

Now, note that due to order constraints, the scores of Watches and Diving Gear, while unknown, cannot be lower than that of Diving Watches; so either of the two could replace Diving Watches in the top-2 answer. To choose between these categories, we observe that the score of Diving Gear must be exactly 0.5 (which bounds it from above and below). In contrast, as the score of Wearable Devices is 0.9, Clothing has a score of at least 0.9, so the score of Watches can be anything between 0.5 and an unknown value which is $\geqslant 0.9$. A

[1] In the latter case, we aggregate the answers of multiple workers to obtain the compatibility score.

better top-2 answer is thus Wearable Devices and Watches, the latter being likely to have a higher score than Diving Watches (or Diving Gear).

Other application domains. Beyond the crowdsourcing example that we illustrate, our setting is relevant to many other application domains involving different types of unknown data. For instance, in the domain of Web services [46], one may wish to find the k-most relevant apartments for a user, given user-provided criteria on the apartment (price range, location, size, etc.) but no precise way to aggregate them into a relevance function. In this case, order constraints may be imposed on the unknown relevance of apartments, whenever one apartment *dominates* another (i.e., is better for each criterion); exact-value constraints may capture user-provided ratings to viewed apartments; and a top-k query could be used, e.g., to select the most relevant apartments among those available for sale.

As another example, in the context of top-k queries over sensor data [25, 35], one may wish to find the k-fastest drivers in a certain region given partial data from speedometers and street cameras; comparing the progress and locations of vehicles may yield partial order constraints on their speed. Other domains include, e.g., data mining [3], managing preference data [48], or finding optimal values of black-box functions expressing cost, running time, etc.

Contributions. As previously mentioned, we assume a uniform probability distribution over valuations of unknown items, which we capture formally in our model via possible-world semantics. We then use the *expected values* of items as an estimate of their unknown values, for the sake of top-k computation (see Section 2). Our work presents three main contributions using this general model, as follows.

First, in Section 3 we present a *general and principled scheme to interpolate unknown values from known ones under partial order constraints*, and thereby obtain the top-k such values. We implement this in an algorithm that is polynomial in the number of possible item orderings, and consequently show that in the worst case it is in $\mathrm{FP}^{\#\mathrm{P}}$ in the size of the input.[2] The problem of finding expected values has a geometric characterization as centroid computation in high-dimensional polytopes (as we explain further); however, our $\mathrm{FP}^{\#\mathrm{P}}$ membership result goes beyond existing computational geometry results since the constraints that we consider, namely, partial order and exact-value constraints, correspond to special classes of polytopes not studied in the context of geometry. Indeed, centroid computation is generally not in $\mathrm{FP}^{\#\mathrm{P}}$ [32, 42]. Our work also departs from previous work on top-k queries over incomplete or probabilistic data [15, 28, 46]: we do not make the simplifying assumption that item distributions are independent and given, but rather study the effect of constraints on individual item distributions.

Our second main contribution, in Section 4, is to establish *hardness results*, and specifically, a matching lower $\mathrm{FP}^{\#\mathrm{P}}$ bound for top-k computation. While the #P-hardness of computing expected values follows from the geometric characterization of the problem [42], we show that top-k is hard even without computing expected values. Hence the $\mathrm{FP}^{\#\mathrm{P}}$ bound is tight for both interpolation and top-k; this shows that the assumption regarding variable independence in previous work, that enables PTIME solutions [15, 28, 46], indeed simplifies the problem. To complete the picture we discuss possible approximation schemes for the interpolation and top-k problems, again by the connection to centroid computation.

[2] #P is the class of counting problems that return the number of solutions of NP problems. $\mathrm{FP}^{\#\mathrm{P}}$ is the class of function problems that can be computed in PTIME using a #P oracle.

Our third main contribution, in Section 5, is the study of *tractable cases*, following the hardness of the general problem and the high complexity of approximation. We devise a PTIME algorithm to compute expected values and top-k results when the order constraints are *tree-shaped* or decomposable to trees. This class of constraints is commonly encountered in the context of unknown values (e.g., taxonomies of products are often trees rather than DAGs); yet, to our knowledge, the corresponding polytopes have no equivalents in the context of computational geometry.

Our results also include a review of existing definitions for top-k over uncertain data, which motivates our particular choice of definition (in Section 6). We survey related work in more depth in Section 7 and conclude in Section 8. Full proofs of our results can be found in [5], an extended version of this paper.

2 Preliminaries and Problem Statement

This section introduces the formal definitions for the problem that we study in this paper. We model known and unknown item values as *variables*, and order constraints as *equalities* and *inequalities* over them. Then we define the possible valuations for the variables via possible-world semantics, and use this semantics to define a uniform distribution where all worlds are equally likely. The problem of top-k querying over unknown values can then be formally defined with respect to the expected values of variables in the resulting distribution.

2.1 Unknown Data Values under Constraints

Our input includes a set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of variables with unknown values $v(x_1), \ldots, v(x_n)$, which we assume[3] to be in the range $[0, 1]$. We consider two kinds of constraints over them:

- **order constraints**, written $x_i \leqslant x_j$ for $x_i, x_j \in \mathcal{X}$, encoding that $v(x_i) \leqslant v(x_j)$;
- **exact-value constraints** to represent variables with known values, written[4] $x_i = \alpha$ for $0 \leqslant \alpha \leqslant 1$ and for $x_i \in \mathcal{X}$, encoding that $v(x_i) = \alpha$.

In what follows, a *constraint set* with constraints of both types is typically denoted $\mathcal{C}$. We assume that constraints in $\mathcal{C}$ are *not contradictory* (e.g., we forbid $x = 0.1$, $y = 0.2$, $y \leqslant z$, and $z \leqslant x$), and that they are *closed under implication*: e.g., if $x = \alpha$, $y = \beta$ are given, and $\alpha \leqslant \beta$, then $x \leqslant y$ is implied and thus should also be in $\mathcal{C}$. We can check in PTIME that $\mathcal{C}$ is non-contradictory by simply verifying that it does not entail a false inequality on exact values (e.g., $0.2 \leqslant 0.1$ as in our previous example). The closure of $\mathcal{C}$ can be found in PTIME as a transitive closure computation [27] that also considers exact-value constraints. We denote by $\mathcal{X}_{\text{exact}}$ the subset of $\mathcal{X}$ formed of variables with exact-value constraints.

▶ **Example 1.** In the product classification example from the Introduction, a variable $x_i \in \mathcal{X}$ would represent the compatibility score of the product to the i-th category. If the score is known, we would encode it as a constraint $x_i = \alpha$. In addition, $\mathcal{C}$ would contain the order constraint $x_i \leqslant x_j$ whenever category i is a sub-category of j (recall that the score of a sub-category cannot be higher than that of an ancestor category).

[3] Our results extend to other bounded, continuous ranges, because we can rescale them to fall in $[0, 1]$.
[4] The number α is written as a rational number, represented by its numerator and denominator.

2.2 Possible World Semantics

The unknown data captured by $\mathcal{X}$ and $\mathcal{C}$ makes infinitely many valuations of $\mathcal{X}$ possible (including the true one). We model these options via possible world semantics: a *possible world* w for a constraint set $\mathcal{C}$ over $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a vector of values $w = (v_1, \ldots, v_n) \in [0,1]^n$, corresponding to setting $v(x_i) := v_i$ for all i, such that all the constraints of $\mathcal{C}$ hold under this valuation. The set of all possible worlds is denoted by $\mathrm{pw}_{\mathcal{X}}(\mathcal{C})$, or by $\mathrm{pw}(\mathcal{C})$ when $\mathcal{X}$ is clear from context.

Notice that $\mathcal{C}$ can be encoded as a set of *linear constraints*, i.e., a set of inequalities between linear expressions on $\mathcal{X}$ and constants in $[0,1]$. Thus, following common practice in linear programming, the feasible region of a set of linear constraints ($\mathrm{pw}(\mathcal{C})$ in our setting) can be characterized geometrically as a convex polytope, termed the *admissible polytope*: writing $n := |\mathcal{X}|$, each linear constraint defines a feasible half-space of $\mathbb{R}^n$ (e.g., the half-space where $x \leqslant y$), and the convex polytope $\mathrm{pw}(\mathcal{C})$ is the intersection of all half-spaces. In our setting the polytope $\mathrm{pw}(\mathcal{C})$ is bounded within $[0,1]^n$, and it is non-empty by our assumption that $\mathcal{C}$ is not contradictory. With exact-value constraints, or order constraints such as $x_i \leqslant x_j$ and $x_j \leqslant x_i$, it may be the case that the dimension of this admissible polytope is less than $|\mathcal{X}|$. Computing this dimension can easily be done in PTIME (see, e.g., [44]).

▶ **Example 2.** Let $\mathcal{X} = \{x, y, z\}$. If $\mathcal{C} = \{x \leqslant y\}$, the admissible polytope has dimension 3 and is bounded by the planes defined by $x = y$, $x = 0$, $y = 1$, $z = 0$ and $z = 1$. If we add to $\mathcal{C}$ the constraint $y = 0.3$, the admissible polytope is a 2-dimensional rectangle bounded by $0 \leqslant x \leqslant 0.3$ and $0 \leqslant z \leqslant 1$ on the $y = 0.3$ plane. We cannot add, for example, the constraint $x = 0.5$, because $\mathcal{C}$ would become contradictory.

2.3 Probability Distribution

Having characterized the possible worlds of $\mathrm{pw}(\mathcal{C})$, we assume a *uniform* probability distribution over $\mathrm{pw}(\mathcal{C})$, as indicated in the Introduction. This captures the case when all possible worlds are equally likely, and is a natural choice when we have no information about which valuations are more probable.

Since the space of possible worlds is continuous, we formally define this distribution via a probability density function (pdf), as follows. Let $\mathcal{X}$ and $\mathcal{C}$ define a d-dimensional polytope $\mathrm{pw}_{\mathcal{X}}(\mathcal{C})$ for some integer d. The *d-volume* (also called the Lebesgue measure [29] on $\mathbb{R}^d$) is a measure for continuous subsets of d-dimensional space, which coincides with length, area, and volume for dimensions 1, 2, and 3, respectively. We denote by $V_d(\mathcal{C})$ the d-volume of the admissible polytope, or simply $V(\mathcal{C})$ when d is the dimension of $\mathrm{pw}(\mathcal{C})$.

▶ **Definition 3.** The *uniform pdf* p maps each possible world $w \in \mathrm{pw}(\mathcal{C})$ to the constant $p(w) := 1/V(\mathcal{C})$.

2.4 Top-k Queries

We are now ready to formally define the main problem studied in this paper, namely, the evaluation of *top-k queries* over unknown data values. The queries that we consider retrieve the k items that are estimated to have the highest values, *along with their estimated values*, with ties broken arbitrarily. We further allow queries to apply a *selection operator* σ on the items before performing the top-k computation. In our example from the Introduction, this is what allows us to select the top-k categories among only the end categories. We denote the subset of $\mathcal{X}$ selected by σ as $\mathcal{X}_\sigma$.

If all item values are known, the semantics of top-k queries is clear. In presence of unknown values, however, the semantics must be redefined to determine how the top-k items and their values are estimated. In this paper, we estimate unknown items by their *expected value over all possible worlds*, i.e., their expected value according to the uniform pdf p defined above on $\mathrm{pw}(\mathcal{C})$. This corresponds to *interpolating* the unknown values from the known ones, and then querying the result. We use these interpolated values to define the top-k problem as computing the k variables with the highest expected values, but we also study on its own the interpolation problem of computing the expected values.

To summarize, the two formal problems that we study on constraint sets are:

Interpolation. Given a constraint set $\mathcal{C}$ over $\mathcal{X}$ and variable $x \in \mathcal{X}$, the *interpolation problem* for x is to compute the expected value of x in the uniform distribution over $\mathrm{pw}_{\mathcal{X}}(\mathcal{C})$.

Top-k. Given a constraint set $\mathcal{C}$ over $\mathcal{X}$, a selection predicate σ, and an integer k, the *top-k computation problem* is to compute the ordered list of the k maximal expected values of variables in $\mathcal{X}_\sigma$ (or less if $|\mathcal{X}_\sigma| \leqslant k$), with ties broken arbitrarily.

We review other definitions of top-k on uncertain data in Section 6, where we justify our choice of semantics.

Alternate phrasing. The Interpolation problem can also be defined geometrically, as the computation of the *centroid* (or *center of mass*) of the admissible polytope: the point G such that all vectors relative to G originating at points within the polytope sum to zero. The constraints that we study correspond to a special kind of polytopes, for which we will design a specific algorithm in the next section, and derive an $\mathrm{FP}^{\#\mathrm{P}}$ membership bound which does not hold for general polytopes (as explained in the Introduction). However, the geometric connection will become useful when we study the complexity of our problem in Section 4.1.

3 An Algorithm for Interpolation and Top-k

Having defined formally the problems that we study, we begin our complexity analysis by designing an algorithm that computes the expected value of variables.

The algorithm enumerates all possible orderings of the variables (to be defined formally below), but it is still nontrivial: we must handle exact-value constraints specifically, and we must compute the probability of each ordering to determine its weight in the overall expected value computation. From the algorithm, we will deduce that our interpolation and top-k problems are in $\mathrm{FP}^{\#\mathrm{P}}$.

Eliminating ties. To simplify our study, we will eliminate from the start the problem of *ties*, which will allow us to assume that values in all worlds are totally ordered. We say that a possible world $w = (v_1, \ldots, v_n)$ of $\mathcal{C}$ has a *tie* if $v_i = v_j$ for some i, j. Note that occasional ties, not enforced by $\mathcal{C}$, have *an overall probability of* 0: intuitively, if the admissible polytope is d-dimensional, then all the worlds where $v_i = v_j$ correspond to a $(d-1)$-dimensional hyperplane bounding or intersecting the polytope. A finite set of such hyperplanes (for every pair of variables) has total d-volume 0. Since our computations (volume, expected value) involve integrating over possible worlds, a set of worlds with total probability 0 does not affect the result.

What is left is to consider ties enforced by $\mathcal{C}$ (and thus having probability 1). In such situations, we can rewrite $\mathcal{C}$ by merging these variables to obtain an equivalent constraint set where ties have probability 0. Formally:

▶ **Lemma 4.** *For any constraint set $\mathcal{C}$, we can construct in PTIME a constraint set $\mathcal{C}'$ such that the probability that the possible worlds of $\mathcal{C}'$ have a tie (under the uniform distribution) is zero, and such that any interpolation or top-k computation problem on $\mathcal{C}$ can be reduced in PTIME to the same type of problem on $\mathcal{C}'$.*

Hence, we assume from now on that ties have zero probability in $\mathcal{C}$, so that we can ignore possible worlds with ties without affecting the correctness of our analysis. Note that this implies that all of our results also hold for *strict inequality constraints*, of the forms $x < y$ and $x \neq y$.

Under this assumption, we first study in Section 3.1 the case where $\mathcal{C}$ is a total order. We then handle arbitrary $\mathcal{C}$ by aggregating over possible variable orderings, in Section 3.2.

3.1 Total Orders

In this section we assume $\mathcal{C}$ is a total order $\mathcal{C}_1^n(\alpha, \beta)$ defined as $x_0 \leqslant x_1 \leqslant \cdots \leqslant x_n \leqslant x_{n+1}$, where $x_0 = \alpha$ and $x_{n+1} = \beta$ are variables with exact-value constraints in $\mathcal{X}_{\text{exact}}$.

We first consider *unfragmented* total orders, where $x_1, \ldots, x_n \notin \mathcal{X}_{\text{exact}}$. In this case, we can show that the expected value of x_i, for $1 \leqslant i \leqslant n$, corresponds to a *linear interpolation* of the unknown variables between α and β, namely: $\frac{i}{n+1} \cdot (\beta - \alpha) + \alpha$. This can be shown formally via a connection to the expected value of the order statistics of samples from a uniform distribution, which follows a Beta distribution [22].

Now consider the case of *fragmented* total orders, where $\mathcal{C}$ is allowed to contain more exact-value constraints than the ones on α and β. We observe that we can *split* the total order into *fragments*: by cutting at each variable that has an exact-value constraint, we obtain sub-sequences of variables which follow an unfragmented total order. We can then compute the expected values of each fragment independently, and compute the total order volume as the product of the fragment volumes. The correctness of this computation follows from a more general result (Lemma 12) stated and proven in Section 5.

Hence, given a constraint set $\mathcal{C}$ imposing a (possibly fragmented) total order, the expected value of x_i can be computed as follows. If $x_i \in \mathcal{X}_{\text{exact}}$, analysis is trivial. Otherwise, we consider the fragment $\mathcal{C}_{p+1}^{q-1}(v_p, v_q)$ that contains x_i; namely, p is the maximal index such that $0 \leqslant p < i$ and $x_p \in \mathcal{X}_{\text{exact}}$, and q is the minimal index such that $i < q \leqslant n+1$ and $x_q \in \mathcal{X}_{\text{exact}}$. The expected value of x_i can then be computed within $\mathcal{C}_{p+1}^{q-1}(v_p, v_q)$ using linear interpolation.

The following proposition summarizes our findings:

▶ **Proposition 5.** *Given a constraint set $\mathcal{C}$ implying a total order, the expected value of any variable $x_i \in \mathcal{X}$ can be computed in PTIME.*

3.2 General Constraint Sets

We can now extend the result for total orders to an expression of the expected value for a general constraint set $\mathcal{C}$. We apply the previous process to each possible total ordering of the variables, and aggregate the results. To do this, we define the notion of *linear extensions*, inspired by partial order theory:

▶ **Definition 6.** Given a constraint set $\mathcal{C}$ over $\mathcal{X}$, we say that a constraint set $\mathcal{T}$ is a *linear extension* of $\mathcal{C}$ if (i) $\mathcal{T}$ is a total order; (ii) the exact-value constraints of $\mathcal{T}$ are exactly those of $\mathcal{C}$; and (iii) $\mathcal{C} \subseteq \mathcal{T}$, namely every constraint $x \leqslant y$ in $\mathcal{C}$ also holds[5] in $\mathcal{T}$.

[5] The linear extensions of $\mathcal{C}$ in this sense are thus exactly the linear extensions of the partial order on $\mathcal{X}$ imposed by $\mathcal{C}$: this partial order is indeed antisymmetric because $\mathcal{C}$ has no ties.

Algorithm 1: Compute the expected value of a variable.

Input: Constraint set $\mathcal{C}$ on variables $\mathcal{X}$ with $n := |\mathcal{X}|$ where ties have probability 0 and where value range $[0, 1]$ is enforced by constraints; variable $x \in \mathcal{X}$

Output: Expected value of x

1 **if** *x is in $\mathcal{X}_{\text{exact}}$, i.e., has an exact-value constraint in $\mathcal{C}$ to some v* **then return** v;
2 $\mathbb{E}_{\mathcal{C}}[x] \leftarrow 0$; $V(\mathcal{C}) \leftarrow 0$;
3 $m \leftarrow |\mathcal{X}_{\text{exact}}|$;
4 Write $x_0 < \cdots < x_{m-1}$ the variables of $\mathcal{X}_{\text{exact}}$ and $v_0 < \cdots < v_{m-1}$ their values;
5 **foreach** *linear extension $\mathcal{T}$ of $\mathcal{C}$* **do**
6 Write $i_0 < \cdots < i_{m-1}$ the *indices* in $\mathcal{T}$ of variables $x_0, \ldots, x_{m-1}$ in $\mathcal{X}_{\text{exact}}$;
7 Write k the *index* in $\mathcal{T}$ of the variable x;
8 Write i_j, i_{j+1} the indices of variables from $\mathcal{X}_{\text{exact}}$ s.t. $i_j < k < i_{j+1}$;
9 $\mathbb{E}_{\mathcal{T}}[x] \leftarrow \text{ExpectedValFrag}(i_j, i_{j+1}, k, v_j, v_{j+1})$; `// Expected value of` x `in` $\mathcal{T}$
10 $V(\mathcal{T}) \leftarrow \prod_{l=0}^{m-2} \text{VolumeFrag}(i_l, i_{l+1}, v_l, v_{l+1})$; `// Volume of` $\mathcal{T}$
11 $\mathbb{E}_{\mathcal{C}}[x] \leftarrow \mathbb{E}_{\mathcal{C}}[x] + V(\mathcal{T}) \times \mathbb{E}_{\mathcal{T}}[x]$; `// Sum exp. value of` x `weighted by` $V(\mathcal{T})$
12 $V(\mathcal{C}) \leftarrow V(\mathcal{C}) + V(\mathcal{T})$; `// Sum of total order volumes`
13 **return** $\frac{\mathbb{E}_{\mathcal{C}}[x]}{V(\mathcal{C})}$;
14 **Function** ExpectedValFrag(p, q, k, α, β)
Input: $p < q$: indices; k: requested variable index; $\alpha < \beta$: exact values at p, q resp.
Output: Expected value of variable x_k in the fragment $\mathcal{C}_{p+1}^{q-1}(\alpha, \beta)$
15 $n \leftarrow q - p - 1$; `// num. of variables in the fragment which are` $\notin \mathcal{X}_{\text{exact}}$
16 **return** $\frac{k-p}{n+1} \cdot (\beta - \alpha) + \alpha$; `// linear interpolation (see Section 3.1)`
17 **Function** VolumeFrag(p, q, α, β)
Input: $p < q$: indices; $\alpha < \beta$: exact values at p, q resp.
Output: $V(\mathcal{C}_{p+1}^{q-1}(\alpha, \beta))$: volume of the total order fragment between indices p, q
18 $n \leftarrow q - p - 1$; `// num. of variables in the fragment which are` $\notin \mathcal{X}_{\text{exact}}$
19 **return** $\frac{(\beta-\alpha)^n}{n!}$; `// Volume of` $[\alpha, \beta]^n$ `divided by num. of total orders`

Algorithm 1 presents our general scheme to compute the expected value of a variable $x \in \mathcal{X}$ under an arbitrary constraint set $\mathcal{C}$, assuming the uniform distribution on $\text{pw}(\mathcal{C})$.

The algorithm iterates over each linear extension $\mathcal{T}$ of $\mathcal{C}$, and computes the expected value of x in $\mathcal{T}$ and the overall probability of $\mathcal{T}$ in $\text{pw}(\mathcal{C})$. A linear extension is a total order, so x is within a particular fragment of it, namely, between the indices of two consecutive variables with exact-value constraints, i_j and i_{j+1}. The *expected value of x in $\mathcal{T}$*, denoted by $\mathbb{E}_{\mathcal{T}}[x]$, is then affected only by the constraints and variables of this fragment, and can be computed using linear interpolation by the function ExpectedValFrag (line 9).

Now, the final expected value of x in $\mathcal{C}$ is the average of all $\mathbb{E}_{\mathcal{T}}[x]$ weighted by the probability of each linear extension $\mathcal{T}$, i.e., the volume of $\text{pw}(\mathcal{T})$ divided by the volume of $\text{pw}(\mathcal{C})$. Recall that, by Lemma 4, worlds with ties have total volume 0 and do not affect this expected value. We compute the volume of $\mathcal{T}$ as the *product of volumes of its fragments* (line 10). The volume of a fragment, computed by function VolumeFrag, is the volume of $[\alpha, \beta]^n$, i.e., all assignments to the n variables of the fragment in $[\alpha, \beta]$, divided by the number of orderings of these variables, to obtain the volume of one specific order (line 19).

The complexity of Algorithm 1 is polynomial in the number of linear extensions of $\mathcal{C}$, as we can enumerate them in constant amortized time [41]. However, in the general case, there may be up to $|\mathcal{X}|!$ linear extensions. To obtain an upper bound in the general case, we note that

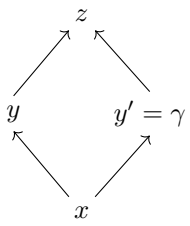
we can rescale all constraints so that all numbers are integers, and then nondeterministically sum over the linear extensions. This yields our $\mathrm{FP}^{\#\mathrm{P}}$ upper bound:

▶ **Theorem 7.** *Given a constraint set $\mathcal{C}$ over $\mathcal{X}$ and $x \in \mathcal{X}$ (resp., and a selection predicate σ, and an integer k), determining the expected value of x in $\mathrm{pw}(\mathcal{C})$ under the uniform distribution (resp., the top-k computation problem over $\mathcal{X}$, $\mathcal{C}$, and σ) is in $\mathrm{FP}^{\#\mathrm{P}}$.*

The $\mathrm{FP}^{\#\mathrm{P}}$ membership for interpolation does not extend to centroid computation in general convex polytopes, which is not in $\mathrm{FP}^{\#\mathrm{P}}$ [32, 42]. Our algorithm thus relies on the fact that the polytope $\mathrm{pw}(\mathcal{C})$ is of a specific form, defined with order and exact-value constraints. The same upper bound for the top-k problem immediately follows. We will show in Section 4 that this $\mathrm{FP}^{\#\mathrm{P}}$ upper bound is tight.

We also provide a complete example to illustrate the constructions of this section.

Full Example. We exemplify our scheme on variables $\mathcal{X} = \{x, y, y', z\}$ and on the constraint set $\mathcal{C}$ generated by the order constraints $x \leqslant y$, $y \leqslant z$, $x \leqslant y'$, $y' \leqslant z$ and the exact-value constraint $y' = \gamma$ for some fixed $0 < \gamma < 1$. Remember that we necessarily have $0 \leqslant x$ and $z \leqslant 1$ as well. The constraints of $\mathcal{C}$ are closed under implication, so they also include $x \leqslant z$. The figure shows the Hasse diagram of the partial order defined by $\mathcal{C}$ on $\mathcal{X}$. Note that ties have a probability of zero in $\mathrm{pw}(\mathcal{C})$.



The two linear extensions of $\mathcal{C}$ are $\mathcal{T}_1 : x \leqslant y \leqslant y' \leqslant z$ and $\mathcal{T}_2 : x \leqslant y' \leqslant y \leqslant z$. Now, $\mathcal{T}_1$ is a fragmented total order, and we have $\mathrm{pw}(\mathcal{T}_1) = \mathrm{pw}_{\{x,y\}}(\mathcal{C}') \times \{\gamma\} \times [\gamma, 1]$ where $\mathcal{C}'$ is defined on variables $\{x, y\}$ by $0 \leqslant x \leqslant y \leqslant \gamma$. We can compute the volume of $\mathrm{pw}(\mathcal{T}_1)$ as $\alpha_1 = \frac{\gamma^2}{2} \times (1 - \gamma)$. Similarly the volume of $\mathrm{pw}(\mathcal{T}_2)$ is $\alpha_2 = \gamma \times \frac{(1-\gamma)^2}{2}$.

Let us compute the expected value of y for $\mathcal{C}$. In $\mathcal{T}_1$ its expected value is $\mu_1 = \mathbb{E}_{\mathcal{T}_1}[y] = \frac{2}{3} \cdot (\gamma - 0) + 0 = \frac{2}{3}\gamma$. In $\mathcal{T}_2$ its expected value is $\mu_2 = \mathbb{E}_{\mathcal{T}_2}[y] = \frac{1}{3} \cdot (1 - \gamma) + \gamma = \frac{1+2\gamma}{3}$. The overall expected value of y is the average of these expected values weighted by total order probabilities (volumes fractions), namely $\mathbb{E}_{\mathcal{C}}[y] = \frac{\alpha_1\mu_1 + \alpha_2\mu_2}{\alpha_1 + \alpha_2}$.

4 Hardness and Approximations

We next show that the intractability of Algorithm 1 in Section 3 is probably unavoidable. We first show matching lower bounds for interpolation and top-k in Section 4.1. We then turn in Section 4.2 to the problem of approximating expected values.

4.1 Hardness of Exact Computation

We now analyze the complexity of computing an exact solution to our two main problems. We show below a new result for the hardness of top-k. But first, we state the lower bound for the interpolation problem, which is obtained via the geometric characterization of the problem. In previous work, centroid computation is proven to be hard for *order polytopes*, namely, polytopes without exact-value constraints, which are a particular case of our setting:

▶ **Theorem 8. ([42], Theorem 1).** *Given a set $\mathcal{C}$ of order constraints and $x \in \mathcal{X}$, determining the expected value of x in $\mathrm{pw}(\mathcal{C})$ under the uniform distribution is $\mathrm{FP}^{\#\mathrm{P}}$-hard.*

We now show a new lower bound for top-k queries: interestingly, these queries are $\mathrm{FP}^{\#\mathrm{P}}$-hard *even if they do not have to return the expected values*. Recall that σ is the selection operator (see Section 2.4), which we use to compute top-k among a restricted subset

of variables. We can show hardness even for top-1 queries, and even when σ only selects two variables:

▶ **Theorem 9.** *Given a constraint set $\mathcal{C}$ over $\mathcal{X}$, a selection predicate σ, and an integer k, the top-k computation problem over $\mathcal{X}$, $\mathcal{C}$ and σ is $\mathrm{FP}^{\#\mathrm{P}}$-hard even if k is fixed to be 1, $|\mathcal{X}_\sigma|$ is 2, and the top-k answer does not include the expected value of the variables.*

Proof sketch. To prove hardness in this case, we reduce from interpolation. We show that a top-1 computation oracle can be used as a comparison oracle to compare the expected value of a variable x to any other rational value α, by adding a fresh element x' with an exact-value constraint to α and using σ to compute the top-1 among $\{x, x'\}$. What is more technical is to show that, given such a comparison oracle, we can perform the reduction and determine *exactly* the expected value v of x (a rational number) using only a *polynomial* number of comparisons to other rationals. This follows from a bound on the denominator of v, and by applying the rational number identification scheme of [39]. ◀

In settings where we do not have a selection operator (i.e., $\mathcal{X}_\sigma = \mathcal{X}$), we can similarly show the hardness of top-k (rather than top-1). See [5] for details.

4.2 Complexity of Approximate Computation

In light of the previous hardness results, we now review approximation algorithms, again via the geometric characterization of our setting. In Section 5, we will show a novel exact solution in PTIME for specific cases.

The *interpolation* problem can be shown to admit a fully polynomial-time randomized approximation scheme (FPRAS). This result follows from existing work [31, 7], using a tractable almost uniform sampling scheme for convex bodies.

▶ **Proposition 10** ([31], Algorithm 5.8)**.** *Let $\mathcal{C}$ be a set of constraints with variable set $\mathcal{X}$ and $x \in \mathcal{X}$. There is an FPRAS that determines an estimate $\hat{\mathbb{E}}_{\mathcal{C}}[x]$ of the expected value $\mathbb{E}_{\mathcal{C}}[x]$ of x in $\mathrm{pw}(\mathcal{C})$ under the uniform distribution.*

This result is mostly of theoretical interest, as the polynomial is in $|\mathcal{X}|^7$ (see [7], Table 1), but recent improved sampling algorithms [37] may ultimately yield a practical approximate interpolation technique for general constraint sets (see [36, 21]).

For completeness, we mention two natural ways to define randomized approximations for *top-k computation*:

- We can define the *approximate top-k* as an ordered list of k items whose expected value does not differ by more than some $\epsilon > 0$ from that of the item in the actual top-k at the same rank. An FPRAS for this definition of approximate top-k can be obtained from that of Proposition 10.
- It is highly unlikely that there exists a PTIME algorithm to return the *actual top-k* with high probability, even without requiring it to return the expected values. Indeed, such an algorithm would be in the BPP (bounded-error probabilistic time) complexity class; yet it follows from Theorem 9 above that deciding whether a set of variables is the top-k is NP-hard, so the existence of the algorithm would entail that NP $\subseteq$ BPP.

5 Tractable Cases

Given the hardness results in the previous section and the impracticality of approximation, we now study whether *exact* interpolation and top-k computation can be tractable on restricted classes of constraint sets. We consider tree-shaped constraints (defined formally below) and generalizations thereof: they are relevant for practical applications (e.g., classifying items into tree- or forest-shaped taxonomies), and we will show that our problems are tractable on them. We start by a splitting lemma to decompose constraint sets into "independent" subsets of variables, and then define and study our tractable class.

5.1 Splitting Lemma

We will formalize the cases in which the valuations of two variables in $\mathcal{X}$ are probabilistically dependent (the variables *influence* each other), according to $\mathcal{C}$. This, in turn, will enable us to define *independent subsets of the variables* and thus *independent subsets of the constraints* over these variables. This abstract result will generalize the notion of fragments from total orders (see Section 3.1) to general constraint sets. In what follows, we use $x_i \prec x_j$ to denote the *covering relation* of the partial order $\leqslant$, i.e., $x_i \leqslant x_j$ is in $\mathcal{C}$ but there exists no $x_k \notin \{x_i, x_j\}$ such that $x_i \leqslant x_k$ and $x_k \leqslant x_j$ are in $\mathcal{C}$.

▶ **Definition 11.** We define the *influence relation* $x \leftrightarrow y$ between variables of $\mathcal{X}\backslash\mathcal{X}_{\text{exact}}$ as the equivalence relation obtained by the symmetric, reflexive, and transitive closure of the $\prec$ relation on $\mathcal{X}\backslash\mathcal{X}_{\text{exact}}$.

The *uninfluenced classes* of $\mathcal{X}$ under $\mathcal{C}$ is the partition of $\mathcal{X}\backslash\mathcal{X}_{\text{exact}}$ as the subsets $\mathsf{X}_1, \ldots, \mathsf{X}_m$ given by the equivalence classes of the influence relation.

The *uninfluence decomposition of* $\mathcal{C}$ is the collection of constraint sets $\mathcal{C}_1, \ldots, \mathcal{C}_m$ of $\mathcal{C}$ where each $\mathcal{C}_i$ has as variables $\mathsf{X}_i \sqcup \mathcal{X}_{\text{exact}}$ and contains all exact-value constraints of $\mathcal{C}$ and all order constraints between variables of $\mathsf{X}_i \sqcup \mathcal{X}_{\text{exact}}$.

We assume w.l.o.g. that $m > 0$, i.e., there are unknown variables in $\mathcal{X}\backslash\mathcal{X}_{\text{exact}}$; otherwise the uninfluence decomposition is meaningless but any analysis is trivial. Intuitively, two unknown variables x, x' are in different uninfluenced classes if in *every* linear extension there is *some* variable from $\mathcal{X}_{\text{exact}}$ between them, or if they belong to disconnected (and thus incomparable) parts of the partial order. In particular, uninfluenced classes correspond to the fragments of a total order: this is used in Section 3.1. The uninfluence decomposition captures only constraints between *variables that influence each other*, and constraints that can *bound the range of a variable* by making it comparable to variables from $\mathcal{X}_{\text{exact}}$. We formally prove the independence of $\mathcal{C}_1, \ldots, \mathcal{C}_m$ via possible-world semantics: every possible world of $\mathcal{C}$ can be decomposed to possible worlds of $\mathcal{C}_1, \ldots, \mathcal{C}_m$, and vice versa.

▶ **Lemma 12.** *Let $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be the uninfluence decomposition of $\mathcal{C}$. There exists a bijective correspondence between* $\text{pw}(\mathcal{C})$ *and* $\text{pw}(\mathcal{C}_1) \times \cdots \times \text{pw}(\mathcal{C}_m)$.

▶ **Example 13.** Let $\mathcal{X}$ be $\{x, y, y', z, w\}$, and let $\mathcal{C}$ be defined by $y' = 0.5$ and $x \leqslant y \leqslant y' \leqslant z$. The uninfluence classes are $\mathsf{X}_1 = \{x, y\}$, $\mathsf{X}_2 = \{z\}$, and $\mathsf{X}_3 = \{w\}$. The uninfluence decomposition thus consists of $\mathcal{C}_1$, with variables $\mathsf{X}_1 \sqcup \{y'\}$, and constraints $x \leqslant y \leqslant y'$ and $y' = 0.5$; $\mathcal{C}_2$, with variables $\mathsf{X}_2 \sqcup \{y'\}$, and constraints $y' \leqslant z$ and $y' = 0.5$; and $\mathcal{C}_3$, with variables $\mathsf{X}_3 \sqcup \{y'\}$, and constraint $y' = 0.5$.

We next use this independence property to analyse restricted classes of constraint sets.

5.2 Tree-Shaped Constraints

We define the first restricted class of constraints that we consider: *tree-shaped* constraints. Recall that a *Hasse diagram* is a representation of a partial order as a directed acyclic graph, whose nodes correspond to $\mathcal{X}$ and where there is an edge (x, y) if $x \prec y$. An example of such a diagram is the one used in Section 3.2.

▶ **Definition 14.** A constraint set $\mathcal{C}$ over $\mathcal{X}$ is *tree-shaped* if the probability of ties is zero, the Hasse diagram of the partial order induced on $\mathcal{X}$ by $\mathcal{C}$ is a directed tree, the root has exactly one child, and exactly the root and leaves are in $\mathcal{X}_{\text{exact}}$. Thus, $\mathcal{C}$ imposes a global minimal value, and maximal values at each leaf, and no other exact-value constraint.

We call $\mathcal{C}$ *reverse-tree-shaped* if the reverse of the Hasse diagram (obtained by reversing the direction of the edges) is tree-shaped.

Tree-shaped constraints are often encountered in practice, in particular in the context of product taxonomies. Indeed, while our example from Figure 1 is a DAG, many real-life taxonomies are trees: in particular, the Google Product Taxonomy [23] and ACM CCS [2].

We now show that for a tree-shaped constraint set $\mathcal{C}$, unlike the general case, we can tractably compute exact expressions of the expected values of variables. In the next two results, we assume arithmetic operations on rationals to have unit cost, e.g., they are performed up to a fixed numerical precision. Otherwise, the complexities remain polynomial but the degrees may be larger. We first show:

▶ **Theorem 15.** *For any tree-shaped constraint set $\mathcal{C}$ over $\mathcal{X}$, we can compute its volume $V(\mathcal{C})$ in time $O(|\mathcal{X}|^2)$.*

Proof sketch. We process the tree bottom-up, propagating a piecewise-polynomial function expressing the volume of the subpolytope on the subtree rooted at each node as a function of the value of the parent node: we compute it using Lemma 12 from the child nodes. ◀

This result can be applied to prove the tractability of computing the marginal distribution of any variable $x \in \mathcal{X} \backslash \mathcal{X}_{\text{exact}}$ in a tree-shaped constraint set, which is defined as the pdf $p_x(v) := V_{d-1}(\mathcal{C} \cup \{x = v\})/V_d(\mathcal{C})$, where d is the dimension of $\text{pw}(\mathcal{C})$:

▶ **Theorem 16.** *For any tree-shaped constraint set $\mathcal{C}$ on variable set $\mathcal{X}$, for any variable $x \in \mathcal{X} \backslash \mathcal{X}_{\text{exact}}$, the marginal distribution for x is piecewise polynomial and can be computed in time $O(|\mathcal{X}_{\text{exact}}| \times |\mathcal{X}|^2)$.*

Proof sketch. We proceed similarly to the proof of Theorem 15 but with two functions: one for x and its descendants, and one for all other nodes. The additional $|\mathcal{X}_{\text{exact}}|$ factor is because the second function depends on how the value given to x compares to the leaves. ◀

We last deduce that our results for tree-shaped constraints extend to a more general tractable case: constraint sets $\mathcal{C}$ whose uninfluence decomposition $\mathcal{C}_1, \ldots, \mathcal{C}_m$ is such that every $\mathcal{C}_i$ is (reverse-)tree-shaped. By Lemma 12, each $\mathcal{C}_i$ (and its variables) can be considered independently, and reverse-tree-shaped trees can be easily transformed into tree-shaped ones. Our previous algorithms thus apply to this general case, by executing them on each constraint set of the uninfluence decomposition that is relevant to the task (namely, containing the variable x to interpolate, or top-k candidates from the selected variables $\mathcal{X}_\sigma$):

▶ **Corollary 17.** *Given any constraint set $\mathcal{C}$ and its uninfluence decomposition $\mathcal{C}_1, \ldots, \mathcal{C}_m$, assuming that each $\mathcal{C}_i$ is a (reverse-)tree-shaped constraint set, we can solve the interpolation problem in time $O(\max_i |\mathsf{X}_i|^3)$ and the top-k problem in PTIME.*

On large tree-shaped taxonomies (e.g., the Google Product Taxonomy [23]), in an interactive setting where we may ask user queries (e.g., the one in the Introduction), we can improve running times by asking more queries. Indeed, each answer about a category adds an exact-value constraint, and reduces the size of the constraint sets of the uninfluence decomposition, which decreases the overall running time, thanks to the superadditivity of $x \mapsto x^3$. We do not study which variables should be queried in order to reduce the running time of the algorithm; see, e.g., [40] for tree-partitioning algorithms.

6 Other Variants

We have defined top-k computation on constraint sets by considering the *expected value* of each variable under the uniform distribution. Comparing to different definitions of top-k on unknown values that have been studied in previous work, our definition has some important properties [15]: it provides a ready estimation for unknown values (namely, their expected value) and guarantees an output of size k. Moreover, it satisfies the *containment property* of [15], defined in our setting as follows:

▶ **Definition 18.** A top-k definition satisfies the *containment property* if for any constraint set $\mathcal{C}$ on variables $\mathcal{X}$, for any predicate σ (where we write $\mathcal{X}_\sigma$ the selected variables), and for any $k < |\mathcal{X}_\sigma|$, letting S_k and S_{k+1} be the ordered lists of top-k and top-$(k+1)$ variables, S_k is a strict prefix of S_{k+1}.

The containment property is a natural desideratum: computing the top-k for some $k \in \mathbb{N}$ should not give different variables or order for the top-k' with $k' < k$. Our definition clearly satisfies the containment property (except in the case of ties). By contrast, we will now review prominent definitions of top-k on uncertain data from related work [47, 15, 52], and show that they do not satisfy the containment property when we apply them to the possible world distributions studied in our setting. We focus on two prominent definitions, *U-top-k* and *global-top-k* and call our own definition *local-top-k* when comparing to them; we also discuss other variants in [5].

U-top-k. The *U-top-k* variant does not study *individual* variables but defines the output as the *sequence* of k variables most likely to be the top-k (in that order), for the uniform distribution on $\mathrm{pw}(\mathcal{C})$. We call this alternative definition *U-top-k* by analogy with [47, 15]. Interestingly, the U-top-k and local-top-k definitions sometimes disagree in our setting:

▶ **Lemma 19.** *There is a constraint set $\mathcal{C}$ and selection predicate σ such that local-top-k and U-top-k do not match, even for $k = 1$ and without returning expected values or probabilities.*

We can easily design an algorithm to compute U-top-k in PSPACE and in polynomial time in the number of linear extensions of $\mathcal{C}$: compute the probability of each linear extension as in Algorithm 1, and then sum on linear extensions depending on which top-k sequence they realize (on the variables selected by σ), to obtain the probability of each answer. Hence:

▶ **Proposition 20.** *For any constraint set $\mathcal{C}$ over $\mathcal{X}$, integer k and selection predicate σ, the U-top-k query for $\mathcal{C}$ and σ can be computed in PSPACE and in time $\mathrm{O}(\mathrm{poly}(N))$, where N is the number of linear extensions of $\mathcal{C}$.*

Unlike Theorem 7, however, this does not imply $\mathrm{FP}^{\#\mathrm{P}}$-membership: when selecting the most probable sequence, the number of candidate sequences may not be polynomial (as k is not fixed). We leave to future work an investigation of the precise complexity of U-top-k.

We show that in our setting U-top-k does not satisfy the *containment property* of [15].

▶ **Lemma 21.** *There is a constraint set $\mathcal{C}$ without ties such that U-top-k does not satisfy the containment property for the uniform distribution on* $\mathrm{pw}(\mathcal{C})$.

Global-top-k. We now study the *global-top-k* definition [52], and show that it does not respect the containment property either, even though it is defined on individual variables:

▶ **Definition 22.** The *global-top-k query*, for a constraint set $\mathcal{C}$, selection predicate σ, and integer k, returns the k variables that have the highest probability in the uniform distribution on $\mathrm{pw}(\mathcal{C})$ to be among the variables with the k highest values, sorted by decreasing probability.

▶ **Lemma 23.** *There is a constraint set $\mathcal{C}$ without ties such that global-top-k does not satisfy the containment property for the uniform distribution on* $\mathrm{pw}(\mathcal{C})$.

7 Related Work

We extend the discussion about related work from the Introduction.

Ranking queries over uncertain databases. A vast body of work has focused on providing semantics and evaluation methods for order queries over uncertain databases, including top-k and ranking queries (e.g., [15, 19, 25, 26, 28, 33, 43, 46, 50, 51]). Such works consider two main uncertainty types: *tuple-level uncertainty*, where the existence of tuples (i.e., variables) is uncertain, and hence affects the query results [15, 19, 26, 28, 33, 43, 50, 51]; and *attribute-level uncertainty*, more relevant to our problem, where the data tuples are known but some of their values are unknown or uncertain [15, 25, 28, 46]. Top-k queries over uncertain data following [46] was recently applied to crowdsourcing applications in [13]. These studies are relevant to our work as they identify multiple possible semantics for order queries in presence of uncertainty, and specify desired properties for such semantics [15, 28]; our definition of top-k satisfies the desiderata that are relevant to attribute-level uncertainty [28].

We depart from this existing work in two main respects. First, existing work assumes that each variable is given with an *independent function* that describes its probability distribution. We do not assume this, and instead *derive* expressions for the expected values of variables in a principled way from a uniform prior on the possible worlds. Our work is thus well-suited to the many situations where probability distributions on variables are not known, or where they are not independent (e.g., when order constraints are imposed on them). For this reason, the problems that we consider are generally computationally harder. For instance, [46] is perhaps the closest to our work, since they consider the total orders compatible with given partial order constraints. However, they assume independent marginal distributions, so they can evaluate top-k queries by only considering k-sized prefixes of the linear extensions; in our setting even computing the top-1 element is hard (Theorem 9).

The second key difference is that other works *do not try to estimate the top-k values*, because they assume that the marginal distribution is given: they only focus on ranks. In our context, we need to compute missing values, and need to account, e.g., for exact-value constraints and their effect on the probability of possible worlds and on expected values (Section 3).

We also mention our previous work [4] which considers the estimation of uncertain values (expectation and variance), but only in a *total order*, and did not consider complexity issues.

Partial order search. Another relevant research topic, *partial order search*, considers queries over elements in a partially ordered set to find a subset of elements with a certain property [3,

17, 20, 24, 40]. This relates to many applications, e.g., crowd-assisted graph search [40], frequent itemset mining with the crowd [3], and knowledge discovery, where the unknown data is queried via oracle calls [24]. These studies are *complementary to ours*: when the target function can be phrased as a top-k or interpolation problem, if the search is stopped before all values are known, we can use our method to estimate the complete output.

Computational geometry. Our work reformulates the interpolation problem as a centroid computation problem in the polytope of possible worlds defined by the constraint set. This problem has been studied independently by computational geometry work [42, 31, 38].

Computational geometry mostly studies *arbitrary* convex polytopes (corresponding to polytopes defined by arbitrary linear constraint sets), and often considers the task of *volume computation*, which is related to the problem of computing the centroid [42]. In this context, it is known that computing the exact volume of a polytope is not in $\mathrm{FP}^{\#\mathrm{P}}$ because the output is generally not of polynomial size [32]. Nevertheless, several (generally exponential) methods for exact volume computation [11] have been developed. The problem of approximation has also been studied, both theoretically and practically [31, 45, 18, 16, 36, 21]. Our problem of *centroid* computation is studied in [38], whose algorithm is based on the idea of computing the volume of a polytope by computing the lower-dimensional volume of its facets. This is different from our algorithm, which divides the polytope along linear extensions into subpolytopes, for which we apply a specific volume and centroid computation method.

Some works in computational geometry specifically study *order polytopes*, i.e., the polytopes defined by constraint sets with only order constraints and no exact-value constraints. For such polytopes, volume computation is known to be $\mathrm{FP}^{\#\mathrm{P}}$-complete [10], leading to a $\mathrm{FP}^{\#\mathrm{P}}$-hardness result for centroid computation [42]. However, these results do not apply to *exact-value constraints*, i.e., when order polytopes can only express order relations, between variables which are in $[0, 1]$. Exact-value constraints are both highly relevant in practice (to represent numerical bounds, or known information, e.g., for crowdsourcing), allow for more general polytopes, and complicate the design of Algorithm 1, which must perform volume computation and interpolation in each *fragmented* linear order.

Furthermore, to our knowledge, computational geometry works do not study the top-k problem, or polytopes that correspond to tree-shaped constraint sets, since these have no clear geometric interpretation.

Tree-shaped partial orders. Our analysis of tractable schemes for tree-shaped partial orders is reminiscent of the well-known tractability of probabilistic inference in tree-shaped graphical models [8], and of the tractability of probabilistic query evaluation on trees [14] and treelike instances [6]. However, we study continuous distributions on numerical values, and the influence between variables when we interpolate does not simply follow the tree structure; so our results do not seem to follow from these settings.

8 Conclusion

In this paper, we have studied the problems of top-k computation and interpolation for data with unknown values and order constraints. We have provided foundational solutions, including a general computation scheme, complexity bounds, and analysis of tractable cases.

One natural direction for future work is to study whether our tractable cases (tree-shaped orders, sampling) can be covered by more efficient PTIME algorithms, or whether more general tractable cases can be identified: for instance, a natural direction to study would be

partial orders with a *bounded-treewidth* Hasse diagram, following recent tractability results for the related problem of linear extension counting [30]. Another question is to extend our scheme to request additional values from the crowd, as in [3, 13], and reduce the expected error on the interpolated values or top-k query, relative to a user goal. In such a setting, how should we choose which values to retrieve, and could we update incrementally the results of interpolation when we receive new exact-value constraints? Finally, it would be interesting to study whether our results generalize to different prior distributions on the polytope.

Acknowledgements This work is partially supported by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, by a grant from the Blavatnik Interdisciplinary Cyber Research Center, by the Israel Science Foundation (grant No. 1157/16), and by the Télécom ParisTech Research Chair on Big Data and Market Insights.

References

1 Serge Abiteboul, T-H. Hubert Chan, Evgeny Kharlamov, Werner Nutt, and Pierre Senellart. Capturing continuous data and answering aggregate queries in probabilistic XML. *TODS*, 36(4), 2011.

2 ACM Computing Classification System, 2012. `https://www.acm.org/about/class/class/2012`.

3 Antoine Amarilli, Yael Amsterdamer, and Tova Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, 2014. `doi:10.5441/002/icdt.2014.06`.

4 Antoine Amarilli, Yael Amsterdamer, and Tova Milo. Uncertainty in crowd data sourcing under structural constraints. In *UnCrowd*, 2014.

5 Antoine Amarilli, Yael Amsterdamer, Tova Milo, and Pierre Senellart. Top-k querying of unknown values under order constraints (extended version). *CoRR*, abs/1701.02634, 2017.

6 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, 2015. `doi:10.1007/978-3-662-47666-6_5`.

7 Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *JACM*, 51(4), 2004.

8 Christopher M. Bishop. Graphical models. In *Pattern Recognition and Machine Learning*, chapter 8. Springer, 2006.

9 Jonathan Bragg, Mausam, and Daniel S. Weld. Crowdsourcing multi-label classification for taxonomy creation. In *HCOMP*, 2013.

10 Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3), 1991.

11 Benno Büeler, Andreas Enge, and Komei Fukuda. Exact volume computation for polytopes: a practical study. In *Polytopes – combinatorics and Computation*, 2000.

12 Reynold Cheng, Dmitri V Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

13 Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Crowdsourcing for top-k query processing over uncertain data. *IEEE TKDE*, 28(1), 2016.

14 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

15 Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.

16 Ben Cousins and Santosh Vempala. A practical volume algorithm. *Mathematical Programming Computation*, 8(2), 2016.

17 Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.

18 Jesús A De Loera, B Dutra, Matthias Köppe, S Moreinis, G Pinto, and J Wu. Software for exact integration of polynomials over polyhedra. *Computational Geometry*, 46(3), 2013.

19 Landon Detwiler, Wolfgang Gatterbauer, Brenton Louie, Dan Suciu, and Peter Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In *ICDE*, 2009.

20 Ulrich Faigle, Laszlo Lovasz, Rainer Schrader, and Gy Turán. Searching in trees, series-parallel and interval orders. *SIAM J. Comput.*, 15(4), 1986.

21 Cunjing Ge and Feifei Ma. A fast and practical method to estimate volumes of convex polytopes. In *FAW*, 2015.

22 James E. Gentle. *Computational Statistics*. Springer, 2009.

23 Google Product Taxonomy, 2016. `https://support.google.com/merchants/answer/1705911?hl=en`.

24 Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *TODS*, 28(2), 2003.

25 Parisa Haghani, Sebastian Michel, and Karl Aberer. Evaluating top-k queries over incomplete data streams. In *CIKM*, 2009.

26 Ming Hua, Jian Pei, and Xuemin Lin. Ranking queries on uncertain data. *VLDB J.*, 20(1), 2011.

27 Yannis E. Ioannidis and Raghu Ramakrishnan. Efficient transitive closure algorithms. In *VLDB*, 1988.

28 Jeffrey Jestes, Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data. *IEEE TKDE*, 23(12), 2011.

29 Frank Jones. *Lebesgue Integration on Euclidean Space*. Jones & Bartlett Learning, 2001.

30 Kustaa Kangas, Teemu Hankala, Teppo Niinimäki, and Mikko Koivisto. Counting linear extensions of sparse posets. In *IJCAI*, 2016.

31 Ravi Kannan, László Lovász, and Miklós Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Struct. Algorithms*, 11(1), 1997.

32 Jim Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195), 1991.

33 Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1), 2009.

34 Xiang Lian and Lei Chen. Probabilistic ranked queries in uncertain databases. In *EDBT*, 2008.

35 Xiang Lian and Lei Chen. A generic framework for handling uncertain data with local correlations. *VLDB*, 4(1), 2010.

36 László Lovász and István Deák. Computational results of an $\mathrm{O}^*(n^4)$ volume algorithm. *European J. Operational Research*, 216(1), 2012.

37 László Lovász and Santosh Vempala. Hit-and-run from a corner. *SIAM J. Comput.*, 35(4), 2006.

38 Frederic Maire. An algorithm for the exact computation of the centroid of higher dimensional polyhedra and its application to kernel machines. In *ICDM*, 2003.

39 Christos H Papadimitriou. Efficient search for rationals. *Information Processing Letters*, 8(1), 1979.

40 A. Parameswaran, A.D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5), 2011.

41 Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM J. Comput.*, 23(2), 1994.

42 Luis A Rademacher. Approximating the centroid is hard. In *SCG*, 2007.

43 Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

44 Alexander Schrijver. The structure of polyhedra. In *Theory of Linear and Integer Programming*, chapter 8. Wiley-Interscience, 1986.

45 Miklós Simonovits. How to compute the volume in high dimension? *Mathematical programming*, 97(1-2), 2003.

46 Mohamed A. Soliman, Ihab F. Ilyas, and Shalev Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDB J.*, 19(4), 2010.

47 Mohamed A. Soliman, Ihab F. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

48 Julia Stoyanovich, Sihem Amer-Yahia, Susan B Davidson, Marie Jacob, Tova Milo, et al. Understanding local structure in ranked datasets. In *CIDR*, 2013.

49 Chong Sun, Narasimhan Rampalli, Frank Yang, and AnHai Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 7(13), 2014.

50 Chonghai Wang, Li-Yan Yuan, Jia-Huai You, Osmar R. Zaïane, and Jian Pei. On pruning for top-k ranking in uncertain databases. *PVLDB*, 4(10), 2011.

51 Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases. In *ICDE*, 2008.

52 Xi Zhang and Jan Chomicki. Semantics and evaluation of top-k queries in probabilistic databases. *DAPD*, 26(1), 2009.

Combined Tractability of Query Evaluation via Tree Automata and Cycluits

Antoine Amarilli[1], Pierre Bourhis[2], Mikaël Monet[3], and Pierre Senellart[4]

1 **LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France**
2 **CRIStAL, CNRS & Université Lille 1, Lille, France**
3 **LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France; and Inria Paris, Paris, France**
4 **DI, École normale supérieure, PSL Research University, Paris, France; and Inria Paris, Paris, France**

Abstract

We investigate parameterizations of both database instances and queries that make query evaluation fixed-parameter tractable in combined complexity. We introduce a new Datalog fragment with stratified negation, intensional-clique-guarded Datalog (ICG-Datalog), with linear-time evaluation on structures of bounded treewidth for programs of bounded rule size. Such programs capture in particular conjunctive queries with simplicial decompositions of bounded width, guarded negation fragment queries of bounded CQ-rank, or two-way regular path queries. Our result is shown by compiling to alternating two-way automata, whose semantics is defined via cyclic provenance circuits (cycluits) that can be tractably evaluated. Last, we prove that probabilistic query evaluation remains intractable in combined complexity under this parameterization.

1998 ACM Subject Classification H.2 Database Management

Keywords and phrases query evaluation, tree automata, provenance, treewidth, circuits

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.6

1 Introduction

Arguably the most fundamental task performed by database systems is *query evaluation*, namely, computing the results of a query over a database instance. Unfortunately, this task is well-known to be intractable in *combined complexity* [55] even for simple query languages.

To address this issue, two main directions have been investigated. The first is to restrict the class of *queries* to ensure tractability, for instance, to α-acyclic conjunctive queries [57], this being motivated by the idea that many real-world queries are simple and usually small. The second approach restricts the structure of database instances, e.g., requiring them to have bounded *treewidth* [52] (we call them *treelike*). This has been notably studied by Courcelle [24], to show the tractability of monadic-second order logic on treelike instances, but in *data complexity* (i.e., for fixed queries); the combined complexity is generally nonelementary [49].

This leaves open the main question studied in this paper: *Which queries can be efficiently evaluated, in combined complexity, on treelike databases?* This question has been addressed by Gottlob, Pichler, and Fei [36] by introducing *quasi-guarded Datalog*; however, an unusual feature of this language is that programs must explicitly refer to the tree decomposition of the instance. Instead, we try to follow Courcelle's approach and investigate which queries can be efficiently *compiled to automata*. Specifically, rather than restricting to a fixed class of "efficient" queries, we study *parameterized* query classes, i.e., we define an efficient class

of queries for each value of the parameter. We further make the standard assumption that the signature is fixed; in particular, its arity is constant. This allows us to aim for low combined complexity for query evaluation, namely, fixed-parameter tractability with linear time complexity in the product of the input query and instance, called *FPT-linear* complexity.

Surprisingly, we are not aware of further existing work on tractable combined query evaluation for parameterized instances and queries, except from an unexpected angle: the compilation of restricted query fragments to tree automata on treelike instances was used in the context of *guarded logics* and other fragments, to decide *satisfiability* [13] and *containment* [11]. To do this, one usually establishes a *treelike model property* to restrict the search to models of low treewidth (but dependent on the formula), and then compiles the formula to an automaton, so that the problems reduce to emptiness testing: expressive automata formalisms, such as *alternating two-way automata*, are typically used. One contribution of our work is to notice this connection, and show how query evaluation on treelike instances can benefit from these ideas: for instance, as we show, some queries can only be compiled efficiently to such concise automata, and not to the more common bottom-up tree automata.

From there, the first main contribution of this paper is to define the language of *intensional-clique-guarded Datalog* (ICG-Datalog), and show an efficient FPT-linear compilation procedure for this language, parameterized by the body size of rules: this implies FPT-linear combined complexity on treelike instances. While we present it as a Datalog fragment, our language shares some similarities with guarded logics; yet, its design incorporates several features (fixpoints, clique-guards, guarded negation, guarding positive subformulae) that are not usually found together in guarded fragments, but are important for query evaluation. We show how the tractability of this language captures the tractability of such query classes as two-way regular path queries [10] and α-acyclic conjunctive queries.

Already for conjunctive queries, we show that the treewidth of queries is not the right parameter to ensure efficient compilability. In fact, a second contribution of our work is a lower bound: we show that bounded treewidth queries cannot be efficiently compiled to automata at all, so we cannot hope to show combined tractability for them via automata methods. By contrast, ICG-Datalog implies the combined tractability of bounded-treewidth queries with an additional requirement (interfaces between bags must be clique-guarded), which is the notion of *simplicial decompositions* previously studied by Tarjan [53]. To our knowledge, our paper is the first to introduce this query class and to show its tractability on treelike instances. ICG-Datalog can be understood as an extension of this fragment to disjunction, clique-guarded negation, and inflationary fixpoints, that preserves tractability.

To derive our main FPT-linear combined complexity result, we define an operational semantics for our tree automata by introducing a notion of *cyclic provenance circuits*, that we call *cycluits*. These cycluits, the third contribution of our paper, are well-suited as a provenance representation for alternating two-way automata encoding ICG-Datalog programs, as they naturally deal with both recursion and two-way traversal of a treelike instance, which is less straightforward with provenance formulae [37] or circuits [26]. While we believe that this natural generalization of Boolean circuits may be of independent interest, it does not seem to have been studied in detail, except in the context of integrated circuit design [45, 51], where the semantics often features feedback loops that involve negation; we prohibit these by focusing on *stratified* circuits, which we show can be evaluated in linear time. We show that the provenance of alternating two-way automata can be represented as a stratified cycluit in FPT-linear time, generalizing results on bottom-up automata and circuits from [5].

Since cycluits directly give us a provenance representation of the query, we then investigate *probabilistic query evaluation*, which we showed in [5] to be linear-time in data complexity

through the use of provenance circuits. We show how to remove cycles, so as to apply message-passing methods [42], yielding a 2EXPTIME upper bound for the combined complexity of probabilistic query evaluation. While we do not obtain tractable probabilistic query evaluation in combined complexity, we give lower bounds showing that this is unlikely.

Outline. We give preliminaries in Section 2, and then position our approach relative to existing work in Section 3. We then present our tractable fragment, first for bounded-simplicial-width conjunctive queries in Section 4, then for our ICG-Datalog language in Section 5. We then define our automata and compile ICG-Datalog to them in Section 6, before introducing cycluits and showing our provenance computation result in Section 7. We last study the conversion of cycluits to circuits, and probability evaluation, in Section 8. Full proofs are provided in the extended version [4].

2 Preliminaries

A *relational signature* σ is a finite set of relation names written R, S, T, ..., each with its associated *arity* $\mathrm{arity}(R) \in \mathbb{N}$. Throughout this work, *we always assume the signature σ to be fixed*: hence, its *arity* $\mathrm{arity}(\sigma)$ (the maximal arity of relations in σ) is constant, and we further assume it is > 0. A *(σ-)instance* I is a finite set of *ground facts* on σ, i.e., $R(a_1, \ldots, a_{\mathrm{arity}(R)})$ with $R \in \sigma$. The *active domain* $\mathrm{dom}(I)$ consists of the elements occurring in I.

We study query evaluation for several *query languages* that are subsets of first-order (FO) logic (e.g., conjunctive queries) or of second-order (SO) logic (e.g., Datalog). Unless otherwise stated, we only consider queries that are *constant-free*, and *Boolean*, so that an instance I either *satisfies* a query q ($I \models q$), or *violates* it ($I \not\models q$), with the standard semantics [1].

We study the *query evaluation problem* (or *model checking*) for a query class $\mathcal{Q}$ and instance class $\mathcal{I}$: given an instance $I \in \mathcal{I}$ and query $Q \in \mathcal{Q}$, check if $I \models Q$. Its *combined complexity* for $\mathcal{I}$ and $\mathcal{Q}$ is a function of I and Q, whereas *data complexity* assumes Q to be fixed. We also study cases where $\mathcal{I}$ and $\mathcal{Q}$ are *parameterized*: given infinite sequences $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\mathcal{Q}_1, \mathcal{Q}_2, \ldots$, the *query evaluation problem parameterized by k_I, k_Q* applies to $\mathcal{I}_{k_\mathrm{I}}$ and $\mathcal{Q}_{k_\mathrm{Q}}$. The parameterized problem is *fixed-parameter tractable* (FPT), for $(\mathcal{I}_n)$ and $(\mathcal{Q}_n)$, if there is a constant $c \in \mathbb{N}$ and computable function f such that the problem can be solved with combined complexity $O\left(f(k_\mathrm{I}, k_\mathrm{Q}) \cdot (|I| \cdot |Q|)^c\right)$. For $c = 1$, we call it *FPT-linear* (in $|I| \cdot |Q|$). Observe that calling the problem FPT is more informative than saying that it is in PTIME for fixed k_I and k_Q, as we are further imposing that the polynomial degree c does not depend on k_I and k_Q: this follows the usual distinction in parameterized complexity between FPT and classes such as XP [29].

Query languages. We first study fragments of FO, in particular, *conjunctive queries* (CQ), i.e., existentially quantified conjunctions of atoms. The *canonical model* of a CQ Q is the instance built from Q by seeing variables as elements and atoms as facts. The *primal graph* of Q has its variables as vertices, and connects all variable pairs that co-occur in some atom.

Second, we study *Datalog with stratified negation*. We summarize the definitions here, see [1] for details. A *Datalog program* P (without negation) over σ (called the *extensional signature*) consists of an *intensional signature* σ_int disjoint from σ (with the arity of σ_int being possibly greater than that of σ), a 0-ary *goal predicate* Goal in σ_int, and a set of *rules*: those are of the form $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, where the *head* $R(\mathbf{x})$ is an atom with $R \in \sigma_\mathrm{int}$, and the *body* ψ is a CQ over $\sigma_\mathrm{int} \sqcup \sigma$ where each variable of $\mathbf{x}$ must occur. The *semantics* $P(I)$ of P over an input σ-instance I is defined by a least fixpoint of the interpretation of σ_int:

we start with $P(I) := I$, and for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$ and tuple $\mathbf{a}$ of $\mathrm{dom}(I)$, when $P(I) \models \exists \mathbf{y}\, \psi(\mathbf{a}, \mathbf{y})$, then we *derive* the fact $R(\mathbf{a})$ and add it to $P(I)$, where we can then use it to derive more facts. We have $I \models P$ iff we derive the fact Goal(). The *arity* of P is $\max(\mathrm{arity}(\sigma), \mathrm{arity}(\sigma_{\mathrm{int}}))$. P is *monadic* if σ_{int} has arity 1.

Datalog with stratified negation [1] allows negated *intensional* atoms in bodies, but requires P to have a *stratification*, i.e., an ordered partition $P_1 \sqcup \cdots \sqcup P_n$ of the rules where:

- **(i)** Each $R \in \sigma_{\mathrm{int}}$ has a *stratum* $\zeta(R) \in \{1, \ldots, n\}$ such that all rules with R in the head are in $P_{\zeta(R)}$;
- **(ii)** For any $1 \leqslant i \leqslant n$ and σ_{int}-atom $R(\mathbf{z})$ in a body of a rule of P_i, we have $\zeta(R) \leqslant i$;
- **(iii)** For any $1 \leqslant i \leqslant n$ and negated σ_{int}-atom $R(\mathbf{z})$ in a body of P_i, we have $\zeta(R) < i$.

The stratification ensures that we can define the semantics of a stratified Datalog program by computing its interpretation for strata $P_1, \ldots, P_n$ in order: atoms in bodies always depend on a lower stratum, and negated atoms depend on strictly lower strata, whose interpretation was already fixed. Hence, there is a unique least fixpoint and $I \models P$ is well-defined.

▶ **Example 1.** The following stratified Datalog program, with $\sigma = \{R\}$ and $\sigma_{\mathrm{int}} = \{T, \mathrm{Goal}\}$, and strata P_1, P_2, tests if there are two elements that are not connected by a directed R-path:

$$P_1 : T(x, y) \leftarrow R(x, y), \quad T(x, y) \leftarrow R(x, z) \wedge T(z, y) \qquad P_2 : \mathrm{Goal}() \leftarrow \neg T(x, y)$$

Treewidth. Treewidth is a measure quantifying how far a graph is to being a tree, which we use to restrict instances and conjunctive queries. The *treewidth* of a CQ is that of its canonical instance, and the *treewidth* of an instance I is the smallest k such that I has a *tree decomposition* of *width* k, i.e., a finite, rooted, unranked tree T, whose nodes b (called *bags*) are labeled by a subset $\mathrm{dom}(b)$ of $\mathrm{dom}(I)$ with $|\mathrm{dom}(b)| \leqslant k + 1$, and which satisfies:

- **(i)** for every fact $R(\mathbf{a}) \in I$, there is a bag $b \in T$ with $\mathbf{a} \subseteq \mathrm{dom}(b)$;
- **(ii)** for all $a \in \mathrm{dom}(I)$, the set of bags $\{b \in T \mid a \in \mathrm{dom}(b)\}$ is a connected subtree of T.

A family of instances is *treelike* if their treewidth is bounded by a constant.

3 Approaches for Tractability

We now review existing approaches to ensure the tractability of query evaluation, starting by query languages whose evaluation is tractable in combined complexity on *all* input instances. We then study more expressive query languages which are tractable on *treelike* instances, but where tractability only holds in data complexity. We then present the goals of our work.

3.1 Tractable Queries on All Instances

The best-known query language to ensure tractable query complexity is *α-acyclic queries* [28], i.e., those that have a tree decomposition where the domain of each bag corresponds exactly to an atom: this is called a *join tree* [34]. With Yannakakis's algorithm [57], we can evaluate an α-acyclic conjunctive query Q on an arbitrary instance I in time $O(|I| \cdot |Q|)$.

Yannakakis's result was generalized in two main directions. One direction [33], moving from linear time to PTIME, has investigated more general CQ classes, in particular CQs of bounded treewidth [30], *hypertreewidth* [34], and *fractional hypertreewidth* [38]. Bounding these query parameters to some fixed k makes query evaluation run in time $O((|I| \cdot |Q|)^{f(k)})$ for some function f, hence in PTIME; for treewidth, since the decomposition can be computed in FPT-linear time [19], this goes down to $O(|I|^k \cdot |Q|)$. However, query evaluation on arbitrary instances is unlikely to be FPT when parameterized by the query treewidth, since it would

imply that the exponential-time hypothesis fails (Theorem 5.1 of [47]). Further, even for treewidth 2 (e.g., triangles), it is not known if we can achieve linear data complexity [2].

In another direction, α-acyclicity has been generalized to queries with more expressive operators, e.g., disjunction or negation. The result on α-acyclic CQs thus extends to the *guarded fragment* (GF) of first-order logic, which can be evaluated on arbitrary instances in time $O(|I| \cdot |Q|)$ [44]. Tractability is independently known for FO^k, the fragment of FO where subformulae use at most k variables, with a simple evaluation algorithm in $O(|I|^k \cdot |Q|)$ [56].

Another important operator are *fixpoints*, which can be used to express, e.g., reachability queries. Though FO^k is no longer tractable when adding fixpoints [56], query evaluation is tractable [17, Theorem 3] for μGF, i.e., GF with some restricted least and greatest fixpoint operators, when *alternation depth* is bounded; without alternation, the combined complexity is in $O(|I| \cdot |Q|)$. We could alternatively express fixpoints in Datalog, but, sadly, most known tractable fragments are nonrecursive: nonrecursive stratified Datalog is tractable [30, Corollary 5.26] for rules with restricted bodies (i.e., strictly acyclic, or bounded strict treewidth). This result was generalized in [35] when bounding the number of guards: this nonrecursive fragment is shown to be equivalent to the k-guarded fragment of FO, with connections to the bounded-hypertreewidth approach. One recursive tractable fragment is Datalog LITE, which is equivalent to alternation-free μGF [32]. Fixpoints were independently studied for graph query languages such as reachability queries and *regular path queries* (RPQ), which enjoy linear combined complexity on arbitrary input instances: this extends to two-way RPQs (2RPQs) and even strongly acyclic conjunctions of 2RPQs (SAC2RPQs), which are expressible in alternation-free μGF. Tractability also extends to acyclic RPQs but with PTIME complexity [10].

3.2 Tractability on Treelike Instances

We now study another approach for tractable query evaluation: this time, we restrict the shape of the *instances*, using treewidth. This ensures that we can translate them to a tree for efficient query evaluation. Informally, having fixed the signature σ, for a fixed treewidth $k \in \mathbb{N}$, there is a finite tree alphabet Γ_σ^k such that σ-instances of treewidth $\leqslant k$ can be translated in FPT-linear time (parameterized by k), following the structure of a tree decomposition, to a Γ_σ^k*-tree*, i.e., a rooted full ordered binary tree with nodes labeled by Γ_σ^k, which we call a *tree encoding*. We omit the formal construction: see the extended version [4] for more details.

We can then evaluate queries on treelike instances by running *tree automata* on the tree encoding that represents them. Formally, given an alphabet Γ, a *bottom-up nondeterministic tree automaton* on Γ-trees (or Γ-bNTA) is a tuple $A = (Q, F, \iota, \Delta)$, where:

- **(i)** Q is a finite set of *states*;
- **(ii)** $F \in Q$ is a subset of *accepting states*;
- **(iii)** $\iota : \Gamma \to 2^Q$ is an *initialization function* determining the state of a leaf from its label;
- **(iv)** $\Delta : \Gamma \times Q^2 \to 2^Q$ is a *transition function* determining the possible states for an internal node from its label and the states of its two children.

Given a Γ-tree $\langle T, \lambda \rangle$ (where $\lambda : T \to \Gamma$ is the *labeling function*), we define a *run* of A on $\langle T, \lambda \rangle$ as a function $\varphi : T \to Q$ such that (1) $\varphi(l) \in \iota(\lambda(l))$ for every leaf l of T; and (2) $\varphi(n) \in \Delta(\lambda(n), \varphi(n_1), \varphi(n_2))$ for every internal node n of T with children n_1 and n_2. The bNTA A *accepts* $\langle T, \lambda \rangle$ if it has a run on T mapping the root of T to a state of F.

We say that a bNTA A *tests* a query Q for treewidth k if, for any Γ_σ^k-encoding $\langle E, \lambda \rangle$ coding an instance I (of treewidth $\leqslant k$), A accepts $\langle E, \lambda \rangle$ iff $I \models Q$. By a well-known result of Courcelle [24] on graphs (extended to higher-arity in [30]), we can use bNTAs to evaluate

all queries in *monadic second-order logic* (MSO), i.e., first-order logic with second-order variables of arity 1. MSO subsumes in particular CQs and monadic Datalog (but not general Datalog). Courcelle showed that MSO queries can be compiled to a bNTA that tests them:

▶ **Theorem 2** ([24, 30]). *For any MSO query Q and treewidth $k \in \mathbb{N}$, we can compute a bNTA that tests Q for treewidth k.*

This implies that evaluating any MSO query Q has FPT-linear *data complexity* when parameterized by Q and the instance treewidth [24, 30], i.e., it is in $O\left(f(|Q|, k) \cdot |I|\right)$ for some computable function f. However, this tells little about the combined complexity, as f is generally nonelementary in Q [49]. A better combined complexity bound is known for unions of conjunctions of two-way regular path queries (UC2RPQs) that are further required to be acyclic and to have a constant number of edges between pairs of variables: these can be compiled into polynomial-sized alternating two-way automata [11].

3.3 Restricted Queries on Treelike Instances

Our approach combines both ideas: we use instance treewidth as a parameter, but also restrict the queries to ensure tractable compilability. We are only aware of two approaches in this spirit. First, Gottlob, Pichler, and Wei [36] have proposed a *quasiguarded* Datalog fragment on *relational structures and their tree decompositions*, with query evaluation is in $O(|I| \cdot |Q|)$. However, this formalism requires queries to be expressed in terms of the tree decomposition, and not just in terms of the relational signature. Second, Berwanger and Grädel [17] remark (after Theorem 4) that, when alternation depth and *width* are bounded, μCGF (the *clique-guarded* fragment of FO with fixpoints) enjoys FPT-linear query evaluation when parameterized by instance treewidth. Their approach does not rely on automata methods, and subsumes the tractability of α-acyclic CQs and alternation-free μGF (and hence SAC2RPQs), on treelike instances. However, μCGF is a restricted query language (the only CQs that it can express are those with a chordal primal graph), whereas we want a richer language, with a parameterized definition.

Our goal is thus to develop an expressive parameterized query language, which can be compiled in *FPT-linear time* to an automaton that tests it (with the treewidth of instances also being a parameter). We can then evaluate the automaton, and obtain FPT-linear combined complexity for query evaluation. Further, as we will show, the use of tree automata will yield *provenance representations* for the query as in [5] (see Section 7).

4 Conjunctive Queries on Treelike Instances

To identify classes of queries that can be efficiently compiled to tree automata, we start by the simplest queries: *conjunctive queries*.

α-acyclic queries. A natural candidate for a tractable query class via automata methods would be *α-acyclic* CQs, which, as we explained in Section 3.1, can be evaluated in time $O(|I| \cdot |Q|)$ on all instances. Sadly, we show that such queries cannot be compiled efficiently to bNTAs, so our compilation result (Theorem 2) does not extend directly:

▶ **Proposition 3.** *There is an arity-two signature σ and an infinite family $Q_1, Q_2, \ldots$ of α-acyclic CQs such that, for any $i \in \mathbb{N}$, any bNTA that tests Q_i for treewidth 1 must have $\Omega(2^{|Q_i|^{1-\varepsilon}})$ states for any $\varepsilon > 0$.*

The intuition of the proof is that bNTAs can only make one traversal of the encoding of the input instance. Faced by this, we propose to use different tree automata formalisms, which are generally more concise than bNTAs. There are two classical generalizations of nondeterministic automata, on words [18] and on trees [23]: one goes from the inherent existential quantification of nondeterminism to *quantifier alternation*; the other allows *two-way* navigation instead of imposing a left-to-right (on words) or bottom-up (on trees) traversal. On words, both of these extensions independently allow for exponentially more compact automata [18]. In this work, we combine both extensions and use *alternating two-way tree automata* [23, 20], formally introduced in Section 6, which leads to tractable combined complexity for evaluation. Our general results in the next section will then imply:

▶ **Proposition 4.** *For any treewidth bound $k_{\mathrm{I}} \in \mathbb{N}$, given an α-acyclic CQ Q, we can compute in FPT-linear time in $O(|Q|)$ (parameterized by k_{I}) an alternating two-way tree automaton that tests it for treewidth k_{I}.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-linear time in $|I| \cdot |Q|$ (parameterized by k_{I}).

Bounded-treewidth queries. Having re-proven the combined tractability of α-acyclic queries (on bounded-treewidth instances), we naturally try to extend to *bounded-treewidth* CQs. Recall from Section 3.1 that these queries have PTIME combined complexity on all instances, but are unlikely to be FPT when parameterized by the query treewidth [47]. Can they be efficiently evaluated on treelike instances by compiling them to automata? We answer in the negative: that bounded-treewidth CQs *cannot* be efficiently compiled to automata to test them, even when using the expressive formalism of alternating two-way tree automata [23]:

▶ **Theorem 5.** *There is an arity-two signature σ for which there is no algorithm $\mathcal{A}$ with exponential running time and polynomial output size for the following task: given a conjunctive query Q of treewidth $\leqslant 2$, produce an alternating two-way tree automaton A_Q on Γ_σ^5-trees that tests Q on σ-instances of treewidth $\leqslant 5$.*

This result is obtained from a variant of the 2EXPTIME-hardness of monadic Datalog containment [12]. We show that efficient compilation of bounded-treewidth CQs to automata would yield an EXPTIME containment test, and conclude by the time hierarchy theorem.

Bounded simplicial width. We have shown that we cannot compile bounded-treewidth queries to automata efficiently. We now show that efficient compilation can be ensured with an additional requirement on tree decompositions. As it turns out, the resulting decomposition notion has been independently introduced for graphs:

▶ **Definition 6** ([27])**.** A *simplicial decomposition of a graph G* is a tree decomposition T of G such that, for any bag b of T and child bag b' of b, letting S be the intersection of the domains of b and b', then the subgraph of G induced by S is a complete subgraph of G.

We extend this notion to CQs, and introduce the *simplicial width* measure:

▶ **Definition 7.** A *simplicial decomposition of a CQ Q* is a simplicial decomposition of its primal graph. Note that any CQ has a simplicial decomposition (e.g., the trivial one that puts all variables in one bag). The *simplicial width* of Q is the minimum, over all simplicial tree decompositions, of the size of the largest bag minus 1.

Bounding the simplicial width of CQs is of course more restrictive than bounding their treewidth, and this containment relation is strict: cycles have treewidth $\leqslant 2$ but have

unbounded simplicial width. This being said, bounding the simplicial width is less restrictive than imposing α-acyclicity: the join tree of an α-acyclic CQ is in particular a simplicial decomposition, so α-acyclic CQs have simplicial width at most $\text{arity}(\sigma) - 1$, which is constant as σ is fixed. Again, the containment is strict: a triangle has simplicial width 2 but is not α-acyclic.

To our knowledge, simplicial width for CQs has not been studied before. Yet, we show that bounding the simplicial width ensures that CQs can be efficiently compiled to automata. This is unexpected, because the same is not true of treewidth, by Theorem 5. Hence:

▶ **Theorem 8.** *For any $k_{\mathrm{I}}, k_{\mathrm{Q}} \in \mathbb{N}$, given a CQ Q and a simplicial decomposition T of simplicial width k_{Q} of Q, we can compute in FPT-linear in $|Q|$ (parameterized by k_{I} and k_{Q}) an alternating two-way tree automaton that tests Q for treewidth k_{I}.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-linear time in $|I| \cdot (|Q| + |T|)$ (parameterized by k_{I} and k_{Q}).

Notice the technicality that the simplicial decomposition T must be provided as input to the procedure, because it is not known to be computable in FPT-linear time, unlike tree decompositions. While we are not aware of results on the complexity of this specific task, quadratic time algorithms are known for the related problem of computing the *clique-minimal separator decomposition* [43, 16].

The intuition for the efficient compilation of bounded-simplicial-width CQs is as follows. The *interface* variables shared between any bag and its parent must be "clique-guarded" (each pair is covered by an atom). Hence, consider any subquery rooted at a bag of the query decomposition, and see it as a non-Boolean CQ with the interface variables as free variables. Each result of this CQ must then be covered by a clique of facts of the instance, which ensures [31] that it occurs in some bag in the instance tree decomposition and can be "seen" by a tree automaton. This intuition can be generalized, beyond conjunctive queries, to design an expressive query language featuring disjunction, negation, and fixpoints, with the same properties of efficient compilation to automata and FPT-linear combined complexity of evaluation on treelike instances. We introduce such a Datalog variant in the next section.

5 ICG-Datalog on Treelike Instances

To design a Datalog fragment with efficient compilation to automata, we must of course impose some limitations, as we did for CQs. In fact, we can even show that the full Datalog language (even without negation) *cannot* be compiled to automata, no matter the complexity:

▶ **Proposition 9.** *There is a signature σ and Datalog program P such that the language of Γ^1_σ-trees that encode instances satisfying P is not a regular tree language.*

Hence, there is no bNTA or alternating two-way tree automaton that tests P for treewidth 1. To work around this problem and ensure that compilation is possible and efficient, the key condition that we impose on Datalog programs, pursuant to the intuition of simplicial decompositions, is that intensional predicates in rule bodies must be *clique-guarded*, i.e., their variables must co-occur in *extensional* predicates of the rule body. We can then use the *body size* of the program rules as a parameter, and will show that the fragment can then be compiled to automata in FPT-linear time.

▶ **Definition 10.** Let P be a stratified Datalog program. An intensional literal $A(\mathbf{x})$ or $\neg A(\mathbf{x})$ in a rule body ψ of P is *clique-guarded* if, for any two variables $x_i \neq x_j$ of $\mathbf{x}$, x_i and x_j co-occur in some extensional atom of ψ. P is *intensional-clique-guarded* (ICG) if, for any

rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, every *intensional* literal in ψ is clique-guarded in ψ. The *body size* of P is the maximal number of atoms in the body of its rules, multiplied by its arity.

The main result of this paper is that evaluation of ICG-Datalog is *FPT-linear* in *combined complexity*, when parameterized by the body size of the program and the instance treewidth.

▶ **Theorem 11.** *Given an ICG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, checking if $I \models P$ is FPT-linear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}).*

We will show this result in the next section by compiling ICG-Datalog programs in FPT-linear time to a special kind of tree automata (Theorem 22), and showing in Section 7 that we can efficiently evaluate such automata and even compute *provenance representations*. The rest of this section presents consequences of our main result for various languages.

Conjunctive queries. Our tractability result for bounded-simplicial-width CQs (Theorem 8), including α-acyclic CQs, is shown by rewriting to ICG-Datalog of bounded body size:

▶ **Proposition 12.** *There is a function f_σ (depending only on σ) such that for all $k \in \mathbb{N}$, for any conjunctive query Q and simplicial tree decomposition T of Q of width at most k, we can compute in $O(|Q| + |T|)$ an equivalent ICG-Datalog program with body size at most $f_\sigma(k)$.*

This implies that ICG-Datalog can express any CQ up to increasing the body size parameter, unlike, e.g., μCGF. Conversely, we can show that bounded-simplicial-width CQs *characterize* the queries expressible in ICG-Datalog when disallowing negation, recursion and disjunction. Specifically, a Datalog program is *positive* if it contains no negated atoms. It is *nonrecursive* if there is no cycle in the directed graph on σ_{int} having an edge from R to S whenever a rule contains R in its head and S in its body. It is *conjunctive* [14] if each intensional relation R occurs in the head of at most one rule. We can then show:

▶ **Proposition 13.** *For any positive, conjunctive, nonrecursive ICG-Datalog program P with body size k, there is a CQ Q of simplicial width $\leqslant k$ that is equivalent to P.*

However, our ICG-Datalog fragment is still exponentially more *concise* than such CQs:

▶ **Proposition 14.** *There is a signature σ and a family $(P_n)_{n\in\mathbb{N}}$ of ICG-Datalog programs with body size at most 9 which are positive, conjunctive, and nonrecursive, such that $|P_n| = O(n)$ and any CQ Q_n equivalent to P_n has size $\Omega(2^n)$.*

Guarded negation fragments. Having explained the connections between ICG-Datalog and CQs, we now study its connections to the more expressive languages of guarded logics, specifically, the *guarded negation fragment* (GNF), a fragment of first-order logic [9]. Indeed, when putting GNF formulae in *GN-normal form* [9] or even *weak GN-normal form* [15], we can translate them to ICG-Datalog, and we can use the *CQ-rank* parameter [15] (that measures the maximal number of atoms in conjunctions) to control the body size parameter.

▶ **Proposition 15.** *There is a function f_σ (depending only on σ) such that, for any weak GN-normal form GNF query Q of CQ-rank r, we can compute in time $O(|Q|)$ an equivalent nonrecursive ICG-Datalog program P of body size $f_\sigma(r)$.*

In fact, the efficient compilation of bounded-CQ-rank normal-form GNF programs (using the fact that subformulae are "answer-guarded", like our guardedness requirements) has been used recently (e.g., in [13]), to give efficient procedures for GNF *satisfiability*, compiling them to automata (to a treewidth which is not fixed, unlike in our context, but depends on

the formula). ICG-Datalog further allows clique guards (similar to CGNFO [9]), can reuse subformulae (similar to the idea of DAG-representations in [15]), and supports recursion (similar to GNFP [9], or GN-Datalog [8] but whose combined complexity is intractable — P^{NP}-complete). ICG-Datalog also resembles μCGF [17], but remember that it is not a guarded *negation* logic, so, e.g., μCGF cannot express all CQs.

Hence, the design of ICG-Datalog, and its compilation to automata, has similarities with guarded logics. However, to our knowledge, the idea of applying it to query evaluation is new, and ICG-Datalog is designed to support all relevant features to capture interesting query languages (e.g., clique guards are necessary to capture bounded-simplicial-width queries).

Recursive languages. The use of fixpoints in ICG-Datalog, in particular, allows us to capture the combined tractability of interesting recursive languages. First, observe that our guardedness requirement becomes trivial when all intensional predicates are monadic (arity-one), so our main result implies that *monadic Datalog* of bounded body size is tractable in combined complexity on treelike instances. This is reminiscent of the results of [36]:

▶ **Proposition 16.** *The combined complexity of monadic Datalog query evaluation on bounded-treewidth instances is FPT when parameterized by instance treewidth and body size (as in Definition 10) of the monadic Datalog program.*

Second, ICG-Datalog can capture *two-way regular path queries* (2RPQs) [21, 10], a well-known query language in the context of graph databases and knowledge bases:

▶ **Definition 17.** We assume that the signature σ contains only binary relations. A *regular path query* (RPQ) Q_L is defined by a regular language L on the alphabet Σ of the relation symbols of σ. Its semantics is that Q_L has two free variables x and y, and $Q_L(a, b)$ holds on an instance I for $a, b \in \mathrm{dom}(I)$ precisely when there is a directed path π of relations of σ from a to b such that the label of π is in L. A *two-way regular path query* (2RPQ) is an RPQ on the alphabet $\Sigma^{\pm} := \Sigma \sqcup \{R^- \mid R \in \Sigma\}$, which holds whenever there is a path from a to b with label in L, with R^- meaning that we traverse an R-fact in the reverse direction. A *Boolean 2RPQ* is a 2RPQ which is existentially quantified on its two free variables.

▶ **Proposition 18** ([48, 10])**.** *2RPQ query evaluation (on arbitrary instances) has linear time combined complexity.*

ICG-Datalog allows us to capture this result for Boolean 2RPQs on treelike instances. In fact, the above result extends to SAC2RPQs, which are trees of 2RPQs with no multi-edges or loops. We can prove the following result, for Boolean 2RPQs and SAC2RPQs, which further implies compilability to automata (and efficient compilation of provenance representations). We do not know whether this extends to the more general classes studied in [11].

▶ **Proposition 19.** *Given a Boolean SAC2RPQ Q, we can compute in time $O(|Q|)$ an equivalent ICG-Datalog program P of body size 4.*

6 Compilation to Automata

In this section, we study how we can compile ICG-Datalog queries on treelike instances to tree automata, to be able to evaluate them efficiently. As we showed with Proposition 3, we need more expressive automata than bNTAs. Hence, we use instead the formalism of *alternating two-way automata* [23], i.e., automata that can navigate in trees in any direction, and can express transitions using Boolean formulae on states. Specifically, we introduce for

our purposes a variant of these automata, which are *stratified* (i.e., allow a form of stratified negation), and *isotropic* (i.e., no direction is privileged, in particular order is ignored).

As in Section 3.2, we will define tree automata that run on Γ-*trees* for some alphabet Γ: a Γ-tree $\langle T, \lambda\rangle$ is a finite rooted ordered tree with a labeling function λ from the nodes of T to Γ. The *neighborhood* $\mathrm{Nbh}(n)$ of a node $n \in T$ is the set which contains n, all children of n, and the parent of n if it exists.

Stratified isotropic alternating two-way automata. To define the transitions of our alternating automata, we write $\mathcal{B}(X)$ the set of propositional formulae (not necessarily monotone) over a set X of variables: we will assume w.l.o.g. that *negations are only applied to variables*, which we can always enforce using de Morgan's laws. A *literal* is a propositional variable $x \in X$ (*positive* literal) or the negation of a propositional variable $\neg x$ (*negative* literal).

A *satisfying assignment* of $\varphi \in \mathcal{B}(X)$ consists of two *disjoint* sets $P, N \subseteq X$ (for "positive" and "negative") such that φ is a tautology when substituting the variables of P with 1 and those of N with 0, i.e., when we have $\nu(\varphi) = 1$ for every valuation ν of X such that $\nu(x) = 1$ for all $x \in P$ and $\nu(x) = 0$ for all $x \in N$. Note that we allow satisfying assignments with $P \sqcup N \subsetneq X$, which will be useful for our technical results. We now define our automata:

▶ **Definition 20.** A *stratified isotropic alternating two-way automata* on Γ-trees (Γ-SATWA) is a tuple $A = (\mathcal{Q}, q_{\mathrm{I}}, \Delta, \zeta)$ with $\mathcal{Q}$ a finite set of *states*, q_{I} the *initial state*, Δ the *transition function* from $\mathcal{Q} \times \Gamma$ to $\mathcal{B}(\mathcal{Q})$, and ζ a *stratification function*, i.e., a surjective function from $\mathcal{Q}$ to $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$, such that for any $q, q' \in \mathcal{Q}$ and $f \in \Gamma$, if $\Delta(q, f)$ contains q' as a positive literal (resp., negative literal), then $\zeta(q') \leqslant \zeta(q)$ (resp. $\zeta(q') < \zeta(q)$).

We define by induction on $0 \leqslant i \leqslant m$ an *i-run* of A on a Γ-tree $\langle T, \lambda\rangle$ as a finite tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}}\rangle$, with labels of the form (q, w) or $\neg(q, w)$ for $w \in T$ and $q \in \mathcal{Q}$ with $\zeta(q) \leqslant i$, by the following recursive definition for all $w \in T$:

- For $q \in \mathcal{Q}$ such that $\zeta(q) < i$, the singleton tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}}\rangle$ with one node labeled by (q, w) (resp., by $\neg(q, w)$) is an i-run if there is a $\zeta(q)$-run of A on $\langle T, \lambda\rangle$ whose root is labeled by (q, w) (resp., if there is no such run);
- For $q \in \mathcal{Q}$ such that $\zeta(q) = i$, if $\Delta(q, \lambda(w))$ has a satisfying assignment (P, N), if we have a $\zeta(q')$-run T_{q^-} for each $q^- \in N$ with root labeled by $\neg(q^-, w)$, and a $\zeta(q^+)$-run T_{q^+} for each $q^+ \in P$ with root labeled by (q^+, w_{q^+}) for some w_{q^+} in $\mathrm{Nbh}(w)$, then the tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}}\rangle$ whose root is labeled (q, w) and has as children all the T_{q^-} and T_{q^+} is an i-run.

A *run* of A starting in a state $q \in \mathcal{Q}$ at a node $w \in T$ is a m-run whose root is labeled (q, w). We say that A *accepts* $\langle T, \lambda\rangle$ (written $\langle T, \lambda\rangle \models A$) if there exists a run of A on $\langle T, \lambda\rangle$ starting in the initial state q_{I} at the root of T.

Observe that the internal nodes of a run starting in some state q are labeled by states q' in the same stratum as q. The leaves of the run may be labeled by states of a strictly lower stratum or negations thereof, or by states of the same stratum whose transition function is tautological, i.e., by some (q', w) such that $\Delta(q', \lambda(w))$ has $\emptyset, \emptyset$ as a satisfying assignment. Intuitively, if we disallow negation in transitions, our automata amount to the alternating two-way automata used by [20], with the simplification that they do not need parity acceptance conditions (because we only work with finite trees), and that they are *isotropic*: the run for each positive child state of an internal node may start indifferently on *any* neighbor of w in the tree (its parent, a child, or w itself), no matter the direction. (Note, however, that the run for negated child states must start on w itself.)

We will soon explain how the compilation of ICG-Datalog is performed, but we first note that evaluation of Γ-SATWAs is in linear time:

▶ **Proposition 21.** *For any alphabet Γ, given a Γ-tree T and a Γ-SATWA A, we can determine whether $T \models A$ in time $O(|T| \cdot |A|)$.*

In fact, this result follows from the definition of provenance cycluits for SATWAs in the next section, and the claim that these cycluits can be evaluated in linear time.

Compilation. We now give our main compilation result: we can efficiently compile any ICG-Datalog program of bounded body size into a SATWA that *tests* it (in the same sense as for bNTAs). This is our main technical claim, which is proven in the extended version [4].

▶ **Theorem 22.** *Given an ICG-Datalog program P of body size k_{P} and $k_{\mathrm{I}} \in \mathbb{N}$, we can build in FPT-linear time in $|P|$ (parameterized by $k_{\mathrm{P}}, k_{\mathrm{I}}$) a SATWA A_P testing P for treewidth k_{I}.*

Proof Sketch. The idea is to have, for every relational symbol R, states of the form $q^{\nu}_{R(\mathbf{x})}$, where ν is a partial valuation of $\mathbf{x}$. This will be the starting state of a run if it is possible to navigate the tree encoding from some starting node and build in this way a total valuation ν' that extends ν and such that $R(\nu'(\mathbf{x}))$ holds. When R is intensional, once ν' is total on $\mathbf{x}$, we go into a state of the form $q^{\nu',\mathcal{A}}_r$ where r is a rule with head relation R and $\mathcal{A}$ is the set of atoms in the body of r (whose size is bounded by the body size). This means that we choose a rule to prove $R(\nu'(\mathbf{x}))$. The automaton can then navigate the tree encoding, build ν' and coherently partition $\mathcal{A}$ so as to inductively prove the atoms of the body. The clique-guardedness condition ensures that, when there is a match of $R(\mathbf{x})$, the elements to which $\mathbf{x}$ is mapped can be found together in a bag. The fact that the automaton is isotropic relieves us from the syntactic burden of dealing with directions in the tree, as one usually has to do with alternating two-way automata. ◀

7 Provenance Cycluits

In the previous section, we have seen how ICG-Datalog programs could be compiled efficiently to tree automata (SATWAs) that test them on treelike instances. To show that SATWAs can be evaluated in linear time (stated earlier as Proposition 21), we will introduce an operational semantics for SATWAs based on the notion of *cyclic circuits*, or *cycluits* for short.

We will also use these cycluits as a new powerful tool to compute (Boolean) *provenance information*, i.e., a representation of how the query result depends on the input data:

▶ **Definition 23.** A (Boolean) *valuation* of a set S is a function $\nu : S \to \{0, 1\}$. A *Boolean function* φ on variables S is a mapping that associates to each valuation ν of S a Boolean value in $\{0, 1\}$ called the *evaluation* of φ according to ν; for consistency with further notation, we write it $\nu(\varphi)$. The *provenance* of a query Q on an instance I is the Boolean function φ, whose variables are the facts of I, which is defined as follows: for any valuation ν of the facts of I, we have $\nu(\varphi) = 1$ iff the subinstance $\{F \in I \mid \nu(F) = 1\}$ satisfies Q.

We can represent Boolean provenance as Boolean formulae [39, 37], or (more recently) as Boolean circuits [26, 5]. In this section, we first introduce *monotone cycluits* (monotone Boolean circuits with cycles), for which we define a semantics (in terms of the Boolean function that they express); we also show that cycluits can be evaluated in linear time, given a valuation. Second, we extend them to *stratified cycluits*, allowing a form of stratified negation. We conclude the section by showing how to construct the *provenance* of a SATWA as a cycluit, in FPT-linear time. Together with Theorem 22, this claim implies our main provenance result:

▶ **Theorem 24.** *Given an ICG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, we can construct in FPT-linear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}) a representation of the provenance of P on I as a stratified cycluit. Further, for fixed k_{I}, this cycluit has treewidth $O(|P|)$.*

Of course, this result implies the analogous claims for query languages that are captured by ICG-Datalog parameterized by the body size, as we studied in Section 5. When combined with the fact that cycluits can be tractably evaluated, it yields our main result, Theorem 11. The rest of this section formally introduces cycluits and proves Theorem 24.

Cycluits. We define *cycluits* as Boolean circuits without the acyclicity requirement, as in [51]. To avoid the problem of feedback loops, however, we first study *monotone cycluits*, and then cycluits with stratified negation.

▶ **Definition 25.** A *monotone Boolean cycluit* is a directed graph $C = (G, W, g_0, \mu)$ where G is the set of *gates*, $W \subseteq G^2$ is the set of directed edges called *wires* (and written $g \to g'$), $g_0 \in G$ is the *output gate*, and μ is the *type* function mapping each gate $g \in G$ to one of inp (input gate, with no incoming wire in W), $\wedge$ (AND gate) or $\vee$ (OR gate).

We now define the semantics of monotone cycluits. A (Boolean) *valuation* of C is a function $\nu : C_{\mathsf{inp}} \to \{0, 1\}$ indicating the value of the input gates. As for standard monotone circuits, a valuation yields an *evaluation* $\nu' : C \to \{0, 1\}$, that we will define shortly, indicating the value of each gate under the valuation ν: we abuse notation and write $\nu(C) \in \{0, 1\}$ for the *evaluation result*, i.e., $\nu'(g_0)$ where g_0 is the output gate of C. The Boolean function *captured* by a cycluit C is thus the Boolean function φ on C_{inp} defined by $\nu(\varphi) := \nu(C)$ for each valuation ν of C_{inp}. We define the evaluation ν' from ν by a least fixed-point computation (see the algorithm in the extended version [4]): we set all input gates to their value by ν, and other gates to 0. We then iterate until the evaluation no longer changes, by evaluating OR-gates to 1 whenever some input evaluates to 1, and AND-gates to 1 whenever all their inputs evaluate to 1. The Knaster–Tarski theorem [54] gives an equivalent characterization:

▶ **Proposition 26.** *For any monotone cycluit C and Boolean valuation ν of C, the set $S := \{g \in C \mid \nu'(g) = 1\}$ is the minimal set of gates (under inclusion) such that:*
(i) *S contains the true input gates, i.e., it contains $\{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\}$;*
(ii) *for any g such that $\mu(g) = \vee$, if some input gate of g is in S, then g is in S;*
(iii) *for any g such that $\mu(g) = \wedge$, if all input gates of g are in S, then g is in S.*

We show that this definition is computable in linear time (see the extended version [4]):

▶ **Proposition 27.** *Given any monotone cycluit C and Boolean valuation ν of C, we can compute the evaluation ν' of C in linear time.*

Stratified cycluits. We now move from monotone cycluits to general cycluits featuring negation. However, allowing arbitrary negation would make it difficult to define a proper semantics, because of possible cycles of negations. Hence, we focus on *stratified cycluits*:

▶ **Definition 28.** A *Boolean cycluit* C is defined like a *monotone cycluit*, but further allows NOT-gates ($\mu(g) = \neg$), which are required to have a single input. It is *stratified* if there exists a *stratification function* ζ mapping its gates surjectively to $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$ such that $\zeta(g) = 0$ iff $g \in C_{\mathsf{inp}}$, and $\zeta(g) \leqslant \zeta(g')$ for each wire $g \to g'$, the inequality being strict if $\mu(g') = \neg$.

Equivalently, C contains no cycle of gates involving a $\neg$-gate. If C is stratified, we can compute a stratification function in linear time by a topological sort, and use it to define the evaluation of C (which will clearly be independent of the choice of stratification function):

▶ **Definition 29.** Let C be a stratified cycluit with stratification function $\zeta : C \to \{0, \ldots, m\}$, and let ν be a Boolean valuation of C. We inductively define the *i-th stratum evaluation* ν_i, for i in the range of ζ, by setting $\nu_0 := \nu$, and letting ν_i extend the ν_j ($j < i$) as follows:

- For g such that $\zeta(g) = i$ with $\mu(g) = \neg$, set $\nu_i(g) := \neg\nu_{\zeta(g')}(g')$ for its one input g'.
- Evaluate all other g with $\zeta(g) = i$ as for monotone cycluits, considering the $\neg$-gates of point **1.** and all gates of lower strata as input gates fixed to their value in ν_{i-1}.

Letting g_0 be the output gate of C, the Boolean function φ *captured* by C is then defined as $\nu(\varphi) := \nu_m(g_0)$ for each valuation ν of C_{inp}.

▶ **Proposition 30.** *We can compute $\nu(C)$ in linear time in the stratified cycluit C and in ν.*

Building provenance cycluits. Having defined cycluits as our provenance representation, we compute the provenance of a query on an instance as the *provenance* of its SATWA on a tree encoding. To do so, we must give a general definition of the provenance of SATWAs. Consider a Γ-tree $\mathcal{T} := \langle T, \lambda\rangle$ for some alphabet Γ, as in Section 6. We define a (Boolean) *valuation* ν of $\mathcal{T}$ as a mapping from the nodes of T to $\{0, 1\}$. Writing $\overline{\Gamma} := \Gamma \times \{0, 1\}$, each valuation ν then defines a $\overline{\Gamma}$-tree $\nu(\mathcal{T}) := \langle T, (\lambda \times \nu)\rangle$, obtained by annotating each node of $\mathcal{T}$ by its ν-image. As in [5], we define the provenance of a $\overline{\Gamma}$-SATWA A on $\mathcal{T}$, which intuitively captures all possible results of evaluating A on possible valuations of $\mathcal{T}$:

▶ **Definition 31.** The *provenance* of a $\overline{\Gamma}$-SATWA A on a Γ-tree $\mathcal{T}$ is the Boolean function φ defined on the nodes of T such that, for any valuation ν of $\mathcal{T}$, $\nu(\varphi) = 1$ iff A accepts $\nu(\mathcal{T})$.

We then show that we can efficiently build provenance representations of SATWAs on trees as stratified cycluits:

▶ **Theorem 32.** *For any fixed alphabet Γ, given a $\overline{\Gamma}$-SATWA A and a Γ-tree $\mathcal{T}$, we can build a stratified cycluit capturing the provenance of A on $\mathcal{T}$ in time $O(|A| \cdot |\mathcal{T}|)$. Moreover, this stratified cycluit has treewidth $O(|A|)$.*

Note that the proof can be easily modified to make it work for standard alternating two-way automata rather than our isotropic automata. This result allows us to prove Theorem 24, by applying it to the SATWA obtained from the ICG-Datalog program (Theorem 22), slightly modified so as to extend it to the alphabet $\overline{\Gamma}$. Recalling that nodes of the tree encodings each encode at most one fact of the instance, we use the second coordinate of $\overline{\Gamma}$ to indicate whether the fact is actually present or should be discarded. This allows us to range over possible subinstances, and thus to compute the provenance. This concludes the proof of our main result (Theorem 11 in Section 5): we can evaluate an ICG-Datalog program on a treelike instance in FPT-linear time by computing its provenance by Theorem 24 and evaluating the provenance in linear time (Proposition 30).

8 From Cycluits to Circuits and Probability Bounds

We have proven our main result on ICG-Datalog, Theorem 11, in the previous section, introducing stratified cycluits in the process as a way to capture the provenance of ICG-Datalog. In this section, we study how these stratified cycluits can be transformed into *equivalent* acyclic Boolean circuits, and we then show how we can use this to derive bounds for the *probabilistic query evaluation* problem (PQE).

From cycluits to circuits. We call two cycluits or circuits C_1 and C_2 *equivalent* if they have the same set of inputs C_{inp} and, for each valuation ν of C_{inp}, we have $\nu(C_1) = \nu(C_2)$. A first result from existing work is that we can remove cycles in cycluits and convert them to circuits, with a quadratic blowup, by creating linearly many copies to materialize the fixpoint computation. This allows us to remain FPT in combined complexity, but not FPT-linear:

▶ **Proposition 33** (([51], Theorem 2))**.** *For any stratified cycluit C, we can compute in time $O(|C|^2)$ a Boolean circuit C' which is equivalent to C.*

In addition to being quadratic rather than linear, another disadvantage of this approach is that bounds on the treewidth of the cycluit (which we will need later for probability computation) are generally not preserved on the output. Hence, we prove a second cycle removal result, that proceeds in FPT-linear time when parameterized by the treewidth of the input cycluit. When we use this result, we no longer preserve FPT combined complexity of the overall computation, because the stratified cycluits produced by Theorem 24 generally have treewidth $\Omega(|P|)$. On the other hand, we obtain an FPT-linear data complexity bound, and a bounded-treewidth circuit as a result.

▶ **Theorem 34.** *There is an $\alpha \in \mathbb{N}$ s.t., for any stratified cycluit C of treewidth k, we can compute in time $O(2^{k^\alpha}\,|C|)$ a circuit C' which is equivalent to C and has treewidth $O(2^{k^\alpha})$.*

Probabilistic query evaluation. We can then apply the above result to the probabilistic query evaluation (PQE) problem, which we now define:

▶ **Definition 35.** A *TID instance* is a relational instance I and a function π mapping each fact $F \in I$ to a rational probability $\pi(F)$. A TID instance (I, π) defines a probability distribution Pr on $I' \subseteq I$, where $\Pr(I') := \prod_{F \in I'} \pi(F) \times \prod_{F \in I \setminus I'} (1 - \pi(F))$.

The *probabilistic query evaluation* (PQE) problem asks, given a Boolean query Q and a TID instance (I, π), the probability that the query Q is satisfied in the distribution Pr of (I, π). Formally, we want to compute $\sum_{I' \subseteq I \text{ s.t. } I' \models Q} \Pr(I')$. The *data complexity* of PQE is its complexity when Q is fixed and the TID instance (I, π) is given as input. Its *combined complexity* is its complexity when both the query and TID instance are given as input.

Earlier work [25] showed that PQE has #P-hard data complexity even for some CQs of a simple form, but [5, 6] shows that PQE is tractable in data complexity for any Boolean query in monadic second-order (MSO) if the input instances are required to be treelike.

We now explain how to use Theorem 34 for PQE. Let P be an ICG-Datalog program of body size k_{P}. Given a TID instance (I, π) of treewidth k_{I}, we compute a provenance cycluit for P on I of treewidth $O(|P|)$ in FPT-linear time in $|I| \cdot |P|$ by Theorem 24. By Theorem 34, we compute in $O(2^{|P|^\alpha}\,|I|\,|P|)$ an equivalent circuit of treewidth $O(2^{|P|^\alpha})$. Now, by Theorem D.2 of [6], we can solve PQE for P and (I, π) in $O(2^{2^{|P|^\alpha}}\,|I|\,|P| + |\pi|)$ up to PTIME arithmetic costs. Linear-time data complexity was known from [5], but 2EXPTIME combined complexity is novel, as [5] only gave non-elementary combined complexity bounds.

Acyclic queries on tree TIDs. A natural question is then to understand whether better bounds are possible. In particular, is PQE tractable in *combined complexity* on treelike instances? We show that, unfortunately, treewidth bounds are *not* sufficient to ensure this. The proof draws some inspiration from earlier work [40] on the topic of tree-pattern query evaluation in *probabilistic XML* [41].

▶ **Proposition 36.** *There is a fixed arity-two signature on which PQE is #P-hard even when imposing that the input instances have treewidth 1 and the input queries are α-acyclic CQs.*

Path queries on tree TIDs. We must thus restrict the query language further to achieve combined tractability. One natural restriction is to go from α-*acyclic queries* to *path queries*, i.e., Boolean CQs of the form $R_1(x_1, x_2), \ldots, R_n(x_{n-1}, x_n)$, where each R_i is a binary relation of the signature. For instance, $R(x, y), S(y, z), T(z, w)$ is a path query, but $R(x, y), S(z, y)$ is not (we do not allow inverse relations). We can strengthen the previous result to show:

▶ **Proposition 37.** *There is a fixed arity-two signature on which PQE is #P-hard even when imposing that the input instances have treewidth* 1 *and the input queries are path queries.*

Tractable cases. In which cases, then, could PQE be tractable in combined complexity? One example is in [22]: PQE is tractable in combined complexity over *probabilistic XML*, when queries are written as *deterministic* tree automata. In this setting, that the edges of the XML document are *directed* (preventing, e.g., the inverse construction used in the proof of Proposition 37). Further, as the result works on *unranked* trees, it is important that children of a node are *ordered* as well (see [3] for examples where this matters).

We leave open the question of whether there are some practical classes of instances and of queries for which such a deterministic tree automaton can be obtained from the query in polynomial time to test the query for a given treewidth. As we have shown, path queries and instances of treewidth 1, even though very restricted, do not suffice to ensure this. Note that, in terms of data complexity, we have shown in [7] that treelike instances are essentially the only instances for which first-order tractability is achievable.

9 Conclusion

We introduced ICG-Datalog, a new stratified Datalog fragment whose evaluation has FPT-linear complexity when parameterized by instance treewidth and program body size. The complexity result is obtained via compilation to alternating two-way automata, and via the computation of a provenance representation in the form of stratified cycluits, a generalisation of provenance circuits that we hope to be of independent interest.

We believe that ICG-Datalog can be further improved by removing the guardedness requirement on negated atoms, which would make it more expressive and step back from the world of guarded negation logics. In particular, we conjecture that our FPT-linear tractability result generalizes to *frontier-guarded Datalog*, and its extensions with clique-guards and stratified (but unguarded) negation, taking the rule body size and instance treewidth as the parameters. We further hope that our results could be used to derive PTIME combined complexity results on instances of arbitrary treewidth, e.g., XP membership when parametrizing by program size; this could in particular recapture the tractability of bounded-treewidth queries. Last, we intend to extend our cycluit framework to support more expressive provenance semirings than Boolean provenance (e.g., formal power series [37]).

We leave open the question of practical implementation of the methods we developed, but we have good hopes that this approach can give efficient results in practice, in part from our experience with a preliminary provenance prototype [50]. Optimization is possible, for instance by not representing the full automata but building them on the fly when needed in query evaluation. Another promising direction supported by our experience, to deal with real-world datasets that are not treelike, is to use partial tree decompositions [46].

Acknowledgements. This work was partly funded by the Télécom ParisTech Research Chair on Big Data and Market Insights.

References

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

2 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3), 1997.

3 Antoine Amarilli. The possibility problem for probabilistic XML. In *AMW*, 2014. URL: `http://ceur-ws.org/Vol-1189/paper_2.pdf`.

4 Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits (extended version). *CoRR*, abs/1612.04203, 2017. `https://arxiv.org/abs/1612.04203`.

5 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, volume 9135 of *LNCS*, 2015.

6 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances (extended version). *CoRR*, abs/1511.08723, 2015. Extended version of [5].

7 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, 2016.

8 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11), 2012.

9 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3), 2015.

10 Pablo Barceló. Querying graph databases. In *PODS*, 2013.

11 Pablo Barceló, Miguel Romero, and Moshe Y Vardi. Does query evaluation tractability help query containment? In *PODS*, 2014.

12 Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic datalog containment. In *ICALP*, 2012.

13 Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *LICS*, 2016.

14 Michael Benedikt and Georg Gottlob. The impact of virtual views on containment. *PVLDB*, 3(1-2), 2010.

15 Michael Benedikt, Balder ten Cate, and Michael Vanden Boom. Effective interpolation and preservation in guarded logics. In *LICS*, 2014.

16 Anne Berry, Romain Pogorelcnik, and Genevieve Simonet. An introduction to clique minimal separator decomposition. *Algorithms*, 3(2), 2010.

17 Dietmar Berwanger and Erich Grädel. Games and model checking for guarded logics. In *LPAR*, 2001.

18 Jean-Camille Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical systems theory*, 26(3), 1993.

19 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6), 1996.

20 Thierry Cachat. Two-way tree automata solving pushdown games. In *Automata logics, and infinite games*, chapter 17. Springer, 2002.

21 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzeniri, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

22 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

23 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications, 2007. Available from `http://tata.gforge.inria.fr/`.

24 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990.

25 Nilesh Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.

26 Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

27 Reinhard Diestel. Simplicial decompositions of graphs: A survey of applications. *Discrete Math.*, 75(1), 1989.

28 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3), 1983.

29 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

30 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.

31 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combinatorial Theory*, 16(1), 1974.

32 Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1), 2002.

33 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Treewidth and hypertree width. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, chapter 1. Cambridge University Press, 2014.

34 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *JCSS*, 64(3), 2002.

35 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *JCSS*, 66(4), 2003.

36 Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic Datalog over finite structures of bounded treewidth. *TOCL*, 12(1), 2010.

37 Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

38 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *TALG*, 11(1), 2014.

39 Tomasz Imielinski and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984. doi:10.1145/1634.1886.

40 Benny Kimelfeld, Yuri Kosharovsky, and Yehoshua Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD*, 2008.

41 Benny Kimelfeld and Pierre Senellart. Probabilistic XML: Models and complexity. In Zongmin Ma and Li Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*. Springer, 2013.

42 Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society. Series B*, 1988.

43 Hanns-Georg Leimer. Optimal decomposition by clique separators. *Discrete Math.*, 113(1-3), 1993.

44 Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14(3), 2005.

45 Sharad Malik. Analysis of cyclic combinational circuits. In *ICCAD*, 1993.

46 Silviu Maniu, Reynold Cheng, and Pierre Senellart. ProbTree: A query-efficient representation of probabilistic graphs. In *BUDA*, June 2014. Workshop without formal proceedings.

47 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1), 2010.

48 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. In *VLDB*, 1989.

49 Albert R. Meyer. Weak monadic second order theory of succesor is not elementary-recursive. In *Logic Colloquium*, 1975.

50 Mikaël Monet. Probabilistic evaluation of expressive queries on bounded-treewidth instances. In *SIGMOD/PODS PhD Symposium*, June 2016.

51 Marc D. Riedel and Jehoshua Bruck. Cyclic Boolean circuits. *Discrete Applied Mathematics*, 160(13-14), 2012.

52 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3), 1986.

53 Robert E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55(2), 1985.

54 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 1955.

55 Moshe Y Vardi. The complexity of relational query languages. In *STOC*, 1982.

56 Moshe Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, pages 266–276, 1995.

57 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.

The Complexity of Reverse Engineering Problems for Conjunctive Queries

Pablo Barceló[1] and Miguel Romero[2]

1 **Center for Semantic Web Research & Department of Computer Science, University of Chile, Santiago de Chile, Chile**
pbarcelo@dcc.uchile.cl

2 **Center for Semantic Web Research & Department of Computer Science, University of Chile, Santiago de Chile, Chile**
mromero@dcc.uchile.cl

Abstract

Reverse engineering problems for conjunctive queries (CQs), such as query by example (QBE) or definability, take a set of user examples and convert them into an explanatory CQ. Despite their importance, the complexity of these problems is prohibitively high (coNEXPTIME-complete). We isolate their two main sources of complexity and propose relaxations of them that reduce the complexity while having meaningful theoretical interpretations. The first relaxation is based on the idea of using existential pebble games for approximating homomorphism tests. We show that this characterizes QBE/definability for CQs up to treewidth k, while reducing the complexity to EXPTIME. As a side result, we obtain that the complexity of the QBE/definability problems for CQs of treewidth k is EXPTIME-complete for each $k \geq 1$. The second relaxation is based on the idea of "desynchronizing" direct products, which characterizes QBE/definability for unions of CQs and reduces the complexity to coNP. The combination of these two relaxations yields tractability for QBE and characterizes it in terms of unions of CQs of treewidth at most k. We also study the complexity of these problems for conjunctive regular path queries over graph databases, showing them to be no more difficult than for CQs.

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

Keywords and phrases reverse engineering, conjunctive queries, query by example, definability, treewidth, complexity of pebble games

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.7

1 Introduction

Reverse engineering is the general problem of abstracting user examples into an explanatory query. An important instance of this problem corresponds to *query-by-example* (QBE) for a query language $\mathcal{L}$. In QBE, the system is presented with a database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$ of *positive* and *negative* examples, respectively. The question is whether there exists a query q in $\mathcal{L}$ such that its evaluation $q(\mathcal{D})$ over $\mathcal{D}$ contains all the positive examples (i.e., $S^+ \subseteq q(\mathcal{D})$) but none of the negative ones (i.e., $q(\mathcal{D}) \cap S^- = \emptyset$). In case such q exists, it is also desirable to return its result $q(\mathcal{D})$. Another version of this problem assumes that the system is given the set S^+ of positive examples only, and the question is whether there is a query q in $\mathcal{L}$ that precisely defines S^+, i.e., $q(\mathcal{D}) = S^+$. This is often known as the *definability problem* for $\mathcal{L}$. As of late, QBE and definability have received quite some attention in different contexts; e.g., for first-order logic and the class of conjunctive queries over relational databases [26, 23, 19, 7, 2, 24, 22]; for regular path queries over graph databases [1, 6]; for SPARQL queries over RDF [3]; and for tree patterns over XML [10, 20].

In data management, a particularly important instance of QBE and definability corresponds to the case when $\mathcal{L}$ is the class of conjunctive queries (CQs). Nevertheless, the relevance of such instance is counterbalanced by its inherent complexity: Both QBE and definability for CQs are coNEXPTIME-complete [24, 22]. Moreover, in case that a CQ-*explanation* q for S^+ and S^- over $\mathcal{D}$ exists (i.e., a CQ q such that $S^+ \subseteq q(\mathcal{D})$ and $q(\mathcal{D}) \cap S^- = \emptyset$ for QBE), it might take double exponential time to compute its result $q(\mathcal{D})$. While several heuristics have been proposed that alleviate this complexity in practice [26, 23, 19, 7], up to date there has been (essentially) no theoretical investigation identifying the sources of complexity of these problems and proposing principled solutions for them. The general objective of this article is to make a first step in such direction.

A semantic characterization of QBE for CQs has been known for a long time in the community. Formally, there exists a CQ q such that $S^+ \subseteq q(\mathcal{D})$ and $q(\mathcal{D}) \cap S^- = \emptyset$ (i.e., a CQ-explanation) if and only if (essentially) the following *QBE test for CQs* succeeds:

- QBE test for CQs: For each tuple $\bar{b}$ in S^- it is the case that $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \not\to (\mathcal{D}, \bar{b})$, i.e., $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ does not homomorphically map to $(\mathcal{D}, \bar{b})$. (Here, $\prod$ denotes the usual direct product of databases with distinguished tuples of constants).

(A similar test characterizes CQ-definability, save that now $\bar{b}$ is an arbitrary tuple over $\mathcal{D}$ outside S^+). Moreover, in case there is a CQ-explanation q for S^+ and S^- over $\mathcal{D}$, then there is a *canonical* such explanation given by the CQ whose body corresponds to $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$. As shown by Willard [24], the QBE test for CQs yields optimal bounds for determining (a) the existence of a CQ-explanation q for S^+ and S^- over $\mathcal{D}$ (namely, coNEXPTIME), and (b) the size of such q (i.e., exponential). More important, it allows to identify the two main sources of complexity of the problem, each one of which increases its complexity by one exponential:

1. The construction of the canonical explanation $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$, which takes exponential time in the combined size of $\mathcal{D}$ and S^+.
2. The homomorphism test $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$ for each tuple $\bar{b} \in S^-$. Since, in general, checking for the existence of a homomorphism is an NP-complete problem, this step involves an extra exponential blow up.

Our contributions: We propose relaxations of the QBE test for CQs that alleviate one or both sources of complexity and have meaningful theoretical interpretations in terms of the QBE problem (our results also apply to definability). They are based on standard approximation notions for the homomorphism test and the construction of the direct product $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$, as found in the context of constraint satisfaction and definability, respectively.

1. We start by relaxing the second source of complexity, i.e., the one given by the homomorphism tests of the form $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$, for $\bar{b} \in S^-$. In order to approximate the notion of homomorphism, we use the *strong consistency tests* often applied in the area of constraint satisfaction [13]. As observed by Kolaitis and Vardi [18], such consistency tests can be recast in terms of the *existential pebble game* [17], first defined in the context of database theory as a tool for studying the expressive power of Datalog, and also used to show that CQs of bounded treewidth can be evaluated efficiently [12].
   As opposed to the homomorphism test, checking for the existence of a *winning duplicator strategy* in the existential k-pebble game on $(\mathcal{D}, \bar{a})$ and $(\mathcal{D}', \bar{b})$, denoted $(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}', \bar{b})$, can be solved in polynomial time for each fixed $k > 1$ [17]. Therefore, replacing the homomorphism test $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$ with its "approximation" $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}, \bar{b})$ reduces the complexity of the QBE test for CQs to EXPTIME. Furthermore, this approximation has a neat theoretical interpretation: The relaxed version of the QBE

test accepts the input given by $(\mathcal{D}, S^+, S^-)$ if and only if there is a CQ-explanation q for S^+ and S^- over $\mathcal{D}$ such that q is of *treewidth at most* $(k-1)$. While the latter is not particularly surprising in light of the strong existing connections between the existential k-pebble game and the evaluation of CQs of treewidth at most $(k-1)$ [12], we believe our characterization to be of conceptual importance.

Interestingly, when this relaxed version of the QBE test yields a CQ-explanation q of treewidth at most $(k-1)$, its result $q(\mathcal{D})$ can be evaluated in exponential time (recall that for general CQs this might require double exponential time).

2. We then prove that the previous bound is optimal, i.e., checking whether the relaxed version of the QBE test accepts the input given by $(\mathcal{D}, S^+, S^-)$, or, equivalently, if there is a CQ-explanation q for S^+ and S^- over $\mathcal{D}$ of treewidth at most k, for each $k \geq 1$, is EXPTIME-complete. (This also holds for the definability problem for CQs of treewidth at most k). Intuitively, this states that relaxing the second source of complexity of the test by using existential pebble games does not eliminate the first one.
3. Finally, we look at the second source of complexity, i.e., the construction of the exponential size canonical explanation $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$. While it is not clear which techniques are better suited for approximating this construction, we look at a particular one that appears in the context of definability: Instead of constructing the synchronized product $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ with respect to all tuples in S, we look at them one by one. That is, we check whether for each tuples $\bar{a} \in S^+$ and $\bar{b} \in S^-$ it is the case that $(\mathcal{D}, \bar{a}) \not\to (\mathcal{D}, \bar{b})$. By using a characterization developed in the context of definability [1], we observe that this relaxed version of the QBE test is coNP-complete and has a meaningful interpretation: It corresponds to finding explanations based on *unions* of CQs. Moreover, when combined with the previous relaxation (i.e., replacing the homomorphism test $(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$ with $(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}, \bar{b})$) we obtain tractability. This further relaxed test corresponds to finding explanations over the set of unions of CQs of treewidth at most $(k-1)$.

We then switch to study QBE in the context of graph databases, where CQs are often extended with the ability to check whether two nodes are linked by a path whose label satisfies a given regular expression. This gives rise to the class of *conjunctive regular path queries*, or CRPQs (see, e.g., [11, 8, 25, 5]). CRPQ-definability was first studied by Antonopulos et al. [1]. In particular, it is shown that CRPQ-definability is in EXPSPACE by exploting automata-based techniques, in special, pumping arguments. Our contributions in this context are the following:

1. We first provide a QBE test for CRPQs in the spirit of the one for CQs given above. With such characterization we prove that QBE and definability for CRPQs are in coNEXPTIME, improving the EXPSPACE upper bound of Antonopoulos et al. This tells us that these problems are at least not more difficult than for CQs.
2. We also develop relaxations of the QBE test for CRPQs based on the existential pebble game and the "desynchronization" of the direct product $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$. As before, we show that they reduce the complexity of the test and have meaningful interpretations in terms of the class of queries we use to construct explanations.

Organization: Preliminaries are in Section 2. A review of QBE/definability for CQs is provided in Section 3. Relaxations of the homomorphism tests are studied in Section 4 and the desynchronization of the direct product in Section 5. In Section 6 we consider QBE/definability for CRPQs. Future work is presented in Section 7.

2 Preliminaries

Databases, homomorphisms, and direct products. A *schema* is a finite set of relation symbols, each one of which has an associated arity $n > 0$. A *database* over schema σ is a finite set of atoms of the form $R(\bar{a})$, where R is a relation symbol in σ of arity $n > 0$ and $\bar{a}$ is an n-ary tuple of constants. We slightly abuse notation, and sometimes write $\mathcal{D}$ also for the set of elements mentioned in $\mathcal{D}$.

Let $\mathcal{D}$ and $\mathcal{D}'$ be databases over the same schema σ. A *homomorphism* from $\mathcal{D}$ to $\mathcal{D}'$ is a mapping h from the elements of $\mathcal{D}$ to the elements of $\mathcal{D}'$ such that for every atom $R(\bar{a})$ in $\mathcal{D}$ it is the case that $R(h(\bar{a})) \in \mathcal{D}'$. We often need to talk about distinguished tuples of elements in databases. We then write $(\mathcal{D}, \bar{a})$ to define the pair that corresponds to the database $\mathcal{D}$ and the tuple $\bar{a}$ of elements in $\mathcal{D}$. Let $\bar{a}$ and $\bar{b}$ be n-ary ($n \geq 0$) tuples of elements in $\mathcal{D}$ and $\mathcal{D}'$, respectively. A homomorphism from $(\mathcal{D}, \bar{a})$ to $(\mathcal{D}', \bar{b})$ is a homomorphism from $\mathcal{D}$ to $\mathcal{D}'$ such that $h(\bar{a}) = \bar{b}$. We write $(\mathcal{D}, \bar{a}) \to (\mathcal{D}', \bar{b})$ if there is a homomorphism from $(\mathcal{D}, \bar{a})$ to $(\mathcal{D}', \bar{b})$. Checking if $(\mathcal{D}, \bar{a}) \to (\mathcal{D}', \bar{b})$ is a well-known NP-complete problem.

In this work, the notion of *direct product* of databases is particularly important. Let $\bar{a} = (a_1, \ldots, a_n)$ and $\bar{b} = (b_1, \ldots, b_n)$ be n-ary tuples of elements over A and B, respectively. Their direct product $\bar{a} \otimes \bar{b}$ is the n-ary tuple $((a_1, b_1), \ldots, (a_n, b_n))$ over $A \times B$. If $\mathcal{D}$ and $\mathcal{D}'$ are databases over the same schema σ, we define $\mathcal{D} \otimes \mathcal{D}'$ to be the following database over σ:

$$\{R(\bar{a} \otimes \bar{b}) \mid R \in \sigma,\ R(\bar{a}) \in \mathcal{D},\ \text{and}\ R(\bar{b}) \in \mathcal{D}'\}.$$

Further, we use $(\mathcal{D}, \bar{a}) \otimes (\mathcal{D}', \bar{b})$ to denote the pair $(\mathcal{D} \otimes \mathcal{D}', \bar{a} \otimes \bar{b})$, and write $\prod_{1 \leq i \leq m}(\mathcal{D}_i, \bar{a}_i)$ as a shorthand for $(\mathcal{D}_1, \bar{a}_1) \otimes \cdots \otimes (\mathcal{D}_m, \bar{a}_m)$. This is allowed since $\otimes$ is an associative operation.

The elements in the tuple $\prod_{1 \leq i \leq m} \bar{a}_i$ may or may not appear in $\prod_{1 \leq i \leq m} \mathcal{D}_i$. If they do appear, we call $\prod_{1 \leq i \leq m}(\mathcal{D}_i, \bar{a}_i)$ *safe*. The notion of safeness is important in our work for reasons that will become apparent later. The next example better explains this notion:

▶ **Example 1.** If $\mathcal{D} = \{R(a, b), S(c, d)\}$, $\bar{a}_1 = (a, b)$, and $\bar{a}_2 = (c, d)$, then $(\mathcal{D}, \bar{a}_1) \otimes (\mathcal{D}, \bar{a}_2)$ is unsafe. In fact, $\bar{a}_1 \otimes \bar{a}_2 = \big((a, c), (b, d)\big)$ and $\mathcal{D} \otimes \mathcal{D} = \{R((a, a), (b, b)), S((c, c), (d, d))\}$. That is, none of the elements in $\bar{a}_1 \otimes \bar{a}_2$ belongs to $\mathcal{D} \otimes \mathcal{D}$. ◀

It is worth remarking that the direct product $\otimes$ defines the least upper bound in the lattice of databases defined by the notion of homomorphism. In particular:

1. $\prod_{1 \leq i \leq m}(\mathcal{D}_i, \bar{a}_i) \to (\mathcal{D}_i, \bar{a}_i)$ for each $1 \leq i \leq m$, and
2. if $(\mathcal{D}, \bar{a}) \to (\mathcal{D}_i, \bar{a}_i)$ for each $1 \leq i \leq m$, then $(\mathcal{D}, \bar{a}) \to \prod_{1 \leq i \leq m}(\mathcal{D}_i, \bar{a}_i)$.

Conjunctive queries. A *conjunctive query* (CQ) q over relational schema σ is an FO formula of the form:

$$\exists \bar{y}\big(R_1(\bar{x}_1) \wedge \cdots \wedge R_m(\bar{x}_m)\big), \tag{1}$$

such that (a) each $R_i(\bar{x}_i)$ is an atom over σ, for $1 \leq i \leq m$, and (b) $\bar{y}$ is a sequence of variables taken from the $\bar{x}_i$'s. In order to ensure domain-independence for queries, we only consider CQs without constants. We often write $q(\bar{x})$ to denote that $\bar{x}$ is the sequence of *free* variables of q, i.e., the ones that do not appear existentially quantified in $\bar{y}$.

Let $\mathcal{D}$ be a database over σ. We define the evaluation of a CQ $q(\bar{x})$ of the form (1) over $\mathcal{D}$ in terms of the homomorphisms from $\mathcal{D}_q$ to $\mathcal{D}$, where $\mathcal{D}_q$ is the *canonical database* of q, that is, $\mathcal{D}_q$ is the database $\{R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)\}$ that contains all atoms in q. The *evaluation of* $q(\bar{x})$ *over* $\mathcal{D}$, denoted $q(\mathcal{D})$, contains exactly those tuples $h(\bar{x})$ such that h is a homomorphism from $\mathcal{D}_q$ to $\mathcal{D}$.

CQs of bounded treewidth. The evaluation problem for CQs (i.e., determining whether $q(\mathcal{D}) \neq \emptyset$, given a database $\mathcal{D}$ and a CQ q) is NP-complete, but becomes tractable for several syntactically defined classes. One of the most prominent such classes corresponds to the CQs of *bounded treewidth* [9]. Recall that treewidth is a graph-theoretical concept that measures how much a graph resembles a tree (see, e.g., [14]). For instance, trees have treewidth one, cycles treewidth two, and K_k, the clique on k elements, treewidth $k-1$.

Formally, let $G = (V, E)$ be an undirected graph. A *tree decomposition* of G is a pair (T, λ), where T is a tree and λ is a mapping that assigns a nonempty set of nodes in V to each node t in T, for which the following holds:

1. For each $v \in V$ it is the case that the set of nodes $t \in T$ such that $v \in \lambda(t)$ is connected.
2. For each edge $\{u, v\} \in E$ there exists a node $t \in T$ such that $\{u, v\} \subseteq \lambda(t)$.

The *width* of (T, λ) corresponds to $(\max\{|\lambda(t)| \mid t \in T\}) - 1$. The treewidth of G is then defined as the minimum width of its tree decompositions.

We define the treewidth of a CQ $q = \exists \bar{y} \bigwedge_{1 \leq i \leq m} R_i(\bar{x}_i)$ as the treewidth of the *Gaifman graph* of its existentially quantified variables. Recall that this is the undirected graph whose vertices are the existentially quantified variables of q (i.e., those in $\bar{y}$) and there is an edge between distinct existentially quantified variables y and y' if and only they appear together in some atom of q, that is, they both appear in a tuple $\bar{x}_i$ for $1 \leq i \leq m$. For $k \geq 1$, we denote by $\mathsf{TW}(k)$ the class of CQs of treewidth at most k. It is known that the evaluation problem for the class $\mathsf{TW}(k)$ (for each fixed $k \geq 1$) can be solved in polynomial time [9, 12].

The QBE and definability problems. Let $\mathcal{C}$ be a class of queries (e.g., the class CQ of all conjunctive queries, or $\mathsf{TW}(k)$ of CQs of treewidth at most k). Suppose that $\mathcal{D}$ is a database and S^+ and S^- are n-ary relations over $\mathcal{D}$ of positive and negative examples, respectively. A *$\mathcal{C}$-explanation* for S^+ and S^- over $\mathcal{D}$ is a query q in $\mathcal{C}$ such that $S^+ \subseteq q(\mathcal{D})$ and $q(\mathcal{D}) \cap S^- = \emptyset$. Analogously, a *$\mathcal{C}$-definition* of S^+ over $\mathcal{D}$ is a query q in $\mathcal{C}$ such that $q(\mathcal{D}) = S^+$. The *query by example* and *definability* problems for $\mathcal{C}$ are as follows:

PROBLEM :	$\mathcal{C}$-QUERY-BY-EXAMPLE (resp., $\mathcal{C}$-DEFINABILITY)
INPUT :	A database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$ (resp., a database $\mathcal{D}$ and an n-ary relation S^+ over $\mathcal{D}$)
QUESTION :	Is there a $\mathcal{C}$-explanation for S^+ and S^- over $\mathcal{D}$? (resp., is there a $\mathcal{C}$-definition of S^+ over $\mathcal{D}$?)

3 Query by example and definability for CQs

Let us start by recalling what is known about these problems for CQs. We first establish characterizations of the notions of CQ-explanations/definitions based on the following tests:

- QBE test for CQs: Takes as input a database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$. It accepts if and only if:
  1. $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ is safe, and
  2. $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \not\to (\mathcal{D}, \bar{b})$ for each tuple $\bar{b} \in S^-$.
- Definability test for CQs: Takes as input a database $\mathcal{D}$ and an n-ary relation S^+ over $\mathcal{D}$. It accepts if and only if:
  1. $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ is safe, and
  2. $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \not\to (\mathcal{D}, \bar{b})$ for each n-ary tuple $\bar{b}$ over $\mathcal{D}$ that is not in S^+.

The following characterizations are considered to be folklore in the literature:

▶ **Proposition 2.** *The following statements hold:*
1. *Let $\mathcal{D}$ be a database and S^+, S^- relations over $\mathcal{D}$. There is a* CQ*-explanation for S^+ and S^- over $\mathcal{D}$ if and only if the QBE test for CQs accepts $\mathcal{D}$, S^+, and S^-.*
2. *Let $\mathcal{D}$ be a database and S^+ a relation over $\mathcal{D}$. There is a* CQ*-definition for S^+ over $\mathcal{D}$ if and only if the definability test for CQs accepts $\mathcal{D}$ and S^+.*

This provides us with a simple method for obtaining a coNEXPTIME upper bound for CQ-QUERY-BY-EXAMPLE and CQ-DEFINABILITY. Let us concentrate on the first problem (a similar argument works for the second one). Assume that S^+ and S^- are relations of positive and negative examples over a database $\mathcal{D}$. It follows from Proposition 2 that to check that there is *not* CQ-explanation for S^+ and S^- over $\mathcal{D}$, we need to either show that $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ is unsafe or guess a tuple $\bar{b} \in S^-$ and a homomorphism h from $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ to $(\mathcal{D},\bar{b})$. Since $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ is of exponential size, checking its safety can be carried out in exponential time. On the other hand, the guess of h is also of exponential size, and therefore checking that h is indeed a homomorphism from $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ to $(\mathcal{D},\bar{b})$ can be performed in exponential time. The whole procedure can then be carried out in NEXPTIME. As it turns out, this bound is also optimal:

▶ **Theorem 3.** [24, 22] *The problems* CQ-QUERY-BY-EXAMPLE *and* CQ-DEFINABILITY *are* coNEXPTIME*-complete.*

The lower bound for CQ-DEFINABILITY was established by Willard using a complicated reduction from the complement of a tiling problem. A simpler proof was then obtained by ten Cate and Dalmau [22]. Their techniques also establish a lower bound for CQ-QUERY-BY-EXAMPLE. Notably, these lower bounds hold even when S^+ and S^- are unary relations.

The cost of evaluating CQ-explanations. Recall that in query by example not only we want to find a CQ-explanation q for S^+ and S^- over $\mathcal{D}$, but also compute its result $q(\mathcal{D})$ if possible. It follows from the proof of Proposition 2 that in case there is a CQ-explanation for S^+ and S^- over $\mathcal{D}$, then we can assume such CQ to be $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$, i.e., the CQ whose set of atoms is $\mathcal{D}^{|S^+|}$ and whose tuple of free variables is $\prod_{\bar{a}\in S^+}\bar{a}$ (notice that we are using here the assumption that $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ is safe, i.e., that the free variables in $\prod_{\bar{a}\in S^+}\bar{a}$ do in fact appear in the atoms in $\mathcal{D}^{|S^+|}$). The CQ $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ is known as the *canonical* CQ-explanation. We could then simply evaluate this canonical CQ-explanation over $\mathcal{D}$ in order to meet the requirements of query by example. This, however, takes double exponential time since $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ itself is of exponential size. It is not known whether there are better algorithms for computing the result of *some* CQ-explanation, but the results in this section suggest that this is unlikely.

Size of CQ explanations and definitions. It follows from the previous observations that CQ-explanations are of at most exponential size (by taking the canonical CQ-explanation as witness). The same holds for CQ-definitions. Interestingly, these bounds are optimal:

▶ **Proposition 4.** [24, 22] *The following statements hold:*
1. *If there is a* CQ*-explanation for S^+ and S^- over $\mathcal{D}$, then there is a* CQ*-explanation of at most exponential size; namely, $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$. Similarly, for* CQ*-definitions.*
2. *There is a family $(\mathcal{D}_n, S_n^+, S_n^-)_{n\geq 0}$ of tuples of databases $\mathcal{D}_n$ and relations S_n^+ and S_n^- over $\mathcal{D}_n$, such that (a) the combined size of $\mathcal{D}_n$, S_n^+, and S_n^- is polynomial in n, (b) there is a* CQ*-explanation for S_n^+ and S_n^- over $\mathcal{D}_n$, and (c) the size of the smallest such* CQ*-explanation is at least 2^n. Similarly, for* CQ*-definitions.*

Sources of complexity. The QBE test performs the following steps on input $(\mathcal{D}, S^+, S^-)$: (1) It computes $\prod_{\bar{a}\in S^+}(\mathcal{D}, \bar{a})$, and (2) it checks whether $\prod_{\bar{a}\in S^+}(\mathcal{D}, \bar{a})$ is unsafe or it is the case that $\prod_{\bar{a}\in S^+}(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$ for some $\bar{b} \in S^-$. The definability test is equivalent, but the homomorphism test is then extended to each tuple over $\mathcal{D}$ but outside S^+. Two sources of complexity are involved in these tests, each one of which incurs in one exponential blow up: (a) The construction of $\prod_{\bar{a}\in S^+}(\mathcal{D}, \bar{a})$, and (b) the homomorphism tests $\prod_{\bar{a}\in S^+}(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$. In order to alleviate the high complexity of the tests we thus propose relaxations of these two sources of complexity. The proposed relaxations are based on well-studied approximation notions with strong theoretical support. As such, they give rise to clean reformulations of the notions of CQ-explanations/definitions. We start with the homomorphism test in the following section.

4 A relaxation of the homomorphism test

We use an approximation technique for the homomorphism test based on the existential pebble game. This technique finds several applications in database theory [17, 12] and can be shown to be equivalent to the strong consistency tests for homomorphism approximation used in the area of constraint satisfaction [18]. The complexity of the (existential) pebble game is by now well-understood [15, 16]. We borrow several techniques used in such analysis to understand the complexity of our problems. We also prove some results on the complexity of such games that are of independent interest. We define the existential pebble game below.

The existential pebble game. Let $k > 1$. The existential k-pebble game is played by the spoiler and the duplicator on pairs $(\mathcal{D}, \bar{a})$ and $(\mathcal{D}', \bar{b})$, where $\mathcal{D}$ and $\mathcal{D}'$ are databases over the same schema and $\bar{a}$ and $\bar{b}$ are n-ary ($n \geq 0$) tuples over $\mathcal{D}$ and $\mathcal{D}'$, respectively. The spoiler plays on $\mathcal{D}$ only, and the duplicator responds on $\mathcal{D}'$. In the first round the spoiler places his pebbles $\mathtt{p}_1, \ldots, \mathtt{p}_k$ on (not necessarily distinct) elements $c_1, \ldots, c_k$ in $\mathcal{D}$, and the duplicator responds by placing his pebbles $\mathtt{q}_1, \ldots, \mathtt{q}_k$ on elements $d_1, \ldots, d_k$ in $\mathcal{D}'$. In every further round, the spoiler removes one of his pebbles, say $\mathtt{p}_i$, for $1 \leq i \leq k$, and places it on an element of $\mathcal{D}$, and the duplicator responds by placing his corresponding pebble $\mathtt{q}_i$ on some element of $\mathcal{D}'$. The duplicator wins if he has a *winning strategy*, i.e., he can indefinitely continue playing the game in such way that at each round, if $c_1, \ldots, c_k$ and $d_1, \ldots, d_k$ are the elements covered by pebbles $\mathtt{p}_1, \ldots, \mathtt{p}_k$ and $\mathtt{q}_1, \ldots, \mathtt{q}_k$ on $\mathcal{D}$ and $\mathcal{D}'$, respectively, then

$$\big((c_1, \ldots, c_k, \bar{a}), (d_1, \ldots, d_k, \bar{b})\big)$$

is a *partial homomorphism* from $\mathcal{D}$ to $\mathcal{D}'$. Recall that this means that for every atom of the form $R(\bar{c}) \in \mathcal{D}$, where each element c of $\bar{c}$ appears in $(c_1, \ldots, c_k, \bar{a})$, it is the case that $R(\bar{d}) \in \mathcal{D}'$, where $\bar{d}$ is the tuple that is obtained from $\bar{c}$ by replacing each element c of $\bar{c}$ by its corresponding element d in $(d_1, \ldots, d_k, \bar{b})$. If such winning strategy for the duplicator exists, we write $(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}', \bar{b})$.

It is easy to see that the relations $\to_k$, for $k > 1$, provide an approximation of the notion of homomorphism in the following sense:

$$\to \subsetneq \ \cdots \ \subsetneq \to_{k+1} \subsetneq \to_k \subsetneq \ \cdots \subsetneq \to_2 .$$

Furthermore, these approximations are convenient from a complexity point of view: While checking for the existence of a homomorphism from $(\mathcal{D}, \bar{a})$ to $(\mathcal{D}', \bar{b})$ is NP-complete, checking for the existence of a winning strategy for the duplicator in the existential k-pebble game can be solved efficiently:

▶ **Proposition 5.** [17] *Fix $k > 1$. Checking if $(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}', \bar{b})$, given databases $\mathcal{D}$ and $\mathcal{D}'$ and n-ary tuples $\bar{a}$ and $\bar{b}$ over $\mathcal{D}$ and $\mathcal{D}'$, respectively, can be solved in polynomial time.*

Furthermore, there is an interesting connection between the existential pebble game and the evaluation of CQs of bounded treewidth as established in the following proposition:

▶ **Proposition 6.** [4] *Fix $k \geq 1$. Consider databases $\mathcal{D}$ and $\mathcal{D}'$ over the same schema and n-ary tuples $\bar{a}$ and $\bar{b}$ over $\mathcal{D}$ and $\mathcal{D}'$, respectively. Then $(\mathcal{D}, \bar{a}) \to_{k+1} (\mathcal{D}', \bar{b})$ if and only if for each CQ $q(\bar{x})$ in $\mathsf{TW}(k)$ such that $|\bar{x}| = n$ the following holds:*

$$\bar{a} \in q(\mathcal{D}) \implies \bar{b} \in q(\mathcal{D}'),$$

or, equivalently, $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{a})$ implies $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}', \bar{b})$, where as before $\mathcal{D}_q$ is the database that contains all the atoms of q.

Moreover, in case that $(\mathcal{D}, \bar{a}) \not\to_{k+1} (\mathcal{D}', \bar{b})$ there exists an exponential size CQ $q(\bar{x})$ in $\mathsf{TW}(k)$ such that $\bar{a} \in q(\mathcal{D})$ but $\bar{b} \notin q(\mathcal{D}')$.

The relaxed test. We study the following relaxed version of the QBE test for CQs that replaces the notion of homomoprhism $\to$ with its approximation $\to_k$, for a fixed $k > 1$:

- k-pebble QBE test for CQs: Takes as input a database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$. It accepts if and only if:
  1. $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ is safe, and
  2. $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \not\to_k (\mathcal{D}, \bar{b})$ for each tuple $\bar{b} \in S^-$.

Analogously, we define the k-pebble definability test for CQs. It immediately follows from the fact that the relation $\to_k$ can be decided in polynomial time (Proposition 5) that the k-pebble tests for CQs reduce the complexity of the general test from coNEXPTIME to EXPTIME. Later, in Section 4.2, we show that this is optimal.

4.1 A characterization of the k-pebble tests for CQs

Using Proposition 6 we can now establish the theoretical meaningfulness of the relaxed tests: They admit a clean characterization in terms of the CQs of bounded treewidth. In fact, recall that the QBE (resp., definability) test for CQs precisely characterizes the existence of CQ-explanations (resp., CQ-definitions). As we show next, their relaxed versions based on the existential $(k+1)$-pebble game preserve these characterizations up to treewidth k:

▶ **Theorem 7.** *Fix $k \geq 1$. Consider a database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$.*

1. *There is a $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$ if and only if the $(k+1)$-pebble QBE test for CQs accepts $\mathcal{D}$, S^+, and S^-.*
2. *There is a $\mathsf{TW}(k)$-definition for S^+ over $\mathcal{D}$ if and only if the $(k+1)$-pebble definability test for CQs accepts $\mathcal{D}$ and S^+.*

Proof. We concentrate on explanations (the proof for definitions is analogous). From left to right, assume for the sake of contradiction that q is a $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$, yet the $(k+1)$-pebble QBE test for CQs fails over $\mathcal{D}$, S^+, and S^-. Since there is a $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$, we have from Proposition 2 that $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ is safe. Therefore, it must be the case that $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \to_{k+1} (\mathcal{D}, \bar{b})$ for some $\bar{b} \in S^-$. Since $S^+ \subseteq q(\mathcal{D})$ it is the case that $\bar{a} \in q(\mathcal{D})$ for each $\bar{a} \in S^+$. That is, $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{a})$ for each $\bar{a} \in S^+$. Due to basic properties of direct products, this implies that $(\mathcal{D}_q, \bar{x}) \to \prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$.

From Proposition 6 we conclude that $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{b})$, i.e., $\bar{b} \in q(\mathcal{D})$. This is a contradiction since $\bar{b} \in S^-$ and $q(\mathcal{D}) \cap S^- = \emptyset$.

From right to left, assume that the $(k+1)$-pebble QBE test for CQs accepts $\mathcal{D}$, S^+, and S^-, i.e., $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ is safe and for every tuple $\bar{b} \in S^-$ it is the case that $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a}) \not\to_{k+1} (\mathcal{D}, \bar{b})$. Since $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ is safe we can apply Proposition 6, which tells us that for each $\bar{b} \in S^-$ there is a CQ $q_{\bar{b}}(\bar{x})$ such that $(\mathcal{D}_{q_{\bar{b}}}, \bar{x}) \to \prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ but $(\mathcal{D}_{q_{\bar{b}}}, \bar{x}) \not\to (\mathcal{D}, \bar{b})$. Suppose first that $S^- \neq \emptyset$ and let:

$$q(\bar{x}) \ := \bigwedge_{\bar{b} \in S^-} q_{\bar{b}}(\bar{x}).$$

It is easy to see that $q(\bar{x})$ is well-defined (since S^- is nonempty) and can be expressed as a CQ in $\mathsf{TW}(k)$. For the latter we simply use fresh existentially quantified variables for each CQ $q_{\bar{b}}$ such that $\bar{b} \in S^-$ and then move all existentially quantified variables in $\bigwedge_{\bar{b} \in S^-} q_{\bar{b}}(\bar{x})$ to the front. We now prove that $q(\bar{x})$ is a $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$. It easily follows that $(\mathcal{D}_q, \bar{x}) \to \prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ from the fact that $(\mathcal{D}_{q_{\bar{b}}}, \bar{x}) \to \prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$ for each $\bar{b} \in S^-$. But then $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{a})$ for each $\bar{a} \in S^+$. This means that $\bar{a} \in q(\mathcal{D})$ for each $\bar{a} \in S^+$, i.e., $S^+ \subseteq q(\mathcal{D})$. Assume now for the sake of contradiction that $q(\mathcal{D}) \cap S^- \neq \emptyset$, that is, there is a tuple $\bar{b} \in q(\mathcal{D}) \cap S^-$. Then $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{b})$, which implies that $(\mathcal{D}_{q_{\bar{b}}}, \bar{x}) \to (\mathcal{D}, \bar{b})$. This is a contradiction. The case when $S^- = \emptyset$ can be proved using similar techniques. ◀

4.2 The complexity of the k-pebble tests for CQs

As mentioned before, the k-pebble tests for CQs can be evaluated in exponential time. We show here that such bounds are also optimal:

▶ **Theorem 8.** *Deciding whether the k-pebble QBE test for CQs accepts $(\mathcal{D}, S^+, S^-)$ is* EXPTIME*-complete for each $k > 1$. Similarly, for the k-pebble definability test for CQs. This holds even if restricted to the case when S^+ and S^- are unary relations.*

As a corollary to Theorems 7 and 8, we obtain the following interesting result:

▶ **Corollary 9.** *The problems* $\mathsf{TW}(k)$-QUERY-BY-EXAMPLE *and* $\mathsf{TW}(k)$-DEFINABILITY *are* EXPTIME*-complete for each fixed $k \geq 1$. This holds even if restricted to the case when the relations to be explained/defined are unary.*

We now provide a brief outline of the main ideas used for proving the lower bounds in Theorem 8. Let us first notice that in the case of the general QBE/definability tests for CQs, a coNEXPTIME lower bound is obtained in [22] as follows:

1. It is first shown that the following *product homomorphism problem* (PHP) is NEXPTIME-hard: Given databases $\mathcal{D}_1, \ldots, \mathcal{D}_m$ and $\mathcal{D}$, is it the case that $\prod_{1 \leq i \leq m} \mathcal{D}_i \to \mathcal{D}$?
2. It is then shown that there is an easy polynomial-time reduction from PHP to the problem of checking whether the QBE/definability test fails on its input.

The ideas used for proving (2) can be easily adapted to show that there is a polynomial-time reduction from the following *relaxed* version of PHP to the problem of checking whether the k-pebble QBE/definability test fails on its input:

PROBLEM :	k-PEBBLE PHP (for $k > 1$)
INPUT :	Databases $\mathcal{D}_1, \ldots, \mathcal{D}_m$ and $\mathcal{D}$ over the same schema
QUESTION :	Is it the case that $\prod_{1 \leq i \leq m} \mathcal{D}_i \to_k \mathcal{D}$?

We establish that this relaxed version of PHP is EXPTIME-complete for each fixed $k > 1$:

▶ **Theorem 10.** *The problem* k-PEBBLE PHP *is* EXPTIME*-complete for each fixed* $k > 1$.

To prove this result, we exploit techniques from [16, 15] that study the complexity of pebble games. In particular, it is shown in [16] that for each fixed $k > 1$, checking whether $\mathcal{D} \to_k \mathcal{D}'$ is P-complete. The proof uses an involved reduction from the *monotone circuit value problem*, that is, given a monotone circuit C, it constructs two databases $\mathcal{D}_C$ and $\mathcal{D}'_C$ such that the value of C is 1 if and only if $\mathcal{D}_C \to_k \mathcal{D}'_C$.

In our case, to show that k-PEBBLE PHP is EXPTIME-hard for each fixed $k > 1$, we reduce from the following well-known EXPTIME-complete problem: Given an alternating Turing machine M and a positive integer n, decide whether M accepts the empty tape using n space. The latter problem can be easily recast as a circuit value problem: We can construct a circuit $C_{M,n}$ such that the value of $C_{M,n}$ is 1 if and only if M accepts the empty tape using n space. The main idea of our reduction is to construct databases $\mathcal{D}_1, \ldots, \mathcal{D}_m$ and $\mathcal{D}$, given M and n, such that:

$$\prod_{1 \leq i \leq m} \mathcal{D}_i \to_k \mathcal{D} \iff \mathcal{D}_{C_{M,n}} \to_k \mathcal{D}'_{C_{M,n}},$$

where $\mathcal{D}_{C_{M,n}}$ and $\mathcal{D}'_{C_{M,n}}$ are defined as in [16].

A natural approach then is to construct $\mathcal{D}_1, \ldots, \mathcal{D}_m, \mathcal{D}$ such that $\prod_{1 \leq i \leq m} \mathcal{D}_i$ and $\mathcal{D}$ roughly coincide with $\mathcal{D}_{C_{M,n}}$ and $\mathcal{D}'_{C_{M,n}}$. However, there is a problem with this: the databases $\mathcal{D}_{C_{M,n}}$ and $\mathcal{D}'_{C_{M,n}}$ closely resemble the circuit $C_{M,n}$, but the size of $C_{M,n}$ is exponential in $|M|$ and n, and so are the sizes of $\mathcal{D}_{C_{M,n}}$ and $\mathcal{D}'_{C_{M,n}}$. Although it is possible to codify the exponential size database $\mathcal{D}_{C_{M,n}}$ using a product of polynomial size databases $\mathcal{D}_1, \ldots, \mathcal{D}_m$, we cannot do the same with the exponential size $\mathcal{D}'_{C_{M,n}}$ using $\mathcal{D}$ only. To overcome this, we need to extend the techniques in [16] and show that the complexity of the existential k-pebble game is P-complete even over a *fixed template*:

▶ **Lemma 11.** *For each fixed* $k > 1$*, there is a database* $\mathcal{D}_k$ *that only depends on* k*, such that the following problem is* P*-complete: Given a database* $\mathcal{D}$*, decide whether* $\mathcal{D} \to_k \mathcal{D}_k$.

To prove this, we again use a reduction from the circuit value problem that given a circuit C constructs a database $\tilde{\mathcal{D}}_C$ such that C takes value 1 if and only if $\tilde{\mathcal{D}}_C \to_k \mathcal{D}_k$. We then use the following idea to prove that k-PEBBLE PHP is EXPTIME-complete: Given M and n, we construct in polynomial time databases $\mathcal{D}_1, \ldots, \mathcal{D}_m$ and $\mathcal{D}$ such that $\prod_{1 \leq i \leq m} \mathcal{D}_i$ and $\mathcal{D}$ roughly coincide with $\tilde{\mathcal{D}}_{C_{M,n}}$ and $\mathcal{D}_k$, respectively. It then follows that:

$$\prod_{1 \leq i \leq m} \mathcal{D}_i \to_k \mathcal{D} \iff \tilde{\mathcal{D}}_{C_{M,n}} \to_k \mathcal{D}_k \iff M \text{ accepts the empty tape using } n \text{ space.}$$

4.3 Evaluating the result of TW(k)-explanations

Recall that computing the result of CQ-explanations might require double exponential time. For TW(k)-explanations, instead, we can do this in single exponential time.

▶ **Theorem 12.** *Fix* $k \geq 1$*. There is a single exponential time algorithm that, given a database* $\mathcal{D}$ *and* n*-ary relations* S^+ *and* S^- *over* $\mathcal{D}$*, does the following:*

1. *It checks whether there is a* TW(k)*-explanation for* S^+ *and* S^- *over* $\mathcal{D}$*, and*
2. *if the latter holds, it computes the evaluation* $q(\mathcal{D})$ *of one such* TW(k)*-explanation* q.

Proof. We first check in exponential time the existence of one such $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$ using the $(k+1)$-pebble QBE test for CQs. If such $\mathsf{TW}(k)$-explanation exists, we compute in exponential time the set S^e of all n-ary tuples $\bar{b}$ over $\mathcal{D}$ such that $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a}) \to_{k+1} (\mathcal{D},\bar{b})$. Notice, in particular, that $S^+ \subseteq S^e$ and $S^e \cap S^- = \emptyset$. Moreover, it can be shown that $S^e = q(\mathcal{D})$ for some $\mathsf{TW}(k)$-explanation q for S^+ and S^- over $\mathcal{D}$. ◀

Notably, the previous result computes the result of a $\mathsf{TW}(k)$-explanation q for S^+ and S^- over $\mathcal{D}$ without explicitly computing q. One might wonder whether it is possible to also include q in the output of the algorithm. The answer is negative, and the reason is that $\mathsf{TW}(k)$-explanations/definitions can be double exponentially large in the worst case:

▶ **Proposition 13.** *Fix $k \geq 1$. The following holds:*

1. *Assume that there is a $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$. Then there is one such $\mathsf{TW}(k)$-explanation of at most double exponential size.*
2. *There is a family $(\mathcal{D}_n, S_n^+, S_n^-)_{n\geq 0}$ of tuples of databases $\mathcal{D}_n$ and relations S_n^+ and S_n^- over $\mathcal{D}_n$, such that (a) the combined size of $\mathcal{D}_n$, S_n^+, and S_n^- is polynomial in n, (b) there is a $\mathsf{TW}(k)$-explanation for S_n^+ and S_n^- over $\mathcal{D}_n$, and (c) the size of the smallest such $\mathsf{TW}(k)$-explanation is at least 2^{2^n}.*

The same holds for $\mathsf{TW}(k)$-definitions.

Proof. From the proof of Theorem 7, whenever there is a $\mathsf{TW}(k)$-explanation for S^+ and S^- over $\mathcal{D}$ this can be assumed to be the CQ $q = \bigwedge_{\bar{b}\in S^-} q_{\bar{b}}(\bar{x})$. From Proposition 6, each such $q_{\bar{b}}$ is of exponential size in the combined size of $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$ and $(\mathcal{D},\bar{b})$, i.e., double exponential in the size of $\mathcal{D}$, S^+ and S^-. Thus, the size of q is at most double exponential in that of $\mathcal{D}$, S^+ and S^-. The lower bound follows by inspection of the proof of Theorem 8. ◀

Notice that this establishes a difference with CQ-explanations/definitions, which are at most of exponential size (see Proposition 4).

5 Desynchronizing the direct product

We now look at the other source of complexity for the QBE and definability tests for CQs: The construction of the direct product $\prod_{\bar{a}\in S^+}(\mathcal{D},\bar{a})$. It is a priori not obvious how to define reasonable approximations of this construction with a meaningful theoretical interpretation. As a first step in this direction, we look at a simple idea that has been applied in the study of CQ-definability: We "desynchronize" this direct product and consider each tuple $\bar{a} \in S^+$ in isolation. This leads to the following relaxed test:

- Desynchronized QBE test for CQs: Takes as input a database $\mathcal{D}$ and n-ary relations S^+, S^- over $\mathcal{D}$. It accepts iff for each $\bar{a} \in S^+$ and $\bar{b} \in S^-$ it is the case that $(\mathcal{D},\bar{a}) \not\to (\mathcal{D},\bar{b})$.

Similarly, we define the desynchronized definability test for CQs. Notice that, unlike the previous tests we have presented in the paper, the desynchronized tests do not require any safeness condition (for reasons we explain below).

It follows from [1] that these tests capture the notion of explanations/definitions for the class of *unions* of CQs (UCQs). Recall that a UCQ is a formula Q of the form $\bigvee_{1\leq i\leq m} q_i(\bar{x})$, where the $q_i(\bar{x})$'s are CQs over the same schema. The evaluation $Q(\mathcal{D})$ of Q over database $\mathcal{D}$ corresponds to $\bigcup_{1\leq i\leq m} q_i(\mathcal{D})$. We denote by UCQ the class of UCQs. We then obtain the following:

▶ **Theorem 14** (implicit in [1]). *Consider a database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$. There is a UCQ-explanation for S^+ and S^- over $\mathcal{D}$ if and only if the desynchronized*

QBE test for CQs accepts $\mathcal{D}$, S^+, and S^-. Similarly, for the UCQ*-definitions of S^+ and the desynchronized definability test for CQs.*

In this case, the *canonical* UCQ-explanation/definition corresponds to $Q = \bigcup_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$. This explains why no safeness condition is required on the desynchronized tests, as each pair of the form $(\mathcal{D}, \bar{a})$, for $\bar{a} \in S^+$, is safe by definition. Notice that Q consists of polynomially many CQs of polynomial size. Its evaluation $Q(\mathcal{D})$ over a database $\mathcal{D}$ can thus be computed in single exponential time (as opposed to the double exponential time needed to evaluate the canonical CQ-explanation $\prod_{\bar{a} \in S^+}(\mathcal{D}, \bar{a})$).

It is easy to see that the desynchronization of the direct product reduces the complexity of the general tests from coNEXPTIME to coNP. It follows from [1] that this bound is optimal. As a corollary to Theorem 14 we thus obtain that QBE/definability for UCQs are coNP-complete:

▶ **Proposition 15.** [1] *The following statements hold:*
1. *Deciding whether the desynchronized QBE test for CQs accepts $(\mathcal{D}, S^+, S^-)$ is* coNP*-complete. Similarly, for the desynchronized definability test for CQs.*
2. UCQ-QUERY-BY-EXAMPLE *and* UCQ-DEFINABILITY *are* coNP*-complete.*

5.1 Combining both relaxations

By combining both relaxations (replacing homomorphism tests with relations $\to_k$, for $k > 1$, and desynchronizing direct products) we obtain the *desynchronized k-pebble QBE (resp., definability) test for CQs*. Its definition coincides with that of the desynchronized QBE (resp., definability) test for CQs given above, save that now the homomorphism test $(\mathcal{D}, \bar{a}) \to (\mathcal{D}, \bar{b})$ is replaced by $(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}, \bar{b})$. As is to be expected from the previous charaterizations, this test captures definability by the class of UCQs of bounded treewidth. Formally, let UTW(k) be the class of unions of CQs in TW(k) (for $k \geq 1$). Then:

▶ **Theorem 16.** *Fix $k \geq 1$. Consider a database $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$. There is a* UTW(k)*-explanation for S^+ and S^- over $\mathcal{D}$ if and only if the desynchronized $(k+1)$-pebble QBE test for CQs accepts $\mathcal{D}$, S^+, and S^-. Similarly, for the* UTW(k)*-definitions of S^+ and the desynchronized $(k+1)$-pebble definability test for CQs.*

Furthermore, in case there is a UTW(k)-explanation for S^+ and S^- over $\mathcal{D}$ (resp., a UTW(k)-definition of S^+ over $\mathcal{D}$), then there is one such explanation/definition given by a union of polynomially many CQs in TW(k), each one of which is of at most exponential size.

Interestingly, the combination of both relaxations yields tractability for the QBE test. In contrast, the definability test remains coNP-complete. The difference lies on the fact that the QBE test only needs to perform a polynomial number of tests of the form $(\mathcal{D}, \bar{a}) \to_k (\mathcal{D}, \bar{b})$ for each $\bar{a} \in S^+$ (one for each tuple $\bar{b} \in S^-$), while the definability test needs to perform exponentially many such tests (one for each tuple $\bar{b}$ outside S^+). Then:

▶ **Proposition 17.** *The following statements hold:*
1. *Deciding whether the desynchronized k-pebble QBE test for CQs accepts $(\mathcal{D}, S^+, S^-)$ can be solved in polynomial time for each fixed $k > 1$. As a consequence,* UTW(k)-QUERY-BY-EXAMPLE *is in polynomial time for each fixed $k \geq 1$.*
2. *If a* UTW(k)*-explanation for S^+ and S^- over $\mathcal{D}$ exists, we can compute the evaluation $Q(\mathcal{D})$ of one such explanation Q in exponential time.*
3. *Deciding whether the desynchronized k-pebble definability test for CQs accepts $(\mathcal{D}, S^+)$ is* coNP*-complete for each fixed $k > 1$. As a consequence,* UTW(k)-DEFINABILITY *is* coNP*-complete for each $k \geq 1$.

6 Conjunctive regular path queries

We now switch to study the QBE and definability problems in the context of graph databases. Let Σ be a finite alphabet. Recall that a *graph database* $\mathcal{G} = (V, E)$ over Σ consists of a finite set V of nodes and a set $E \subseteq V \times \Sigma \times V$ of directed edges labeled in Σ (i.e., $(v, a, v') \in E$ represents the fact that there is an a-labeled edge from node v to node v' in $\mathcal{G}$). A *path* in $\mathcal{G}$ is a sequence

$$\eta \;=\; v_0 a_1 v_1 a_2 v_2 \dots v_{k-1} a_k v_k, \qquad \text{for } k \geq 0,$$

such that $(v_{i-1}, a_i, v_i) \in E$ for each $1 \leq i \leq k$. The *label* of η, denoted $\mathsf{label}(\eta)$, is the word $a_1 a_2 \dots a_k$ in Σ^*. Notice that v is a path for each node $v \in V$. The label of such path is the empty word ε.

The basic navigational mechanism for querying graph databases is the class of *regular path queries*, or RPQs (see, e.g., [25, 5]). An RPQ L over alphabet Σ is a regular expression over Σ. The *evaluation* $L(\mathcal{G})$ of L over graph database $\mathcal{G}$ consists of those pairs (v, v') of nodes in $\mathcal{G}$ such that there is a path η in $\mathcal{G}$ from v to v' whose label $\mathsf{label}(\eta)$ satisfies L. The analogue of CQs in the context of graph databases is the class of *conjunctive* RPQs, or CRPQs [8]. Formally, a CRPQ γ over Σ is an expression of the form:

$$\exists \bar{z}(L_1(x_1, y_1) \wedge \dots \wedge L_m(x_m, y_m)),$$

where each L_i is a RPQ over Σ, for $1 \leq i \leq m$, and $\bar{z}$ is a tuple of variables among $\{x_1, y_1, \dots, x_m, y_m\}$. We write $\gamma(\bar{x})$ to denote that $\bar{x}$ is the tuple of free variables of γ. A homomorphism from γ to the graph database $\mathcal{G}$ is a mapping h from $\{x_1, y_1, \dots, x_m, y_m\}$ to the nodes of $\mathcal{G}$, such that $(h(x_i), h(y_i)) \in L_i(\mathcal{G})$ for each $1 \leq i \leq m$. The evaluation $\gamma(\mathcal{G})$ of $\gamma(\bar{x})$ over $\mathcal{G}$ is the set of tuples $h(\bar{x})$ such that h a homomorphism from γ to $\mathcal{G}$. We denote the class of CRPQs by CRPQ.

6.1 The QBE and definability tests for CRPQs

We present QBE/definability tests for CRPQs in the same spirit than the tests for CQs, save that we now use a notion of *strong homomorphism* from a product $\prod_{1 \leq i \leq n} \mathcal{G}_i$ of directed graphs to a single directed graph $\mathcal{G}$. This notion preserves, in a precise sense defined below, the languages defined by pairs of nodes in $\prod_{1 \leq i \leq n} \mathcal{G}_i$. Interestingly, these tests yield a coNEXPTIME upper bound for the QBE/definability problems for CRPQs, which improves the EXPSPACE upper bound from [1]. In conclusion, QBE/definability for CRPQs is no more difficult than for CQs.

We start with some notation. Let v and v' be nodes in a graph database $\mathcal{G}$. We define the following language in Σ^*:

$$L^{\mathcal{G}}_{v,v'} \;:=\; \{\mathsf{label}(\eta) \mid \eta \text{ is a path in } \mathcal{G} \text{ from } v \text{ to } v'\}.$$

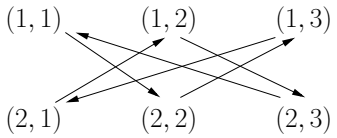
Moreover, if $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$ are graph databases over Σ, their direct product $\mathcal{G}_1 \otimes \mathcal{G}_2$ is the graph database (V, E) such that $V = V_1 \times V_2$ and there is an a-labeled edge in E from node (v_1, v_2) to node (v'_1, v'_2) if and only if $(v_1, a, v_2) \in E_1$ and $(v'_1, a, v'_2) \in E_2$.

Let then $\mathcal{G}_1, \dots, \mathcal{G}_n$ and $\mathcal{G}$ be graph databases over Σ. A *strong homomorphism* from $\prod_{1 \leq i \leq n} \mathcal{G}_i$ to $\mathcal{G}$ is a mapping h from the nodes of $\prod_{1 \leq i \leq n} \mathcal{G}_i$ to the nodes of $\mathcal{G}$ such that for each pair $\bar{v} = (v_1, \dots, v_n)$ and $\bar{v}' = (v'_1, \dots, v'_n)$ of nodes in $\prod_{1 \leq i \leq n} \mathcal{G}_i$, it is the case that:

$$L^{\mathcal{G}_i}_{v_i, v'_i} \;\subseteq\; L^{\mathcal{G}}_{h(\bar{v}), h(\bar{v}')}, \qquad \text{for some coordinate } i \text{ with } 1 \leq i \leq n.$$

We write $\prod_{1\leq i\leq n}\mathcal{G}_i \Rightarrow \mathcal{G}$ when there is a strong homomorphism h from $\prod_{1\leq i\leq n}\mathcal{G}_i$ to $\mathcal{G}$. Note that in this case, h must also be a (usual) homomorphism from $\prod_{1\leq i\leq n}\mathcal{G}_i$ to $\mathcal{G}$, i.e., $\prod_{1\leq i\leq n}\mathcal{G}_i \Rightarrow \mathcal{G}$ implies $\prod_{1\leq i\leq n}\mathcal{G}_i \to \mathcal{G}$. The next example shows that the converse does not hold in general:

▶ **Example 18.** Let $\vec{C}_n$ be the directed cycle of length n over $\{1, 2, \ldots, n\}$. We assume $\vec{C}_n$ to be represented as a graph database over the unary alphabet $\Sigma = \{a\}$. We then have that $\vec{C}_2 \otimes \vec{C}_3 \to \vec{C}_6$, since $\vec{C}_2 \otimes \vec{C}_3$ is isomorphic to $\vec{C}_6$ as shown below (we omit the labels):



On the other hand, $\vec{C}_2 \otimes \vec{C}_3 \not\Rightarrow \vec{C}_6$. To see this, take e.g. the homomorphism h defined as

$$\{(1,1) \mapsto 1,\ (2,2) \mapsto 2,\ (1,3) \mapsto 3,\ (2,1) \mapsto 4,\ (1,2) \mapsto 5,\ (2,3) \mapsto 6\}.$$

This is not a strong homomorphism as witnessed by the pair $(1, 1)$ and $(2, 2)$. Indeed, we have that:

$$\big(h(1,1) = 1 \ \text{ and } \ h(2,2) = 2\big) \ \text{ but } \ \big(L^{\vec{C}_2}_{1,2} \not\subseteq L^{\vec{C}_6}_{1,2} \ \text{ and } \ L^{\vec{C}_3}_{1,2} \not\subseteq L^{\vec{C}_6}_{1,2}.\big)$$

The reason is that $aaa \in L^{\vec{C}_2}_{1,2}$, $aaaa \in L^{\vec{C}_3}_{1,2}$, but none of these words is in $L^{\vec{C}_6}_{1,2}$. The same holds for any homomorphism $h : \vec{C}_2 \otimes \vec{C}_3 \to \vec{C}_6$. ◀

If $(\mathcal{G}_1, \bar{a}_1), \ldots, (\mathcal{G}_n, \bar{a}_n)$ and $(\mathcal{G}, \bar{b})$ are graph databases with distinguished tuple of elements, then we write $\prod_{1\leq i\leq n}(\mathcal{G}_i, \bar{a}_i) \Rightarrow (\mathcal{G}, \bar{b})$ if there is a strong homomorphism h from $\prod_{1\leq i\leq n}\mathcal{G}_i$ to $\mathcal{G}$ such that $h(\bar{a}_1 \otimes \cdots \otimes \bar{a}_n) = \bar{b}$. Next we present our tests for CRPQs:

- QBE test for CRPQs: Takes as input a graph database $\mathcal{G}$ and n-ary relations S^+ and S^- over $\mathcal{G}$. It accepts if and only if $\prod_{\bar{a}\in S^+}(\mathcal{G}, \bar{a}) \not\Rightarrow (\mathcal{G}, \bar{b})$ for each tuple $\bar{b} \in S^-$.
- Definability test for CRPQs: Takes as input a graph database $\mathcal{G}$ and an n-ary relation S^+ over $\mathcal{G}$. It accepts if and only if $\prod_{\bar{a}\in S^+}(\mathcal{G}, \bar{a}) \not\Rightarrow (\mathcal{G}, \bar{b})$ for each n-ary tuple $\bar{b} \notin S^+$.

As it turns out, our tests characterize the non-existence of CRPQ-explanations/definitions. (Notice that unlike Proposition 2, we need no safety conditions on QBE/definability tests for CRPQs for this characterization to hold).

▶ **Theorem 19.** *The following hold:*

1. *Let $\mathcal{G}$ be a database and S^+, S^- relations over $\mathcal{G}$. There is a* CRPQ*-explanation for S^+ and S^- over $\mathcal{G}$ if and only if the QBE test for CRPQs accepts $\mathcal{G}$, S^+, and S^-.*
2. *Let $\mathcal{G}$ be a database and S^+ a relation over $\mathcal{G}$. There is a* CRPQ*-definition for S^+ over $\mathcal{G}$ if and only if the definability test for CRPQs accepts $\mathcal{G}$ and S^+.*

Since containment of regular languages can be checked in polynomial space [21], it is straightforward to check that both tests can be carried out in coNEXPTIME. Thus:

▶ **Theorem 20.** CRPQ-QUERY-BY-EXAMPLE *and* CRPQ-DEFINIBILITY *are in* coNEXPTIME.

Whether these problems are complete for coNEXPTIME is left as an open question.

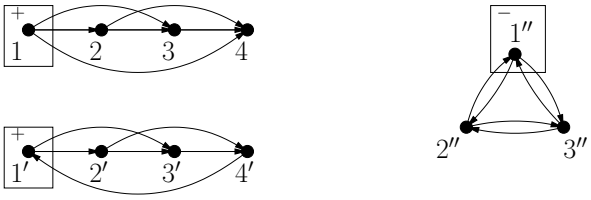


Figure 1 The graph database $\mathcal{G}$ from Example 21.

CRPQ vs UCQ explanations. It is easy to see that if there is a CRPQ-explanation for S^+ and S^- over $\mathcal{G}$, then there is also a UCQ-explanation [1]. One may wonder then if QBE for CRPQs and UCQs coincide. If this was the case, we would directly obtain a coNP upper bound for CRPQ-QUERY-BY-EXAMPLE from Proposition 15 (which establishes that UCQ-QUERY-BY-EXAMPLE is in coNP). The next example shows that this is not the case:

▶ **Example 21.** Consider the graph database $\mathcal{G}$ over $\Sigma = \{a\}$ given by the three connected components depicted in Figure 1 (we omit the labels). Let $S^+ = \{1, 1'\}$ and $S^- = \{1''\}$. Clearly, $(\mathcal{G}, 1) \not\to (\mathcal{G}, 1'')$ and $(\mathcal{G}, 1') \not\to (\mathcal{G}, 1'')$, since the underlying graph of each component on the left-hand side is a clique of size 4, while the one on the right-hand side is a clique of size 3. It follows that there is a UCQ-explanation for S^+ and S^- over $\mathcal{G}$. On the other hand, a straightforward construction shows that $(\mathcal{G}, 1) \otimes (\mathcal{G}, 1') \Rightarrow (\mathcal{G}, 1'')$. The intuition is that, since $(4', 1')$ and $(1, 4)$ have opposite direction, they do not synchronize in the product and, thus, the product does not contain a clique of size 4. We conclude that there is no CRPQ-explanation for S^+ and S^- over $\mathcal{G}$. ◀

6.2 Relaxing the QBE and definability tests for CRPQs

In this section, we develop relaxations of the tests for CRPQs based on the ones we studied for CQs in the previous sections. Let us start by observing that desynchronizing the direct product trivializes the problem in this case: In fact, as expected the *desynchronized QBE/definability tests for CRPQs* characterize QBE/definability for the *unions* of CRPQs (UCRPQ). It is known, on the other hand, that QBE/definability for UCRPQ and UCQ coincide [1]. The results then follow directly from the ones obtained in Section 5 for UCQs. In particular, UCRPQ-QUERY-BY-EXAMPLE and UCRPQ-DEFINABILITY are coNP-complete.
+

We thus concentrate on the most interesting case, which is the relaxation of the homomorphism tests. In order to approximate the strong homomorphism test, we consider a variant of the existential pebble game. Fix $k > 1$. Let $(\mathcal{G}_1, \bar{a}_1), \ldots, (\mathcal{G}_n, \bar{a}_n)$ and $(\mathcal{G}, \bar{b})$ be graph databases over Σ with distinguished tuples of elements. We define $\bar{a} := \bar{a}_1 \otimes \cdots \otimes \bar{a}_n$. The *strong existential k-pebble game* on $\prod_{1 \leq i \leq n}(\mathcal{G}_i, \bar{a}_i)$ and $(\mathcal{G}, \bar{b})$ is played as the existential k-pebble game on $\prod_{1 \leq i \leq n}(\mathcal{G}_i, \bar{a}_i)$ and $(\mathcal{G}, \bar{b})$, but now, at each round, if $c_1, \ldots, c_k$ and $d_1, \ldots, d_k$ are the elements covered by pebbles on $\prod_{1 \leq i \leq n} \mathcal{G}_i$ and $\mathcal{G}$, respectively, then the duplicator needs to ensure that $((c_1, \ldots, c_k, \bar{a}), (d_1, \ldots, d_k, \bar{b}))$ is a *strong partial homomorphism* from $\prod_{1 \leq i \leq n} \mathcal{G}_i$ and $\mathcal{G}$. This means that for every pair $\bar{v} = (v_1, \ldots, v_n)$ and $\bar{v}' = (v'_1, \ldots, v'_n)$ of nodes in $\prod_{1 \leq i \leq n} \mathcal{G}_i$ that appear in $(c_1, \ldots, c_k, \bar{a})$, if u and u' are the elements in $(d_1, \ldots, d_k, \bar{b})$ that correspond to $\bar{v}$ and $\bar{v}'$, respectively, then:

$$L^{\mathcal{G}_i}_{v_i, v'_i} \subseteq L^{\mathcal{G}}_{u,u'}, \quad \text{for some coordinate } i \text{ with } 1 \leq i \leq n.$$

We write $\prod_{1 \leq i \leq n}(\mathcal{G}_i, \bar{a}_i) \Rightarrow_k (\mathcal{G}, \bar{b})$ if the duplicator has a winning strategy in the strong existential k-pebble game on $\prod_{1 \leq i \leq n}(\mathcal{G}_i, \bar{a}_i)$ and $(\mathcal{G}, \bar{b})$.

By replacing the notion of strong homomorphism $\Rightarrow$ with its approximation $\Rightarrow_k$, for a fixed $k > 1$, we can then define the following relaxed test:

- k-pebble QBE test for CRPQs: Takes as input a graph database $\mathcal{G}$ and n-ary relations S^+ and S^- over $\mathcal{G}$. It accepts iff $\prod_{\bar{a} \in S^+}(\mathcal{G}, \bar{a}) \not\Rightarrow_k (\mathcal{G}, \bar{b})$ for each tuple $\bar{b} \in S^-$.

The k-pebble definability test for CRPQs is defined analogously. As in the case of CQs, these tests characterize the existence of CRPQs-explanations/definitions of treewidth at most k. Formally, the treewidth of a CRPQ $\gamma = \exists \bar{y} \bigwedge_{1 \leq i \leq m} L_i(x_i, y_i)$ is the treewidth of the undirected graph that contains as nodes the existentially quantified variables of γ, i.e., those in $\bar{y}$, and whose set of edges is $\{\{x_i, y_i\} \mid 1 \leq i \leq m,\ x_i \neq y_i\}$. We denote by $\mathsf{TW}_{\text{crpq}}(k)$ the class of CRPQs of treewidth at most k (for $k \geq 1$). Then:

▶ **Theorem 22.** *Fix $k \geq 1$. Consider a database $\mathcal{G}$ and n-ary relations S^+ and S^- over $\mathcal{G}$.*

1. *There is a $\mathsf{TW}_{\text{crpq}}(k)$-explanation for S^+ and S^- over $\mathcal{G}$ if and only if the $(k+1)$-pebble QBE test for CRPQs accepts $\mathcal{G}$, S^+ and S^-.*
2. *There is a $\mathsf{TW}_{\text{crpq}}(k)$-definition for S^+ over $\mathcal{G}$ if and only if the $(k+1)$-pebble definability test for CRPQs accepts $\mathcal{G}$ and S^+.*

Using similar ideas as for the existential k-pebble game, it is possible to prove that the problem of checking whether $\prod_{1 \leq i \leq n}(\mathcal{G}_i, \bar{a}_i) \Rightarrow_k (\mathcal{G}, \bar{b})$, given $(\mathcal{G}_1, \bar{a}_1), \ldots, (\mathcal{G}_n, \bar{a}_n)$ and $(\mathcal{G}, \bar{b})$, can be solved in exponential time for each fixed $k > 1$. We then obtain that the k-pebble QBE/definability tests for CRPQs take exponential time, and from Theorem 22 that $\mathsf{TW}_{\text{crpq}}(k)$-QUERY-BY-EXAMPLE and $\mathsf{TW}_{\text{crpq}}(k)$-DEFINABILITY are in EXPTIME (same than for $\mathsf{TW}(k)$ as stated in Corollary 9). We also obtain an exponential upper bound on the cost of evaluating a $\mathsf{TW}_{\text{crpq}}(k)$-explanation (in case it exists):

▶ **Proposition 23.** *Fix $k \geq 1$. The following statements hold:*

1. *$\mathsf{TW}_{\text{crpq}}(k)$-QUERY-BY-EXAMPLE and $\mathsf{TW}_{\text{crpq}}(k)$-DEFINABILITY are in EXPTIME.*
2. *Moreover, in case that there is a $\mathsf{TW}_{\text{crpq}}(k)$-explanation of S^+ and S^- over $\mathcal{G}$, the evaluation $\gamma(\mathcal{G})$ of one such explanation γ over $\mathcal{G}$ can be computed in exponential time.*

7 Future work

We have left some problems open. The most notable one is determining the precise complexity of QBE/definability for CRPQs (resp., CRPQs of bounded treewidth). We have only obtained upper bounds for these problems that show that they are no more difficult than for CQs, but proving matching lower bounds seems challenging.

An interesting line for future research is studying what to do when no explanation/definition exists for a set of examples. In such cases one might want to compute a query that minimizes the "error", e.g., the number of misclassified examples. We plan to study whether the techniques presented in this paper can be extended to deal with such problems.

Acknowledgements. We are grateful to Leonid Libkin for their helpful comments in earlier versions of the paper and to Timos Antonopoulos for enlightening discussions about the notion of definability. We are also indebted to the reviewers of this article who helped us improving the presentation. In particular, one of the revieweres identified a subtle but important issue with the QBE tests presented in the paper that would have gone unnoticed otherwise. Barceló and Romero are funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. Romero is also funded by a Conicyt PhD scholarship.

References

1 Timos Antonopoulos, Frank Neven, and Frédéric Servais. Definability problems for graph query languages. In *ICDT*, pages 141–152, 2013.
2 Marcelo Arenas and Gonzalo I. Díaz. The exact complexity of the first-order logic definability problem. *ACM Trans. Database Syst.*, 41(2), to 2016.
3 Marcelo Arenas, Gonzalo I. Díaz, and Egor V. Kostylev. Reverse engineering sparql queries. In *WWW*, 2016.
4 Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In *CP*, pages 77–91, 2004.
5 Pablo Barceló. Querying graph databases. In *PODS*, pages 175–188, 2013.
6 Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. Learning path queries on graph databases. In *EDBT*, pages 109–120, 2015.
7 Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24, 2016.
8 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.
9 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.
10 Sara Cohen and Yaacov Y. Weiss. Learning tree patterns from example graphs. In *ICDT*, pages 127–143, 2015.
11 Mariano P. Consens and Alberto O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *PODS*, pages 404–416, 1990.
12 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP*, pages 310–326, 2002.
13 Rina Dechter. From local to global consistency. *Artif. Intell.*, 55(1):87–108, 1992.
14 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
15 Martin Grohe. Equivalence in finite-variable logics is complete for polynomial time. *Combinatorica*, 19(4):507–532, 1999.
16 Phokion G. Kolaitis and Jonathan Panttaja. On the complexity of existential pebble games. In *CSL*, pages 314–329, 2003.
17 Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of datalog: Tools and a case study. *J. Comput. Syst. Sci.*, 51(1):110–134, 1995.
18 Phokion G. Kolaitis and Moshe Y. Vardi. A game-theoretic approach to constraint satisfaction. In *AAAI*, pages 175–181, 2000.
19 Hao Li, Chee-Yong Chan, and David Maier. Query from examples: An iterative, data-driven approach to query construction. *PVLDB*, 8(13):2158–2169, 2015.
20 Slawek Staworko and Piotr Wieczorek. Characterizing XML twig queries with examples. In *ICDT*, pages 144–160, 2015.
21 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.
22 Balder ten Cate and Víctor Dalmau. The product homomorphism problem and applications. In *ICDT*, pages 161–176, 2015.
23 Quoc Trung Tran, Chee Yong Chan, and Srinivasan Parthasarathy. Query reverse engineering. *VLDB J.*, 23(5):721–746, 2014.
24 Ross Willard. Testing expressibility is hard. In *CP*, pages 9–23, 2010.
25 Peter T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.
26 Meihui Zhang, Hazem Elmeleegy, Cecilia M. Procopiuc, and Divesh Srivastava. Reverse engineering complex join queries. In *SIGMOD*, pages 809–820, 2013.

Answering FO+MOD Queries Under Updates on Bounded Degree Databases*

Christoph Berkholz[1], Jens Keppeler[2], and Nicole Schweikardt[3]

1 **Humboldt-Universität zu Berlin, Berlin, Germany**
`berkholz@informatik.hu-berlin.de`
2 **Humboldt-Universität zu Berlin, Berlin, Germany**
`keppelej@informatik.hu-berlin.de`
3 **Humboldt-Universität zu Berlin, Berlin, Germany**
`schweikn@informatik.hu-berlin.de`

Abstract

We investigate the query evaluation problem for fixed queries over fully dynamic databases, where tuples can be inserted or deleted. The task is to design a dynamic algorithm that immediately reports the new result of a fixed query after every database update.

We consider queries in first-order logic (FO) and its extension with modulo-counting quantifiers (FO+MOD), and show that they can be efficiently evaluated under updates, provided that the dynamic database does not exceed a certain degree bound.

In particular, we construct a data structure that allows to answer a Boolean FO+MOD query and to compute the size of the query result within constant time after every database update. Furthermore, after every update we are able to immediately enumerate the new query result with constant delay between the output tuples. The time needed to build the data structure is linear in the size of the database.

Our results extend earlier work on the evaluation of first-order queries on static databases of bounded degree and rely on an effective Hanf normal form for FO+MOD recently obtained by Heimberg, Kuske, and Schweikardt (LICS 2016).

1998 ACM Subject Classification H.2.4 [Systems] Relational Databases, Query Processing, H.2.3 [Languages] Query Languages, F.1.2 [Modes of Computation] Interactive and Reactive Computation

Keywords and phrases dynamic databases, query enumeration, counting problem, first-order logic with modulo-counting quantifiers, Hanf locality

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.8

1 Introduction

Query evaluation is a fundamental task in databases, and a vast amount of literature is devoted to the complexity of this problem. In this paper we study query evaluation on relational databases in the "dynamic setting", where the database may be updated by inserting or deleting tuples. In this setting, an evaluation algorithm receives a query φ and an initial database D and starts with a preprocessing phase that computes a suitable data structure to represent the result of evaluating φ on D. After every database update, the data structure is updated so that it represents the result of evaluating φ on the updated

* We acknowledge the financial support by the German Research Foundation DFG under grant SCHW 837/5-1.

database. The data structure shall be designed in such a way that it quickly provides the query result, preferably in constant time (i. e., independent of the database size). We focus on the following flavours of query evaluation.

- *Testing:* Decide whether a given tuple $\overline{a}$ is contained in $\varphi(D)$.
- *Counting:* Compute $|\varphi(D)|$ (i.e., the number of tuples that belong to $\varphi(D)$).
- *Enumeration:* Enumerate $\varphi(D)$ with a bounded delay between the output tuples.

Here, as usual, $\varphi(D)$ denotes the k-ary relation obtained by evaluating a k-ary query φ on a relational database D. For *Boolean* queries, all three tasks boil down to

- *Answering:* Decide if $\varphi(D) \neq \emptyset$.

Compared to the *dynamic descriptive complexity* framework introduced by Patnaik and Immerman [17], which focuses on the *expressive power* of first-order logic on dynamic databases and has led to a rich body of literature (see [18] for a survey), we are interested in the *complexity* of query evaluation. The query language studied in this paper is FO+MOD, the extension of first-order logic FO with modulo-counting quantifiers of the form $\exists^{i \bmod m} x\, \psi$, expressing that the number of witnesses x that satisfy ψ is congruent to i modulo m. FO+MOD can be viewed as a subclass of SQL that properly extends the relational algebra.

Following [2], we say that a query evaluation algorithm is efficient if the update time is either constant or at most polylogarithmic ($\log^c n$) in the size of the database. As a consequence, efficient query evaluation in the dynamic setting is only possible if the static problem (i.e., the setting without database updates) can be solved in linear or pseudo-linear ($n^{1+\varepsilon}$) time. Since this is not always possible, we provide a short overview on known results about first-order query evaluation on static databases and then proceed by discussing our results in the dynamic setting.

First-order query evaluation on static databases. The problem of deciding whether a given database D satisfies a FO-sentence φ is AW[$*$]-complete (parameterised by $\|\varphi\|$) and it is therefore generally believed that the evaluation problem cannot be solved in time $f(\|\varphi\|)\|D\|^c$ for any computable f and constant c (here, $\|\varphi\|$ and $\|D\|$ denote the size of the query and the database, respectively). For this reason, a long line of research focused on increasing classes of sparse instances ranging from databases of *bounded degree* [19] (where every domain element occurs only in a constant number of tuples in the database) to classes that are *nowhere dense* [9]. In particular, Boolean first-order queries can be evaluated on classes of databases of bounded degree in linear time $f(\|\varphi\|)\|D\|$, where the constant factor $f(\|\varphi\|)$ is 3-fold exponential in $\|\varphi\|$ [19, 7]. As a matter of fact, Frick and Grohe [7] showed that the 3-fold exponential blow-up in terms of the query size is unavoidable assuming FPT $\neq$ AW[$*$].

Durand and Grandjean [5] and Kazana and Segoufin [11] considered the task of enumerating the result of a k-ary first-order query on bounded degree databases and showed that after a linear time preprocessing phase the query result can be enumerated with constant delay. This result was later extended to classes of databases of bounded expansion [12]. Kazana and Segoufin [12] also showed that counting the number of result tuples of a k-ary first-order query on databases of bounded expansion (and hence also on databases of bounded degree) can be done in time $f(\|\varphi\|)\|D\|$. In [6] an analogous result was obtained for classes of databases of low degree (i. e., degree at most $\|D\|^{o(1)}$) and pseudo-linear time $f(\|\varphi\|)\|D\|^{1+\varepsilon}$; the paper also presented an algorithm for enumerating the query results with constant delay after pseudo-linear time preprocessing.

Our contribution. We extend the known linear time algorithms for first-order logic on classes of databases of bounded degree to the more expressive query language FO+MOD.

Moreover, and more importantly, we lift the tractability to the dynamic setting and show that the result of FO and FO+MOD-queries can be maintained with constant update time. In particular, we obtain the following results. Let φ be a fixed k-ary FO+MOD-query and d a fixed degree bound on the databases under consideration. Given an initial database D, we construct in linear time $f(\|\varphi\|, d)\|D\|$ a data structure that can be updated in constant time $f(\|\varphi\|, d)$ when a tuple is inserted into or deleted from a relation of D. After each update the data structure allows to

- immediately answer φ on D if φ is a Boolean query (Theorem 4.1),
- test for a given tuple $\overline{a}$ whether $\overline{a} \in \varphi(D)$ in time $\mathcal{O}(k^2)$ (Theorem 6.1),
- immediately output the number of result tuples $|\varphi(D)|$ (Theorem 8.1), and
- enumerate all tuples $(a_1, \ldots, a_k) \in \varphi(D)$ with $\mathcal{O}(k^3)$ delay (Theorem 9.4).

For fixed d, the parameter function $f(\|\varphi\|, d)$ is 3-fold exponential in terms of the query size, which is (by Frick and Grohe [7]) optimal assuming FPT $\neq$ AW[$*$].

Outline. Our dynamic query evaluation algorithm crucially relies on the locality of FO+MOD and in particular an effective Hanf normal form for FO+MOD on databases of bounded degree recently obtained by Heimberg, Kuske, and Schweikardt [10]. After some basic definitions in Section 2 we briefly state their result in Section 3 and obtain a dynamic algorithm for Boolean FO+MOD-queries in Section 4. After some preparations for non-Boolean queries in Section 5, we present the algorithm for testing in Section 6. In Section 7 we reduce the task of counting and enumerating FO+MOD-queries in the dynamic setting to the problem of counting and enumerating independent sets in graphs of bounded degree. We use this reduction to provide efficient dynamic counting and enumeration algorithms in Section 8 and 9, respectively, and we conclude in Section 10. Due to space constraints some technical proofs are deferred to the full version of the paper [3].

2 Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers and let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$ and $[n] := \{1, \ldots, n\}$ for all $n \in \mathbb{N}_{\geqslant 1}$. By 2^M we denote the power set of a set M. For a partial function f we write $\mathrm{dom}(f)$ and $\mathrm{codom}(f)$ for the domain and the codomain of f, respectively.

Databases. We fix a countably infinite set $\mathbf{dom}$, the *domain* of potential database entries. Elements in $\mathbf{dom}$ are called *constants*. A *schema* is a finite set σ of relation symbols, where each $R \in \sigma$ is equipped with a fixed *arity* $\mathrm{ar}(R) \in \mathbb{N}_{\geqslant 1}$. Let us fix a schema $\sigma = \{R_1, \ldots, R_{|\sigma|}\}$. A *database* D of schema σ (σ-db, for short), is of the form $D = (R_1^D, \ldots, R_{|\sigma|}^D)$, where each R_i^D is a finite subset of $\mathbf{dom}^{\mathrm{ar}(R_i)}$. The *active domain* $\mathrm{adom}(D)$ of D is the smallest subset A of $\mathbf{dom}$ such that $R_i^D \subseteq A^{ar(R_i)}$ for each R_i in σ.

The *Gaifman graph* of a σ-db D is the undirected simple graph $G^D = (V, E)$ with vertex set $V := \mathrm{adom}(D)$, where there is an edge between vertices u and v whenever $u \neq v$ and there are $R \in \sigma$ and $(a_1, \ldots, a_{\mathrm{ar}(R)}) \in R^D$ such that $u, v \in \{a_1, \ldots, a_{\mathrm{ar}(R)}\}$. A σ-db D is called *connected* if its Gaifman graph G^D is connected; the *connected components* of D are the connected components of G^D. The *degree* of a database D is the degree of its Gaifman graph G^D, i.e., the maximum number of neighbours of a node of G^D. Throughout this paper we fix a number $d \in \mathbb{N}$ and restrict attention to databases of degree at most d.

Updates. We allow to update a given database of schema σ by inserting or deleting tuples as follows (note that both types of commands may change the database's active domain and

the database's degree). A *deletion* command is of the form $\mathsf{delete}\, R(a_1, \ldots, a_r)$ for $R \in \sigma$, $r = \mathrm{ar}(R)$, and $a_1, \ldots, a_r \in \mathbf{dom}$. When applied to a σ-db D, it results in the updated σ-db D' with $R^{D'} = R^D \setminus \{(a_1, \ldots, a_r)\}$ and $S^{D'} = S^D$ for all $S \in \sigma \setminus \{R\}$.

An *insertion* command is of the form $\mathsf{insert}\, R(a_1, \ldots, a_r)$ for $R \in \sigma$, $r = \mathrm{ar}(R)$, and $a_1, \ldots, a_r \in \mathbf{dom}$. When applied to a σ-db D in the unrestricted setting, it results in the updated σ-db D' with $R^{D'} = R^D \cup \{(a_1, \ldots, a_r)\}$ and $S^{D'} = S^D$ for all $S \in \sigma \setminus \{R\}$. In this paper, we restrict attention to databases of degree at most d. Therefore, when applying an insertion command to a σ-db D of degree $\leqslant d$, the command is carried out only if the resulting database D' still has degree $\leqslant d$; otherwise D remains unchanged and instead of carrying out the insertion command, an error message is returned.

Queries. We fix a countably infinite set $\mathbf{var}$ of *variables*. We consider the extension FO+MOD of first-order logic FO with modulo-counting quantifiers. For a fixed schema σ, the set FO+MOD$[\sigma]$ is built from atomic formulas of the form $x_1{=}x_2$ and $R(x_1, \ldots, x_{\mathrm{ar}(R)})$, for $R \in \sigma$ and variables $x_1, x_2, \ldots, x_{\mathrm{ar}(R)} \in \mathbf{var}$, and is closed under Boolean connectives $\neg$, $\wedge$, existential first-order quantifiers $\exists x$, and modulo-counting quantifiers $\exists^{i \bmod m} x$, for a variable $x \in \mathbf{var}$ and integers $i, m \in \mathbb{N}$ with $m \geqslant 2$ and $i < m$. The intuitive meaning of a formula of the form $\exists^{i \bmod m} x\, \psi$ is that the number of witnesses x that satisfy ψ is congruent i modulo m. As usual, $\forall x$, $\vee$, $\rightarrow$, $\leftrightarrow$ will be used as abbreviations when constructing formulas. It will be convenient to add the quantifier $\exists^{\geqslant m} x$, for $m \in \mathbb{N}_{\geqslant 1}$; a formula of the form $\exists^{\geqslant m} x\, \psi$ expresses that the number of witnesses x which satisfy ψ is $\geqslant m$. This quantifier is just syntactic sugar an does not increase the expressive power of FO+MOD.

The *quantifier rank* $\mathrm{qr}(\varphi)$ of a FO+MOD-formula φ is the maximum nesting depth of quantifiers that occur in φ. By $\mathrm{free}(\varphi)$ we denote the set of all *free variables* of φ, i.e., all variables x that have at least one occurrence in φ that is not within a quantifier of the form $\exists x$, $\exists^{\geqslant m} x$, or $\exists^{i \bmod m} x$. A *sentence* is a formula φ with $\mathrm{free}(\varphi) = \emptyset$.

An *assignment* for φ in a σ-db D is a partial mapping α from $\mathbf{var}$ to $\mathrm{adom}(D)$, where $\mathrm{free}(\varphi) \subseteq \mathrm{dom}(\alpha)$. We write $(D, \alpha) \models \varphi$ to indicate that φ is satisfied when evaluated in D with respect to *active domain semantics* while interpreting every free occurrence of a variable x with the constant $\alpha(x)$. Recall from [1] that "active domain semantics" means that quantifiers are evaluated with respect to the database's active domain. In particular, $(D, \alpha) \models \exists x\, \psi$ iff there exists an $a \in \mathrm{adom}(D)$ such that $(D, \alpha\frac{a}{x}) \models \psi$, where $\alpha\frac{a}{x}$ is the assignment α' with $\alpha'(x) = a$ and $\alpha'(y) = \alpha(y)$ for all $y \in \mathrm{dom}(\alpha) \setminus \{x\}$. Accordingly, $(D, \alpha) \models \exists^{\geqslant m} x\, \psi$ iff $\big|\{\, a \in \mathrm{adom}(D) : (D, \alpha\frac{a}{x}) \models \psi \,\}\big| \geqslant m$, and $(D, \alpha) \models \exists^{i \bmod m} x\, \psi$ iff $\big|\{\, a \in \mathrm{adom}(D) : (D, \alpha\frac{a}{x}) \models \psi \,\}\big| \equiv i \bmod m$.

A *k-ary* FO+MOD *query of schema σ* is of the form $\varphi(x_1, \ldots, x_k)$ where $k \in \mathbb{N}$, $\varphi \in$ FO+MOD$[\sigma]$, and $\mathrm{free}(\varphi) \subseteq \{x_1, \ldots, x_k\}$. We will often assume that the tuple $(x_1, \ldots, x_k)$ is clear from the context and simply write φ instead of $\varphi(x_1, \ldots, x_k)$ and $\big(D, (a_1, \ldots, a_k)\big) \models \varphi$ instead of $\big(D, \frac{a_1, \ldots, a_k}{x_1, \ldots, x_k}\big) \models \varphi$, where $\frac{a_1, \ldots, a_k}{x_1, \ldots, x_k}$ denotes the assignment α with $\alpha(x_i) = a_i$ for all $i \in [k]$. When evaluated in a σ-db D, the k-ary query $\varphi(x_1, \ldots, x_k)$ yields the k-ary relation

$$\varphi(D) \quad := \quad \Big\{\, (a_1, \ldots, a_k) \in \mathrm{adom}(D)^k \;:\; \big(D, \tfrac{a_1, \ldots, a_k}{x_1, \ldots, x_k}\big) \models \varphi \,\Big\}.$$

Boolean queries are k-ary queries with $k = 0$. As usual, for Boolean queries we will write $\varphi(D) = \mathtt{no}$ instead of $\varphi(D) = \emptyset$, and $\varphi(D) = \mathtt{yes}$ instead of $\varphi(D) \neq \emptyset$; and we write $D \models \varphi$ to indicate that $(D, \alpha) \models \varphi$ for any assignment α.

Sizes and Cardinalities. The *size* $\|\sigma\|$ of a schema σ is the sum of the arities of its relation symbols. The size $\|\varphi\|$ of an FO+MOD query φ of schema σ is the length of φ when

viewed as a word over the alphabet $\sigma \cup \mathbf{var} \cup \mathbb{N} \cup \{=, \wedge, \neg, \exists, \mathsf{mod}, (,)\}$. For a k-ary query $\varphi(x_1, \ldots, x_k)$ and a σ-db D, the *cardinality of the query result* is the number $|\varphi(D)|$ of tuples in $\varphi(D)$. The *cardinality* $|D|$ of a σ-db D is defined as the number of tuples stored in D, i.e., $|D| := \sum_{R \in \sigma} |R^D|$. The *size* $\|D\|$ of D is defined as $\|\sigma\| + |\mathrm{adom}(D)| + \sum_{R \in \sigma} \mathrm{ar}(R) \cdot |R^D|$ and corresponds to the size of a reasonable encoding of D.

Dynamic Algorithms for Query Evaluation. We adopt the framework for dynamic algorithms for query evaluation of [2]; the next paragraphs are taken almost verbatim from [2]. Following [4], we use Random Access Machines (RAMs) with $\mathcal{O}(\log n)$ word-size and a uniform cost measure to analyse our algorithms. We will assume that the RAM's memory is initialised to 0. In particular, if an algorithm uses an array, we will assume that all array entries are initialised to 0, and this initialisation comes at no cost (in real-world computers this can be achieved by using the *lazy array initialisation technique*, cf. e.g. [16]). A further assumption is that for every fixed dimension $k \in \mathbb{N}_{\geqslant 1}$ we have available an unbounded number of k-ary arrays $\mathtt{A}$ such that for given $(n_1, \ldots, n_k) \in \mathbb{N}^k$ the entry $\mathtt{A}[n_1, \ldots, n_k]$ at position $(n_1, \ldots, n_k)$ can be accessed in constant time.[1]

Our algorithms will take as input a k-ary FO+MOD-query $\varphi(x_1, \ldots, x_k)$, a parameter d, and a σ-db D_0 of degree $\leqslant d$. For all query evaluation problems considered in this paper, we aim at routines **preprocess** and **update** which achieve the following.

Upon input of $\varphi(x_1, \ldots, x_k)$ and D_0, **preprocess** builds a data structure $\mathtt{D}$ which represents D_0 (and which is designed in such a way that it supports the evaluation of φ on D_0). Upon input of a command $\mathsf{update}\ R(a_1, \ldots, a_r)$ (with $\mathsf{update} \in \{\mathsf{insert}, \mathsf{delete}\}$), calling **update** modifies the data structure $\mathtt{D}$ such that it represents the updated database D. The *preprocessing time* t_p is the time used for performing **preprocess**; the *update time* t_u is the time used for performing an **update**. In this paper, t_u will be independent of the size of the current database D. By **init** we denote the particular case of the routine **preprocess** upon input of a query $\varphi(x_1, \ldots, x_k)$ and the *empty* database $D_\emptyset$ (where $R^{D_\emptyset} = \emptyset$ for all $R \in \sigma$). The *initialisation time* t_i is the time used for performing **init**. In all dynamic algorithms presented in this paper, the **preprocess** routine for input of $\varphi(x_1, \ldots, x_k)$ and D_0 will carry out the **init** routine for $\varphi(x_1, \ldots, x_k)$ and then perform a sequence of $|D_0|$ update operations to insert all the tuples of D_0 into the data structure. Consequently, $t_p = t_i + |D_0| \cdot t_u$.

In the following, D will always denote the database that is currently represented by the data structure $\mathtt{D}$.

To solve the *enumeration problem under updates*, apart from the routines **preprocess** and **update**, we aim at a routine **enumerate** such that calling **enumerate** invokes an enumeration of all tuples (without repetition) that belong to the query result $\varphi(D)$. The *delay* t_d is the maximum time used during a call of **enumerate**

- until the output of the first tuple (or the end-of-enumeration message $\mathtt{EOE}$, if $\varphi(D) = \emptyset$),
- between the output of two consecutive tuples, and
- between the output of the last tuple and the end-of-enumeration message $\mathtt{EOE}$.

To *test* if a given tuple belongs to the query result, instead of **enumerate** we aim at a routine **test** which upon input of a tuple $\overline{a} \in \mathbf{dom}^k$ checks whether $\overline{a} \in \varphi(D)$. The *testing time* t_t is the time used for performing a **test**. To solve the *counting problem under updates*, instead of **enumerate** or **test** we aim at a routine **count** which outputs the cardinality $|\varphi(D)|$ of the query result. The *counting time* t_c is the time used for performing a **count**.

[1] While this can be accomplished easily in the RAM-model, for an implementation on real-world computers one would probably have to resort to replacing our use of arrays by using suitably designed hash functions.

To *answer* a *Boolean* query under updates, instead of **enumerate**, **test**, or **count** we aim at a routine **answer** that produces the answer `yes` or `no` of φ on D. The *answer time* t_a is the time used for performing **answer**. Whenever speaking of a *dynamic algorithm*, we mean an algorithm that has routines **preprocess** and **update** and, depending on the problem at hand, at least one of the routines **answer**, **test**, **count**, and **enumerate**.

Throughout the paper, we often adopt the view of *data complexity* and suppress factors that may depend on the query φ or the degree bound d, but not on the database D. E.g., "linear preprocessing time" means $t_p \leqslant f(\varphi, d) \cdot \|D_0\|$ and "constant update time" means $t_u \leqslant f(\varphi, d)$, for a function f with codomain $\mathbb{N}$. When writing *poly*(n) we mean $n^{\mathcal{O}(1)}$.

3 Hanf Normal Form for FO+MOD

Our algorithms for evaluating FO+MOD queries rely on a decomposition of FO+MOD queries into *Hanf normal form*. To describe this normal form, we need some more notation.

Two formulas φ and ψ of schema σ are called *d-equivalent* (in symbols: $\varphi \equiv_d \psi$) if $\varphi(D) = \psi(D)$ for all σ-dbs D *of degree* $\leqslant d$. For a σ-db D and a set $A \subseteq \mathrm{adom}(D)$ we write $D[A]$ to denote the restriction of D to the domain A, i.e., $R^{D[A]} = \{\overline{a} \in R^D : \overline{a} \in A^{\mathrm{ar}(R)}\}$, for all $R \in \sigma$. For two σ-dbs D and D' and two k-tuples $\overline{a} = (a_1, \ldots, a_k)$ and $\overline{a}' = (a'_1, \ldots, a'_k)$ of elements in $\mathrm{adom}(D)$ and $\mathrm{adom}(D')$, resp., we write $\big(D, \overline{a}\big) \cong \big(D', \overline{a}'\big)$ to indicate that there is an isomorphism[2] π from D to D' that maps a_i to a'_i for all $i \in [k]$.

The *distance* $dist^D(a, b)$ between two elements $a, b \in \mathrm{adom}(D)$ is the minimal length (i.e., the number of edges) of a path from a to b in D's Gaifman graph G^D (if no such path exists, we let $dist^D(a, b) = \infty$; note that $dist^D(a, a) = 0$). For $r \geqslant 0$ and $a \in \mathrm{adom}(D)$, the *r-ball* around a in D is the set $N_r^D(a) := \{b \in \mathrm{adom}(D) : dist^D(a, b) \leqslant r\}$. For a σ-db D and a tuple $\overline{a} = (a_1, \ldots, a_k)$ we let $N_r^D(\overline{a}) := \bigcup_{i \in [k]} N_r^D(a_i)$. The *$r$-neighbourhood* around $\overline{a}$ in D is defined as the σ-db $\mathcal{N}_r^D(\overline{a}) := D[N_r^D(\overline{a})]$.

For $r \geqslant 0$ and $k \geqslant 1$, a *type τ (over σ) with k centres and radius r* (for short: *r-type with k centres*) is of the form $(T, \bar{t})$, where T is a σ-db, $\bar{t} \in \mathrm{adom}(T)^k$, and $\mathrm{adom}(T) = N_r^T(\bar{t})$. The elements in $\bar{t}$ are called the *centres* of τ. For a tuple $\overline{a} \in \mathrm{adom}(D)^k$, the *$r$-type of $\overline{a}$ in D* is defined as the r-type with k centres $\big(\mathcal{N}_r^D(\overline{a}), \overline{a}\big)$.

For a given r-type with k centres $\tau = (T, \bar{t})$ it is straightforward to construct a first-order formula $\mathrm{sph}_\tau(\overline{x})$ (depending on r and τ) with k free variables $\overline{x} = (x_1, \ldots, x_k)$ which expresses that the r-type of $\overline{x}$ is isomorphic to τ, i.e., for every σ-db D and all $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{adom}(D)^k$ we have $\big(D, \overline{a}\big) \models \mathrm{sph}_\tau(\overline{x}) \iff \big(\mathcal{N}_r^D(\overline{a}), \overline{a}\big) \cong \big(T, \bar{t}\big)$. The formula $\mathrm{sph}_\tau(\overline{x})$ is called a *sphere-formula* (over σ and $\overline{x}$); the numbers r and k are called *locality radius* and *arity*, resp., of the sphere-formula.

A *Hanf-sentence* (over σ) is a sentence of the form $\exists^{\geqslant m} x\, \mathrm{sph}_\tau(x)$ or $\exists^{i \bmod m} x\, \mathrm{sph}_\tau(x)$, where τ is an r-type (over σ) with 1 centre, for some $r \geqslant 0$. The number r is called *locality radius* of the Hanf-sentence. A formula in *Hanf normal form* (over σ) is a Boolean combination[3] of sphere-formulas and Hanf-sentences (over σ). The *locality radius* of a formula ψ in Hanf normal form is the maximum of the locality radii of the Hanf-sentences and the sphere-formulas that occur in ψ. The formula is *d-bounded* if all types τ that occur in sphere-formulas or Hanf-sentences of ψ are d-bounded, i.e., T is of degree $\leqslant d$, where $\tau = (T, \bar{t})$. Our query evaluation algorithms for FO+MOD rely on the following result by Heimberg, Kuske, and Schweikardt [10].

---

[2] An *isomorphism* $\pi\colon D \to D'$ is a bijection from $\mathrm{adom}(D)$ to $\mathrm{adom}(D')$ with $(b_1, \ldots, b_r) \in R^D \iff (\pi(b_1), \ldots, \pi(b_r)) \in R^{D'}$ for all $R \in \sigma$, for $r := \mathrm{ar}(R)$, and for all $b_1, \ldots, b_r \in \mathrm{adom}(D)$.

[3] Throughout this paper, whenever we speak of *Boolean combinations* we mean *finite* Boolean combinations.

▶ **Theorem 3.1** ([10]). *There is an algorithm which receives as input a degree bound $d \in \mathbb{N}$ and a* FO+MOD$[\sigma]$*-formula φ, and constructs a d-equivalent formula ψ in Hanf normal form (over σ) with the same free variables as φ. For any $d \geqslant 2$, the formula ψ is d-bounded and has locality radius $\leqslant 4^{\mathrm{qr}(\varphi)}$, and the algorithm's runtime is $2^{d^{2^{\mathcal{O}(\|\varphi\|+\|\sigma\|)}}}$.*

The first step of all our query evaluation algorithms is to use Theorem 3.1 to transform a given query $\varphi(\overline{x})$ into a d-equivalent query $\psi(\overline{x})$ in Hanf normal form. The following lemma summarises easy facts that are useful for evaluating the sphere-formulas that occur in ψ.

▶ **Lemma 3.2.** *Let $d \geqslant 2$ and let D be a σ-db of degree $\leqslant d$. Let $r \geqslant 0$, $k \geqslant 1$, and $\overline{a} = (a_1, \ldots, a_k) \in \mathrm{adom}(D)$.*

(a) $\left|N_r^D(\overline{a})\right| \leqslant k \sum_{i=0}^{r} d^i \leqslant k d^{r+1}$.

(b) *Given D and $\overline{a}$, the r-neighbourhood $\mathcal{N}_r^D(\overline{a})$ can be computed in time $\left(kd^{r+1}\right)^{\mathcal{O}(\|\sigma\|)}$.*

(c) *$\mathcal{N}_r^D(a_1, a_2)$ is connected if and only if $dist^D(a_1, a_2) \leqslant 2r+1$.*

(d) *If $\mathcal{N}_r^D(\overline{a})$ is connected, then $N_r^D(\overline{a}) \subseteq N_{r+(k-1)(2r+1)}^D(a_i)$, for all $i \in [k]$.*

(e) *Let D' be a σ-db of degree $\leqslant d$ and let $\overline{b} = (b_1, \ldots, b_k) \in \mathrm{adom}(D')$. It can be tested in time $(kd^{r+1})^{\mathcal{O}(\|\sigma\|+kd^{r+1})} \leqslant 2^{\mathcal{O}(\|\sigma\| k^2 d^{2r+2})}$ whether $\left(\mathcal{N}_r^D(\overline{a}), \overline{a}\right) \cong \left(\mathcal{N}_r^{D'}(\overline{b}), \overline{b}\right)$.*

The time bound stated in part (e) of Lemma 3.2 is obtained by a brute-force approach. When using Luks' polynomial time isomorphism test for bounded degree graphs [15], the time bound of Lemma 3.2(e) can be improved to $\left(kd^{r+1}\right)^{poly(d\|\sigma\|)}$. However, the asymptotic overall runtime of our algorithms for evaluating FO+MOD-queries won't improve when using Luks algorithm instead of the brute-force isomorphism test of Lemma 3.2(e).

4 Answering Boolean FO+MOD Queries Under Updates

In [7], Frick and Grohe showed that in the static setting (i.e., without database updates), Boolean FO-queries φ can be answered on databases D of degree $\leqslant d$ in time $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}} \cdot \|D\|$. Our first main theorem extends their result to FO+MOD-queries and the dynamic setting.

▶ **Theorem 4.1.** *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a Boolean* FO+MOD$[\sigma]$*-query φ, and a σ-db D_0 of degree $\leqslant d$, and computes within $t_p = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_u = f(\varphi, d)$ and allows to return the query result $\varphi(D)$ with answer time $t_a = \mathcal{O}(1)$. The function $f(\varphi, d)$ is of the form $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$.*

If φ is a d-bounded Hanf-sentence of locality radius r, then $f(\varphi, d) = 2^{\mathcal{O}(\|\sigma\| d^{2r+2})}$, and the initialisation time is $t_i = \mathcal{O}(\|\varphi\|)$.

Proof. W.l.o.g. we assume that all the symbols of σ occur in φ (otherwise, we remove from σ all symbols that do not occur in φ). In the preprocessing routine, we first use Theorem 3.1 to transform φ into a d-equivalent sentence ψ in Hanf normal form; this takes time $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$. The sentence ψ is a Boolean combination of d-bounded Hanf-sentences (over σ) of locality radius at most $r := 4^{\mathrm{qr}(\varphi)}$. Let $\rho_1, \ldots, \rho_s$ be the list of all types that occur in ψ. Thus, every Hanf-sentence in ψ is of the form $\exists^{\geqslant k} x\, \mathrm{sph}_{\rho_j}(x)$ or $\exists^{i \bmod m} x\, \mathrm{sph}_{\rho_j}(x)$ for some $j \in [s]$ and $k, i, m \in \mathbb{N}$ with $k \geqslant 1$, $m \geqslant 2$, and $i < m$. For each $j \in [s]$ let r_j be the radius of ρ_j. Thus, ρ_j is an r_j-type with 1 centre (over σ).

For each $j \in [s]$ our data structure will store the number $\mathtt{A}[j]$ of all elements $a \in \mathrm{adom}(D)$ whose r_j-type is isomorphic to ρ_j, i.e., $(\mathcal{N}_{r_j}^D(a), a) \cong \rho_j$. The initialisation for the empty

database $D_\emptyset$ lets $\mathtt{A}[j] = 0$ for all $j \in [s]$. In addition to the array $\mathtt{A}$, our data structure stores a Boolean value $\mathtt{Ans}$ where $\mathtt{Ans} = \varphi(D)$ is the answer of the Boolean query φ on the current database D. This way, the query can be answered in time $\mathcal{O}(1)$ by simply outputting $\mathtt{Ans}$. The initialisation for the empty database $D_\emptyset$ computes $\mathtt{Ans}$ as follows. Every Hanf-sentence of the form $\exists^{\geqslant k} x\, \mathrm{sph}_{\rho_j}(x)$ in ψ is replaced by the Boolean constant $\mathtt{false}$. Every Hanf-sentence of the form $\exists^{i \bmod m} x\, \mathrm{sph}_{\rho_j}(x)$ is replaced by $\mathtt{true}$ if $i = 0$ and by $\mathtt{false}$ otherwise. The resulting formula, a Boolean combination of the Boolean constants $\mathtt{true}$ and $\mathtt{false}$, then is evaluated, and we let $\mathtt{Ans}$ be the obtained result. The entire initialisation takes time at most $t_i = f(\varphi, d) = 2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$. If φ is a Hanf-sentence, we even have $t_i = \mathcal{O}(\|\varphi\|)$.

To update our data structure upon a command $\mathsf{update}\, R(a_1, \ldots, a_k)$, for $k = \mathrm{ar}(R)$ and $\mathsf{update} \in \{\mathsf{insert}, \mathsf{delete}\}$, we proceed as follows. The idea is to remove from the data structure the information on all the database elements whose r_j-neighbourhood (for some $j \in [s]$) is affected by the update, and then to recompute the information concerning all these elements on the updated database.

Let D_{old} be the database before the update is received and let D_{new} be the database after the update has been performed. We consider each $j \in [s]$. All elements whose r_j-neighbourhood might have changed, belong to the set $U_j := N_{r_j}^{D'}(\overline{a})$, where $D' := D_{new}$ if the update command is $\mathsf{insert}\ R(\overline{a})$, and $D' := D_{old}$ if the update command is $\mathsf{delete}\ R(\overline{a})$.

To remove the old information from $\mathtt{A}[j]$, we compute for each $a \in U_j$ the neighbourhood $T_a := \mathcal{N}_{r_j}^{D_{old}}(a)$, check whether $(T_a, a) \cong \rho_j$, and if so, decrement the value $\mathtt{A}[j]$.

To recompute the new information for $\mathtt{A}[j]$, we compute for all $a \in U_j$ the neighbourhood $T'_a := \mathcal{N}_{r_j}^{D_{new}}(a)$, check whether $(T'_a, a) \cong \rho_j$, and if so, increment the value $\mathtt{A}[j]$.

Using Lemma 3.2 we obtain for each $j \in [s]$ that $|U_j| \leqslant k d^{r_j+1}$. For each $a \in U_j$, the neighbourhoods T_a and T'_a can be computed in time $\big(d^{r_j+1}\big)^{\mathcal{O}(\|\sigma\|)}$, and testing for isomorphism with ρ_j can be done in time $\big(d^{r_j+1}\big)^{\mathcal{O}(\|\sigma\|+d^{r_j+1})}$. Thus, the update of $\mathtt{A}[j]$ is done in time $k{\cdot}\big(d^{r_j+1}\big)^{\mathcal{O}(\|\sigma\|+d^{r_j+1})} \leqslant 2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$ (note that $k \leqslant \|\sigma\| \leqslant \|\varphi\|$ and $r_j \leqslant 4^{\mathrm{qr}(\varphi)} \leqslant 2^{\mathcal{O}(\|\varphi\|))}$).

After having updated $\mathtt{A}[j]$ for each $j \in [s]$, we recompute the query answer $\mathtt{Ans}$ as follows. Every Hanf-sentence of the form $\exists^{\geqslant k} x\, \mathrm{sph}_{\rho_j}(x)$ in ψ is replaced by the Boolean constant $\mathtt{true}$ if $\mathtt{A}[j] \geqslant k$, and by the Boolean constant $\mathtt{false}$ otherwise. Every Hanf-sentence of the form $\exists^{i \bmod m} x\, \mathrm{sph}_{\rho_j}(x)$ is replaced by $\mathtt{true}$ if $\mathtt{A}[j] \equiv i \bmod m$, and by $\mathtt{false}$ otherwise. The resulting formula, a Boolean combination of the Boolean constants $\mathtt{true}$ and $\mathtt{false}$, then is evaluated, and we let $\mathtt{Ans}$ be the obtained result. Thus, recomputing $\mathtt{Ans}$ takes time $\mathit{poly}(\|\psi\|)$.

In summary, the entire update time is $t_u = f(\varphi, d) = 2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$. In case that φ is a d-bounded Hanf-sentence of locality radius r, we even have $t_u = k{\cdot}\big(d^{r+1}\big)^{\mathcal{O}(\|\sigma\|+d^{r+1})} \leqslant 2^{\mathcal{O}(\|\sigma\| d^{2r+2})}$. This completes the proof of Theorem 4.1. ◀

In [7], Frick and Grohe obtained a matching lower bound for answering Boolean FO-queries of schema $\sigma = \{E\}$ on databases of degree at most $d := 3$ in the static setting. They used the (reasonable) complexity theoretic assumption $\mathsf{FPT} \neq \mathsf{AW}[*]$ and showed that if this assumption is correct, then there is no algorithm that answers Boolean FO-queries φ on σ-dbs D of degree $\leqslant 3$ in time $2^{2^{2^{o(\|\varphi\|)}}} \cdot \mathit{poly}(\|D\|)$ in the static setting (see Theorem 2 in [7]). As a consequence, the same lower bound holds in the dynamic setting and shows that in Theorem 4.1, the 3-fold exponential dependency on the query size $\|\varphi\|$ cannot be substantially lowered (unless $\mathsf{FPT} = \mathsf{AW}[*]$):

▶ **Corollary 4.2.** *Let $\sigma := \{E\}$ and let $d := 3$. If $\mathsf{FPT} \neq \mathsf{AW}[*]$, then there is no dynamic algorithm that receives a Boolean $\mathsf{FO}[\sigma]$-query φ and a σ-db D_0, and computes within $t_p \leqslant f(\varphi)\cdot poly(\|D_0\|)$ preprocessing time a data structure that can be updated in time $t_u \leqslant f(\varphi)$ and allows to return the query result $\varphi(D)$ with answer time $t_a \leqslant f(\varphi)$, for a function f with $f(\varphi) = 2^{2^{2^{o(\|\varphi\|)}}}$.*

5 Technical Lemmas on Types and Spheres Useful for Handling Non-Boolean Queries

For our algorithms for evaluating non-Boolean queries it will be convenient to work with a fixed list of representatives of d-bounded r-types, provided by the following straightforward lemma.

▶ **Lemma 5.1.** *There is an algorithm which upon input of a schema σ, a degree bound $d \geqslant 2$, a radius $r \geqslant 0$, and a number $k \geqslant 1$, computes a list $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$ (for a suitable $\ell \geqslant 1$) of d-bounded r-types with k centres (over σ), such that for every d-bounded r-type τ with k centres (over σ) there is exactly one $i \in [\ell]$ such that $\tau \cong \tau_i$. The algorithm's runtime is $2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$. Furthermore, upon input of a d-bounded r-type τ with k centres (over σ), the particular $i \in [\ell]$ with $\tau \cong \tau_i$ can be computed in time $2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$.*

Throughout the remainder of this paper, $\mathcal{L}_r^{\sigma,d}(k)$ will always denote the list provided by Lemma 5.1. The following lemma will be useful for evaluating Boolean combinations of sphere-formulas.

▶ **Lemma 5.2.** *Let σ be a schema, let $r \geqslant 0$, $k \geqslant 1$, $d \geqslant 2$, and let $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$. Let $\overline{x} = (x_1, \ldots, x_k)$ be a list of k pairwise distinct variables. For every Boolean combination $\psi(\overline{x})$ of d-bounded sphere-formulas of radius at most r (over σ), there is an $I \subseteq [\ell]$ such that $\psi(\overline{x}) \equiv_d \bigvee_{i\in I} sph_{\tau_i}(\overline{x})$.*
Furthermore, given $\psi(\overline{x})$, the set I can be computed in time $poly(\|\psi\|) \cdot 2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$.

The lemma's proof is based on the following observations. Negations can be eliminated by the equivalence $\neg \operatorname{sph}_{\tau_j}(\overline{x}) \equiv_d \bigvee_{i\in[\ell]\setminus\{j\}} \operatorname{sph}_{\tau_i}(\overline{x})$. To eliminate conjunctions, observe that for $i \neq i'$ the formula $\operatorname{sph}_{\tau_i}(\overline{x}) \wedge \operatorname{sph}_{\tau_{i'}}(\overline{x})$ is unsatisfiable. Thus, by the distributive law we obtain for all $m \geqslant 1$ and all $I_1, \ldots, I_m \subseteq [\ell]$ that

$$\bigwedge_{j\in[m]} \Big(\bigvee_{i\in I_j} \operatorname{sph}_{\tau_i}(\overline{x}) \Big) \quad \equiv_d \quad \bigvee_{i_1\in I_1} \cdots \bigvee_{i_m\in I_m} \Big(\operatorname{sph}_{\tau_{i_1}}(\overline{x}) \wedge \cdots \wedge \operatorname{sph}_{\tau_{i_m}}(\overline{x}) \Big) \quad \equiv_d \quad \bigvee_{i\in I} \operatorname{sph}_{\tau_i}(\overline{x})$$

for $I := I_1 \cap \cdots \cap I_m$.

For evaluating a Boolean combination $\psi(\overline{x})$ of sphere-formulas *and Hanf-sentences* on a given σ-db D, an obvious approach is to first consider every Hanf-sentence χ that occurs in ψ, to check if $D \models \chi$, and replace every occurrence of χ in ψ with `true` (resp., `false`) if $D \models \chi$ (resp., $D \not\models \chi$). The resulting formula $\psi'(\overline{x})$ is then transformed into a disjunction $\psi''(\overline{x}) := \bigvee_{i\in I} \operatorname{sph}_{\tau_i}(\overline{x})$ by Lemma 5.2, and the query result $\psi(D) = \psi''(D)$ is obtained as the union of the query results $\operatorname{sph}_{\tau_i}(D)$ for all $i \in I$.

While this works well in the static setting (i.e., without database updates), in the dynamic setting we have to take care of the fact that database updates might change the status of a Hanf-sentence χ in ψ, i.e., an update operation might turn a database D with $D \models \chi$ into a database D' with $D' \not\models \chi$ (and vice versa). Consequently, the formula $\psi''(\overline{x})$ that is equivalent to $\psi(\overline{x})$ on D might be inequivalent to $\psi(\overline{x})$ on D'.

To handle the dynamic setting correctly, at the end of each update step we will use the following lemma (the lemma's proof is an easy consequence of Lemma 5.2).

▶ **Lemma 5.3.** *Let σ be a schema. Let $s \geqslant 0$ and let $\chi_1, \ldots, \chi_s$ be* FO+MOD$[\sigma]$*-sentences. Let $r \geqslant 0$, $k \geqslant 1$, $d \geqslant 2$, and let $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$. Let $\overline{x} = (x_1, \ldots, x_k)$ be a list of k pairwise distinct variables. For every Boolean combination $\psi(\overline{x})$ of the sentences $\chi_1, \ldots, \chi_s$ and of d-bounded sphere-formulas of radius at most r (over σ), and for every $J \subseteq [s]$ there is a set $I \subseteq [\ell]$ such that*

$$\psi_J(\overline{x}) \quad \equiv_d \quad \bigvee_{i \in I} sph_{\tau_i}(\overline{x}),$$

where ψ_J is the formula obtained from ψ by replacing every occurrence of a sentence χ_j with `true` *if $j \in J$ and with* `false` *if $j \notin J$ (for every $j \in [s]$).*
Given ψ and J, the set I can be computed in time poly$(\|\psi\|) \cdot 2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$.

To evaluate a single sphere-formula $\mathrm{sph}_\tau(\overline{x})$ for a given r-type τ with k centres (over σ), it will be useful to decompose τ into its connected components as follows. Let $\tau = (T, \bar{t})$ with $\bar{t} = (t_1, \ldots, t_k)$. Consider the Gaifman graph G^T of T and let $C_1, \ldots, C_c$ be the vertex sets of the c connected components of G^T. For each connected component C_j of G^T, let $\bar{t}_j$ be the subsequence of $\bar{t}$ consisting of all elements of $\bar{t}$ that belong to C_j, and let k_j be the length of $\bar{t}_j$. Since $(T, \bar{t})$ is an r-type with the k centres, we have $T = \mathcal{N}_r^T(\bar{t})$, and thus $c \leqslant k$ and $k_j \geqslant 1$ for all $j \in [c]$. To avoid ambiguity, we make sure that the list $C_1, \ldots, C_c$ is sorted in such a way that for all $j < j'$ we have $i < i'$ for the smallest i with $t_i \in C_j$ and the smallest i' with $t_{i'} \in C_{j'}$.

For each C_j consider the r-type with k_j centres $\rho_j = \big(T[C_j], \bar{t}_j\big)$. Let ν_j be the unique integer such that ρ_j is isomorphic to the ν_j-th element in the list $\mathcal{L}_r^{\sigma,d}(k_j)$, and let τ_{j,ν_j} be the ν_j-th element in this list.

It is straightforward to see that the formula $\mathrm{sph}_\tau(\overline{x})$ is d-equivalent to the formula

$$\text{conn-sph}_\tau(\overline{x}) \quad := \quad \bigwedge_{j \in [c]} \mathrm{sph}_{\tau_{j,\nu_j}}(\overline{x}_j) \ \wedge \ \bigwedge_{j \neq j'} \neg\, dist^{k_j,k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'}), \tag{1}$$

where $\overline{x}_j$ is the subsequence of $\overline{x}$ obtained from $\overline{x}$ in the same way as $\bar{t}_j$ is obtained from $\bar{t}$, and $dist^{k_j,k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'})$ is a formula of schema σ which expresses that for some variable y in $\overline{x}_j$ and some variable y' in $\overline{x}_{j'}$ the distance between y and y' is $\leqslant 2r{+}1$. I.e., for $\overline{a} = (a_1, \ldots, a_{k_j})$ and $\overline{b} = (b_1, \ldots, b_{k_{j'}})$ we have $(\overline{a}, \overline{b}) \in dist^{k_j,k_{j'}}_{\leqslant 2r+1}(D) \iff dist^D(\overline{a}; \overline{b}) \leqslant 2r{+}1$, where

$$dist^D(\overline{a}; \overline{b}) \leqslant 2r{+}1 \text{ means that } dist^D(a_i, b_{i'}) \leqslant 2r{+}1 \text{ for some } i \in [k_j] \text{ and } i' \in [k_{j'}]. \tag{2}$$

Using the Lemmas 3.2 and 5.1, the following lemma is straightforward.

▶ **Lemma 5.4.** *There is an algorithm which upon input of a schema σ, numbers $r \geqslant 0$, $k \geqslant 1$, and $d \geqslant 2$, and an r-type τ with k centres (over σ) computes the formula conn-sph$_\tau(\overline{x})$, along with the corresponding parameters c and k_j, ν_j, $\overline{x}_j$, τ_{j,ν_j} for all $j \in [c]$.*
The algorithm's runtime is $2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$.

We define the *signature of* τ to be the tuple $\mathrm{sgn}(\tau)$ built from the parameters c and $\big(k_j, \nu_j, \{\mu \in [k] : x_\mu \text{ belongs to } \overline{x}_j\}\big)_{j \in [c]}$ obtained from the above lemma. The signature $\mathrm{sgn}^D(\overline{a})$ of a tuple $\overline{a}$ in a database D (w.r.t. radius r) is defined as $\mathrm{sgn}(\rho)$ for $\rho := \big(\mathcal{N}_r^D(\overline{a}), \overline{a}\big)$. Note that $\overline{a} \in \mathrm{sph}_\tau(D) \iff \mathrm{sgn}^D(\overline{a}) = \mathrm{sgn}(\tau)$.

6 Testing Non-Boolean FO+MOD Queries Under Updates

This section is devoted to the proof of the following theorem.

▶ **Theorem 6.1.** *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary $\mathsf{FO}{+}\mathsf{MOD}[\sigma]$-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$, and computes within $t_p = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_u = f(\varphi, d)$ and allows to test for any input tuple $\overline{a} \in \boldsymbol{dom}^k$ whether $\overline{a} \in \varphi(D)$ within testing time $t_t = \mathcal{O}(k^2)$. The function $f(\varphi, d)$ is of the form $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$.*

For the proof, we use the lemmas provided in Section 5 and the following lemma.

▶ **Lemma 6.2.** *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, numbers $r \geqslant 0$ and $k \geqslant 1$, an r-type τ with k centres (over σ), and a σ-db D_0 of degree $\leqslant d$, and computes within $t_p = 2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}} \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_u = 2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$ and allows to test for any input tuple $\overline{a} \in \boldsymbol{dom}^k$ whether $\overline{a} \in sph_\tau(D)$ within testing time $t_t = \mathcal{O}(k^2)$.*

Proof sketch. The preprocessing routine starts by using Lemma 5.4 to compute the formula $\text{conn-sph}_\tau(\overline{x})$, along with the according parameters c and k_j, ν_j, $\overline{x}_j$, τ_{j,ν_j} for each $j \in [c]$. This is done in time $2^{(kd^{r+1})^{\mathcal{O}(\|\sigma\|)}}$. We let $\text{sgn}(\tau)$ be the signature of τ (defined directly after Lemma 5.4). Recall that $\text{conn-sph}_\tau(\overline{x}) \equiv_d \text{sph}_\tau(\overline{x})$, and recall from equation (1) the precise definition of the formula $\text{conn-sph}_\tau(\overline{x})$. Our data structure will store the following information on the database D:

- the set Γ of all tuples $\overline{b} \in \text{adom}(D)^{k'}$ where $k' \leqslant k$ and $\mathcal{N}_r^D(\overline{b})$ *is connected*, and
- for every $j \in [c]$ and every k_j-tuple $\overline{b} \in \Gamma$, the unique number $\nu_{\overline{b}}$ such that $\rho_{\overline{b}} := \big(\mathcal{N}_r^D(\overline{b}), \overline{b}\big)$ is isomorphic to the $\nu_{\overline{b}}$-th element in the list $\mathcal{L}_r^{\sigma,d}(k_j)$.

We want to store this information in such a way that for any given tuple $\overline{b} \in \mathbf{dom}^{k'}$ it can be checked in time $\mathcal{O}(k)$ whether $\overline{b} \in \Gamma$. To ensure this, we use a k'-ary array $\Gamma_{k'}$ that is initialised to 0, and where during update operations the entry $\Gamma_{k'}[\overline{b}]$ is set to 1 for all $\overline{b} \in \Gamma$ of arity k'. In a similar way we can ensure that for any given $j \in [c]$ and any $\overline{b} \in \Gamma$ of arity k_j, the number $\nu_{\overline{b}}$ can be looked up in time $\mathcal{O}(k)$.

The **test** routine upon input of a tuple $\overline{a}$ computes the signature $\text{sgn}^D(\overline{a})$ of $\overline{a}$ in D, tests whether $\text{sgn}^D(\overline{a}) = \text{sgn}(\tau)$ and outputs "yes" if this is the case and "no" otherwise. Using the information stored in our data structure, all this can be done in time $\mathcal{O}(k^2)$. The bound on the update time follows from the fact that the insertion or deletion of a tuple affects only a small number of entries in the data structure. ◀

Theorem 6.1 is now obtained by combining Theorem 3.1, Lemma 6.2, Theorem 4.1, and Lemma 5.3.

Proof of Theorem 6.1. For $k = 0$, the theorem immediately follows from Theorem 4.1. Consider the case where $k \geqslant 1$. As in the proof of Theorem 4.1, we assume w.l.o.g. that all the symbols of σ occur in φ. We start the preprocessing routine by using Theorem 3.1 to transform $\varphi(\overline{x})$ into a d-equivalent query $\psi(\overline{x})$ in Hanf normal form; this takes time $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$. The formula ψ is a Boolean combination of d-bounded Hanf-sentences and sphere-formulas (over σ) of locality radius at most $r := 4^{\text{qr}(\varphi)}$, and each sphere-formula is of arity at most k. Let $\chi_1, \ldots, \chi_s$ be the list of all Hanf-sentences that occur in ψ.

We use Lemma 5.1 to compute the list $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$. In parallel for each $i \in [\ell]$, we use the algorithm provided by Lemma 6.2 for $\tau := \tau_i$. Furthermore, for each $j \in [s]$, we

use the algorithm provided by Theorem 4.1 upon input of the Hanf-sentence $\varphi := \chi_j$. In addition to the components used by these dynamic algorithms, our data structure also stores

- the set $J := \{j \in [s] : D \models \chi_j\}$,
- the particular set $I \subseteq [\ell]$ provided by Lemma 5.3 for $\psi(\overline{x})$ and J, and
- the set $K = \{\mathrm{sgn}(\tau_i) : i \in I\}$, where for each type τ, $\mathrm{sgn}(\tau)$ is the signature of τ defined directly after Lemma 5.4.

The **test** routine upon input of a tuple $\overline{a} = (a_1, \ldots, a_k)$ proceeds in the same way as in the proof of Lemma 6.2 to compute in time $\mathcal{O}(k^2)$ the signature $\mathrm{sgn}^D(\overline{a})$ of the tuple $\overline{a}$. For every $i \in [\ell]$ we have $\overline{a} \in \mathrm{sph}_{\tau_i}(D) \iff \mathrm{sgn}^D(\overline{a}) = \mathrm{sgn}(\tau_i)$. Thus, $\overline{a} \in \varphi(D) \iff \mathrm{sgn}^D(\overline{a}) \in K$. Therefore, the **test** routine checks whether $\mathrm{sgn}^D(\overline{a}) \in K$ and outputs "yes" if this is the case and "no" otherwise. To ensure that this test can be done in time $\mathcal{O}(k^2)$, we use an array construction for storing K (similar to the one for storing Γ in the proof of Lemma 6.2).

The **update** routine runs in parallel the update routines for all the used dynamic data structures. Afterwards, it recomputes J by calling the **answer** routine for χ_j for all $j \in [s]$. Then, it uses Lemma 5.3 to recompute I. The set K is then recomputed by applying Lemma 5.4 for $\tau := \tau_i$ for all $i \in I$. It is straightforward to see that the overall runtime of the **update** routine is $t_u = 2^{d^{2^{\mathcal{O}(\|\sigma\|)}}}$. This completes the proof of Theorem 6.1. ◀

7 Representing Databases by Coloured Graphs

To obtain dynamic algorithms for counting and enumerating query results, it will be convenient to work with a representation of databases by coloured graphs that is similar to the representation used in [6]. For defining this representation, let us consider a fixed d-bounded r-type τ with k centres (over a schema σ). Use Lemma 5.4 to compute the formula $\text{conn-sph}_\tau(\overline{x})$ (for $\overline{x} = (x_1, \ldots, x_k)$) and the according parameters c and $k_j, \nu_j, \overline{x}_j, \tau_{j,\nu_j}$, and let $\mathrm{sgn}(\tau)$ be the signature of τ. To keep the notation simple, we assume w.l.o.g. that $\overline{x}_1 = x_1, \ldots, x_{k_1}$, $\overline{x}_2 = x_{k_1+1}, \ldots, x_{k_1+k_2}$ etc.

Recall that $\mathrm{sph}_\tau(\overline{x})$ is d-equivalent to the formula

$$\text{conn-sph}_\tau(\overline{x}) \quad := \quad \bigwedge_{j\in[c]} \mathrm{sph}_{\tau_{j,\nu_j}}(\overline{x}_j) \;\wedge\; \bigwedge_{j\neq j'} \neg\, \mathit{dist}^{k_j,k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'}).$$

To count or enumerate the results of the formula $\mathrm{sph}_\tau(\overline{x})$ we represent the database D by a *c-coloured graph* $\mathcal{G}_D$. Here, a *c-coloured graph* $\mathcal{G}$ is a database of the particular schema

$$\sigma_c \quad := \quad \{E, C_1, \ldots, C_c\},$$

where E is a binary relation symbol and $C_1, \ldots, C_c$ are unary relation symbols. We define $\mathcal{G}_D$ in such a way that the task of counting or enumerating the results of the query $\mathrm{sph}_\tau(\overline{x})$ on the database D can be reduced to counting or enumerating the results of the query

$$\varphi_c(z_1, \ldots, z_c) \quad := \quad \bigwedge_{j\in[c]} C_j(z_j) \;\wedge\; \bigwedge_{j\neq j'} \neg\, E(z_j, z_{j'}) \tag{3}$$

on the c-coloured graph $\mathcal{G}_D$. The vertices of $\mathcal{G}_D$ correspond to tuples over $\mathrm{adom}(D)$ whose r-neighbourhood is connected; a vertex has colour C_j if its associated tuple $\overline{a}$ is in $\mathrm{sph}_{\tau_{j,\nu_j}}(D)$; and an edge between two vertices indicates that $\mathit{dist}^D(\overline{a}; \overline{b}) \leqslant 2r+1$, for their associated tuples $\overline{a}$ and $\overline{b}$. The following lemma allows to translate a dynamic algorithm for counting or enumerating the results of the query $\varphi_c(z_1, \ldots, z_c)$ on c-coloured graphs into a dynamic algorithm for counting or enumerating the result of the query $\mathrm{sph}_\tau(\overline{x})$ on D.

▶ **Lemma 7.1.** *Suppose that the counting problem (the enumeration problem) for $\varphi_c(\overline{z})$ on σ_c-dbs of degree at most d' can be solved by a dynamic algorithm with initialisation time $t_i(c,d')$, update time $t_u(c,d')$, and counting time $t_c(c,d')$ (delay $t_d(c,d')$). Then for every schema σ and every $d \geqslant 2$ the following holds.*

(1) Let $r \geqslant 0$, $k \geqslant 1$, τ a d-bounded r-type with k centres, and fix $d' := d^{2k^2(2r+1)}$ and $\widetilde{t_x} := \max_{c=1}^{k} t_x(c,d')$ for $t_x \in \{t_i, t_u, t_c, t_d\}$. The counting problem (the enumeration problem) for $\mathrm{sph}_\tau(\overline{x})$ on σ-dbs of degree at most d can be solved by a dynamic algorithm with counting time $\widetilde{t_c}$ (delay $\mathcal{O}(\widetilde{t_d}k)$), update time $t'_u \leqslant \widetilde{t_u} d^{\mathcal{O}(k^2 r + k\|\sigma\|)} + 2^{\mathcal{O}(\|\sigma\| k^2 d^{2r+2})}$, and initialisation time $\widetilde{t_i}$.

(2) The counting problem (the enumeration problem) for k-ary FO+MOD-queries $\varphi(\overline{x})$ on σ-dbs of degree at most d can be solved with counting time $\mathcal{O}(1)$ (delay $\mathcal{O}(\widehat{t_d}k)$), update time $(\widehat{t_u} + \widehat{t_c})2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$, and initialisation time $\widehat{t_i}2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$ where $\widehat{t_x} = \max_{c=1}^{k} t_x\big(c, d^{2^{\mathcal{O}(\|\varphi\|)}}\big)$ for $t_x \in \{t_i, t_u, t_c, t_d\}$.

Proof sketch. The first part is a simple reduction from $\text{conn-sph}_\tau(\overline{x})$ to φ_c and can be found in the full version of the paper. The second part for $k = 0$ follows immediately from Theorem 4.1. Consider the case where $k \geqslant 1$. W.l.o.g. we assume that all the symbols of σ occur in φ (otherwise, we remove from σ all symbols that do not occur in φ). We start the preprocessing routine by using Theorem 3.1 to transform $\varphi(\overline{x})$ into a d-equivalent query $\psi(\overline{x})$ in Hanf normal form; this takes time $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$. The formula ψ is a Boolean combination of d-bounded Hanf-sentences and sphere-formulas (over σ) of locality radius at most $r := 4^{\mathrm{qr}(\varphi)}$, and each sphere-formula is of arity at most k. Note that for $d' := d^{2k^2(2r+1)}$ as used in the first part it holds that $d' = d^{2^{\mathcal{O}(\|\varphi\|)}}$. Let $\chi_1, \ldots, \chi_s$ be the list of all Hanf-sentences that occur in ψ (recall that $s \leqslant 2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$).

We use Lemma 5.1 to compute the list $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$ (note that $\ell \leqslant 2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$). In parallel for each $i \in [\ell]$, we use the dynamic algorithm for $\mathrm{sph}_{\tau_i}(\overline{x})$ provided from the lemma's part (1). Furthermore, for each $j \in [s]$, we use the dynamic algorithm provided by Theorem 4.1 upon input of the Hanf-sentence $\varphi := \chi_j$. In addition to the components used by these dynamic algorithms, our data structure also stores

- the set $J := \{j \in [s] : D \models \chi_j\}$,
- the particular set $I \subseteq [\ell]$ provided by Lemma 5.3 for $\psi(\overline{x})$ and J, and
- the cardinality $n = |\varphi(D)|$ of the query result.

The **count** routine simply outputs the value n in time $\mathcal{O}(1)$. The **enumerate** routine runs the **enumerate** routine on $\mathrm{sph}_{\tau_i}(D)$ for every $i \in I$. Note that this enumerates, without repetition, all tuples in $\varphi(D)$, because by Lemma 5.3, $\varphi(D)$ is the union of the sets $\mathrm{sph}_{\tau_i}(D)$ for all $i \in I$, and this is a union of pairwise disjoint sets. The **update** routine runs in parallel the update routines for all used dynamic data structures. Afterwards, it recomputes J by calling the **answer** routine for χ_j for all $j \in [s]$. Then, it uses Lemma 5.3 to recompute I. The number n is then recomputed by letting $n = \sum_{i \in I} n_i$, where n_i is the result of the **count** routine for τ_i. It is straightforward to verify that the overall runtime of the **update** routine is bounded by $(\widehat{t_u} + \widehat{t_c})2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$. ◀

8 Counting Results of FO+MOD Queries Under Updates

This section is devoted to the proof of the following theorem.

▶ **Theorem 8.1.** *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary $\mathsf{FO{+}MOD}[\sigma]$-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$, and computes within $t_p = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_u = f(\varphi, d)$ and allows to return the cardinality $|\varphi(D)|$ of the query result within time $\mathcal{O}(1)$. The function $f(\varphi, d)$ is of the form $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$.*

The theorem follows immediately from Lemma 7.1 and the following dynamic counting algorithm for the query $\varphi_c(\overline{z})$.

▶ **Lemma 8.2.** *There is a dynamic algorithm that receives a number $c \geqslant 1$, a degree bound $d \geqslant 2$, and a σ_c-db $\mathcal{G}_0$ of degree $\leqslant d$, and computes $|\varphi_c(\mathcal{G})|$ with $d^{\mathcal{O}(c^2)}$ initialisation time, $\mathcal{O}(1)$ counting time, and $d^{\mathcal{O}(c^2)}$ update time.*

Proof. Recall that $\varphi_c(z_1, \ldots, z_c) = \bigwedge_{i\in[c]} C_i(z_i) \wedge \bigwedge_{j\neq j'} \neg E(z_j, z_{j'})$. For all $j, j' \in [c]$ with $j \neq j'$ consider the formula $\theta_{j,j'}(z_1, \ldots, z_c) := E(z_j, z_{j'}) \wedge \bigwedge_{i\in[c]} C_i(z_i)$. Furthermore, let $\alpha(z_1, \ldots, z_c) := \bigwedge_{i\in[c]} C_i(z_i)$. Clearly, for every σ_c-db $\mathcal{G}$ we have

$$\begin{aligned} \alpha(\mathcal{G}) &= C_1^{\mathcal{G}} \times \cdots \times C_c^{\mathcal{G}}, \\ \varphi_c(\mathcal{G}) &= \alpha(\mathcal{G}) \setminus \Big(\bigcup_{j\neq j'} \theta_{j,j'}(\mathcal{G}) \Big), \quad \text{and hence,} \quad |\varphi_c(\mathcal{G})| = |\alpha(\mathcal{G})| - \Big| \bigcup_{j\neq j'} \theta_{j,j'}(\mathcal{G}) \Big|. \end{aligned}$$

By the *inclusion-exclusion principle* we obtain for $J := \{(j, j') : j, j' \in [c],\ j \neq j'\}$ that

$$\Big| \bigcup_{j\neq j'} \theta_{j,j'}(\mathcal{G}) \Big| = \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|-1} \Big| \bigcap_{(j,j')\in K} \theta_{j,j'}(\mathcal{G}) \Big| = \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|-1} \, |\varphi_K(\mathcal{G})|$$

for the formula $\varphi_K(z_1, \ldots, z_c) := \bigwedge_{i\in[c]} C_i(z_i) \wedge \bigwedge_{(j,j')\in K} E(z_j, z_{j'})$.

Our data structure stores the following values:

- $|C_i^{\mathcal{G}}|$, for each $i \in [c]$, and $n_1 := |\alpha(\mathcal{G})| = \prod_{i\in[c]} |C_i^{\mathcal{G}}|$,
- $|\varphi_K(\mathcal{G})|$, for each $K \subseteq J$ with $K \neq \emptyset$, and
- $n_2 := \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|-1} \, |\varphi_K(\mathcal{G})|$ and $n_3 := n_1 - n_2$.

Note that $n_3 = |\varphi_c(\mathcal{G})|$ is the desired size of the query result. Therefore, the **count** routine can answer in time $\mathcal{O}(1)$ by just outputting the number n_3.

It remains to show how these values can be initialised and updated during updates of $\mathcal{G}$. The initialisation for the empty graph initialises all the values to 0. In the **update** routine, the values for $|C_i^{\mathcal{G}}|$ and n_1 can be updated in a straightforward way (using time $\mathcal{O}(c)$). For each $K \subseteq J$, the update of $|\varphi_K(\mathcal{G})|$ is provided within time $d^{\mathcal{O}(c^2)}$ by the following Claim 8.3, whose proof can be found in the full version of the paper.

▶ **Claim 8.3.** *For every $K \subseteq J$, the cardinality $|\varphi_K(\mathcal{G})|$ of a σ_c-db $\mathcal{G}$ of degree at most d can be updated within time $d^{\mathcal{O}(c^2)}$ after $d^{\mathcal{O}(c^2)} \cdot |\mathcal{G}_0|$ preprocessing time.*

Once we have available the updated numbers $|\varphi_K(\mathcal{G})|$ for all $K \subseteq J$, the value n_2 can be computed in time $\mathcal{O}(|2^J|) \leqslant 2^{\mathcal{O}(c^2)}$. And n_3 is then obtained in time $\mathcal{O}(1)$. Altogether, performing the **update** routine takes time at most $d^{\mathcal{O}(c^2)}$. The **preprocess** routine initialises all values for the empty graph and then uses $|\mathcal{G}_0|$ update steps to insert all the tuples of $\mathcal{G}_0$ into the data structure. This completes the proof of Lemma 8.2. ◀

9 Enumerating Results of FO+MOD Queries Under Updates

In this section we prove (and afterwards, improve) the following theorem.

▶ **Theorem 9.1.** *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary $\mathsf{FO}{+}\mathsf{MOD}[\sigma]$-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$, and computes within $t_p = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_u = f(\varphi, d)$ and allows to enumerate $\varphi(D)$ with $d^{2^{\mathcal{O}(\|\varphi\|)}}$ delay.*

The function $f(\varphi, d)$ is of the form $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$.

The theorem follows immediately from Lemma 7.1 and the following dynamic enumeration algorithm for the query $\varphi_c(\overline{z})$.

▶ **Lemma 9.2.** *There is a dynamic algorithm that receives a number $c \geqslant 1$, a degree bound $d \geqslant 2$, and a σ_c-db $\mathcal{G}_0$ of degree $\leqslant d$, and computes within $t_p = d^{poly(c)} \cdot |\mathcal{G}_0|$ preprocessing time a data structure that can be updated in time $d^{poly(c)}$ and allows to enumerate the query result $\varphi_c(\mathcal{G})$ with $\mathcal{O}(c^3 d)$ delay.*

Proof. For a σ_c-db $\mathcal{G}$ and a vertex $v \in \mathrm{adom}(\mathcal{G})$ we let $N^{\mathcal{G}}(v)$ be the set of all neighbours of v in $\mathcal{G}$. I.e., $N^{\mathcal{G}}(v)$ is the set of all $w \in \mathrm{adom}(\mathcal{G})$ such that (v, w) or (w, v) belongs to $E^{\mathcal{G}}$.

The underlying idea of the enumeration procedure is the following greedy strategy. We cycle through all vertices $u_1 \in C_1^{\mathcal{G}}$, $u_2 \in C_2^{\mathcal{G}} \setminus N^{\mathcal{G}}(u_1)$, $u_3 \in C_3^{\mathcal{G}} \setminus \big(N^{\mathcal{G}}(u_1) \cup N^{\mathcal{G}}(u_2)\big)$, …, $u_c \in C_c^{\mathcal{G}} \setminus \bigcup_{i \leqslant c-1} N^{\mathcal{G}}(u_i)$ and output $(u_1, \ldots, u_c)$. This strategy does not yet lead to a constant delay enumeration, as there might be vertex tuples $(u_1, \ldots, u_i)$ (for $i < c$) that do extend to an output tuple $(u_1, \ldots, u_c)$, but where many possible extensions are checked before this output tuple is encountered. We now show how to overcome this problem and describe an enumeration procedure with $\mathcal{O}(c^3 d)$ delay and update time $d^{poly(c)}$.

Note that for every $J \subseteq [c]$ we have $\big|\bigcup_{j \in J} N^{\mathcal{G}}(u_j)\big| \leqslant cd$. Hence, if a set $C_i^{\mathcal{G}}$ contains more than cd elements, we know that *every* considered tuple has an extension $u_i \in C_i^{\mathcal{G}}$ that is not a neighbour of any vertex in the tuple. Let $I := \{i \in [c] : |C_i^{\mathcal{G}}| \leqslant cd\}$ be the set of *small* colour classes in $\mathcal{G}$ and to simplify the presentation we assume without loss of generality that $I = \{1, \ldots, s\}$. In our data structure we store the current index set I and the set

$$\mathcal{S} \quad := \quad \big\{\, (u_1, \ldots, u_s) \in C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}} \;:\; (u_j, u_{j'}) \notin E^{\mathcal{G}}, \text{ for all } j \neq j' \,\big\} \tag{4}$$

of tuples on the small colours. Note that a tuple $(u_1, \ldots, u_s) \in C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}}$ extends to an output tuple $(u_1, \ldots, u_c) \in \varphi_c(\mathcal{G})$ if and only if it is contained in $\mathcal{S}$. As $|\mathcal{S}| \leqslant (cd)^c$, it is not hard to see that we can recompute the sets I and $\mathcal{S}$ in time $d^{poly(c)}$ after every update. The enumeration procedure is given in Algorithm 1.

It is straightforward to see that this procedure enumerates $\varphi_c(\mathcal{G})$. Let us analyse the delay. Since for all $i > s$ we have that $\big|C_i^{\mathcal{G}}\big| > cd$, it follows that every call of $\textsc{Enum}(u_1, \ldots, u_i)$ leads to at least one recursive call of $\textsc{Enum}(u_1, \ldots, u_i, u_{i+1})$. Furthermore, there are at most cd iterations of the loop in line 7 that do *not* lead to a recursive call. As every test in line 8 can be done in time $\mathcal{O}(c)$, it follows that the time spans until the first recursive call, between the calls, and after the last call are bounded by $\mathcal{O}(c^2 d)$. As the recursion depth is c, the overall delay between two output tuples is bounded by $\mathcal{O}(c^3 d)$. ◀

By using similar techniques as in [6], we can obtain the following improved version of Lemma 9.2 where the delay is independent of the degree bound d.

Algorithm 1 Enumeration procedure.

1: **for all** $(u_1, \ldots, u_s) \in \mathcal{S}$ **do** ENUM$(u_1, \ldots, u_s)$.
2: Output the end-of-enumeration message `EOE`.
3:
4: **function** ENUM$(u_1, \ldots, u_i)$
5: **if** $i = c$ **then** output the tuple $(u_1, \ldots, u_c)$.
6: **else**
7: **for all** $u_{i+1} \in C^{\mathcal{G}}_{i+1}$ **do**
8: **if** $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ **then** ENUM$(u_1, \ldots, u_i, u_{i+1})$.

▶ **Lemma 9.3.** *There is a dynamic algorithm that receives a number $c \geqslant 1$, a degree bound $d \geqslant 2$, and a σ_c-db $\mathcal{G}_0$ of degree $\leqslant d$, and computes within $t_p = d^{poly(c)} \cdot |\mathcal{G}_0|$ preprocessing time a data structure that can be updated in time $d^{poly(c)}$ and allows to enumerate the query result $\varphi_c(\mathcal{G})$ with $\mathcal{O}(c^2)$ delay.*

Proof idea. We proceed in a similar way as in the proof of Lemma 9.2. But in order to enumerate the tuples with only $\mathcal{O}(c^2)$ delay, we replace the loop in lines 7–8 of Algorithm 1 by a precomputed "skip" function that allows to iterate through all elements in $C^{\mathcal{G}}_{i+1} \setminus \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ with $\mathcal{O}(c)$ delay. This technique has been introduced for static databases in [6]. It turns out that it is possible to maintain the additional information with $d^{poly(c)}$ update time. For details we refer the reader to the full version of the paper. ◀

By Lemma 7.1, this directly improves the delay in Theorem 9.1 from $d^{2^{\mathcal{O}(\|\varphi\|)}}$ to $\mathcal{O}(k^3)$ and leads to the following theorem.

▶ **Theorem 9.4.** *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary* FO+MOD$[\sigma]$*-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$, and computes within $t_p = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_u = f(\varphi, d)$ and allows to enumerate $\varphi(D)$ with $\mathcal{O}(k^3)$ delay. The function $f(\varphi, d)$ is of the form $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$.*

10 Conclusion

Our main results show that in the dynamic setting (i.e., allowing database updates), the results of k-ary FO+MOD-queries on bounded degree databases can be tested and counted in constant time and enumerated with constant delay, after linear time preprocessing and with constant update time. Here, "constant time" refers to data complexity and is of size $poly(k)$ concerning the delay and the time for testing and counting. The time for performing a database update is 3-fold exponential in the size of the query and the degree bound, and is worst-case optimal.

The starting point of our algorithms is to decompose the given query into a query in Hanf normal form, using a recent result of [10]. This normal form is only available for the setting with a fixed maximum degree bound d, i.e., the setting considered in this paper.

Recently, Kuske and Schweikardt [13] introduced a new kind of Hanf normal form for a variant of *first-order logic with counting* that contains and extends Libkin's logic FO(Cnt) [14] and Grohe's logic FO+C [8]. As an application it is shown in [13] that the present paper's techniques can be lifted from FO+MOD to full first-order logic with counting.

An obvious future task is to investigate to which extent further query evaluation results that are known for the static setting can be lifted to the dynamic setting. More specifically: Are there efficient dynamic algorithms for evaluating (i.e., answering, testing, counting, or enumerating) results of first-order queries on other sparse classes of databases (e.g. planar, bounded treewidth, bounded expansion, nowhere dense) or databases of low degree, lifting the "static" results accumulated in [12, 9, 6] to the dynamic setting?

References

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

2 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'17*, 2017. To appear.

3 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. *CoRR*, abs/1702.08764, 2017. URL: `http://arxiv.org/abs/1702.08764`.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009. URL: `http://mitpress.mit.edu/books/introduction-algorithms`.

5 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007. `doi:10.1145/1276920.1276923`.

6 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 121–131, 2014. `doi:10.1145/2594538.2594539`.

7 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004. `doi:10.1016/j.apal.2004.01.007`.

8 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Association for Symbolic Logic in conjunction with Cambridge University Press, to appear. Preliminary version available at `https://www.lii.rwth-aachen.de/de/13-mitarbeiter/professoren/39-book-descriptive-complexity.html`.

9 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 89–98, 2014. `doi:10.1145/2591796.2591851`.

10 Lucas Heimberg, Dietrich Kuske, and Nicole Schweikardt. Hanf normal form for first-order logic with unary counting quantifiers. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16, New York, NY, USA, July 5-8, 2016*, pages 277–286, 2016. `doi:10.1145/2933575.2934571`.

11 Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011. `doi:10.2168/LMCS-7(2:20)2011`.

12 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22–27, 2013*, pages 297–308, 2013. `doi:10.1145/2463664.2463667`.

13 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting: At least, *weak* Hanf normal forms always exist and can be computed! Unpublished manuscript, 2017.

14 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

15 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

16 Bernard M. E. Moret and Henry D. Shapiro. *Algorithms from P to NP: Volume 1: Design & Efficiency.* Benjamin-Cummings, 1991.

17 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.

18 Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. doi:10.1145/2948896.2948899.

19 Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

How Many Variables Are Needed to Express an Existential Positive Query?

Simone Bova[1] **and Hubie Chen**[2]

1 **TU Wien, Vienna, Austria**
`simone.bova@ac.tuwien.ac.at`
2 **Departamento LSI, Facultad de Informática, Universidad del País Vasco, San Sebastián, Spain; and**
IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
`hubie.chen@ehu.es`

Abstract

The number of variables used by a first-order query is a fundamental measure which has been studied in numerous contexts, and which is known to be highly relevant to the task of query evaluation. In this article, we study this measure in the context of existential positive queries. Building on previous work, we present a combinatorial quantity defined on existential positive queries; we show that this quantity not only characterizes the minimum number of variables needed to express a given existential positive query by another existential positive query, but also that it characterizes the minimum number of variables needed to express a given existential positive query, over all first-order queries. Put differently and loosely, we show that for any existential positive query, no variables can ever be saved by moving out of existential positive logic to first-order logic. One component of this theorem's proof is the construction of a winning strategy for a certain Ehrenfeucht-Fraïssé type game.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic, G.2.2 Graph Theory, H.2.1 Logical Design

Keywords and phrases Existential positive queries, finite-variable logics, first-order logic, query optimization

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.9

1 Introduction

Background. The number of variables used by a first-order query is recognized as a highly useful and fundamental measure, and has been studied in numerous settings, including descriptive complexity, finite model theory, and query evaluation. By the number of variables used by a query, we refer to the total number of variables that appear in the query. Note that this measure is, in essence, equivalent to the *width* of a query, which is defined as the maximum number of free variables over all subformulas of the query: a query having width k can be rewritten, just by syntactically renaming variables, as a query using k variables; and, a query using k variables clearly has width at most k. Within this article, all queries dealt with are relational and first-order.

In the setting of query evaluation, the number of variables is a measure of prime and crucial interest. A first reason for this is that the natural bottom-up algorithm for evaluating a first-order query on a finite structure exhibits, in general, an exponential dependence on the number of variables; it also runs in polynomial-time when a constant bound is placed on the number of variables [21]. Furthermore, there are complexity classification

theorems [17, 14, 9, 10] on classes of Boolean queries in which the number of variables emerges as the decisive measure for describing whether or not a class of Boolean queries enjoys tractable query evaluation; in particular, these classification theorems show that, if a class of queries enjoys tractable query evaluation at all, then there exists a constant $k \geq 1$ such that each query in the class can be expressed by (that is, is logically equivalent to) a query using at most k variables. Let us remark that these classification theorems are given in a parameterized complexity setting in which a query can be preprocessed independently of the structure on which it is to be evaluated; and, that these theorems concern classes of queries having bounded arity.[1] (We refer the reader to the cited articles for precise theorem statements and more information.)

Given the computational relevance of *the number of variables* as a query measure, it is natural to inquire, given a query, to what extent the number of variables can be minimized; indeed, it is a natural desire to attempt to rewrite/optimize a given query as one that uses the fewest number of variables (and which retains logical equivalence to the original query). In this article, we study this question on existential positive queries. They include and are semantically equivalent to the so-called *unions of conjunctive queries*, also known as *select-project-join-union queries*; these have been argued to be the most common database queries [1]. Previous work [6] due to the present authors yields a combinatorial characterization (Theorem 21) of the minimum number of variables needed to express a given Boolean existential positive query, *by another existential positive query.* Let FO denote the class of first-order queries, let EP denote the class of existential positive queries, and let FO^k and EP^k denote the restrictions of these classes to queries using at most k variables, respectively; say that a query ϕ is *L-expressible* if there exists a query $\psi \in L$ that is logically equivalent to ϕ. Rephrasing, the combinatorial characterization yields, given a Boolean existential positive query ϕ, the minimum value k such that ϕ is EP^k-expressible. This characterization thus indicates how to minimize number of variables *within* the class of existential positive queries. However, this characterization does not preclude the possibility that a query requiring k variables to be expressed as an existential positive query, could be expressed by a first-order query that uses strictly fewer than k variables.

Contributions. We prove that the just-mentioned possibility can never occur. We generalize the aforementioned combinatorial quantity so that it is defined on all existential positive queries (both Boolean and non-Boolean), and dub this quantity the *combinatorial width.* Our primary theorem states that, for *any* existential positive query ϕ, when k is set equal to the combinatorial width,

- ϕ is can be expressed by an existential positive query using k variables, but
- ϕ cannot be expressed by any first-order query using a number of variables that is strictly less than k.

That is, the combinatorial width not only gives the minimum value k such that ϕ is EP^k-expressible, it in fact more sharply gives the minimum value k such that ϕ is FO^k-expressible. This theorem can be viewed as a collapse result, namely, that for existential positive queries, FO^k-expressibility implies (and hence coincides with) EP^k-expressibility. We want to emphasize that the theorem applies individually to every single existential positive query; in our view, the theorem contains a certain element of surprise, since it states (essentially)

[1] A class of queries has *bounded arity* if there is a constant upper bound on the arity of all relation symbols appearing in a query of the class.

that there is *no existential positive query whatsoever* for which one can save variables by moving out of existential positive logic to the more general first-order logic.

One corollary of our development is that deciding FO^k-expressibility of existential positive sentences is complete for the class Π_2^p of the polynomial hierarchy (Corollary 25); this follows from the present theory in conjunction with a previous theorem on the complexity of deciding EP^k-expressibility of existential positive sentences ([6, Theorem 6]). Let us remark that FO^k-expressibility is undecidable on first-order sentences ([2, Remark 5.3]), and that (to our knowledge) prior to this work, FO^k-expressibility of existential positive sentences was not even known to be decidable.

To establish the inexpressibility portion of our primary theorem, for each Boolean existential positive query ϕ, we show how to construct two finite structures $\mathbf{B}$, $\mathbf{B}'$ on which the query differs, but which are not distinguishable from each other by any first-order query using a number of variables strictly less than the combinatorial width of ϕ. To show this non-distinguishability, we make use of a known Ehrenfeucht-Fraïssé type game [5, 19] designed for showing non-FO^m-expressibility. We in fact first perform this construction for Boolean conjunctive queries (phrased in terms of homomorphisms; see Section 3.1, Theorem 6); after this, we observe that this result extends to Boolean existential positive queries (Theorem 22), and then build on this understanding to treat general existential positive queries (Theorem 23).

The construction of the aforementioned two structures is based on a construction due to Atserias et al. [3]. This previous work characterized, for each finite structure $\mathbf{A}$, the number of pebbles needed for the *existential pebble game* [20] to act as a solution procedure for deciding if there exists a homomorphism from $\mathbf{A}$ to an given structure $\mathbf{B}$ (or, equivalently, if the conjunctive query corresponding to $\mathbf{A}$ evaluates to true on an given structure $\mathbf{B}$). As we show in the present article (see the discussion of Theorem 28 in Section 5), this previous characterization theorem can be readily derived from our primary theorem, and hence our primary theorem provides a strengthening of and broader perspective on this previous theorem.

Let us mention that, in related work, there are numerous articles that investigate the applicability of pebble games to query evaluation problems, which issue was a motivation for the Atserias et al. article [3]. As examples, we mention the work of Dalmau et al. on conjunctive queries and the existential pebble game [15]; the works of Chen and Dalmau on quantified conjunctive queries [13, 8]; the work of Chen and Dalmau on conjunctive queries and generalized hypertree width [12]; and, the work of Barceló et al. [4] on semantically acyclic query evaluation under database constraints.

2 Preliminaries

For an integer $k \geq 0$, we use $[k]$ to denote the set $\{1, \ldots, k\}$, with the convention that $[0] = \emptyset$. For $i = 1, 2$, we freely let π_i denote the ith projection both over pairs and over sets of pairs, so for instance $\pi_i((a_1, a_2)) = a_i$ and $\pi_i(A_1 \times A_2) = A_i$. For an integer n, we let $n \pmod 2$ denote the value $b \in \{0, 1\}$ such that n and b are congruent modulo 2, that is, such that $n - b$ is an integer multiple of 2.

When $f\colon A \to B$ and $g\colon B \to C$ are functions, we use $g(f)$ to denote their composition. When h is a partial function, we use $\mathsf{dom}(h)$ to denote the domain of h.

Graphs, Structures, and Logic

Graphs. All graphs $G = (V, E)$ in this article are undirected and simple, that is, E is a set of 2-element subsets of V.

A *walk* in G is a sequence $W = (a_1, \ldots, a_m) \in V^m$, $m \geq 0$, such that $a_1, \ldots, a_m \in V$ and $\{a_i, a_{i+1}\} \in E$ for all $i \in [m-1]$. Relative to a walk $W = (a_1, \ldots, a_m)$, we use the following terminology. We say that W: *contains* $a \in V$ if $a = a_i$ for some $i \in [m]$; is *from* $s \in V$ *to* $t \in V$ if $a_1 = s$ and $a_m = t$; is *from* $s \in V$ *to* $T \subseteq V$ if $a_1 = s$ and $a_m \in T$; is *s-cyclic* if $a_1 = a_m = s$. A graph $G = (V, E)$ is *connected* if for every two vertices v and v' in V there exists a walk from v to v'.

A *tree decomposition* of a graph $G = (V, E)$ consists of a tree T where each node u is associated to a nonempty subset B_u of V (also called *bag*) such that the following holds:

- For each vertex $v \in V$, the nodes u of T such that $v \in B_u$ form a non-empty connected subtree of T.
- For each edge $e \in E$, there exists a node u in T such that $e \subseteq B_u$.

The *width* of a tree decomposition T of a graph G is defined as the maximum size attained by its bags minus 1, that is, $\max_{u \in T} |B_u| - 1$. The *treewidth* of a graph G, denoted by $\mathsf{tw}(G)$, is the minimum width over all tree decompositions of G.

Let $G = (V, E)$ be a graph. We say that two subsets U and U' of V *touch* if $U \cap U' \neq \emptyset$ or there exist $u \in U$, $u' \in U'$ such that $\{u, u'\} \in E$. A set $\mathcal{M}$ of mutually touching connected subsets of V is called a *bramble* of G. A subset H of V *hits* $\mathcal{M}$ if $H \cap M \neq \emptyset$ for all $M \in \mathcal{M}$. The *order* of a bramble is the minimum size attained over its hitting sets. We will make use of the following duality theorem.

▶ **Theorem 1.** *(refer to [16]) For $k \geq 1$, a graph has treewidth $\geq k$ if and only if it has a bramble of order $> k$.*

Structures. A *relational vocabulary* σ is a set of *relation symbols* R, each of which has an associated natural number $\mathsf{ar}(R)$ called its *arity*.

Let σ be a relational vocabulary. A *σ-structure* $\mathbf{A}$ is specified by a nonempty set A, called the *universe* of $\mathbf{A}$ and denoted by the corresponding italic letter, and a relation $R^{\mathbf{A}} \subseteq A^{\mathsf{ar}(R)}$ for each relation symbol $R \in \sigma$. A structure is *finite* if its universe is finite.

Let $\mathbf{A}$ and $\mathbf{B}$ be σ-structures. A *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ is a mapping $h : A \to B$ such that for each symbol $R \in \sigma$: if the tuple $(a_1, \ldots, a_{\mathsf{ar}(R)})$ is in $R^{\mathbf{A}}$, then the tuple $(h(a_1), \ldots, h(a_{\mathsf{ar}(R)}))$ is in $R^{\mathbf{B}}$. We write $\mathbf{A} \to \mathbf{B}$ to indicate that there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. $\mathbf{A}$ and $\mathbf{B}$ are *homomorphically equivalent* if $\mathbf{A} \to \mathbf{B}$ and $\mathbf{B} \to \mathbf{A}$ both hold. An *endomorphism* of $\mathbf{A}$ is a homomorphism from $\mathbf{A}$ to $\mathbf{A}$. An *automorphism* of $\mathbf{A}$ is a bijective mapping $h\colon A \to A$ such that $(a_1, \ldots, a_{\mathsf{ar}(R)}) \in R^{\mathbf{A}}$ if and only if $(h(a_1), \ldots, h(a_{\mathsf{ar}(R)})) \in R^{\mathbf{A}}$, for all $R \in \sigma$ and $(a_1, \ldots, a_{\mathsf{ar}(R)}) \in A^{\mathsf{ar}(R)}$; note that the inverse of an automorphism is an automorphism. A structure $\mathbf{A}$ is a *core* if every endomorphism of $\mathbf{A}$ is an automorphism of $\mathbf{A}$.

The structure $\mathbf{B}$ is a *substructure* of the structure $\mathbf{A}$ if $B \subseteq A$ and $R^{\mathbf{B}} \subseteq R^{\mathbf{A}}$ for all relation symbols $R \in \sigma$. When $\mathbf{B}$ is a substructure of $\mathbf{A}$, there exists a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$, and h fixes each element $b \in B$, the mapping h is said to be a *retraction* from $\mathbf{A}$ to $\mathbf{B}$; when there exists a retraction from $\mathbf{A}$ to $\mathbf{B}$, it is said that $\mathbf{A}$ *retracts* to $\mathbf{B}$. A *core of a structure* $\mathbf{A}$ is a structure $\mathbf{C}$ such that $\mathbf{A}$ retracts to $\mathbf{C}$, but $\mathbf{A}$ does not retract to any proper substructure of $\mathbf{C}$. It is well known that each finite structure has a core and all cores of a finite structure are isomorphic [18]; we therefore freely refer to *the* core of a finite structure $\mathbf{A}$, and denote this object by $\mathsf{core}(\mathbf{A})$.

The *Gaifman graph* of a structure $\mathbf{A}$ is the graph with vertex set A and having an edge $\{a, a'\}$ if and only if $a \neq a'$ and a and a' cooccur in a tuple of $\mathbf{A}$. The treewidth of a structure $\mathbf{A}$, denoted by $\mathsf{tw}(\mathbf{A})$, is defined as the treewidth of its Gaifman graph.

Logic. In this article, we deal with first-order logic. An *atom* (over vocabulary σ) is an equality of variables $(x = y)$ or is a predicate application $R(x_1, \ldots, x_r)$, where $x_1, \ldots, x_r$ are variables, and $R \in \sigma$ is a relation symbol of arity r. A *formula* (over vocabulary σ) is built from atoms (over σ), negation $(\neg)$, conjunction $(\wedge)$, disjunction $(\vee)$, universal quantification $(\forall)$, and existential quantification $(\exists)$. We define $\mathsf{free}(\phi)$ to be the set of free variables of a formula ϕ. A *sentence* is a formula having no free variables.

We let FO denote the class of first-order formulas. An *existential positive* formula (over vocabulary σ) is a formula built from atoms (over σ) using conjunction, disjunction, and existential quantification; we let EP denote the class of existential positive formulas. A *primitive positive* formula (over vocabulary σ) is a formula built from atoms (over σ) using conjunction and existential quantification; we let PP denote the class of primitive positive formulas. Let $\mathrm{L} \subseteq \mathrm{FO}$. A formula $\phi \in \mathrm{FO}$ is called an L-formula (respectively, an L-sentence) if ϕ is in L (respectively, if ϕ is a sentence in L).

When $\mathbf{A}$ is a structure, f is an assignment of variables in A, and ϕ is a formula over the vocabulary of $\mathbf{A}$, we write $\mathbf{A}, f \models \phi$ to indicate that ϕ is true in $\mathbf{A}$ under f; if ϕ is a sentence, we simply write $\mathbf{A} \models \phi$. Let ϕ and ψ be formulas over the vocabulary σ having the same free variables. We write $\phi \models \psi$ to indicate that ϕ entails ψ, that is, for all σ-structures $\mathbf{A}$ and assignments f in A it holds that, $\mathbf{A}, f \models \phi$ implies $\mathbf{A}, f \models \psi$. We say that ϕ and ψ are *logically equivalent* (denoted $\phi \equiv \psi$) if $\phi \models \psi$ and $\psi \models \phi$. Let ϕ be an FO-formula and let $\mathrm{L} \subseteq \mathrm{FO}$. We say that ϕ is L-*expressible* if there exists an L-formula ϕ' such that $\phi \equiv \phi'$.

For any PP-sentence ϕ over σ, we let $\mathbf{C}[\phi]$ denote the canonical structure induced by ϕ. The *canonical structure* of ϕ is obtained by first prenexing the quantifiers and eliminating the equalities in ϕ, obtaining a logically equivalent PP-sentence ϕ' in prenex form and equality free; and second by defining $\mathbf{C}[\phi]$ to be the structure having a universe element for each existentially quantified variable in ϕ', and where, for each $R \in \sigma$, the relation $R^{\mathbf{C}[\phi]}$ contains $(x_1, \ldots, x_r)$ if and only if $R(x_1, \ldots, x_r)$ appears in the quantifier free part of ϕ'.

For a finite σ-structure $\mathbf{A}$, we let $Q[\mathbf{A}]$ denote the *canonical query* of $\mathbf{A}$, namely, if $A = \{a_1, \ldots, a_n\}$, then

$$Q[\mathbf{A}] = \exists a_1 \ldots \exists a_n \bigwedge_{R \in \sigma} \bigwedge_{(a'_1, \ldots, a'_k) \in R^{\mathbf{A}}} R(a'_1, \ldots, a'_k).$$

It is straightforward to verify that any PP-sentence ϕ is logically equivalent to $Q[\mathbf{C}[\phi]]$, and that every finite structure $\mathbf{A}$ is homomorphically equivalent to $\mathbf{C}[Q[\mathbf{A}]]$. We will use the following known fact [7].

▶ **Theorem 2** (Chandra and Merlin [7]). *Let ϕ be a* PP*-sentence and let $\mathbf{A}$ be a finite structure, such that $\phi = Q[\mathbf{A}]$ or $\mathbf{A} = \mathbf{C}[\phi]$. Then, for any structure $\mathbf{B}$, it holds that $\mathbf{A} \to \mathbf{B}$ if and only if $\mathbf{B} \models \phi$.*

Finite Variable Logics

For each class L of first-order formulas and each integer $k \geq 1$, we let L^k denote those formulas in L that use at most k distinct variable symbols. Fragments of first-order logic using only finitely many variables, called *finite variable logics*, are central in finite model theory. Equivalence in these fragments can be characterized by Ehrenfeucht-Fraïssé style games called *pebble games* [19], which we now introduce.

▶ **Definition 3.** Let σ be a relational vocabulary and let $\mathbf{A}$ and $\mathbf{B}$ be σ-structures. A *partial isomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$ is an injective partial function h from A to B such that for all $R \in \sigma$ and $a_1, \ldots, a_{\mathsf{ar}(R)} \in \mathsf{dom}(h)$ it holds that $(a_1, \ldots, a_{\mathsf{ar}(R)}) \in R^{\mathbf{A}}$ if and only if $(h(a_1), \ldots, h(a_{\mathsf{ar}(R)})) \in R^{\mathbf{B}}$.

The k-pebble game is played by two players, a spoiler and a duplicator, over two (finite) relational structures $\mathbf{A}$ and $\mathbf{B}$ over the same vocabulary. A position of the game is a subset of $A \times B$ of size at most k. The game starts in the empty position and continues in a sequence of rounds. In each round of the game, the spoiler removes a pair from the current position if its size is k, and then selects an element $a \in A$ or $b \in B$; the duplicator answers by selecting an element $b \in B$ or $a \in A$, respectively. The new position is defined by adding the pair (a, b) to the old position. The duplicator wins the game if each position occurring along the rounds is a partial isomorphism from $\mathbf{A}$ to $\mathbf{B}$.

▶ **Definition 4.** [19] Let $k \geq 0$. A *duplicator winning strategy in the k-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$ is a family $\mathcal{S}$ of partial isomorphisms h from $\mathbf{A}$ to $\mathbf{B}$ with $|\mathsf{dom}(h)| \leq k$ such that:
(S1) $\emptyset \to \emptyset$ is in $\mathcal{S}$.
(S2) If $h \in \mathcal{S}$ and $|\mathsf{dom}(h)| < k$, then:
  (S2.F) For every $a \in A$ there exists $b \in B$ such that $h \cup \{(a, b)\}$ is in $\mathcal{S}$.
  (S2.B) For every $b \in B$ there exists $a \in A$ such that $h \cup \{(a, b)\}$ is in $\mathcal{S}$.
(S3) If $h \in \mathcal{S}$ and $a \in \mathsf{dom}(h)$, then $h|_{\mathsf{dom}(h) \setminus \{a\}}$ is in $\mathcal{S}$.

It is clear that the duplicator wins the above described k-pebble game on $\mathbf{A}$ and $\mathbf{B}$ if and only if the game admits a duplicator winning strategy.

We say that two structures $\mathbf{A}$, $\mathbf{B}$ are *indistinguishable by* FO^k*-sentences* (in short FO^k*-indistinguishable*), if for each FO^k-sentence ϕ it holds that $\mathbf{A} \models \phi$ if and only if $\mathbf{B} \models \phi$. As anticipated, k-pebble games characterize expressibility in the k-variable fragment of first-order logic.

▶ **Theorem 5** (Barwise [5], Immerman [19])**.** *Let $k \geq 0$ and let $\mathbf{A}$ and $\mathbf{B}$ be relational structures on the same vocabulary. The following are equivalent.*
1. *There exists a duplicator winning strategy in the k-pebble game on $\mathbf{A}$ and $\mathbf{B}$.*
2. *$\mathbf{A}$ and $\mathbf{B}$ are FO^k-indistinguishable.*

3 Construction of Structures

In this section, we show a theorem implying that certain PP-sentences, namely those corresponding (via Chandra-Merlin, Theorem 2) to cores of treewidth at least k, cannot be expressed by FO-sentences using k variables (Theorem 6). This inexpressibility result allows us to later derive our primary theorem (Theorem 23).

▶ **Theorem 6.** *Let $\mathbf{A}$ be a core on the relational vocabulary σ such that $\mathsf{tw}(\mathbf{A}) \geq k \geq 1$. There exist σ-structures $\mathbf{B}$ and $\mathbf{B}'$ such that $\mathbf{B} \to \mathbf{A}$; $\mathbf{A} \to \mathbf{B}'$; $\mathbf{A} \not\to \mathbf{B}$; and, $\mathbf{B}$ and $\mathbf{B}'$ are FO^k-indistinguishable.*

The remainder of the current section is devoted to the construction (Section 3.1) and the study (Section 3.2 and Section 3.3) of the structures $\mathbf{B}$ and $\mathbf{B}'$ mentioned in Theorem 6. We remark that the structure $\mathbf{B}$ is essentially equal to the structure defined by Atserias et al. in [3, Section 4].

We give a proof of Theorem 6 that makes forward references; this proof might serve as a guide to the layout of this section.

Proof of Theorem 6. It is readily verified that $\mathsf{tw}(\mathbf{A}) \geq k \geq 1$ implies the existence of a connected component (C, E) in the Gaifman graph of $\mathbf{A}$ such that $\mathsf{tw}((C, E)) \geq k \geq 1$. Let s be an arbitrary but fixed vertex in C. Let $\mathbf{B}$ and $\mathbf{B}'$ be the two σ-structures defined relative to $\mathbf{A}$ and s in Section 3.1; then $\mathbf{B} \to \mathbf{A}$ by Observation 9 in Section 3.1. In Section 3.2, Lemma 11 and Lemma 12 show that $\mathbf{A} \to \mathbf{B}'$ and $\mathbf{A} \not\to \mathbf{B}$, respectively. In Section 3.3, Lemma 16 gives a duplicator winning strategy in the k-pebble game on $(\mathbf{B}, \mathbf{B}')$, which suffices via Theorem 5. ◀

▶ **Notation 7.** *The following names are reserved throughout the current section:*

- $\mathbf{A}$ *denotes a core on the relational vocabulary* σ*, with universe* A*, such that* $\mathsf{tw}(\mathbf{A}) \geq k \geq 1$.
- $G = (C, E)$ *denotes a connected component in the Gaifman graph of* $\mathbf{A}$ *such that* $\mathsf{tw}(G) \geq k \geq 1$*. Note that, in particular,* $|C| \geq 2$.
- s *denotes an arbitrary but fixed vertex in* C.
- $\mathcal{M}$ *denotes an arbitrary but fixed bramble of* G *having order* $> k$ *(this exists by Theorem 1). Recall that, therefore, any hitting set for* $\mathcal{M}$ *has size at least* $k + 1$.

3.1 Construction of $\mathbb{B}$ and $\mathbb{B}'$

In this section, relative to $\mathbf{A}$ and s, we define two σ-structures $\mathbf{B}$ and $\mathbf{B}'$ as follows.

For all $a \in A$, let E_a denote the edges incident on a in the Gaifman graph of $\mathbf{A}$. Let:

$$U_C = \left\{ (a, f) \;\middle|\; a \in C, f\colon E_a \to \{0,1\} \text{ is such that } \sum_{e \in E_a} f(e) \pmod 2 = \begin{cases} 0 & \text{if } a \neq s \\ 1 & \text{if } a = s \end{cases} \right\}$$

$$U'_C = \left\{ (a, f) \;\middle|\; a \in C, f\colon E_a \to \{0,1\} \text{ is such that } 0 = \sum_{e \in E_a} f(e) \pmod 2 \right\}$$

$$U_{A \setminus C} = \{(a, f\colon E_a \to \{0\}) \mid a \in A \setminus C\}.$$

Then $\mathbf{B}$ and $\mathbf{B}'$ have universes B and B' defined as follows:

$$B = U_C \cup U_{A \setminus C}$$
$$B' = U'_C \cup U_{A \setminus C}$$

The vocabulary is interpreted as follows. For all $R \in \sigma$, let $(a_1, f_1), \ldots, (a_r, f_r)$ be elements of $\mathbf{B}$ (respectively, of $\mathbf{B}'$), where $r = \mathsf{ar}(R)$. Then $((a_1, f_1), \ldots, (a_r, f_r))$ is in $R^{\mathbf{B}}$ (respectively, in $R^{\mathbf{B}'}$) if and only if

- $(a_1, \ldots, a_r) \in R^{\mathbf{A}}$;
- for all $i, j \in [r]$, if $e = \{a_i, a_j\} \in E$ then $f_i(e) = f_j(e)$.

For the sake of intuition, suppose that $\mathbf{A}$ is a connected graph, so that $\mathbf{A}$ is isomorphic to its Gaifman graph and $C = A$. The universes of $\mathbf{B}$ and $\mathbf{B}'$ are formed by pairs (a, f) where a is a vertex of $\mathbf{A}$ and f is a Boolean labelling of the edges incident on a. The Boolean labellings have even parity with the only exception of those paired with s in B which have odd parity. Moreover there is an edge $\{(a, f), (a', f')\}$ in $\mathbf{B}$ (respectively, $\mathbf{B}'$) if and only if the edge $\{a, a'\}$ is in $\mathbf{A}$ and the labellings f and f' agree on $\{a, a'\}$. A concrete example follows.

▶ **Example 8.** Let $\mathbf{A} = (A, E^{\mathbf{A}})$ where $A = \{a, s\}$ and $E^{\mathbf{A}} = \{(a, s), (s, a)\}$, that is, $\mathbf{A}$ is the graph formed by the single edge $\{a, s\}$. Let $f_i\colon \{\{a, s\}\} \to \{0, 1\}$ be such that $f_i(\{a, s\}) = i$ for $i = 0, 1$. Then $\mathbf{B} = (B, E^{\mathbf{B}})$ where $B = \{(a, f_0), (s, f_1)\}$ and $E^{\mathbf{B}} = \emptyset$, and $\mathbf{B}' = (B', E^{\mathbf{B}'})$ where $B' = \{(a, f_0), (s, f_0)\}$ and $E^{\mathbf{B}'} = \{((a, f_0), (s, f_0)), ((s, f_0), (a, f_0))\}$.

Note that, by construction of $\mathbf{B}$, the map $\pi_1 \colon B \to A$ is a homomorphism from $\mathbf{B}$ to $\mathbf{A}$. We inline this observation for later use.

▶ **Observation 9.** $\mathbf{B} \to \mathbf{A}$.

3.2 $\mathbb{A}$ Treats $\mathbb{B}$ and $\mathbb{B}'$ Differently

Let $\mathbf{B}$ and $\mathbf{B}'$ be the two σ-structures defined relative to $\mathbf{A}$ and s in Section 3.1. We show that $\mathbf{A}$ maps homomorphically to $\mathbf{B}'$ (Lemma 11) but not to $\mathbf{B}$ (Lemma 12).

▶ **Example 10.** Let $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{B}'$ be as in Example 8. Then the mapping h defined by $h(a') = (a', f_0)$ for all $a' \in A$ is a homomorphism from $\mathbf{A}$ to $\mathbf{B}'$, but $\mathbf{A}$ has no homomorphisms to $\mathbf{B}$ by direct inspection, as $E^{\mathbf{B}} = \emptyset$.

A direct inspection of the construction in Section 3.1 shows that a copy of $\mathbf{A}$ sits inside $\mathbf{B}'$, by looking at the pairs in B' carrying an identically 0 labelling. Therefore $\mathbf{A}$ maps homomorphically to $\mathbf{B}'$.

▶ **Lemma 11.** $\mathbf{A} \to \mathbf{B}'$.

On the other hand, we claim that $\mathbf{B}$ has no homomorphisms from $\mathbf{A}$. For the sake of intuition, let $\mathbf{A}$ be a connected graph. A homomorphism from $\mathbf{A}$ to $\mathbf{B}$ maps A to B preserving all edges. By construction (here we use that $\mathbf{A}$ is a core), the homomorphic image of $\mathbf{A}$ in $\mathbf{B}$ is a copy of $\mathbf{A}$ where all edges carry *two* Boolean labels *equal* to each other. Summing these Boolean labels in two ways, edgewise and vertexwise, we get the contradiction that the edgewise sum has even parity but the vertexwise sum, by the contribution of the labels of s, has odd parity.

▶ **Lemma 12.** *[3, Lemma 1]* $\mathbf{A} \not\to \mathbf{B}$.

3.3 $\mathbb{B}$ and $\mathbb{B}'$ are FO^k-Indistinguishable

Let $\mathbf{B}$ and $\mathbf{B}'$ be the two σ-structures defined relative to $\mathbf{A}$ and s in Section 3.1. We show that $\mathbf{B}$ and $\mathbf{B}'$ are FO^k-indistinguishable (Lemma 16).

We describe informally a winning strategy for the duplicator in the k-pebble game on $\mathbf{B}$ and $\mathbf{B}'$. For the sake of illustration, consider the simple case where $\mathbf{B}$ and $\mathbf{B}'$ are constructed relative to a *connected graph* $\mathbf{A}$ so that $\mathbf{A} = G$ (the lemma lifts the idea to arbitrary relational structures $\mathbf{A}$, whose Gaifman graphs are possibly disconnected). The duplicator fixes a bramble $\mathcal{M}$ of G and maintains along the rounds of the game the following position ($i = 0, 1, \ldots, k$):

- i pebble pairs are placed on elements $(a_1, f_1), \ldots, (a_i, f_i) \in B$ and correspondingly on elements $(a_1, f'_1), \ldots, (a_i, f'_i) \in B'$ such that for some walk W in G from s to a bramble set in $\mathcal{M}$ avoiding $a_1, \ldots, a_i$ it holds that $f_j(e)$ equals the parity of $f'_j(e)$ plus the number of times e occurs in W (for all $j \in [i]$ and $e \in E_{a_j}$).

That such a position exists and is a partial isomorphism is the content of Lemma 14; that the duplicator can maintain such a position along the rounds of the game is the content of Lemma 16.

We now start proving our statement. Recall Notation 7 for the meaning of G, $\mathcal{M}$, et cetera. We further prepare the following notation.

▶ **Notation 13.** *Let $W = (a_1, \ldots, a_m)$ be a walk in G. For $e \in E$, we let*

$$\mathsf{occ}_W(e) = |\{i \in [m-1] \mid e = \{a_i, a_{i+1}\}\}|$$

denote the number of times the edge e is used in W. Moreover, for every $S \subseteq A$, let

$$\mathsf{avoid}_{\mathcal{M}}(S) = \bigcup_{\{M \in \mathcal{M} \mid M \cap S = \emptyset\}} M$$

be the union of the bramble sets in $\mathcal{M}$ disjoint from S.

▶ **Lemma 14.** *Let $0 \leq i \leq k$ and let $\{(a_1, f_{a_1}), \ldots, (a_i, f_{a_i})\} \subseteq B$. Let W be a walk in G from s to $\mathsf{avoid}_{\mathcal{M}}(\{a_1, \ldots, a_i\})$. For all $j \in [i]$, let $f'_{a_j} : E_{a_j} \to \{0,1\}$ be defined by $f'_{a_j}(e) = f_{a_j}(e) + \mathsf{occ}_W(e) \pmod 2$. Then the mapping h sending (a_j, f_{a_j}) to (a_j, f'_{a_j}) for all $j \in [i]$ is a partial isomorphism from $\mathbf{B}$ to $\mathbf{B}'$.*

Towards proving Lemma 14, we claim the following.

▶ **Claim 15.** *Let $a \in A$ and let W be a walk in G from s to $t \neq a$. Then:*

- *If $(a, f) \in B$ and $f'(e) = f(e) + \mathsf{occ}_W(e) \pmod 2$ for all $e \in E_a$, then $(a, f') \in B'$.*
- *If $(a, f') \in B'$ and $f(e) = f'(e) - \mathsf{occ}_W(e) \pmod 2$ for all $e \in E_a$, then $(a, f) \in B$.*

The idea underlying the claim is that, for $(a, f) \in B$ with $a = s$, both the sum of the labellings of E_a under f, and the number of occurrences of edges in E_a in a walk W that starts at a and does not end at a, are odd. For $(a, f) \in B$ with $a \neq s$ both the sum of the labellings under f of edges incident on a, and the number of occurrences of edges incident on a in a walk W that neither starts nor ends at a, are even. It follows that $(a, f') \in B'$ by construction.

We are now ready to prove the lemma. For the sake of intuition, consider the case where $\mathbf{A}$ is a connected graph, so that $\mathbf{A} = G$. If the edge $\{(a, f), (b, g)\}$ is in $\mathbf{B}$ and $h((a, f)) = (a, f')$, $h((b, g)) = (b, g')$, then on the one hand $f(\{a, b\}) = g(\{a, b\})$, which is the content of (3); and on the other hand $f'(\{a, b\})$ and $g'(\{a, b\})$ equal the parity of the sum of $f(\{a, b\})$ and $g(\{a, b\})$, respectively, and the number of occurrences of $\{a, b\}$ in a fixed walk W, which is the content of (2). Therefore $f'(\{a, b\}) = g'(\{a, b\})$ and the edge $\{(a, f), (b, g)\}$ is in $\mathbf{B}'$. The converse is symmetric.

Proof of Lemma 14. By Claim 15, $(a_j, f'_{a_j}) \in B'$ for all $j \in [i]$, hence h is a partial function from B to B'; moreover, h is injective by definition. Let $R \in \sigma$, let $\mathsf{ar}(R) = r$, and let $(b_1, f_{b_1}), \ldots, (b_r, f_{b_r}) \in \mathsf{dom}(h)$. It is sufficient to show that

$$((b_1, f_{b_1}), \ldots, (b_r, f_{b_r})) \in R^{\mathbf{B}} \iff (h(b_1, f_{b_1}), \ldots, h(b_r, f_{b_r})) \in R^{\mathbf{B}'}$$

We prove the forward direction; the backward direction is similar.

Assume $((b_1, f_{b_1}), \ldots, (b_r, f_{b_r})) \in R^{\mathbf{B}}$. Then by construction $(b_1, \ldots, b_r) \in R^{\mathbf{A}}$. We distinguish two cases.

Case: $\{b_1, \ldots, b_r\} \cap C = \emptyset$. Note that if $b_j \in A \setminus C$, then $f_{b_j}(e) = 0$ for all $e \in E_{b_j}$ by construction and $\mathsf{occ}_W(e) = 0$ for all $e \in E_{b_j}$ because W lies entirely in C. Then $f'_{b_j}(e) = 0$ for all $e \in E_{b_j}$. Therefore, by construction, $(h(b_1, f_{b_1}), \ldots, h(b_r, f_{b_r})) \in R^{\mathbf{B}'}$.

Case: If $\{b_1, \ldots, b_r\} \subseteq C$, then let $e = \{b_j, b_{j'}\} \in E$, $j, j' \in [r]$. We claim that $f'_{b_j}(e) = f'_{b_{j'}}(e)$. By hypothesis, there exists a walk W in G from s to $t \in \mathsf{avoid}_{\mathcal{M}}(\{a_1, \ldots, a_i\})$ such that for all $j \in [i]$ and all $e \in E_{a_j}$ it holds that

$$f'_{a_j}(e) = f_{a_j}(e) + \mathsf{occ}_W(e) \pmod 2. \tag{1}$$

It follows from (1) that, for all $j \in [r]$ and $e \in E_{b_j}$,

$$f'_{b_j}(e) = f_{b_j}(e) + \mathsf{occ}_W(e) \pmod 2. \tag{2}$$

Moreover, by construction, if $j, j' \in [r]$ and $e = \{b_j, b_{j'}\} \in E$, then

$$f_{b_j}(e) = f_{b_{j'}}(e). \tag{3}$$

Therefore,

$$\begin{aligned} f'_{b_j}(e) &= f_{b_j}(e) + \mathsf{occ}_W(e) \pmod 2 && \text{by (2)} \\ &= f_{b_{j'}}(e) + \mathsf{occ}_W(e) \pmod 2 && \text{by (3)} \\ &= f'_{b_{j'}}(e) && \text{by (2)} \end{aligned}$$

We conclude that $((b_1, f'_{b_1}), \ldots, (b_r, f'_{b_r})) = (h(b_1, f_{b_1}), \ldots, h(b_r, f_{b_r})) \in R^{\mathbf{B}'}$. ◀

We now formalize the strategy informally described above and show that, indeed, it is a winning strategy for the duplicator in the k-pebble game on $\mathbf{B}$ and $\mathbf{B}'$.

▶ **Lemma 16.** *Let $\mathcal{S}$ be the family of partial isomorphisms from $\mathbf{B}$ to $\mathbf{B}'$ that contains an injective map $h\colon B \to B'$ when $|\mathsf{dom}(h)| \leq k$ and there exists a walk W in G from s to $\mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h)))$ such that, for all $(a, f) \in \mathsf{dom}(h)$, it holds that $h((a, f)) = (a, f')$ where*

$$f'(e) = f(e) + \mathsf{occ}_W(e) \pmod 2$$

for all $e \in E_a$. Then $\mathcal{S}$ is a duplicator winning strategy in the k-pebble game on $\mathbf{B}$ and $\mathbf{B}'$.

The crux of the proof is the following. Suppose that $i < k$ pebble pairs are placed on elements $(a_1, f_1), \ldots, (a_i, f_i) \in B$ and correspondingly on elements $(a_1, f'_1), \ldots, (a_i, f'_i) \in B'$ such that for some walk W in G from s to t in a bramble set in $\mathcal{M}$ avoiding $a_1, \ldots, a_i$ it holds that $f'_j(e)$ equals the parity of $f_j(e)$ plus the number of times e occurs in W (for all $j \in [i]$ and $e \in E_{a_j}$).

The spoiler pebbles, say, $(a_{i+1}, f_{i+1}) \in B$. The duplicator obtains a walk W' in G from s to a vertex t' lying in a bramble set of $\mathcal{M}$ that avoids $a_1, \ldots, a_i, a_{i+1}$ (such a set exists because the bramble has order greater than $k \geq i + 1$) by walking from s to t over W and then from t to t' using only vertices in the bramble sets of t and t' (which is feasible by the properties of the bramble). The duplicator pebbles $(a_{i+1}, f'_{i+1}) \in B$, where $f'_{i+1}(e)$ equals the parity of $f_{i+1}(e)$ plus the number of times e occurs in W' for all $e \in E_{a_{i+1}}$; thus maintaining its winning position, because the number of occurrences of edges incident to any of $a_1, \ldots, a_i$ does not change in passing from W to W'.

Proof of Lemma 16. We check that $\mathcal{S}$ satisfies Definition 4 relative to $\mathbf{B}$ and $\mathbf{B}'$.

For (S1), we show that the function $h\colon \emptyset \to \emptyset$ is in $\mathcal{S}$. Since any hitting set of the bramble $\mathcal{M}$ has size at least $k + 1 \geq 2$, there exists a set M in $\mathcal{M}$ such that $s \notin M$. Let $a \in M$. Then $a \in \mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h)))$. Moreover, G is connected, hence there is a walk W in G from s to a. Thus $h \in \mathcal{S}$.

We now verify that $\mathcal{S}$ satisfies (S2.F) and (S2.B). Let $h \in \mathcal{S}$ and let $|\mathsf{dom}(h)| < k$. By definition of $\mathcal{S}$, there exists a walk W in G from s to $t \in \mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h)))$ such that, for all $(a, f) \in \mathsf{dom}(h)$ it holds that $h((a, f)) = (a, f')$ where $f'(e) = f(e) + \mathsf{occ}_W(e) \pmod 2$ for all $e \in E_a$.

For (S2.F), let $(a, f) \in B$. Since $|\mathsf{dom}(h)| < k$, we have that $|\pi_1(\mathsf{dom}(h)) \cup \{a\}| \leq k$. So, by the observation about any hitting set of the bramble $\mathcal{M}$, there exists a set $M' \in \mathcal{M}$ such that $M' \cap (\pi_1(\mathsf{dom}(h)) \cup \{a\}) = \emptyset$. Let $t \in M \in \mathcal{M}$. Obtain a walk W' in G from s to $t' \in \mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h)) \cup \{a\})$ by concatenating W and a walk in G from $t \in M$ to $t' \in M'$

containing only vertices in M and M'. Let (a, f') be such that $f'(e) = f(e)+\mathsf{occ}_{W'}(e) \pmod 2$ for all $e \in E_a$. Now define $h' = h \cup \{((a, f), (a, f'))\}$.

We want to show that h' is in $\mathcal{S}$. We claim that, for all $(b, f_b) \in \mathsf{dom}(h')$, if $h'((b, f_b)) = (b, f'_b)$, then $f'_b(e) = f_b(e) + \mathsf{occ}_{W'}(e) \pmod 2$ for all all $e \in E_b$. It follows that h' is a partial isomorphism from $\mathbf{B}$ to $\mathbf{B}'$ by Lemma 14, so that h' is in $\mathcal{S}$ witnessed by W'. Hence $h' \in \mathcal{S}$.

For the claim, let $(b, f_b) \in \mathsf{dom}(h')$ and let $h'((b, f_b)) = (b, f'_b)$. If $b = a$, then $(b, f_b) = (a, f)$ and $h'((a, f)) = (a, f')$ such that $f'(e) = f(e) + \mathsf{occ}_{W'}(e) \pmod 2$ for all $e \in E_a$ by construction. If $b \neq a$, then notice that $b \notin M$ and $b \notin M'$, so that for every $e \in E_b$ it holds that $\mathsf{occ}_W(e) = \mathsf{occ}_{W'}(e)$, and therefore,

$$\begin{aligned} f'_b(e) &= f_b(e) + \mathsf{occ}_W(e) \pmod 2 && \text{by hypothesis on } h \\ &= f_b(e) + \mathsf{occ}_{W'}(e) \pmod 2 \end{aligned}$$

and the claim is settled.

For (S2.B), let $(a, f') \in B'$. Along the lines above, we obtain a walk W' in G from s to $\mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h)) \cup \{a\})$, and $f \colon E_a \to \{0, 1\}$ such that $f'(e) = f(e)+\mathsf{occ}_{W'}(e) \pmod 2$ for all $e \in E_a$; we put $h' = h \cup \{((a, f), (a, f'))\}$, and show that $h' \in \mathcal{S}$ appealing to Lemma 14.

We conclude by verifying that $\mathcal{S}$ satisfies (S3). Let $h \in \mathcal{S}$ and let $(a, f) \in \mathsf{dom}(h)$. We want to show that the restriction of h to $\mathsf{dom}(h) \setminus \{(a, f)\}$, namely, $h' = h|_{\mathsf{dom}(h)\setminus\{(a,f)\}}$ is in $\mathcal{S}$. Partial isomorphisms are closed under restrictions, hence h' is a partial isomorphism from $\mathbf{B}$ to $\mathbf{B}'$ of domain size at most k. Let W be a walk in G from s to $\mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h)))$ witnessing that h is in $\mathcal{S}$. We claim that W also witnesses that h' is in $\mathcal{S}$. By definition, W is from s to $\mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h))) \subseteq \mathsf{avoid}_{\mathcal{M}}(\pi_1(\mathsf{dom}(h')))$. Moreover, if $h'((a', f')) = (a', f'')$, then $h((a', f')) = (a', f'')$ and for all $e \in E_{a'}$ it holds that $f''(e) = f'(e) + \mathsf{occ}_W(e) \pmod 2$. ◀

4 Existential Positive Logic

In this section, we present *combinatorial width*, the combinatorial measure on EP-formulas. While the specialization of this measure to EP-sentences is due (implicitly) to previous work, we here give a definition that applies to all EP-formulas.

In order to define our measure, we first associate a structure to each PP-formula, as follows. In the following definition, one should conceive of ψ as a disjunct of an EP-formula which is the disjunction of PP-formulas and which has free variables $v_1, \ldots, v_\ell$.

▶ **Definition 17.** For each vocabulary σ and each integer $\ell \geq 1$, we fix $U_{\sigma,\ell}$ to be a relation symbol of arity ℓ outside of σ.

For each vocabulary σ, each list $v_1, \ldots, v_\ell$ of variables, and each PP-formula ψ over vocabulary σ with $\mathsf{free}(\psi) \subseteq \{v_1, \ldots, v_\ell\}$, we define a structure $\mathbf{C}[\psi; \sigma; v_1, \ldots, v_\ell]$ as follows:

- Define ψ' as the formula obtained from ψ by prenexing ψ and then renaming quantified variables (if necessary) so that none of the variables $v_1, \ldots, v_\ell$ are quantified.
- Define ψ^+ as $\exists v_1 \ldots v_\ell (U_{\sigma,\ell}(v_1, \ldots, v_\ell) \wedge \psi')$ if $\ell > 0$, and as ψ' if $\ell = 0$.
- Define $\mathbf{C}[\psi; \sigma; v_1, \ldots, v_\ell]$ as $\mathbf{C}[\psi^+]$.

Note that, in the case that $\ell = 0$, it holds that $\mathbf{C}[\psi; \sigma; v_1, \ldots, v_\ell]$ is isomorphic to $\mathbf{C}[\psi]$, since in this case ψ' and ψ are identical up to renaming variables, and $\psi^+ = \psi'$.

In essence, the structure just defined is the canonical structure obtained by conjoining, to the PP-formula, the atom $U_{\sigma,\ell}(v_1, \ldots, v_\ell)$, where the symbol $U_{\sigma,\ell}$ is a fresh one; and then existentially quantifying all free variables.

We now define a notion of *normalized* EP-formula.

▶ **Definition 18.** (extends [6, Definition 3]) Let ϕ be an EP-formula over vocabulary σ and whose free variables are $v_1, \ldots, v_\ell$. We say that ϕ is *normalized* if it is equal to a disjunction $\bigvee_{i\in[m]} \psi_i$ (with $m \geq 0$) where:

- each ψ_i is a prenex PP-formula,
- for each $i \in [m]$, the structure $\mathbf{C}[\psi_i; \sigma; v_1, \ldots, v_\ell]$ is a core, and
- for each $i, j \in [m]$ with $i \neq j$, it holds that $\mathbf{C}[\psi_i; \sigma; v_1, \ldots, v_\ell] \not\to \mathbf{C}[\psi_j; \sigma; v_1, \ldots, v_\ell]$.

▶ **Proposition 19.** *There exists an algorithm that, given as input an* EP*-formula ϕ, outputs a normalized* EP*-formula $\phi' = \bigvee_{i\in[m]} \psi'_i$ that is logically equivalent to ϕ, and such that (for any $k \geq 0$) if ϕ is* EP^k*-expressible, then ψ'_i is* PP^k*-expressible for every $i \in [m]$.*

This proposition was observed in the particular case of sentences by [6, Section 3]; the construction is, in essence, a solution to a classic exercise in database theory [1, Exercise 6.14(c)].

We now define the notion of *combinatorial width*. Although we only define it directly on normalized EP-formulas, the definition can be naturally extended to all EP-formulas in light of Proposition 19.

▶ **Definition 20.** Let $\phi = \bigvee_{i\in[m]} \psi_i$ be a normalized EP-formula with free variables $v_1, \ldots, v_\ell$ and over vocabulary σ. We define $\mathsf{comb\text{-}width}(\phi) = \max_{i\in[m]}(\mathsf{tw}(\mathbf{C}[\psi_i; \sigma; v_1, \ldots, v_\ell]) + 1)$. Note that, in the case that ϕ is a sentence (equivalently, when $\ell = 0$), it holds that $\mathsf{comb\text{-}width}(\phi) = \max_{i\in[m]}(\mathsf{tw}(\mathbf{C}[\psi_i]) + 1)$.

The notion of the *combinatorial width* of a normalized sentence was studied implicitly in previous work; see Proposition 3.4 of [9] and Section 3 of [6]. The following theorem was known [15, 6].

▶ **Theorem 21.** *Let $\phi = \bigvee_{i\in[m]} \psi_i$ be a normalized* EP*-sentence. Let $k = \mathsf{comb\text{-}width}(\phi)$. The sentence ϕ is* EP^k*-expressible, but (assuming $k > 0$) is not* EP^{k-1}*-expressible.*

Proof. We will use the fact, which follows from [15, Theorem 12], that (when $w \geq 1$) a PP-sentence ψ has $\mathsf{tw}(\mathsf{core}(\mathbf{C}[\psi])) < w$ if and only if ψ is PP^w-expressible.

We have that ϕ is EP^k-expressible via this fact, since each disjunct ψ_i has $\mathsf{tw}(\mathbf{C}[\psi_i]) < \mathsf{comb\text{-}width}(\phi)$.

For the non-expressibility result, suppose that ϕ is EP^n-expressible; we prove that $n \geq k$. It follows from Proposition 19 that ϕ is logically equivalent to a disjunction $\bigvee_{i\in[m']} \psi'_i$ of PP^n-formulas. By the fact, we may assume that each $\mathbf{C}[\psi'_i]$ is a core; we obtain that $\mathsf{tw}(\mathbf{C}[\psi'_i]) + 1 \leq n$. By Lemma 4(2) of [6], it follows that $k = \max_{i\in[m']}(\mathsf{tw}(\mathbf{C}[\psi'_i]) + 1)$. We thus have $k \leq n$. ◀

5 Main Theorems and Consequences

We first prove our number-of-variables characterization for EP-sentences.

▶ **Theorem 22.** *Let $\phi = \bigvee_{i\in[m]} \psi_i$ be a normalized* EP*-sentence; let $w = \mathsf{comb\text{-}width}(\phi)$. The sentence ϕ is* EP^w*-expressible, but (assuming $w > 1$) is not* FO^{w-1}*-expressible.*

This theorem is transparently seen to be a strengthening of Theorem 21 (when the stated assumption holds).

Proof. That the sentence ϕ is EP^w-expressible follows directly from Theorem 21, so we prove that ϕ is not FO^{w-1}-expressible. Choose $i \in [m]$ such that $\mathsf{comb\text{-}width}(\phi) = \mathsf{tw}(\mathbf{C}[\psi_i]) + 1$;

set $\mathbf{A} = \mathbf{C}[\psi_i]$. By the definition of *normalized*, we have that $\mathbf{A}$ is a core. Let $\mathbf{B}$, $\mathbf{B}'$ be the structures provided by Theorem 6 relative to $\mathbf{A}$ and $k = \mathsf{tw}(\mathbf{A})$; note that $k = w - 1$.

We show in the next paragraph that $\mathbf{B}' \models \phi$ and $\mathbf{B} \not\models \phi$. Then, since $\mathbf{B}'$ and $\mathbf{B}$ are FO^k-indistinguishable by Theorem 6, and since ϕ distinguishes between $\mathbf{B}'$ and $\mathbf{B}$, it follows that ϕ is not FO^k-expressible.

We prove the claim. We have that $\mathbf{A} \to \mathbf{B}'$, hence $\mathbf{B}' \models \psi_i$ by Theorem 2, and $\mathbf{B}' \models \phi$. Now, assume for a contradiction that $\mathbf{B} \models \phi$. Then $\mathbf{B} \models \psi_j$ for some $j \in [m]$. Then $\mathbf{C}[\psi_j] \to \mathbf{B}$ by Theorem 2. We have $\mathbf{B} \to \mathbf{A} = \mathbf{C}[\psi_i]$ by Theorem 6. Hence $\mathbf{C}[\psi_j] \to \mathbf{C}[\psi_i]$, so $i = j$ by the hypothesis that ϕ is normalized. But we also have $\mathbf{A} = \mathbf{C}[\psi_i] \not\to \mathbf{B}$, hence $i \neq j$, a contradiction. ◀

We now extend the previous theorem to address all normalized EP-formulas. The following is our primary theorem.

▶ **Theorem 23** (Primary theorem). *Let $\phi = \bigvee_{i \in [m]} \psi_i$ be a normalized* EP*-formula; let $w = \mathsf{comb\text{-}width}(\phi)$. The formula ϕ is EP^w-expressible, but (assuming $w > 1$) is not FO^{w-1}-expressible.*

Proof. Let σ denote the vocabulary of ϕ. Let $v_1, \ldots, v_\ell$ denote the free variables of ϕ. We assume that $\ell \geq 1$ (otherwise, the theorem follows from Theorem 22).

We first establish that ϕ is EP^w-expressible. It suffices to prove that, for each $i \in [m]$, the formula ψ_i is expressible using $b = \mathsf{tw}(\mathbf{C}[\psi_i; \sigma; v_1, \ldots, v_\ell]) + 1$ variables. By definition of treewidth, there exists a tree decomposition of $\mathbf{C}[\psi_i; \sigma; v_1, \ldots, v_\ell]$ where each bag has size b or less. It is readily verified that this tree decomposition is a tree decomposition of $\mathbf{C}[\psi_i']$ (where ψ_i' is derived from ψ_i as described in Definition 17) which has a bag B_s with $v_1, \ldots, v_\ell \in B_s$. The result now essentially follows from the argument of Lemma 5.11 of [10, 11]. We give an explanation for the sake of completeness. Add a vertex r adjacent to s to the tree and define $B_r = \{v_1, \ldots, v_\ell\}$. Let T be the resulting object, which is straightforwardly verified to be a tree decomposition of $\mathbf{C}[\psi_i']$ where each bag has size b or less.

We now describe how to construct the desired formula. Each variable of ψ_i', other than $v_1, \ldots, v_\ell$, will be existentially quantified exactly once in the formula; and all atoms of ψ_i', will appear in the formula. In this way, the constructed formula will be clearly logically equivalent to ψ_i', and hence ψ_i.

- Root the tree T at r; note that r has a single child, s.
- For each non-root vertex t of T, we define a PP-formula θ_t inductively, as follows. Define θ_t as $\exists w_1 \ldots w_m \theta_t'$ where $w_1, \ldots, w_m$ is a list of variables that are in the bag B_t of t but not in the bag of t's parent, and θ_t' is the conjunction of θ_u over all children u of t and all atoms $R(z_1, \ldots, z_k)$ of ψ_i' where $\{z_1, \ldots, z_k\} \subseteq B_t$.
- The desired formula is θ_s.

Observe (by induction) that, for each non-root vertex t of T, it holds that $\mathsf{free}(\theta_t') \subseteq B_t$ and $\mathsf{free}(\theta_t) \subseteq B_p$ where p is the parent of t. It follows that each formula θ_t, and in particular θ_s, has width $\leq b$, and is thus PP^b-expressible.

We now establish that ϕ is not FO^{w-1}-expressible.

If $w \leq \ell$, then it is straightforward to verify that the formula ϕ is not FO^{w-1}-expressible, since $w - 1$ is strictly lower than the number of free variables of ϕ. (We can remark that $w \geq \ell$, since each structure $\mathbf{C}[\psi_i; \sigma; v_1, \ldots, v_\ell]$ interprets $U_{\sigma,\ell}$ as $\{(v_1, \ldots, v_\ell)\}$, and so the treewidth of each such structure plus 1 is ℓ or more.)

We thus assume in the sequel that $w > \ell$. For each $i \in [m]$, we may assume without loss of generality that each ψ_i is prenexed and that in each ψ_i, none of the variables $v_1, \ldots, v_\ell$ are

quantified. Define ψ_i^+ as in Definition 17; we then have $\psi_i^+ = \exists v_1 \dots \exists v_\ell (U_{\sigma,\ell}(v_1, \dots, v_\ell) \wedge \psi_i)$. Define ϕ^+ as $\bigvee_{i \in [m]} \psi_i^+$. We have that $\mathbf{C}[\psi_i; \sigma; v_1, \dots, v_\ell] = \mathbf{C}[\psi_i^+]$. We have

$$\mathsf{comb\text{-}width}(\phi) = \max_{i \in [m]}(\mathsf{tw}(\mathbf{C}[\psi_i; \sigma; v_1, \dots, v_\ell])+1) = \max_{i \in [m]}(\mathsf{tw}(\mathbf{C}[\psi_i^+])+1) = \mathsf{comb\text{-}width}(\phi^+),$$

where the last equality holds by the note in Definition 20.

Observe that

$$\begin{aligned}\phi^+ &= \bigvee_{i \in [m]} \psi_i^+ = \bigvee_{i \in [m]} \exists v_1 \dots \exists v_\ell (U_{\sigma,\ell}(v_1, \dots, v_\ell) \wedge \psi_i) \\ &\equiv \exists v_1 \dots \exists v_\ell \bigvee_{i \in [m]} (U_{\sigma,\ell}(v_1, \dots, v_\ell) \wedge \psi_i) \equiv \exists v_1 \dots \exists v_\ell (U_{\sigma,\ell}(v_1, \dots, v_\ell) \wedge (\bigvee_{i \in [m]} \psi_i)) \\ &= \exists v_1 \dots \exists v_\ell (U_{\sigma,\ell}(v_1, \dots, v_\ell) \wedge \phi).\end{aligned}$$

Suppose, for a contradiction, that ϕ is FO^{w-1}-expressible. Then, ϕ can be expressed just using the variables $x_1, \dots, x_{w-1}$ (recall the assumption that $w > \ell$, which gives $w - 1 \geq \ell$). Since $\phi^+ \equiv \exists v_1 \dots \exists v_\ell (U_{\sigma,\ell}(v_1, \dots, v_\ell) \wedge \phi)$, this immediately implies that ϕ^+ can be expressed just using the variables $x_1, \dots, x_{w-1}$, and that ϕ^+ is FO^{w-1}-expressible. As $w = \mathsf{comb\text{-}width}(\phi) = \mathsf{comb\text{-}width}(\phi^+)$, this contradicts Theorem 22. ◀

▶ **Corollary 24.** *For each $k \geq 1$, FO^k-expressibility and EP^k-expressibility coincide on existential positive formulas; that is, an existential positive formula is FO^k-expressible if and only if it is EP^k-expressible.*

▶ **Corollary 25.** *The problem of deciding FO^k-expressibility of EP-sentences is Π_2^p-complete. By this, we refer to the problem of deciding, given an EP-sentence ϕ and an integer $k \geq 1$, whether ϕ is FO^k-expressible.*

Proof. The problem of deciding, given an EP-sentence ϕ and an integer $k \geq 1$, whether ϕ is EP^k-expressible, is Π_2^p-complete by [6, Theorem 6]. The present corollary thus follows from Corollary 24. ◀

We conclude the article by explaining how the main result of [3] follows from our development. We first present the necessary definitions.

▶ **Definition 26.** Let σ be a relational vocabulary and let $\mathbf{A}$ and $\mathbf{B}$ be σ-structures. A *partial homomorphism from $\mathbf{A}$ to $\mathbf{B}$* is a partial function h from A to B such that, for all $R \in \sigma$ and all $a_1, \dots, a_{\mathsf{ar}(R)} \in \mathsf{dom}(h)$, it holds that $(a_1, \dots, a_{\mathsf{ar}(R)}) \in R^{\mathbf{A}}$ implies $(h(a_1), \dots, h(a_{\mathsf{ar}(R)})) \in R^{\mathbf{B}}$.

▶ **Definition 27.** [20] Let $k \geq 0$. A *duplicator winning strategy in the existential k-pebble game on* $(\mathbf{A}, \mathbf{B})$ is a family $\mathcal{S}$ of partial homomorphisms h from $\mathbf{A}$ to $\mathbf{B}$ with $|\mathsf{dom}(h)| \leq k$ such that:

(E1) $\emptyset \to \emptyset$ is in $\mathcal{S}$.

(E2) If $h \in \mathcal{S}$ and $|\mathsf{dom}(h)| < k$, then for every $a \in A$ there exists $b \in B$ such that $h \cup \{(a, b)\}$ is in $\mathcal{S}$.

(E3) If $h \in \mathcal{S}$ and $a \in \mathsf{dom}(h)$, then $h|_{\mathsf{dom}(h) \setminus \{a\}}$ is in $\mathcal{S}$.

▶ **Theorem 28.** *(Main theorem of [3]) Let $\mathbf{A}$ be a core on the relational vocabulary σ such that $\mathsf{tw}(\mathbf{A}) \geq k \geq 1$. There exists a σ-structure $\mathbf{B}$ such that $\mathbf{A} \not\to \mathbf{B}$ and there exists a duplicator winning strategy in the existential k-pebble game on $(\mathbf{A}, \mathbf{B})$.*

To prove Theorem 28, we will make use of the following transitivity property.

▶ **Lemma 29.** *Let $k \geq 0$. If there exist duplicator winning strategies in the existential k-pebble games on $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{B}, \mathbf{C})$, then there exists a duplicator winning strategy in the existential k-pebble game on $(\mathbf{A}, \mathbf{C})$.*

We now give a proof of Theorem 28 using the construction of this article.

Proof of Theorem 28. Let $\mathbf{A}$ be a σ-structure. By hypothesis, $\mathbf{A}$ is a core and $\mathsf{tw}(\mathbf{A}) \geq k$; so, by Theorem 6 there exist σ-structures $\mathbf{B}$ and $\mathbf{B}'$, such that $\mathbf{A} \not\to \mathbf{B}$, $\mathbf{A} \to \mathbf{B}'$, and (via Theorem 5) the duplicator has a winning strategy in the k-pebble game on $\mathbf{B}$ and $\mathbf{B}'$.

Therefore the duplicator has a winning strategy $\mathcal{S}$ in the existential k-pebble game on $(\mathbf{B}', \mathbf{B})$, because duplicator winning strategies in the k-pebble game on $\mathbf{B}$ and $\mathbf{B}'$ are also duplicator winning strategies in the existential k-pebble game on $(\mathbf{B}', \mathbf{B})$. Moreover, there exists a homomorphism g from $\mathbf{A}$ to $\mathbf{B}'$. It is straightforward to verify that the family containing each restriction of g to a subset $S \subseteq A$ with $|S| \leq k$ is a duplicator winning strategy in the existential k-pebble game on $(\mathbf{A}, \mathbf{B}')$. It follows, from Lemma 29 that there exists a duplicator winning strategy existential k-pebble game on $(\mathbf{A}, \mathbf{B})$. ◀

Acknowledgments. The first author is supported by the FWF Austrian Science Fund (Parameterized Compilation, P26200). The second author is supported by the Spanish Project MINECO COMMAS TIN2013-46181-C2-R, Basque Project GIU15/30, and Basque Grant UFI11/45.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2 I. Adler and M. Weyer. Tree-Width for First-Order Formulae. *Logical Methods in Computer Science*, 8(1), 2012.
3 A. Atserias, A.A. Bulatov, and V. Dalmau. On the power of k -consistency. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, pages 279–290, 2007.
4 P. Barceló, G. Gottlob, and A. Pieris. Semantic Acyclicity under Constraints. In *Proceedings of the 35th Symposium on Principles of Database Systems (PODS 2016)*, pages 343–354, 2016.
5 J. Barwise. On Moschovakis Closure Ordinals. *Journal of Symbolic Logic*, 42:292–296, 1977.
6 S. Bova and H. Chen. The Complexity of Width Minimization for Existential Positive Queries. In *Proceedings of the 17th International Conference on Database Theory (ICDT 2014)*, pages 235–244, 2014.
7 A.K. Chandra and P.M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the 9th Symposium on Theory of Computing (STOC 1977)*, pages 77–90, 1977.
8 H. Chen. Beyond Q-Resolution and Prenex Form: A Proof System for Quantified Constraint Satisfaction. *Logical Methods in Computer Science*, 10(4), 2014.
9 H. Chen. On the Complexity of Existential Positive Queries. *ACM Transactions on Computational Logic*, 15(1), 2014.
10 H. Chen. The Tractability Frontier of Graph-Like First-Order Query Sets. In *Proceedings of the Joint Meeting of the 23rd Conference on Computer Science Logic and the 29th Symposium on Logic in Computer Science (CSL-LICS 2014)*, pages 31:1–31:9, 2014.
11 H. Chen. The Tractability Frontier of Graph-Like First-Order Query Sets. *CoRR*, abs/1407.3429, 2014.

12 H. Chen and V. Dalmau. Beyond Hypertree Width: Decomposition Methods Without Decompositions. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2015)*, pages 167–181, 2005.

13 H. Chen and V. Dalmau. From Pebble Games to Tractability: An Ambidextrous Consistency Algorithm for Quantified Constraint Satisfaction. In *Proceedings of the 19th Conference on Computer Science Logic (CSL 2005)*, pages 232–247, 2005.

14 H. Chen and M. Müller. One Hierarchy Spawns Another: Graph Deconstructions and the Complexity Classification of Conjunctive Queries. In *Proceedings of the Joint Meeting of the 23rd Conference on Computer Science Logic and the 29th Symposium on Logic in Computer Science (CSL-LICS 2014)*, pages 32:1–32:10, 2014.

15 V. Dalmau, P.G. Kolaitis, and M.Y. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, pages 310–326, 2002.

16 R. Diestel. *Graph Theory*. Springer, 2012.

17 M. Grohe. The Complexity of Homomorphism and Constraint Satisfaction Problems Seen from the Other Side. *Journal of the ACM*, 54(1), 2007.

18 P. Hell and J. Nesetril. The Core of a Graph. *Discrete Mathematics*, 109:117–126, 1992.

19 N. Immerman. Upper and Lower Bounds for First-Order Expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.

20 P.G. Kolaitis and M.Y. Vardi. On the Expressive Power of Datalog: Tools and a Case Study. *Journal of Computer and System Sciences*, 51(1):110–134, 1995.

21 M.Y. Vardi. On the Complexity of Bounded-Variable Queries. In *Proceedings of the 14th Symposium on Principles of Database Systems (PODS 1995)*, pages 266–276, 1995.

Expressive Power of Entity-Linking Frameworks

Douglas Burdick[1], Ronald Fagin[2], Phokion G. Kolaitis[*3], Lucian Popa[4], and Wang-Chiew Tan[5]

1 **IBM Almaden Research Center, San Jose, CA, USA**
2 **IBM Almaden Research Center, San Jose, CA, USA**
3 **University of California Santa Cruz, Santa Cruz, CA, USA; and IBM Almaden Research Center, San Jose, CA, USA**
4 **IBM Almaden Research Center, San Jose, CA, USA**
5 **Recruit Institute of Technology, Mountain View, CA, USA; and University of California Santa Cruz, Santa Cruz, CA, USA**

Abstract

We develop a unifying approach to declarative entity linking by introducing the notion of an entity linking framework and an accompanying notion of the certain links in such a framework. In an entity linking framework, logic-based constraints are used to express properties of the desired link relations in terms of source relations and, possibly, in terms of other link relations. The definition of the certain links in such a framework makes use of weighted repairs and consistent answers in inconsistent databases. We demonstrate the modeling capabilities of this approach by showing that numerous concrete entity linking scenarios can be cast as such entity linking frameworks for suitable choices of constraints and weights. By using the certain links as a measure of expressive power, we investigate the relative expressive power of several entity linking frameworks and obtain sharp comparisons. In particular, we show that we gain expressive power if we allow constraints that capture non-recursive collective entity resolution, where link relations may depend on other link relations (and not just on source relations). Moreover, we show that an increase in expressive power also takes place when we allow constraints that incorporate preferences as an additional mechanism for expressing "goodness" of links.

1998 ACM Subject Classification H.2 Database Management

Keywords and phrases Entity linking, entity resolution, constraints, repairs, certain links

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.10

1 Introduction and Summary of Results

Entity linking is the problem of creating links among records representing real-world entities that are related in certain ways. As an important special case, it includes *entity resolution*, which is the problem of identifying or linking "duplicate" entities. Since the pioneering work of Fellegi and Sunter [11] in 1969, entity linking has been recognized as a fundamental computational problem that has been investigated by several different research communities. While much of the work in this area [8, 10, 15, 18] has focused and continues to focus on the design, implementation, and validation of direct algorithms for entity linking (and, in particular, for entity resolution), recent investigations have developed declarative approaches to entity linking that make it possible to separate the specification of entity linking from its actual implementation (see, for example, [1, 7, 13, 14]).

* Part of this work was done while Phokion G. Kolaitis was visiting the Simons Institute for the Theory of Computing.

In [7], we introduced and explored a declarative approach to entity linking that makes use of logical constraints. Our approach differs from earlier declarative approaches because it uses *link-to-source* constraints, instead of *source-to-link* constraints. Source-to-link constraints constitute, in effect, rules for creating links from source data in an operational manner. Our link-to-source constraints spell out conditions that the links must satisfy, independently of how the links will be created, and thus give rise to *solutions* of the declarative entity-linking specification at hand. In [7], we focused on the class of *maximum-value* solutions as "good" solutions for entity linking; intuitively, these are the solutions in which links have maximum "justification" in terms of the constraints and in terms of the source data. Since there can be multiple maximum-value solutions, we introduced the notion of the *certain links*, which, by definition, are the links that appear in every maximum-value solution and, therefore, are the links that should be kept. We then explored the problem of enumerating all maximum-value solutions and the problem of computing the certain links. This investigation was carried out for several different languages expressing link-to-source constraints, including languages that capture *collective entity resolution*, where interdependence between link relations is allowed.

The variety and multitude of entity-linking approaches raise the question of developing methods and tools for comparing such different approaches. A comparative evaluation of the performance of several different direct algorithms for entity resolution (or entity matching) has been carried in [16] and [17]. Up to now, however, no methodology has been developed for comparing, along some axis, different declarative approaches for entity linking. The main aim of this paper is to develop such a methodology that is centered on the notion of the *expressive power* of declarative entity-linking frameworks.

Our first conceptual contribution is to formulate a unifying notion of an *entity-linking framework* and an accompanying notion of the *certain links* in such a framework. This is achieved by bringing into the picture a notion of *weighted repairs* of inconsistent databases, which are a variant of the notion of weighted repairs of inconsistent databases in description logics studied in [9]. The "good" solutions for entity linking are then identified with the maximum weight repairs of inconsistent databases with respect to suitable choices of constraints and weights, while the certain links are defined to be the *consistent answers* of atomic link queries with respect to the maximum weight repairs, that is, those links that are in every maximum weight repair. The inconsistent database whose weighted repairs we consider gives an upper bound or a domain for the candidate links; it could be provided (e.g., handed in from another system), or could be simply based on the Cartesian product of sets of entities (which we do in many of our definitions and proofs[1]).

This general approach gives rise to a single formalism for declarative entity linking in which the constraint language, the sets of constraints allowed, and the weight function that measures the "strength" of the links are parameters of the definition. We demonstrate the modeling capabilities of this formalism by showing that it not only contains as special cases the concrete declarative entity linking scenarios studied in [7], but also can account for new ones, such as entity linking based on maximum cardinality repairs and entity linking with constraints that incorporate preferences.

Our second conceptual contribution is to use the certain links as a measure of the expressive power of an entity linking framework and define what it means for an entity linking framework to *subsume* another entity linking framework. This makes it possible to compare different entity linking frameworks along the axis of their expressive power.

[1] Note that this is conceptual and it does not mean that such a Cartesian product needs to be materialized.

As regards technical results, we first show that, under some mild hypotheses on entity linking frameworks, it is possible to enumerate with polynomial delay all maximum weight repairs and to compute the certain links in polynomial time. This general result contains as special cases several similar results for concrete entity linking scenarios obtained in [7]. Our main technical contribution, however, is to delineate the relative expressive power of different linking frameworks. Specifically, we show that the entity linking framework of the maximum-value solutions considered in [7] and the entity linking framework of maximum cardinality repairs introduced here are of incomparable expressive power, in the sense that neither of the two can subsume the other. We also show that the entity linking framework for collective entity resolution where the constraints allow the link relations to depend on other link relations is strictly more expressive than in the case where constraints do not allow for interdependence among the link relations. This increase in expressive power takes place even when the dependencies among the link relations are non-recursive. Finally, we show that we also gain expressive power by adding preference constraints, which represent an additional, practical mechanism (see HIL [14]) for specifying the "good" links by letting a user explicitly, and declaratively, give priority to some types of links over other types of links. Concretely, we show that there is an entity linking framework with preference constraints that is not subsumed by the entity linking framework of maximum-value solutions (with no preference constraints).

Note that since the expressive power is measured via the certain links, proving that a specific entity linking framework is not subsumed by some other specific entity linking framework is a much more challenging task than simply showing that the constraints defining the first framework are not logically equivalent to those defining the second framework. The proofs of our results about the expressive power of entity linking frameworks involve a combination of special-purpose techniques with techniques from finite model theory. In particular, the proof of the result concerning the expressive power of entity linking frameworks with preference constraints uses a locality theorem that is interesting in its own right.

In summary, the conceptual and technical contributions in this paper provide a unifying approach to declarative entity linking and pave the way for the systematic comparative evaluation of different entity linking frameworks.

2 Weighted Repairs and Consistent Answers

Let $\mathbf{S}$ and $\mathbf{L}$ be two disjoint relational schemas, and let $\mathbf{R} = \mathbf{S} \cup \mathbf{L}$ be the union of these two schemas. If I is an $\mathbf{S}$-instance and J is an $\mathbf{L}$-instance, then $\langle I, J\rangle$ denotes the $\mathbf{R}$-instance that is the union of I and J viewed as sets of facts. Clearly, every $\mathbf{R}$-instance is of the form $\langle I, J\rangle$, where I is an $\mathbf{S}$-instance and J is an $\mathbf{L}$-instance. If I is an $\mathbf{S}$-instance and S is a relation symbol in $\mathbf{S}$, then S^I denotes the relation of I interpreting the relation symbol S; similarly, if J is an $\mathbf{L}$-instance and L is a symbol in $\mathbf{L}$, then L^J denotes the relation of J interpreting the relation symbol L.

▶ **Definition 1.** A *weight function on* $\mathbf{R}$ is a function w that assigns a non-negative weight $w(\langle I, J\rangle, L^J(a_1, \ldots, a_n))$ for every $\mathbf{R}$-instance $\langle I, J\rangle$ and for every fact $L^J(a_1, \ldots, a_n)$ of J, where L is a relation symbol in $\mathbf{L}$. The weight $w(\langle I, J\rangle, L^J(a_1, \ldots, a_n))$ is called the *weight* of the fact $L^J(a_1, \ldots, a_n)$ in $\langle I, J\rangle$.

Note that, even though only facts in relations interpreting $\mathbf{L}$-symbols have weights, the weight of such a fact may depend on the entire $\mathbf{R}$-instance $\langle I, J\rangle$ and not just on J.

In what follows, we will define the notion of a *maximum weight repair* of an $\mathbf{R}$-instance $\langle I, J\rangle$; this notion is inspired by a similar one introduced by Du, Qi, and Shen [9] in the context of knowledge-bases with constraints expressed in description logics.

▶ **Definition 2.** Let Σ be a set of integrity constraints on $\mathbf{R}$, let w be a weight function on $\mathbf{R}$, and let $\langle I, J\rangle$ be an $\mathbf{R}$-instance. A sub-instance $\langle I, J'\rangle$ of $\langle I, J\rangle$ is a *maximum weight repair of* $\langle I, J\rangle$ *with respect to* Σ *and* w if $\langle I, J'\rangle$ has the following properties:

1. $\langle I, J'\rangle$ is consistent, i.e., $\langle I, J'\rangle$ satisfies every constraint in Σ.
2. J' has maximum weight, i.e., if $\langle I, J''\rangle$ is a consistent sub-instance of $\langle I, J\rangle$, then $\Sigma_{f\in J''} w(\langle I, J''\rangle, f) \leq \Sigma_{f\in J'} w(\langle I, J'\rangle, f)$.

In general, the weight function w may also depend on the set Σ of constraints at hand. If Σ and w are understood from the context, then we will simply talk about maximum weight repairs of $\langle I, J\rangle$, instead of maximum weight repairs of $\langle I, J\rangle$ with respect to Σ and w.

Thus, a maximum weight repair of $\langle I, J\rangle$ is a consistent sub-instance $\langle I, J'\rangle$ of $\langle I, J\rangle$ whose total sum of the weights of its $\mathbf{L}$-facts is maximum across all consistent sub-instances $\langle I, J''\rangle$ of $\langle I, J\rangle$. Note that the notion of maximum weight repairs introduced in Definition 2 differs from the standard notion of subset repairs [2] in two ways: first, in the standard notion, the repair takes place with respect to the entire schema or, more precisely, we have there that $\mathbf{S} = \emptyset$ and $\mathbf{R} = \mathbf{L}$; second, in the standard notion, there is no weight function on the facts. Note also that *maximum cardinality subset repairs* [21] are the special case of maximum weight repairs in which $\mathbf{S} = \emptyset$, $\mathbf{R} = \mathbf{L}$, and the weight function assigns weight 1 to each fact. Finally, note that our notion of maximum weight repairs differs also from the notion of maximum weight repairs introduced in [9] in the following way. In [9], the weight of each fact f depends on the inconsistent instance $\langle I, J\rangle$ under consideration, but remains the same on all consistent sub-instances of $\langle I, J\rangle$ containing f. In contrast, in Definition 2, the weight of each fact f may differ from instance to instance; thus, we may have $w(\langle I, J\rangle, f) \neq w(\langle I, J'\rangle, f)$, where $\langle I, J'\rangle$ is a consistent sub-instance of $\langle I, J\rangle$.

Maximum weight repairs give rise to a notion of consistent answers of queries in exactly the same way subset repairs do.

▶ **Definition 3.** Let Σ be a set of integrity constraints on $\mathbf{R}$ and let w be a weight function on $\mathbf{R}$. If q is a query on $\mathbf{R}$, and $\langle I, J\rangle$ is an $\mathbf{R}$-instance, then a tuple $\mathbf{a}$ is a *consistent answer of* q *on* $\langle I, J\rangle$ *with respect to* Σ *and* w if $\mathbf{a} \in q(\langle I, J'\rangle)$, for every maximum weight repair $\langle I, J'\rangle$ of $\langle I, J\rangle$ with respect to Σ and w.

3 Certain Links and Entity-Linking Frameworks

Here, we will focus on maximum weight repairs in declarative scenarios for entity linking, such as the ones considered in [7]. In such scenarios, $\mathbf{S}$ is the schema of *source* relations, while $\mathbf{L}$ is the schema of *link* relations, where each link relation is binary. Relation symbols in $\mathbf{S}$ will be referred to as *source* symbols, while relation symbols in $\mathbf{L}$ will be referred to as *link* symbols. Some source symbols may be interpreted by *built-in* relations, that is, such symbols may have the same interpretation on every allowable source instance. For example, a source symbol may stand for the substring relation between two strings, or it may stand for a user-defined predicate, such as similarity of names. If J is an $\mathbf{L}$-instance and $(a, b) \in L^J$ for some link symbol L in $\mathbf{L}$, then we say that (a, b) is a *link* of L in J. We sometimes write such a link as the fact $L^J(a, b)$, or $L(a, b)$ when J is clear from the context. We may also refer to J as a *link instance*.

▶ **Definition 4.** Let $\mathbf{S}$ be a schema of source symbols, let $\mathbf{L}$ be a schema of link symbols, let Σ be a set of integrity constraints on $\mathbf{R} = \mathbf{S} \cup \mathbf{L}$, and let w be a weight function on $\mathbf{R} = \mathbf{S} \cup \mathbf{L}$. If L is a link symbol in $\mathbf{L}$ and $\langle I, J\rangle$ is an $\mathbf{R}$-instance, then a *certain link of* L *on* $\langle I, J\rangle$ *with respect to* Σ *and* w is a consistent answer of the atomic query $L(x, y)$ on $\langle I, J\rangle$

with respect to Σ and w, i.e., a pair (a, b) such that $(a, b) \in L^{J'}$, for every maximum weight repair $\langle I, J' \rangle$ of $\langle I, J \rangle$ with respect to Σ and w.

We will also use the notation $L(a, b)$ for a certain link (a, b) of L. It will be clear from the context if $L(a, b)$ refers to a certain link or to a link $L^J(a, b)$ for some instance J.

Intuitively, in the above definition, we are given an instance $\langle I, J \rangle$, not necessarily consistent with respect to the set Σ of integrity constraints, where J represents an initial set of link facts. Then, the certain links of L on $\langle I, J \rangle$ represent precisely the subset of L-facts of J that appear in every maximum weight repair of $\langle I, J \rangle$. In this paper, we focus on links that are certain because it is a standard semantics in information integration, including data exchange and incomplete databases. While other alternatives may be considered (e.g., possible links, which are the links that appear in at least one maximum weight repair), we leave such investigation for future work.

Note that Definition 4 is very general and does not make any assumptions about the class of integrity constraints that is allowed in Σ or about the weight function w. We also note that the weight function w is assumed to be defined over instances of $\mathbf{R} = \mathbf{S} \cup \mathbf{L}$, independently of whether these instances are consistent with Σ or not.

The concrete choices for Σ and w will be incorporated into the notion of *entity-linking frameworks*, which we define next, together with the notion of *entity-linking specifications*.

▶ **Definition 5.** Let $\mathbf{S}$ be a schema of source symbols, let $\mathbf{L}$ be a schema of link symbols, and let $\mathbf{R} = \mathbf{S} \cup \mathbf{L}$.

- An *entity-linking framework on* $\mathbf{R}$ is a triple $(\mathcal{L}, \mathcal{S}, \mathcal{W})$ consisting of a logical language $\mathcal{L}$ on $\mathbf{R}$, a collection $\mathcal{S}$ of finite sets of $\mathcal{L}$-formulas, and a collection $\mathcal{W}$ of weight functions such that, for each $\Sigma \in \mathcal{S}$, there is a weight function w_Σ on $\mathbf{R}$.
- If Σ is a member of $\mathcal{S}$ and w_Σ is the associated weight function in $\mathcal{W}$, then we say that the triple $(\mathcal{L}, \Sigma, w_\Sigma)$ is an *entity-linking specification* in the entity-linking framework $(\mathcal{L}, \mathcal{S}, \mathcal{W})$.

Several different logical languages for expressing entity-linking specifications were introduced in [7] and then used to define and study different scenarios for declarative entity linking. Here, we show that all but one of the scenarios considered in [7] (namely, the scenario of maximal solutions) are concrete instances of the notion of an entity-linking framework in Definition 5, by choosing, in each case, the logical language $\mathcal{L}$, the collection $\mathcal{S}$ of finite sets of constraints from $\mathcal{L}$, and the collection $\mathcal{W}$ of weight functions. As we shall see, the weight functions can become progressively more sophisticated. Furthermore, the logical language $\mathcal{L}$ together with the collection $\mathcal{S}$ can become progressively richer.

We first focus on the language $\mathcal{L}_\mathbf{0}$ introduced in [7], consisting of three types of constraints:

- Inclusion dependencies of the form $L[X] \subseteq S[A]$ and $L[Y] \subseteq T[B]$, where L is a link symbol, and S and T are source symbols. We use X and Y to denote the first and the second attribute of L, while A and B denote attributes in relations S and T, respectively. Note that S and T could be the same source symbol.
- Functional dependencies (FDs) $L : X \to Y$ and $L : Y \to X$, where L is a link symbol and X and Y denote the attributes of L.
- Matching constraints of the form:

$$L(x, y) \to \forall \mathbf{u}(\psi(x, y, \mathbf{u}) \to \alpha_1 \vee \ldots \vee \alpha_k), \tag{1}$$

where L is a link symbol, $\psi(x, y, \mathbf{u})$ is a (possibly empty) conjunction of atomic formulas over $\mathbf{S}$ (with the requirement that the universally quantified variables $\mathbf{u}$ must occur in

ψ), and where α_i is of the form $\exists \mathbf{z_i}\ \phi_i(x, y, \mathbf{u}, \mathbf{z_i})$. Each ϕ_i is a conjunction of atomic formulas[2] over $\mathbf{S}$ along with equalities. We assume that the variables in $\mathbf{z_i}$ are disjoint from the variables in ψ and from $\{x, y\}$. Also, note that x and y are universally quantified, but for simplicity of notation we omit their quantifiers.

The intuition behind the use of disjunction in a matching constraint is that it lists all the possible matching conditions $\alpha_1, \ldots, \alpha_k$ for why a link $L(x, y)$ may exist (provided ψ holds). If a link $L(x, y)$ exists, then one or more of those conditions must be true. We do not require a matching constraint to be given for each link; for those links without a matching constraint, the link relation is implicitly defined by the rest of the constraints.

The inclusion dependencies have the important role of specifying the domain of values that can be used to populate a link relation. While in general there could be more than two inclusion dependencies for each link, all the scenarios considered in [7] focused on the case of exactly two inclusion dependencies and also on the case of exactly one matching constraint per link symbol. The next definition captures these requirements by introducing the collection $\mathcal{S}_0$; it also introduces an initial instance $\langle I, I^* \rangle$ that will be used repeatedly in the sequel (intuitively, as a superset for the repairs).

▶ **Definition 6.** Let $\mathbf{S}$ be a schema of source symbols and let $\mathbf{L}$ be a schema of link symbols.

- We write $\mathcal{S}_0$ to denote the collection of all finite sets Σ of $\mathcal{L}_\mathbf{0}$-formulas such that for each link symbol L, the set Σ contains one inclusion dependency on L for each of its attributes, contains zero, one or both functional dependencies on L, and at most one matching constraint on L.
- If I is an $\mathbf{S}$-instance, then we write I^* to denote the $\mathbf{L}$-instance defined as follows: for each link symbol L in $\mathbf{S}$, we have that $L^{I^*} = \pi_A(S^I) \times \pi_B(T^I)$, where A is the attribute of the source symbol S and B is the attribute of the target symbol T for which $\mathcal{L}_\mathbf{0}$ contains the inclusion dependencies $L[X] \subseteq S[A]$ and $L[Y] \subseteq T[B]$.

In the above definition, the instance $\langle I, I^* \rangle$ satisfies the inclusion dependencies of $\mathcal{L}_\mathbf{0}$ on each link symbol, but it need not satisfy the functional dependencies or the matching constraints of $\mathcal{L}_\mathbf{0}$.

While other combinations of constraints may also be meaningful (e.g., more than two inclusion dependencies per link, as mentioned above, or more than one matching constraint per link), the collection $\mathcal{S}_0$ is one of the simplest; it also has a good practical motivation, since it corresponds to entity linking statements in the HIL language [14].

We next give a concrete example taken from [7] of a set Σ of constraints in $\mathcal{S}_0$. We will make use of this example in the sequel.

▶ **Example 7.** In this scenario, we link subsidiaries in one database with parent companies in another database. Consider the following source schema $\mathbf{S}$:

$$\texttt{Subsid}(sid, sname, location) \qquad \texttt{Company}(cid, cname, hdqrt)$$
$$\texttt{Exec}(eid, cid, name, title)$$

This source schema includes the relation symbols `Subsid` from the first database, and `Company` and `Exec` from the second database. The link schema $\mathbf{L}$ consists of a single link relation $L(sid, cid)$. The following set Σ of constraints can be used to specify declaratively the properties of the link relation in terms of the source relations. First, Σ contains two inclusion

[2] Note that some of these atomic formulas may involve built-in relations.

dependencies $L[sid] \subseteq \texttt{Subsid}[sid]$, $L[cid] \subseteq \texttt{Company}[cid]$, and the functional dependency $L : sid \rightarrow cid$. While the inclusion dependencies specify where L is allowed to take values from, the functional dependency gives the additional requirement that the links must be many-to-one from *sid* to *cid* (i.e., every subsidiary must link to at most one parent company). Additionally, Σ includes the matching constraint:

$$\begin{aligned} L(sid, cid) \rightarrow \forall sn, loc, cn, hd\ (&\texttt{Subsid}(sid, sn, loc) \wedge \texttt{Company}(cid, cn, hd) \\ &\rightarrow (sn \sim cn) \\ &\vee \\ &\exists e, n, t\,(\texttt{Exec}(e, cid, n, t)\ \wedge\ contains(t, sn))\ ), \end{aligned}$$

which lists all possible reasons as to why a link may exist. Concretely, if a subsidiary id (*sid*) and a company id (*cid*) are linked, then for every binding of `Subsid` and `Company` source tuples where *sid* and *cid* respectively occur, it must be that one of the two matching conditions holds: (1) there is a similarity in the names, as specified by $sn \sim cn$, or (2) there is some executive working for the company and this executive has a title that contains the subsidiary's name.

3.1 Concrete Entity-Linking Frameworks Based on $\mathcal{L}_0$

We are now in a position to define several concrete entity-linking frameworks by instantiating the general concepts introduced above. We first consider three different entity-linking frameworks obtained from $\mathcal{L}_\mathbf{0}$ and $\mathcal{S}_0$ by using three different types of weight functions.

▶ **Framework 8.** *The entity-linking framework* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$ *of maximum cardinality repairs.*

Let $\mathbf{1}$ *be the weight function on* $\mathbf{R}$ *such that* $\mathbf{1}(\langle I, J\rangle, L^J(a, b)) = 1$*, for every* $\mathbf{R}$*-instance* $\langle I, J\rangle$ *and every fact* $L^J(a, b)$*. Consider the entity-linking framework* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$*, where, for each* $\Sigma \in \mathcal{S}$*, we have that* $w_\Sigma = \mathbf{1}$*.*

A maximum weight repair of $\langle I, I^*\rangle$ *with respect to* Σ *and* $\mathbf{1}$ *is a repair that maximizes the total cardinality of the link facts. We call such repairs* maximum cardinality repairs*.*

It can be verified that if $\langle I, J\rangle$ is such a maximum cardinality repair of $\langle I, I^*\rangle$, then J is a maximal solution for I, as defined in [7]. The converse, however, does not always hold. Like maximal solutions, the notion of maximum cardinality repairs suffers from the deficiency that they give rise to "too few" certain links. This can be seen in the following example from [7].

▶ **Example 9.** Assume the same schemas and constraints as in Example 7. A source instance I for $\mathbf{S}$ is given below as a set of facts:

$\texttt{Subsid}(s_1, \text{“Citibank N.A.”}, \text{“New York”})$	$\texttt{Company}(c_1, \text{“Citigroup Inc”}, \text{“New York”})$
$\texttt{Subsid}(s_2, \text{“CIT Bank”}, \text{“Salt Lake City”})$	$\texttt{Company}(c_2, \text{“CIT Group Inc”}, \text{“New York”})$
	$\texttt{Exec}(e_1, c_1, \text{“E. McQuade”}, \text{“CEO, Citibank N.A.”})$

In the above, 'Citigroup Inc" and "CIT Group Inc" are two different parent companies, and "Citibank N.A." is the name of a true subsidiary of "Citigroup Inc", while "CIT Bank" is the name of a true subsidiary of "CIT Group Inc". The goal of entity linking is to identify links such as $L(s_1, c_1)$ and $L(s_2, c_2)$.

It can be seen that, given our set Σ of constraints, there are exactly four maximum cardinality repairs for $\langle I, I^*\rangle$, namely $\langle I, J_i\rangle$, $i = 1, 4$, where the J_i's are as follows:

$$J_1 = \{L(s_1, c_1), L(s_2, c_1)\} \qquad J_2 = \{L(s_1, c_1), L(s_2, c_2)\}$$
$$J_3 = \{L(s_1, c_2), L(s_2, c_1)\} \qquad J_4 = \{L(s_1, c_2), L(s_2, c_2)\}$$

It is assumed here that the name similarity predicate $\sim$ evaluates to true for all pairs of subsidiary name and company name occurring in our instance I (thus, "Citibank N.A." $\sim$ "Citigroup Inc" but also "Citibank N.A." $\sim$ "CIT Group Inc", and so on).

It follows that the set of certain links of L on $\langle I, I^* \rangle$ w.r.t. Σ and $\mathbf{1}$ is empty: there is no link that appears in all four maximum cardinality repairs and, hence, no link qualifies as a certain link. However, some links are clearly stronger than others. In particular, the link $L(s_1, c_1)$ relating "Citibank N.A." to "Citigroup Inc." satisfies both the $\sim$ predicate and the `Exec`-based matching constraint, while the other links satisfy only the $\sim$ predicate. Intuitively, there is evidence that suggests that $L(s_1, c_1)$ is a strong link that should be differentiated from the other links. However, the constant weight function $\mathbf{1}$ does not provide such differentiation.

The above example illustrates the need for more refined notions of weights on links.

▶ **Framework 10.** *The entity-linking framework* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ *of maximum-value solutions.*

For each $\Sigma \in \mathcal{S}_0$*, consider the following weight function* w_Σ*. Given an* $\mathbf{R}$*-instance* $\langle I, J \rangle$ *and a fact* $L^J(a, b)$*, we distinguish the following cases:*

1. *If* $L(a, b)$ *does not satisfy the inclusion dependencies, then* $w_\Sigma(\langle I, J \rangle, L^J(a, b)) = 0$*. Otherwise:*
   - **(a)** *If* Σ *contains no matching constraint for* L*, then* $w_\Sigma(\langle I, J \rangle, L^J(a, b)) = 1$*.*
   - **(b)** *If* Σ *contains a matching constraint for* L *(which, by the definition of* $\mathcal{S}_0$*, is the only such matching constraint) and if* (a, b) *does not satisfy the right-hand side of the matching constraint for* L*, then* $w_\Sigma(\langle I, J \rangle, L^J(a, b)) = 0$*.*
   - **(c)** *If* Σ *contains a matching constraint for* L *and if* (a, b) *satisfies the right-hand side of the matching constraint for* L*, then* $w_\Sigma(\langle I, J \rangle, L^J(a, b)) = \mathit{Val}(L^J(a, b))$*, as defined in Section 5.2 of [7]. The precise definition of Val is as follows.*

     First, let us recall that the matching constraint for L *has the form (1). Assume that there is no instantiation* $\mathbf{u_0}$ *of the vector of universally quantified variables* $\mathbf{u}$ *such that* $I \models \psi(a, b, \mathbf{u}_0)$*. This means that the matching constraint for* $L(a, b)$ *is satisfied for vacuous reasons. As in the earlier case of no matching constraint, we take the value of the link to be* 1*. In all other cases, we let the value of the link be:*

$$\mathit{Val}(L^J(a, b)) = \min_{\mathbf{u}_0}\Big(\sum_{\alpha_i, \mathbf{z}_0} 1\Big). \tag{2}$$

     In the above, $\mathbf{u}_0$ *ranges over all the distinct instantiations of the vector of universally quantified variables* $\mathbf{u}$ *such that* $I \models \psi(a, b, \mathbf{u}_0)$*. We take the minimum, over all such* $\mathbf{u}_0$*, of the* strength *with which the source instance* I *satisfies the disjunction* $\alpha_1 \vee \ldots \vee \alpha_k$*. This strength is defined as a sum that gives a value of* 1 *for* every *distinct combination of a disjunct* α_i *such that* I *satisfies* $\alpha_i(a, b, \mathbf{u}_0)$*, and distinct instantiation* $\mathbf{z}_0$ *of the vector* $\mathbf{z}$ *of existentially quantified variables of* α_i *that makes the satisfaction of* α_i *hold. (Recall that* α_i *is, in general, of the form* $\exists \mathbf{z}\ \phi_i(x, y, \mathbf{u}, \mathbf{z})$*.) In the case when* α_i *is satisfied and the existentially quantified variables are missing, then we count only* 1*.*

     We can see that, intuitively, the sum in formula (2) calculates the strength of a link by counting the number of satisfied disjuncts together with the evidence (i.e., the number of existential witnesses). Taking the minimum guarantees that we take the weakest strength among all $\mathbf{u}_0$*.*

We remark that the weights $w_\Sigma(\langle I, J \rangle, L^J(a, b))$ do not actually depend on J.

If we revisit the earlier Example 9, we have that $\mathrm{Val}(L^{J_1}(s_1, c_1)) = 2$, since $L^{J_1}(s_1, c_1)$ satisfies both disjuncts in the matching constraint, while $\mathrm{Val}(L^{J_1}(s_2, c_1)) = 1$. Thus, the total weight of the link instance J_1 is 3. Similarly, the other link instance containing $L(s_1, c_1)$,

namely J_2, also has weight 3. The remaining link instances J_3 and J_4 have weight of 2. Hence, $\langle I, J_1\rangle$ and $\langle I, J_2\rangle$ are the two maximum weight repairs of $\langle I, I^*\rangle$ in this example. It follows that there is precisely one certain link of L on $\langle I, I^*\rangle$ w.r.t. Σ and the weight function w_Σ, namely $L(s_1, c_1)$. This is in contrast with the earlier case of maximum cardinality repairs, where we had no certain links.

Consider the above entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$. It is easy to verify that if I is an $\mathbf{S}$-instance, then the following statements are equivalent for an $\mathbf{L}$-instance J:

1. $\langle I, J\rangle$ is a maximum weight repair of $\langle I, I^*\rangle$ with respect to Σ and w_Σ.
2. J is a maximum-value solution for I with respect to Σ, as defined in [7].

It follows that the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ coincides with the entity-linking scenario given by $\mathcal{L}_\mathbf{0}(\oplus)$ in [7].

▶ **Framework 11.** *The entity-linking frameworks $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{w})$ of maximum-value solutions with weighted disjuncts.*

For each matching constraint $L(x, y) \to \forall\mathbf{u}(\psi(x, y, \mathbf{u}) \to \alpha_1 \vee \ldots \vee \alpha_k)$ of $\mathcal{L}_\mathbf{0}$ and for each disjunct $\alpha_i ::= \exists\mathbf{z}\ \phi_i(x, y, \mathbf{u}, \mathbf{z})$, let $w_{\phi_i}(x, y, \mathbf{u}, \mathbf{z})$ be a function that returns non-negative numbers. Intuitively, with each disjunct that returns true or false, we also have a function that computes a weight for that disjunct. This collection of functions w_{ϕ_i} gives rise to a weight function $\mathcal{V}_\mathbf{w}$ that is computed as in the case of $\mathcal{V}_0$ except that in formula (2) we replace the number 1 by $w_{\phi_i}(a, b, \mathbf{u}_0, \mathbf{z}_0)$.

Note that each different collection of functions w_{ϕ_i} gives rise to a different entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{w})$. This family of frameworks captures the entity-linking scenarios given by $\mathcal{L}_\mathbf{0}(\oplus, \mathbf{w})$, which, as discussed in [7], is of special interest because of its connection to probabilistic methods for entity resolution, including those based on Markov Logic Networks (MLNs) [22].

Next, we state a general theorem for enumerating all maximum weight repairs with polynomial delay and for computing the certain links in polynomial time. Several results in [7], including Theorem 5.4, are special cases of this theorem.

▶ **Theorem 12.** *Let $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W})$ be an entity-linking framework such that for each $\Sigma \in \mathbf{S}_0$, for each $\mathbf{S}$-instance I, for each sub-instance J of I^*, and for each fact $L^J(a, b)$, we have that $w_\Sigma(\langle I, I^*\rangle, L^{I^*}(a, b)) = w_\Sigma(\langle I, J\rangle, L^J(a, b))$. Then the following statements are true.*

1. *There is a polynomial-delay algorithm that, given an $\mathbf{S}$-instance I, enumerates the maximum weight repairs of $\langle I, I^*\rangle$.*
2. *There is a polynomial-time algorithm that, given an $\mathbf{S}$-instance I, computes the certain links of $\langle I, I^*\rangle$ with respect to Σ and w_Σ.*

Note that the hypothesis of Theorem 12 is satisfied by the preceding three entity-linking frameworks. In particular, in all three frameworks, the weight of a link fact does not depend on the link instance J in which it appears. The proof of Theorem 12 is essentially the same as the proof of Theorem 5.4 in [7], where the problem is reduced to computing and enumerating maximum-weight matchings in undirected weighted bipartite graphs.

3.2 Collective Entity-Linking Frameworks

We now consider a language $\mathcal{L}_\mathbf{c}$ that is richer than $\mathcal{L}_\mathbf{0}$ and allows for link relations to appear in the right-hand side of matching constraints. Thus, the language $\mathcal{L}_\mathbf{c}$ allows us to express what is usually called *collective entity linking* [5], that is, the process of creating or specifying multiple inter-dependent links.

Concretely, in $\mathcal{L}_{\mathbf{c}}$, the matching constraint for a link symbol L has the same form

$$L(x, y) \rightarrow \forall \mathbf{u}(\psi(x, y, \mathbf{u}) \rightarrow \alpha_1 \vee \ldots \vee \alpha_k)$$

as in $\mathcal{L}_{\mathbf{0}}$, with the difference that in each disjunct $\alpha_i ::= \exists \mathbf{z}\ \phi_i(x, y, \mathbf{u}, \mathbf{z})$, the formula ϕ_i can now be a conjunction of source *and link* atomic formulas, along with equalities. Thus, the matching constraint for L is allowed to refer to other link symbols (possibly, including L itself). As an example, which we give shortly, in $\mathcal{L}_{\mathbf{c}}$ one can express matching constraints to specify both publication links and venue links, where the matching constraint for publication links may depend on the links between venues, and the matching constraint for venue links may depend on the links between publications.

Based on the language $\mathcal{L}_{\mathbf{c}}$, we can define two entity-linking frameworks, one that does not allow for recursion among the links, and one that does allow for recursion.

▶ **Framework 13.** *The entity-linking framework* $(\mathcal{L}_{\mathbf{c}}, \mathcal{S}_{\mathbf{1}}, \mathcal{V}_{\mathbf{1}})$ *for recursion-free collective entity linking.*

In this framework, $\mathcal{S}_1$ *is the collection of all finite sets of constraints from* $\mathcal{L}_{\mathbf{c}}$*, such that for each link symbol* L*, the set* Σ *contains the two inclusion dependencies on* L*, it contains zero, one or two functional dependencies on* L*, and at most one matching constraint on* L*. Additionally, we require that there is no recursion through the links. Thus, for each* Σ *in* $\mathcal{S}_1$*, there is implicitly a hierarchy of link symbols, and a matching constraint for* L *may call only links that are strictly lower in the hierarchy than* L*. Additionally,* $\mathcal{V}_1$ *is the collection of weight functions that associates with each* Σ *in* $\mathcal{S}_1$ *a weight function* w_Σ *defined in the same way as in the entity-linking framework* $(\mathcal{L}_{\mathbf{0}}, \mathcal{S}_{\mathbf{0}}, \mathcal{V}_{\mathbf{0}})$*.*

▶ **Framework 14.** *The entity-linking framework* $(\mathcal{L}_{\mathbf{c}}, \mathcal{S}_{\mathbf{2}}, \mathcal{V}_{\mathbf{2}})$ *for recursive collective entity linking is defined in the same way as* $(\mathcal{L}_{\mathbf{c}}, \mathcal{S}_{\mathbf{1}}, \mathcal{V}_{\mathbf{1}})$ *except that* $\mathcal{S}_2$ *allows recursion through the links.*

▶ **Example 15.** Consider a bibliographic example from [7], where we link papers (from one database) with articles (from another database), while also linking the corresponding venues. The source schema $\mathbf{S}$ consists of $\texttt{Paper}(pid, title, venue, year)$ and $\texttt{Article}(ano, title, journal, year)$. Here, `pid` is a unique id assigned to `Paper` records, while `venue` could be a conference, a journal, or some other place of publication. The `Article` relation represents publications that appeared in journals, and `ano` is a unique id assigned to such records. The link schema $\mathbf{L}$ consists of two relations: $\texttt{PaperLink}\ (pid, ano)$ and $\texttt{VenueLink}\ (venue, journal)$. The first relation is intended to link paper ids from `Paper` with article numbers from `Article`, when they represent the same publication. The second relation is intended to relate `journal` values that occur in `Article` (e.g., "ACM TODS") to journal values that occur under the `venue` field in `Paper` (e.g., "TODS").

A possible entity linking specification in the framework $(\mathcal{L}_{\mathbf{c}}, \mathcal{S}_{\mathbf{2}}, \mathcal{V}_{\mathbf{2}})$ is $(\mathcal{L}_{\mathbf{c}}, \Sigma, w_\Sigma)$, where Σ contains the following two matching constraints:

$$\begin{aligned}
&\texttt{VenueLink}(ven, jou) \rightarrow (ven \sim_1 jou) \\
&\qquad\qquad \vee \exists pid, t_1, y_1, ano, t_2, y_2\ (\ \texttt{Paper}(pid, t_1, ven, y_1) \\
&\qquad\qquad\qquad\qquad \wedge \texttt{Article}(ano, t_2, jou, y_2) \\
&\qquad\qquad\qquad\qquad \wedge \texttt{PaperLink}(pid, ano)\ )
\end{aligned}$$

$$\begin{aligned}
&\texttt{PaperLink}(pid, ano) \rightarrow \\
&\qquad \forall t_1, ven, y_1, t_2, jou, y_2\ (\ \texttt{Paper}(pid, t_1, ven, y_1) \wedge \texttt{Article}(ano, t_2, jou, y_2) \\
&\qquad\qquad\qquad \rightarrow ((t_1 \sim_2 t_2) \wedge (y_1 = y_2)) \\
&\qquad\qquad\qquad \vee ((t_1 \sim_2 t_2) \wedge \texttt{VenueLink}(ven, jou))\ )
\end{aligned}$$

The first constraint specifies that we may link a venue with a journal only if their string values are similar (via some similarity predicate $\sim_1$), or if there are papers and articles that have been published in the respective venue and journal and that are linked via `PaperLink`. The second constraint specifies that we may link a paper with an article only if their titles are similar (via a similarity predicate $\sim_2$) and their years of publication match exactly, or if their titles are similar and their venues of publications are linked via `VenueLink`.

Additionally, Σ includes two functional dependencies on `PaperLink`: $pid \rightarrow ano$, $ano \rightarrow pid$, to reflect that each paper id in `Paper` must match to at most one article number in `Article`, and vice-versa. We do not require any functional dependencies on `VenueLink`; thus, we could have multiple venue strings in `Paper` matching with a journal string in `Article`, and vice-versa. We also include in Σ the expected inclusion dependencies from the link attributes to the corresponding source attributes (e.g., $\texttt{PaperLink}[pid] \subseteq \texttt{Paper}[pid]$).

With a simple modification, where we remove the second disjunct in the matching constraint for `PaperLink`, we obtain a different entity linking specification that is in the recursion-free collective entity-linking framework $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{1}, \mathcal{V}_\mathbf{1})$.

We point out that the entity-linking framework $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{1}, \mathcal{V}_\mathbf{1})$ coincides with the entity-linking scenario given by $\mathcal{L}_\mathbf{1}(\oplus)$ in [7], while entity-linking framework $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{2}, \mathcal{V}_\mathbf{2})$ coincides with the entity-linking scenario given by $\mathcal{L}_\mathbf{2}(\oplus)$ in [7].

For the above two entity-linking frameworks, it is important to note that the weight functions depend on the link instance in a crucial way. In particular, the hypothesis of the preceding Theorem 12, stating that the weight of a link fact only depends on I^* and not on the link instance J, is no longer satisfied. In fact, as shown in [7] (Theorem 7.3), Theorem 12 fails even for $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{1}, \mathcal{V}_\mathbf{1})$, unless NP = coNP.

4 Comparing the Expressive Power of Entity-Linking Frameworks

The notion of certain links makes it possible to compare the expressive power of entity-linking frameworks. In the next definition, we first introduce the notion of *certain-link equivalence* between entity-linking specifications. This notion is of interest as a tool to compare entity-linking specifications in a way other than logical equivalence (which may be too strict for entity linking purposes). The second part of the definition then makes use of certain-link equivalence to define a notion of *subsumption* between entity-linking frameworks.

▶ **Definition 16.** Let $\mathbf{S}$ be a schema of source symbols, let $\mathbf{L}$ be a schema of link symbols, let $\mathbf{R} = \mathbf{S} \cup \mathbf{L}$. Assume that $\mathcal{F} = (\mathcal{L}, \mathcal{S}, \mathcal{W})$ and $\mathcal{F}' = (\mathcal{L}', \mathcal{S}', \mathcal{W}')$ are two entity-linking frameworks on $\mathbf{R}$.

- Let $\mathcal{E} = (\mathcal{L}, \Sigma, w_\Sigma)$ be an entity-linking specification in $\mathcal{F}$, and let $\mathcal{E}' = (\mathcal{L}', \Sigma', w_{\Sigma'})$ be an entity-linking specification in $\mathcal{F}'$. We say that $\mathcal{E}$ *and* $\mathcal{E}'$ *are certain-link equivalent* if for every link symbol L in $\mathbf{L}$ and every $\mathbf{R}$-instance $\langle I, J\rangle$, we have that the certain links of L on $\langle I, J\rangle$ with respect to Σ and w_Σ coincide with the certain links of L on $\langle I, J\rangle$ with respect to Σ' and $w_{\Sigma'}$.
- We say that $\mathcal{F}$ *is subsumed by* $\mathcal{F}'$, denoted $\mathcal{F} \preceq \mathcal{F}'$, if for every entity-linking specification $\mathcal{E}$ of $\mathcal{F}$ there is an entity-linking specification $\mathcal{E}'$ of $\mathcal{F}'$ such that $\mathcal{E}$ and $\mathcal{E}'$ are certain-link equivalent. Otherwise, we say that $\mathcal{F}$ *is not subsumed by* $\mathcal{F}'$, and write $\mathcal{F} \not\preceq \mathcal{F}'$.
- We say that $\mathcal{F}$ *is strictly subsumed by* $\mathcal{F}'$ if $\mathcal{F} \preceq \mathcal{F}'$, but $\mathcal{F}' \not\preceq \mathcal{F}$.

We note that a weaker notion of subsumption was considered implicitly in [7] for concrete entity linking scenarios. In this weaker notion, certain-link equivalence holds for repairs of the instance $\langle I, I^*\rangle$ instead of arbitrary instances $\langle I, J\rangle$. In effect, Theorem 6.2 in [7] asserts

that *linear* MLNs, an important special case of MLNs, are subsumed under this weaker notion of subsumption by an entity linking framework of maximum-value solutions with weighted disjuncts, where the matching constraints are in the existential fragment $\exists\mathcal{L}_\mathbf{0}$ of the language $\mathcal{L}_\mathbf{0}$.

We note that for of all our subsumption results (Theorems 17, 18, 19, and 23), whenever we prove failure of subsumption, we actually prove it in a stronger sense, by showing that it fails even under the weaker notion.

The next two theorems say that the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ of maximum-value solutions and the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$ of maximum cardinality repairs are incomparable in expressive power, in that neither subsumes the other.

▶ **Theorem 17.** *The entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ of maximum-value solutions is not subsumed by the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$ of maximum cardinality repairs.*

Proof. (Hint) Our entity-linking specification in $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ that is not certain-link equivalent to any entity-linking specification in $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$ has one link symbol L, the matching constraint $L(x, y) \rightarrow R(x, y) \vee S(x, y) \vee T(x, y)$, the FD $L : X \rightarrow Y$, and the inclusion dependencies $L[X| \subseteq D$ and $L[Y| \subseteq D$. ◀

▶ **Theorem 18.** *The entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$ of maximum cardinality repairs is not subsumed by the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ of maximum-value solutions.*

Proof. (Hint) Our entity-linking specification in $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{W}_\mathbf{1})$ that is not certain-link equivalent to any entity-linking specification in $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ has one link symbol L, the matching constraint $L(x, y) \rightarrow R(x, y) \vee S(x, y)$, the FD $L : X \rightarrow Y$, and the inclusion dependencies $L[X| \subseteq D_1$ and $L[Y| \subseteq D_2$. ◀

By definition, the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ is subsumed by the entity-linking framework $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{1}, \mathcal{V}_\mathbf{1})$. The next theorem says that this subsumption is strict. This means that allowing for link relations to appear on the right-hand side of matching constraints gives strictly more expressive power than not allowing this, even when the dependencies among the link relations are non-recursive.

▶ **Theorem 19.** *The entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ of maximum-value solutions is strictly subsumed by the entity-linking framework $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{1}, \mathcal{V}_\mathbf{1})$ for recursion-free collective entity linking.*

Proof. (Hint) Our entity-linking specification in $(\mathcal{L}_\mathbf{c}, \mathcal{S}_\mathbf{1}, \mathcal{V}_\mathbf{1})$ that is not certain-link equivalent to any entity-linking specification in $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ has two link symbols L_1 and L_2, ,the matching constraints $L_1(x, y) \rightarrow (S(x, y) \rightarrow (L_2(x, y) \wedge R(x, y))$ and $L_2(x, y) \rightarrow (P(x, y) \rightarrow T(x, y))$ and the inclusion dependencies $L_1[X| \subseteq D$, $L_1[Y| \subseteq D$, $L_2[X] \subseteq D$, and $L_2[Y] \subseteq D$. There are no FDs. ◀

5 Adding Preference Constraints

In this section, we introduce a family of entity-linking frameworks $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$ that is parameterized by a set of Π *preference constraints*. This family of frameworks can be seen as an extension of the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$, where we use a more refined collection of weight functions that also take into account preferences among the link facts.

We first introduce the language of preference constraints from which Π is drawn. The main motivation for such preference constraints is that they allow a user to specify explicitly

whether some link facts should be considered stronger than other link facts. Such preference constraints are given independently of, and in addition to, the set Σ of constraints in $\mathcal{S}_0$, and will be used to further differentiate among conflicting links (i.e., pairs of link facts that violate one or both of the functional dependencies on a link relation).

A *preference constraint* has the following general form:

$$L(x,y) \wedge L(x',y') \wedge \ \alpha(x,y) \wedge \neg\alpha(x',y') \rightarrow L(x,y) \geq L(x',y') \tag{3}$$

In the above, L can be any of the link symbols in $\mathbf{L}$ while $\alpha(x,y)$ can be any predicate of the form $\exists \mathbf{z}\ \phi(x,y,\mathbf{z})$, where ϕ is a conjunction of source atomic formulas along with equalities.

▶ **Example 20.** Consider a variation of the earlier Example 7 linking subsidiaries with companies, where the set Σ of constraints is as follows. The functional and inclusion dependencies are as before. However, the matching constraint is simplified, for the purposes of this example, so that it now requires only the similarity of the subsidiary name and company name:

$$L(sid,cid) \rightarrow \forall sn, loc, cn, hd\ (\texttt{Subsid}(sid,sn,loc) \wedge \texttt{Company}(cid,cn,hd) \rightarrow (sn \sim cn)\ .$$

We now consider, additionally, a set Π consisting of a single preference constraint, which uses an `Exec`-based condition to differentiate among links:

$$\begin{aligned} & L(sid,cid) \wedge L(sid',cid') \\ & \quad \wedge \exists e,n,t,sn,loc\ (\texttt{Exec}(e,cid,n,t) \wedge \texttt{Subsid}(sid,sn,loc)\ \ \wedge \mathit{contains}(t,sn))\ \\ & \quad \wedge \neg\exists e,n,t,sn,loc\ (\texttt{Exec}(e,cid',n,t) \wedge \texttt{Subsid}(sid',sn,loc)\ \wedge \mathit{contains}(t,sn))\ \\ & \quad \rightarrow L(sid,cid) \geq L(sid',cid') \end{aligned}$$

Thus, whenever we have two links relating a subsidiary with a company, if one of the links satisfies the fact that the company has an executive whose title contains the subsidiary name, while the other link does not satisfy such fact, we prefer the first link over the second link.

Note that a user has the freedom, in general, to choose which conditions to push into the matching constraints of Σ and which ones into the preference constraints of Π. This is manifested, in this example, via the fact that the executive information is used in a preference constraint whereas before it was used as part of a matching constraint.

The notion of a consistent instance when there are preference constraints continues to be the same as that of a consistent instance with respect to an entity-linking specification in $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ where there are no preference constraints. Thus, the set Π of preference constraints plays no role in defining consistent instances under $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$. However, Π plays an important role in defining the weight functions for the links, as we see next.

We are now ready to formally define $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$. First, we recall from Section 3 the instance $\langle I, I^* \rangle$, which for a given source instance I, represents a superset for the repairs that we consider. Thus, I^* represents the domain for all the links that may appear in link relations.

▶ **Framework 21.** *The family of entity-linking frameworks* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$ *with preference constraints.*

For every fixed finite set Π *of preference constraints, we define an entity-linking framework* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$*, by assigning to each* $\Sigma \in \mathcal{S}_0$ *a weight function* $w_{\Sigma,\Pi}$ *that depends on both* Σ *and* Π*. Given an* $\mathbf{R}$*-instance* $\langle I, J \rangle$ *and a fact* $L^J(a,b)$*, we define* $w_{\Sigma,\Pi}(\langle I, J \rangle, L^J(a,b))$ *to be* $w_{\Sigma,\Pi}(\langle I, I^* \rangle, L^{I^*}(a,b))$*, which in turn is defined as follows.*

For each link symbol L, and source instance I, we first compute a preference relation $\geq_L$ on I^* on conflicting links of L, by evaluating each preference constraint of the form (3) that involves L. Concretely, whenever (x_0, y_0) and (x'_0, y'_0) are pairs in I^* such that $L(x_0, y_0)$ and $L(x'_0, y'_0)$ are conflicting (i.e., together violate one or both of the functional dependencies on L), and such that $\alpha(x_0, y_0)$ is true in I but $\alpha(x'_0, y'_0)$ is not true in I, we set $L(x_0, y_0) \geq_L L(x'_0, y'_0)$. In general, $\geq_L$ can have cycles. For example, we can have two distinct pairs $l = L(x_0, y_0)$ and $l' = L(x'_0, y'_0)$ such that $l \geq_L l'$ and $l' \geq_L l$. Such situation may arise when a user gives (at least) two preference constraints for L, the evaluation of which leads to opposite preferences for the particular links.

We then turn $\geq_L$ into an acyclic relation $>_L$ as follows. First, we take the transitive closure $\geq_L^*$ of $\geq_L$. Then, we set $l >_L l'$ whenever $l \geq_L^* l'$ but it is not the case that $l' \geq_L^* l$. Intuitively, $l >_L l'$ means that l is strictly preferred to l'. It can be verified that, for each L, the relation $>_L$ (or rather its inverse $<_L$) forms a strict partial order. We may also drop the subscript L and use the notation $>$ or $(\geq)$ whenever L is understood from the context. We may refer to $>$ as the preference relation.

The weight of a link fact l in I^* is then defined recursively:

$$w_{\Sigma,\Pi}(\langle I, I^*\rangle, l) = w_\Sigma(\langle I, I^*\rangle, l) + \sum_{l > l'} w_{\Sigma,\Pi}(\langle I, I^*\rangle, l'),$$

where w_Σ is the weight function associated with Σ in the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ of maximum-value solutions. Thus, the weight of l is obtained by adding up $w_\Sigma(\langle I, I^*\rangle, l)$, which is calculated solely based on Σ as defined for $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$, with the total aggregated weight of all the links that l dominates (via the preference relation $>$). In the special case when there are no preference constraints, the weight of a link l falls back to $w_\Sigma(\langle I, I^*\rangle, l)$. Thus, for each Π, the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$ is an extension of the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$.

Note that, by definition, the weight of a link is relative to $\langle I, I^*\rangle$, on which we evaluated the preference constraints, but independent of any particular sub-instance $\langle I, J\rangle$. Thus, the hypothesis of Theorem 12 holds, by definition.

▶ **Example 22.** Recall the specification in Example 20. First, it is immediate to see that this is an example of an entity-linking specification in the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$, for the given set Π of preference constraints. Moreover, let us assume the same source instance I as in Example 9. The link $L(s_1, c_1)$ strictly dominates the link $L(s_1, c_2)$ (by the fact that c_1 satisfies the `Exec` condition for s_1 in the preference constraint, while c_2 does not). Since no other strict domination holds, we have that $w_{\Sigma,\Pi}(\langle I, I^*\rangle, L^{I^*}(s_1, c_1)) = 2$, while the weight of any other link is 1. As a consequence, among the four maximal cardinality repairs for $\langle I, I^*\rangle$ that we have seen earlier, we have that $\langle I, J_1\rangle$ and $\langle I, J_2\rangle$ have weight 3, while $\langle I, J_3\rangle$ and $\langle I, J_4\rangle$ have weight 2. Thus, $\langle I, J_1\rangle$ and $\langle I, J_2\rangle$ are the maximum weight repairs with respect to Σ and $w_{\Sigma,\Pi}$. As a result, we also obtain that $L(s_1, c_1)$ is the sole certain link, in this example.

As we noted above, the hypothesis of Theorem 12 holds for $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$ and so we obtain, as a corollary, a polynomial-delay algorithm for the enumeration of maximum weight repairs and a polynomial-time algorithm for the computation of the certain links.

It is clear that every entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ (Framework 10) can be simulated by using an entity-linking framework involving preferences (Framework 21) by simply taking the set Π of preferences to be empty. The next theorem says that, in fact, we gain expressive power by allowing preference constraints. This is our main technical result.

▶ **Theorem 23.** *There is a finite set* Π *of preference constraints such that the corresponding framework* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$ *is not subsumed by the entity-linking framework* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ *of maximum-value solutions.*

A key tool in the proof of Theorem 23 is a locality theorem that is interesting in its own right, and that we use multiple times in the proof of Theorem 23. We first need some preliminaries. For each entry a in a fact in an instance I, define $N_0(a)$ to be $\{a\}$. Inductively, define $N_{i+1}(a)$ to consist of $N_i(a)$ along with each c such that there is a' in $N_i(a)$ where a' and c are both entries in some fact in I. Thus $N_r(a)$ consists of those entries of I within distance r of a in the Gaifman graph [20] of I. Let $N_r(a,b)$ be $N_r(a) \cup N_r(b)$. We may refer to $N_r(a,b)$ as an *r-neighborhood*.

▶ **Theorem 24** (Locality Theorem)**.** *Let* $\mathcal{E}$ *be an entity-linking specification in* $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$*, with link symbol* L*. Then there is* r*, depending only on* $\mathcal{E}$*, such that for every source instance* I*, if* $I \restriction N_r(a_1,b_1)$ *and* $I \restriction N_r(a_2,b_2)$ *are isomorphic under an isomorphism* f *with* $f(a_1) = a_2$ *and* $f(b_1) = b_2$*, then the weights of the links* $L(a_1,b_1)$ *and* $L(a_2,b_2)$ *in* $\mathcal{E}$ *are the same.*

By $I \restriction N_r(a_i,b_i)$ we mean the usual notion of the restriction of I to the domain $N_r(a_i,b_i)$. The proof of the Locality theorem makes use of the Gaifman locality theorem for first-order logic [12], and extensions of that theorem to logics with counting by Libkin [19]. Our proof of the Locality Theorem depends on a certain uniformity in the choice of r.

Sketch of the proof of Theorem 23. Our entity-linking specification $\mathcal{E}$ in the framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{P}_\mathbf{\Pi})$ has one link symbol L, the matching constraint $L(x,y) \to R(x,y)$, both FDs on L, and the inclusion dependencies $L[X| \subseteq R[X]$ and $L[Y| \subseteq R[Y]$. We define a family of source instance K_r and a set of preference constraints such that we get two long chains $L(0,1) > L(2,3) > L(4,5) > \cdots > L(m,m+1)$ and $L(0,1') > L(2',3') > L(4',5') > \cdots > L(n',(n+1)')$ of strict preferences, where $m > n$ (so the first chain is longer than the second). It is shown that $L(0,1)$ has so much weight that it is a certain link for $\mathcal{E}$. However, given an entity-linking specification $\mathcal{E}'$ in the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ of maximum-value solutions, when we select r based on $\mathcal{E}'$, the source instance $K = K_r$ is designed so that the neighborhoods $K \restriction N_r(0,1)$ and $K \restriction N_r(0,1')$ are isomorphic, and so by the Locality Theorem, $L(0,1)$ and $L(0,1')$ have the same weight in $\mathcal{E}'$.

Assume, by way of contradiction, that there is an entity-linking specification $\mathcal{E}'$ in the entity-linking framework $(\mathcal{L}_\mathbf{0}, \mathcal{S}_\mathbf{0}, \mathcal{V}_\mathbf{0})$ that is certain-link equivalent to $\mathcal{E}$. By considering an instance with only one fact $R(0,1)$, we show that $\mathcal{E}'$ has the same inclusion dependencies as $\mathcal{E}$. We show that $\mathcal{E}'$ has both FDs on L with the following argument. Assume first that $\mathcal{E}'$ does not have the FD $L: Y \to X$. Since $L(0,1')$ has the same weight in $\mathcal{E}'$ as $L(0,1)$, in particular $L(0,1')$ satisfies the matching constraint for $\mathcal{E}'$. Now $L(0,1')$ is not a certain link in $\mathcal{E}'$, since it is not a certain link in $\mathcal{E}$. So let $\langle K, N\rangle$ be a maximum weight repair of $\langle K, K^*\rangle$ that does not contain $L(0,1')$. Then of course N contains the certain link $L(0,1)$. Form N' by replacing $L(0,1)$ in N by $L(0,1')$. Now N' satisfies the only possible FD $L: X \to Y$, and it satisfies the inclusion dependencies and matching constraint. Furthermore, N' has the same weight as N, since $L(0,1)$ and $L(0,1')$ have the same weight, and so $\langle K, N'\rangle$ is a maximum weight repair. But this is a contradiction, since $\langle K, N'\rangle$ is a maximum weight repair that does not contain the certain link $L(0,1)$. Now define the instance $U(K)$, where (a,b) is a tuple of a relation of K if and only if $(\underline{b}, \underline{a})$ is a tuple of the corresponding relation of $U(K)$, and where $\underline{a}$ and $\underline{b}$ are new values. The proof that the FD $L: X \to Y$ holds for $\mathcal{E}'$ is the same, except rather than replacing the certain link $L(0,1)$ in a maximum weight repair of $\langle K, K^*\rangle$ by $L(0,1')$, we instead replace the certain link $L(\underline{1}, \underline{0})$ in a maximum weight repair of $\langle U(K), (U(K))^*\rangle$ by $L(\underline{1'}, \underline{0})$.

We explicitly find the set M of certain links for $I = K \cup U(K)$ in $\mathcal{E}$ and prove, using the FDs and inclusion dependencies for $\mathcal{E}'$, that $\langle I, M\rangle$ is the unique maximum weight repair for $\langle I, I^*\rangle$ in $\mathcal{E}'$. Let M' consist precisely of all of the links of $\mathcal{E}$ that are not links in M. We prove, again using the Locality Theorem, that there is a one-to-one correspondence between the links ℓ of M and the links ℓ' of M', where ℓ and ℓ' have the same weight in $\mathcal{E}'$. In particular, each link of M' satisfies the entity-linking specification of $\mathcal{E}'$. Further, since M' also satisfies both FDs and the inclusion dependencies, it follows that $\langle I, M'\rangle$ is a maximum weight repair. But this is a contradiction, since $\langle I, M\rangle$ is the unique maximum weight repair. ◀

6 Concluding Remarks

In this paper, we introduced and explored a unifying approach to entity linking. This approach, which is based on the notion of an entity linking framework and the notion of the certain links in such a framework, provides a single formalism for modeling different entity linking scenarios and for comparing them using the certain links as a measure of their expressive power. To this effect, we defined a notion of *certain-link equivalence* that allows us to compare entity-linking specifications, in a way other than logical equivalence (which may be too strict for entity linking purposes). We then made use of certain-link equivalence to define what it means for an entire entity-linking framework to subsume another one. We established a number of technical results that delineate the comparative expressive power of several concrete entity linking frameworks. Our concrete focus in this paper was on the comparison of the entity linking framework of maximum-value solutions with entity linking frameworks (1) that involve maximum cardinality repairs, (2) that allow recursion-free collective entity linking, and (3) that incorporate preferences among links.

A next step in this investigation is to understand the expressive power of recursive collective entity linking. Specifically, we conjecture that the framework $(\mathcal{L}_{\mathbf{c}}, \mathcal{S}_{\mathbf{2}}, \mathcal{V}_{\mathbf{2}})$ of recursive collective entity linking cannot be subsumed by the framework $(\mathcal{L}_{\mathbf{c}}, \mathcal{S}_{\mathbf{1}}, \mathcal{V}_{\mathbf{1}})$ of non-recursive collective entity linking. Another next step has to do with Markov Logic Networks (MLNs), which were first studied in [22]. As stated earlier, it follows from results in [7] that linear MLNs are subsumed by an entity linking framework of maximum-value solutions with weighted disjuncts, where the constraints are in the existential fragment $\exists\mathcal{L}_{\mathbf{0}}$ of the language $\mathcal{L}_{\mathbf{0}}$. It is an open problem if more general MLNs (i.e., not necessarily linear) can be subsumed by an entity linking framework of maximum-value solutions with weighted disjuncts for some suitable choice of weights and constraints from $\mathcal{L}_{\mathbf{0}}$ or from the more general language $\mathcal{L}_{\mathbf{c}}$.

In a different direction, we note that our unifying approach to entity linking is flexible enough to allow assigning *probabilities* to links in a natural way. Specifically, we can define the probability $Pr(L(a, b))$ of a link $L(a, b)$ to be the number of maximum weight repairs containing $L(a, b)$ divided by the total number of maximum weight repairs. Thus, a link $L(a, b)$ is certain if and only if $Pr(L(a, b)) = 1$. The introduction of probabilities in entity linking frameworks raises several algorithmic questions, including the question of enumerating the links whose probability is above a fixed threshold, say, enumerating all links $L(a, b)$ such that $Pr(L(a, b)) \geq 0.75$. Furthermore, it may be possible to establish tight connections between our approach and other approaches in entity linking and entity resolution, such as Probabilistic Soft Logic (PSL) [3, 4, 6], that derive links with *scores* based on weighted first-order formulas. By utilizing such connections, one may also be able to transfer the formalism of preference constraints, which fits naturally in our declarative approach, into PSL

(or into MLN as well). In general, we may obtain more powerful entity linking approaches that combine declarative, logic-based specification with probabilistic reasoning and with explicit user preference constraints.

References

1 Arvind Arasu, Christopher Re, and Dan Suciu. Large-Scale Deduplication with Constraints using Dedupalog. In *ICDE*, pages 952–963, 2009.

2 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *PODS*, pages 68–79, 1999.

3 Stephen H. Bach. *Hinge-Loss Markov Random Fields and Probabilistic Soft Logic: A Scalable Approach to Structured Prediction.* PhD thesis, University of Maryland, 2015.

4 Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *CoRR*, abs/1505.04406, 2015.

5 Indrajit Bhattacharya and Lise Getoor. Collective Entity Resolution in Relational Data. *TKDD*, 1(1), 2007.

6 Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic Similarity Logic. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pages 73–82, 2010.

7 Douglas Burdick, Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. A Declarative Framework for Linking Entities. *ACM Trans. Database Syst.*, 41(3):17, 2016. Preliminary version appeared in ICDT, pages 25–43, 2015.

8 Xin Dong, Alon Y. Halevy, and Jayant Madhavan. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*, pages 85–96, 2005.

9 Jianfeng Du, Guilin Qi, and Yi-Dong Shen. Weight-Based Consistent Query Answering over Inconsistent SHIQ Knowledge Bases. *Knowl. Inf. Syst.*, 34(2):335–371, 2013.

10 Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE TKDE*, 19(1):1–16, 2007.

11 Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *J. Am. Statistical Assoc.*, 64(328):1183–1210, 1969.

12 Haim Gaifman. On Local and Non-Local Properties. *Proc. Herbrand Symp. - Logic Colloquium '81*, 1982.

13 Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *VLDB*, pages 371–380, 2001.

14 Mauricio A. Hernández, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, and Ryan Wisnesky. HIL: A High-Level Scripting Language for Entity Integration. In *EDBT*, pages 549–560, 2013.

15 Mauricio A. Hernández and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In *SIGMOD*, pages 127–138, 1995.

16 Hanna Köpcke and Erhard Rahm. Frameworks for Entity Matching: A Comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.

17 Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of Entity Resolution Approaches on Real-World Match Problems. *PVLDB*, 3(1):484–493, 2010.

18 Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record Linkage: Similarity Measures and Algorithms. In *SIGMOD*, pages 802–803, 2006.

19 Leonid Libkin. Logics with Counting and Local Properties. *ACM Transactions on Computational Logic*, 1(1):33–59, 2000.

20 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

21 Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *ICDT*, pages 179–193, 2007.

22 Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006.

k-Regret Minimizing Set: Efficient Algorithms and Hardness*

Wei Cao[1], Jian Li[2], Haitao Wang†[3], Kangning Wang[4], Ruosong Wang[5], Raymond Chi-Wing Wong‡[6], and Wei Zhan[7]

1 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
cao-w13@mails.tsinghua.edu.cn
2 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
lapordge@gmail.com
3 **Utah State University, Logon, Utah, USA**
haitao.wang@usu.edu
4 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
wkn13,wrs13,zhan-w13@mails.tsinghua.edu.cn
5 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
wrs13@mails.tsinghua.edu.cn
6 **The Hong Kong University of Science and Technology, Hong Kong, China**
raywong@cse.ust.hk
7 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
zhan-w13@mails.tsinghua.edu.cn

Abstract

We study the k-regret minimizing query (k-RMS), which is a useful operator for supporting multi-criteria decision-making. Given two integers k and r, a k-RMS returns r tuples from the database which minimize the *k-regret ratio*, defined as one minus the worst ratio between the k-th maximum utility score among all tuples in the database and the maximum utility score of the r tuples returned. A solution set contains only r tuples, enjoying the benefits of both top-k queries and skyline queries. Proposed in 2012, the query has been studied extensively in recent years. In this paper, we advance the theory and the practice of k-RMS in the following aspects. First, we develop efficient algorithms for k-RMS (and its decision version) when the dimensionality is 2. The running time of our algorithms outperforms those of previous ones. Second, we show that k-RMS is NP-hard even when the dimensionality is 3. This provides a complete characterization of the complexity of k-RMS, and answers an open question in previous studies. In addition, we present approximation algorithms for the problem when the dimensionality is 3 or larger.

1998 ACM Subject Classification H.2.4 Query Processing

Keywords and phrases multi-criteria decision-making, regret minimizing set, top-k query

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.11

* Wei Cao, Jian Li, Kangning Wang, Ruosong Wang and Wei Zhan are supported in part by the National Basic Research Program of China Grant 2015CB358700, 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61202009, 61033001, 61361136003.
† Haitao Wang's research was supported in part by NSF under Grant CCF-1317143.
‡ The research of Raymond Chi-Wing Wong is supported by the grant GRF 16219816.

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi; Article No. 11; pp. 11:1–11:19

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

1 Introduction

One major task of a database system is to return "representative" records to a user. Usually, there are two goals in the system. The first goal is to return a *limited* number of records to a user when the utility function of this user is *known*. One query type achieving this goal is top-k queries [14, 15, 20, 29, 30, 34], each returning k records that have the greatest scores calculated based on these k records and the utility function of a user where k is a positive integer. Interested readers may refer to [18] for a survey of top-k queries. The second goal is to return a set of records which are interesting to a user even though his/her utility function is *unknown*. One example of a query type for this goal is skyline queries [3, 4, 15, 20, 22, 24, 31], each returning a set of records from the database each of which is not *dominated* by other records in the database. Here, a record x is said to *dominate* another record x' if and only if each attribute value of x is not worse than that of x' and at least one attribute value of x is better than that of x'. Interested readers may also refer to [9] for a survey of skyline queries.

However, as described in [25, 26, 27], the above two popular queries could not achieve these two goals simultaneously. First, a top-k query does not achieve the second goal since it requires that a user is given an *exact* utility function indicating his/her preference, which is not reasonable in some cases because in many situations, the user does not know how to specify his/her exact utility function. Second, a skyline query does not meet the first goal because it returns an *uncontrolled* number of records. In the worst case, all records in the database are returned as an output in a skyline query.

Recently, *r-regret queries* and *k-regret minimizing set (k-RMS)* queries, two new types of queries meeting the above two goals, were proposed [8, 25, 26, 27] and studied extensively due to its usefulness and its wide applicability, where r and k are two positive integers. All applications originally applied to top-k queries and skyline queries could also be applied to r-regret queries and k-RMS queries. Some typical applications are choosing hotels for vacation and choosing items (e.g., cars) for purchase.

The purpose of an r-regret query is to return a set of r records in the database, minimizing the "unhappiness" level of a user when seeing only these r records instead of all records in the database, even though the utility function of this user is unknown. Given a positive integer r and a database D containing a number of records, an r-regret query is to return a set R of r records from D such that the greatest "unhappiness" level of a user, formally called the *maximum regret ratio* of a user, is minimized when the user sees only records in R. Here, the "unhappiness" level of a user, called the *regret ratio* of a user, ranging from 0 to 1, refers to how unhappy the user would be when seeing only the records in R, instead of all records in D. Consider the user with his/her utility function f. The *score* of a record x in D with respect to the utility function f is denoted by $f(x)$. The greater the score of a record is, the better the record is. Given a set R of records, the *best* record in R with respect to the utility function f is defined to be the record in R with its greatest score with respect to the utility function f. The regret ratio of this user is equal to 0 if the score of the best record in the selection set R is equal to the one in the whole database D. It becomes larger if the score of the best record in R is smaller than the one in D. The maximum regret ratio of a user refers to the greatest possible regret ratio of a user (since different users can have different utility functions).

Recently, Chester et al. [8] proposed a *generalized* version of r-regret queries called the *k-regret minimizing set (k-RMS)* problem (or queries) relaxing the concept of the "best" record to the concept of *the best k records*. The original form of an r-regret query assumes

that a user must be satisfied with only the "best" record in D. Chester et al. [8] relaxed this assumption and considered that a user is already satisfied and "happy" with one of the best k records in D. Specifically, given two positive integers r and k, and a database D containing a number of records, the k-RMS problem is to return a set R of r records from D such that the *maximum k-regret ratio* of a user is minimized. Here, the *k-regret ratio* of a user, ranging from 0 to 1, is equal to 0 if the score of the best record in R is at least the score of the k-th best record in D. It becomes larger if the score of the best record in R is smaller than the score of the k-th best record in D. Clearly, when $k = 1$, k-RMS becomes the r-regret query (called 1-RMS, or simply the RMS problem). In this paper, we have the following contributions.

1. For RMS in $\mathbb{R}^2$ (i.e., the dimensionality is 2), we propose an $O(n \log n)$ time exact algorithm, where n is the number of records in the dataset D. The time complexity is better than the previous best-known time complexity of $O(rn^2 + n^2 \log n)$ [8].
2. For k-RMS in $\mathbb{R}^2$, we present an $O(n^2 \log n)$ time algorithm, which improves the $O(rn^2k^{\frac{1}{3}} + n^2 \log n)$ time complexity result in [8]. We also propose an approximation algorithm of $O(nk^{\frac{1}{3}} \log(1/\epsilon) + n \log n + nk^{\frac{1}{3}} \log^{1+\delta} n)$ time, where ϵ is the additive approximation error and δ is any positive constant. For typical parameters, it performs much faster than the previous best-known algorithm [8]. To solve the problem, we also give an efficient algorithm for the decision version of the problem, which is interesting in its own right both theoretically and practically. A summary of our results is in Table 1.
3. We show that for any positive integer k, the k-RMS problem is NP-hard when the dimensionality of the database is 3 (or larger). This is the first-known hardness result for the k-RMS problem in a fixed dimensional database. Although Chester et al. [8] prove the NP-hardness of the RMS problem, it states that the hardness is due to both the *high* dimensionality of the dataset and the *large* number of records in the dataset. It has been open whether the problem is still NP-hard for fixed dimensional cases. Our result settles the open problem and thus provides a complete characterization of the computational complexity of the problem (together with our algorithms in $\mathbb{R}^2$).
4. For RMS in $\mathbb{R}^d$, we show it is closely connected to the notion of *ϵ-kernel*, introduced by Agarwal et al. [2]. Based on the connection, we derive an upper bound $r^{-2/(d-1)}$ of the maximum regret ratio, improving the previous bound $r^{-1/(d-1)}$ in [26]. We also provide an approximation algorithm for k-RMS when $d \geq 3$.

Outline: The rest of the paper is organized as follows. In Section 2, we formally define the problem. Section 3 gives our algorithms for k-RMS when the dimensionality is 2. Section 4 presents the NP-hardness result. Section 5 gives our algorithms in high-dimensional cases. Section 6 discusses the related work. Due to the page limitation, many details and proofs are omitted but can be found in the full version of this paper[1].

2 Problem Formulation

Let D be a database containing n records/points[2] with d attributes/dimensions. Given a point x in D, for each $i \in [1, d]$, the i-th dimensional value of point x is denoted by $x[i]$. We assume that the values $x[i]$ in the database are all non-negative real numbers, which is

[1] http://www.ruosongwang.net/Paper/k_RMS.pdf
[2] In the following, we use term "records" and "points" interchangeably since they refer to the same concept.

the common assumption in related literatures [8, 26]. Each user is associated with a utility function f denoted by a d-dimensional non-negative vector ω called a *weight vector*. Let W be the set of all possible weight vectors.

Given a point x in D and a weight vector ω, the *score* of x with respect to ω is the dot product of x and ω, denoted by $\langle x, \omega \rangle$. That is, $\langle x, \omega \rangle$ is equal to $\sum_{i=1}^{d} x[i]\omega[i]$. If we know the utility function f with the weight vector ω, this score $\langle x, \omega \rangle$ is also written as $f_\omega(x)$. Given an integer $k \geq 1$, we denote the k-th largest score among $x \in D$ with respect to weight vector ω by $\max_{x \in D}^{(k)} \langle x, \omega \rangle$.

Given a non-empty subset R of D and a weight vector ω, the *k-regret ratio* of set R with respect to weight vector ω, denoted by k-regratio(R, ω), is defined to be

$$k\text{-regratio}(R, \omega) = \max \left\{ 0, 1 - \frac{\max_{x \in R} \langle x, \omega \rangle}{\max_{x \in D}^{(k)} \langle x, \omega \rangle} \right\}.$$

If k-regratio(R, ω) is 0, the best score in R is at least as good as the k-th largest score with respect to ω in the original dataset D. The *maximum k-regret ratio* of R, denoted by k-regratio(R), is defined to be k-regratio$(R) = \sup_{\omega \in W} k\text{-regratio}(R, \omega)$.

▶ **Problem 1** (k-RMS [8]). *Given two positive integers k and r, we want to find a set R of r points from D such that k-regratio(R) is minimized.*

This is an *optimization* problem. The following defines its *decision* version, called Dec-k-RMS (we also use Dec-RMS to refer to the case $k = 1$).

▶ **Problem 2** (Dec-k-RMS). *Given two positive integers k and r, and a real value $\theta \in [0, 1]$, determine whether there exists a set R of r points from D such that k-regratio(R) is at most $1 - \theta$ (if yes, find such a solution set R).*

We will also give algorithms for Dec-k-RMS since they will be used as subroutines for solving the optimization version (i.e., Problem 1). On the other hand, in some applications where there is a pre-specified error threshold of k-regratio$(\cdot)$, it would be more suitable to solve the decision version, and thus the decision problem may be interesting in its own right.

3 Efficient Algorithms in $\mathbb{R}^2$

In this section, we develop several algorithms for RMS and k-RMS in $\mathbb{R}^2$. Table 1 summarizes our results. We assume that $\|\omega\|_1 = \omega[1] + \omega[2] = 1$ for any weight vector $\omega \in W$ as scaling does not change the k-regratio. Hence, we can write $\omega = (\lambda, 1 - \lambda)$ for some $\lambda \in [0, 1]$.

For each point $p = (x, y)$ in D, we define a linear function $f_p(\lambda) = \lambda x + (1 - \lambda)y$ with $\lambda \in [0, 1]$. We reformulate both the decision version and the optimization version as follows.

1. (The decision version) In the decision version Dec-k-RMS, we are given a constant θ, and we need to decide whether there is some set $R \subseteq D$ of cardinality r such that the following holds:

$$\forall \lambda \in [0, 1], \ \max_{p \in R} f_p(\lambda) \geq \theta \cdot \max_{p \in D}^{(k)} f_p(\lambda), \tag{1}$$

   where $\max^{(k)}$ is the operator that returns the k-th largest value.
2. (The optimization version) The optimization version k-RMS is to maximize θ in (1).

Table 1 The running times of the previous algorithms and our new algorithms in $\mathbb{R}^2$. Det: Deterministic/randomized algorithms (yes/no); Ex: Exact/approximation algorithms (yes/no); The naming of the algorithms: D- means the decision version, E- means an exact algorithm and A- means an approximation algorithm. $n = |D|$, $m = |\mathsf{LS}_k|$, $r = |R|$. ϵ is the additive error of the approximate regret ratio. δ can be any positive constant. † D-Greedy-k requires $O(n \log n + m \log^{1+\delta} n)$ preprocessing time, and runs in $O(n+m)$ time for any threshold θ. †† In [8], the authors claim their algorithm runs in $O(rn^2)$ time, with the factor depending on k omitted as a constant. However, a more careful examination shows their algorithm runs in $O(rn^2 + n^2 \log n)$ time for RMS and $O(rn^2 k^{\frac{1}{3}} + n^2 \log n)$ time for k-RMS instead: The priority queue requires $O(n^2 \log n)$ time; the best known upper bound of the size of the k-level set is $O(nk^{\frac{1}{3}})$.

Problem	Algorithm	Time Complexity	Det	Ex	Source
Dec-RMS	D-IntCov-1	$O(n \log n)$	yes	yes	Sect. 3.1.1
Dec-k-RMS	D-Greedy-k	$O(n+m)$ †	yes	yes	Sect. 3.2
RMS	E-Pre-1	$O(rn^2 + n^2 \log n)$ ††	yes	yes	[8]
	A-IntCov-1	$O(n \log n \log(1/\epsilon))$	yes	no	Sect. 3.1.2
	A-Greedy-k	$O(n \log n + n \log(1/\epsilon))$	yes	no	Sect. 3.2
	E-Greedy-1	$O(n \log n)$	no	yes	Sect. 3.3.2
k-RMS	E-Pre-k	$O(rn^2 k^{\frac{1}{3}} + n^2 \log n)$ ††	yes	yes	[8]
	A-Greedy-k	$O((n+m)\log(1/\epsilon) + n \log n + m \log^{1+\delta} n)$	yes	no	Sect. 3.2
	E-Greedy-k	$O(n^2 \log n)$	yes	yes	Sect. 3.3.1

It is convenient to view the problem from a geometric perspective as follows. Each record $p \in D$ corresponds to a line $f_p(\lambda)$ ($\lambda \in [0,1]$). All such lines form a line arrangement $\mathbb{A}(D)$. The *k-level set* of $\mathbb{A}(D)$ [5] is a piecewise linear curve (see Figure 1 for an example)

$$\mathsf{LS}_k(\lambda) = \max^{(k)}_{p \in D} f_p(\lambda), \text{ for } \lambda \in [0,1].$$

A *segment* is a maximal linear piece in the k-level set. Let m denote the number of segments of $\mathsf{LS}_k(\lambda)$. It is known that m is bounded by $O(nk^{\frac{1}{3}})$ and $\mathsf{LS}_k(\lambda)$ can be computed in $O(n \log n + nk^{\frac{1}{3}})$ expected time by a randomized algorithm [5] or in $O(n \log m + m \log^{1+\delta} k)$ time by a deterministic algorithm for any constant $\delta > 0$ [5]. Note that the 1-level set LS_1 is always *convex* since it is the *upper envelop* of $\mathbb{A}(D)$ (see the red-dashed curve in Figure 1). However, the convexity property does not necessarily hold for any $k > 1$.

We introduce *scaled level sets*, which generalizes the notion of k-level sets.

▶ **Definition 3.** (Scaled Level Set) Given a threshold $\theta > 0$, define the θ-scaled k-level set as the function $\theta\text{-}\mathsf{LS}_k(\lambda) = \theta \cdot \mathsf{LS}_k(\lambda) = \theta \cdot \max^{(k)}_{p \in D} f_p(\lambda)$, for $\lambda \in [0,1]$.

We can reformulate the decision problem Dec-k-RMS as follows: Decide whether there exists a subset R of r lines, such that the upper envelop of R covers the scaled level set (i.e., the function $\max_{p \in R} f_p(\lambda)$ is above $\theta\text{-}\mathsf{LS}_k$).

▶ **Example 4.** As an example of the decision problem Dec-k-RMS shown in Fig. 2, where $k = 3$, $\theta = 0.9$ and the dashed thick curve is $\theta\text{-}\mathsf{LS}_k$, we aim to find r lines such that they collectively cover the dashed thick curve from above. When $r = 2$, the two thick lines shown in red form a solution. When $r = 1$, the thick blue line is the only valid solution.

In the sequel, we solve the decision problem in Section 3.2. In Section 3.3, we solve the optimization problem, using the algorithms for the decision problem as subroutines. But as warm-ups, we first give some simple but practical algorithms.

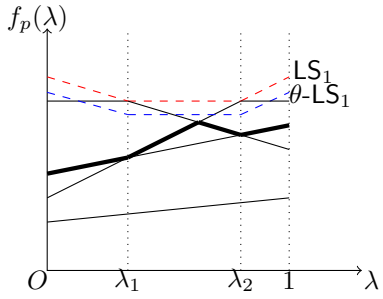


Figure 1 The arrangement $\mathbb{A}(D)$ (black lines), its 1-level set LS_1 (red dashed line), θ-scaled 1-level set (blue lines), and 3-level set (thick black line).

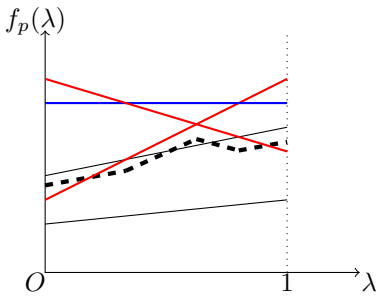


Figure 2 Illustrating Example 4: θ-LS_k is the dashed thick curve.

3.1 The Warm-up Algorithms

In this section we present algorithms for Dec-RMS and RMS. These algorithms are theoretically not as efficient as the algorithms given later, but they are very simple and practical, and may also provide some directions for the later improved ones.

3.1.1 Reducing Dec-RMS to Interval Coverage

Note that since LS_1 (which is actually the upper envelop of $\mathbb{A}(D)$) is convex, the scaled level set θ-LS_1 is also convex. Also note that $m \leq n$ in this case. We can compute LS_1 (and thus θ-LS_1) in $O(n \log n)$ time [16]. We use $\lambda_0 = 0, \lambda_1, \ldots, \lambda_{m-1}, \lambda_m = 1$ to denote all breaking points of θ-LS_1 (see Fig. 1). Our task is to cover θ-LS_1 using at most r lines. For a line f_p, we let $I(p) = \{\lambda \mid f_p(\lambda) \geq \theta\text{-}\mathsf{LS}_1(\lambda)\}$. The convexity of θ-LS_1 implies that $I(p)$ is a closed interval (which may be empty). Moreover, the interval can be computed in $O(\log n)$ time by binary search.

Hence, we can compute the set of intervals $\{I(p)\}_{p \in D}$ in $O(n \log n)$ time.

To solve our problem Dec-RMS, it is sufficient to find a minimum number of intervals in the above set whose union covers the range $[0, 1]$. This can be easily done in $O(n)$ time by a greedy method after the endpoints of the intervals of $\{I(p)\}_{p \in D}$ are sorted [1].

We call the above algorithm D-IntCov-1.

▶ **Theorem 5.** *D-IntCov-1 solves the Dec-RMS problem in $O(n \log n)$ time and $O(n)$ space deterministically.*

3.1.2 An Approximating Algorithm for RMS

To solve the optimization problem RMS, the high-level idea is to perform binary search on a set of "candidate values" for the optimal regret ratio θ, and use our decision algorithms to check whether θ-LS_1 can be covered by r lines from D.

We simply perform binary search directly on the interval $[0, 1]$. Initially, the candidate range of θ is $[0, 1]$. Given $\theta \in [0, 1]$, we run the decision algorithm for Dec-RMS to check whether the regret ratio of $1 - \theta$ is achievable (i.e., whether θ-LS_1 can be covered). If the answer is yes (resp., no), then we say that θ is *feasible* and the optimal value is at least θ (resp., smaller than θ). We stop until the interval for the candidate θ values is shorter than ϵ, a given tolerable error. The decision procedure is evoked for $O(\log \frac{1}{\epsilon})$ times and

the regret ratio of solution is at most the optimal regret plus ϵ (we call such a solution an ϵ-approximation). We refer to the algorithm as A-IntCov-1 (using D-IntCov-1 as the decision procedure). We have the following theorem.

▶ **Theorem 6.** *For any $\epsilon > 0$, A-IntCov-1 can find an ϵ-approximation for RMS in $O(n \log n \log \frac{1}{\epsilon})$ time.*

3.2 The Decision Algorithm for Dec-k-RMS

In this section, we present an algorithm for the problem Dec-k-RMS (and thus also for Dec-RMS). We call our algorithm D-Greedy-k. We will prove the following theorem.

▶ **Theorem 7.** *After $O(n \log n + m \log^{1+\delta} k)$-time preprocessing for any $\delta > 0$, given any $\theta \in [0, 1]$, our algorithm D-Greedy-k solves the problem Dec-k-RMS in $O(n + m)$ time, where n is the number of lines of D and m is the number of segments in the k-level set LS_k.*

Due to the page limitation, we ignored proofs for some lemmas and some implementation details, which can be found in the full version of this paper.

Given any $\theta > 0$, D-Greedy-k first finds a smallest subset $R \subseteq D$ of lines such that the upper envelop of R is above θ-LS_k for $\lambda \in [0, 1]$ and then solves Dec-k-RMS by comparing $|R|$ with the given maximum cardinality r.

As preprocessing, we sort all lines of D by their intersections with the vertical line $\lambda = 0$ from top to bottom. This step can be done in $O(n \log n)$ time. Then we compute LS_k in $O(n \log m + m \log^{1+\delta} k)$ time for any $\delta > 0$, using the algorithm in [5]. The total preprocessing time is $O(n \log n + m \log^{1+\delta} k)$ (since $m = O(nk^{\frac{1}{3}})$ and $k \leq n$).

After the preprocessing, for any given θ, we first compute θ-LS_k, which can be done in $O(m)$ time since LS_k has been computed in the preprocessing step. Let $l_1, l_2, \ldots, l_n$ be the lines of D sorted by their intersections with the vertical line $\lambda = 0$ from top to bottom. For each $i \in [1, n]$, let a_i denote the intersection of l_i and the vertical line $\lambda = 0$, and thus a_i is also from top to bottom. For convenience, we use l_0 to denote the vertical line $\lambda = 0$ and l_{n+1} to denote the vertical line $\lambda = 1$. A simple observation, as formalized in Lemma 8, is that for two lines l and l', if l "dominates" l', then l' can be directly discarded.

▶ **Lemma 8.** *For any $1 \leq i < j \leq n$, if the slope of l_i is larger than or equal to that of l_j, then there exists an optimal solution R that does not contain l_j (and thus l_j can be ignored for solving the problem).*

Notice that Lemma 8 essentially states that points that are not in the skyline (in the original space) can be dropped. This has already been known in previous works (e.g., [8]).

We run a *pruning procedure* on D to remove such lines l_j as specified in the preceding lemma. This can be done by scanning the lines of D in their index order in $O(n)$ time. The following algorithm will work on the remaining lines of D. Hence, after the pruning procedure, we can assume that the remaining lines of D following their index order are also sorted by their slopes in strictly ascending order (renamed as $\{l_1, l_2, ...l_n\}$).

To simplify the notation, we use C to refer to θ-LS_k. For any two values λ_1 and λ_2 with $\lambda_1 \leq \lambda_2$, we use $C[\lambda_1, \lambda_2]$ to denote the portion of C defined on the interval $\lambda \in [\lambda_1, \lambda_2]$. For any two points q_1 and q_2 on C, we also use $C[q_1, q_2]$ to refer to the portion of C between q_1 and q_2. Let $P(C)$ denote the region of the plane above C and between l_0 and l_{n+1}. For any set D' of lines, we use $U(D')$ to denote its upper envelop. For any point q in the plane, let $\lambda(q)$ denote its λ-coordinate. Note that C is λ-monotone, i.e., any vertical line intersects C at most once. Therefore, we can say something like "move a point on C from left to right".

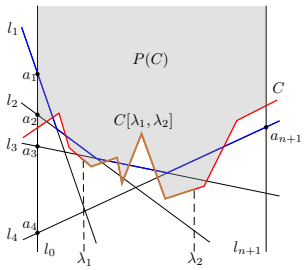


Figure 3 The blue curve denotes $U(\{l_1, l_2, l_3, l_4\})$. The brown curve denotes $C[\lambda_1, \lambda_2]$. The gray area denotes $P(C)$.

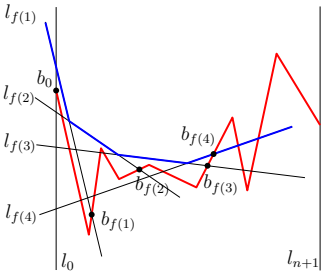


Figure 4 Illustrating the set $R_i = \{l_0, l_{f(1)}, l_{f(2)}, l_{f(3)}, l_{f(4)}\}$ with $g_i = 4$. The red curve is C and the blue curve is $U(R_i)$. The point p is at $b_{f(4)}$.

Let C' be another λ-monotone curve in the plane. We say that C' is *above* $C[\lambda_1, \lambda_2]$ for some $\lambda_1 \leq \lambda_2$ if for any value $\lambda' \in [\lambda_1, \lambda_2]$, the intersection of the vertical line $\lambda = \lambda'$ and C' is not lower than that of the vertical line $\lambda = \lambda'$ and C. For two points $p_1 \neq p_2$, we use $\overline{p_1 p_2}$ to denote the line segment with endpoints p_1 and p_2. See Figure 3 for an illustration of the definitions given above.

Our algorithm processes the lines of $l_0, l_1, \ldots, l_{n+1}$ in their index order from l_0 to l_{n+1}. In general, suppose line l_i has just been processed and we are about to process l_{i+1} for some i with $0 \leq i \leq n$. Our algorithm maintains a set $R_i = \{l_{f(0)}, l_{f(1)}, \ldots, l_{f(g_i)}\}$ of $g_i + 1$ lines in $\{l_0, l_1, l_2 \ldots, l_i\}$ and a set $B_i = \{b_{f(0)}, b_{f(1)}, \ldots, b_{f(g_i)}\}$ of $g_i + 1$ points with $f(0) < f(1) < \cdots < f(g_i)$, for some integer $g_i \geq 0$, such that R_i and B_i have the following properties. Refer to Figure 4 for an example.

1. $f(0) = 0$, i.e., $l_{f(0)}$ is l_0. Since l_0 is vertical, we tilt it slightly such that it has a negative slope so that the definition of the upper envelop $U(R_i)$ is clear. Similarly, we tilt l_{n+1} slightly such that it has a positive slope.
2. Each line of R_i has a segment that appears in $U(R_i)$. The segments of lines of R_i appear on $U(R_i)$ from left to right following the index order.
3. $0 = \lambda(b_{f(0)}) < \lambda(b_{f(1)}) < \ldots < \lambda(b_{f(g_i)})$. For $0 \leq t \leq g_i$, $b_{f(t)} \in l_{f(t)} \cap C$. Recall that a_i denote the intersection of l_i and the vertical line $\lambda = 0$, thus the line segment $\overline{a_{f(t)} b_{f(t)}}$ is segment of the line $l_{f(t)}$. For $0 < t \leq g_i$, l_t is above $C[b_{f(t-1)}, b_{f(t)}]$. Thus, $U(R_i)$ is above $C[0, \lambda(b_{f(g_i)})]$.
4. For each line $l_{f(t)} \in R_i$ with $0 \leq t \leq i$, the point $b_{f(t)}$ is defined as follows.
   When $t = 0$, $b_{f(t)}$ (i.e., b_0) is defined as the intersection of C and l_0. Here, for convenience of discussion, we also let C include the half-line of l_0 above b_0 and the half-line of l_{n+1} above a_{n+1} (i.e., the intersection of C and l_{n+1}). In this way, C is the boundary of the region $P(C)$.
   When $t > 0$, suppose point $b_{f(t-1)}$ has been defined on C. Denote q to be the intersection of $l_{f(t)}$ and the vertical line through $b_{f(t-1)}$, it holds that q is above $b_{f(t-1)}$. If we move q rightwards on $l_{f(t)}$, then $b_{f(t)}$ is defined as the first point of $P(C)$ we encounter after which q will move out of $P(C)$.
5. $\overline{a_{f(t)} b_{f(t)}}$ intersects $\overline{a_{f(t-1)} b_{f(t-1)}}$ for any t with $1 \leq t \leq i$. For any $2 \leq t \leq i$, either $\overline{a_{f(t)} b_{f(t)}}$ does not intersect $\overline{a_{f(t-2)} b_{f(t-2)}}$, or they intersect but $l_{f(t)}$ is not completely above $C[b_{f(t-2)}, b_{f(t-1)}]$. In Figure 4, although $\overline{a_{f(1)} b_{f(1)}}$ intersects with $\overline{a_{f(3)} b_{f(3)}}$, $l_{f(3)}$ is not completely above $C[b_{f(1)}, b_{f(2)}]$.
6. For each $1 \leq t \leq i$, the upper hull of the convex hull of $C[b_{f(t-1)}, b_{f(t)}]$ is maintained in a linked list $L(b_{f(t-1)}, b_{f(t)})$. More specifically, $L(b_{f(t-1)}, b_{f(t)})$ stores the edges of the

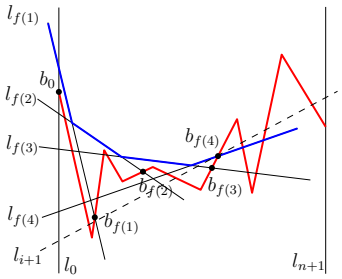


Figure 5 Illustrating the *special case*. Notice that $l_{i+1} \cap \overline{a_{f(4)}b_{f(4)}} = b_{f(4)}$.

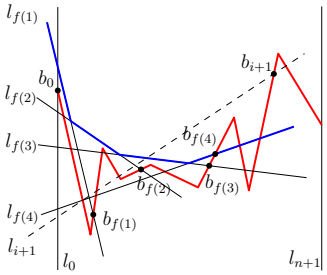


Figure 6 Illustrating the case where l_{i+1} (the dashed line) intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ with $g_i = 4$. In this example, $R_{i+1} = \{l_0, l_{f(1)}, l_{f(2)}, l_{i+1}\}$ because l_{i+1} intersects $\overline{a_{f(1)}b_{f(1)}}$ but is not above $C[b_{f(1)}, b_{f(2)}]$.

upper hull of $C[b_{f(t-1)}, b_{f(t)}]$ from left to right.

3.2.1 The Algorithm

Initially, for $i = 0$, we let $R_0 = \{l_0\}$ and $B_0 = \{b_0\}$. In general, suppose we have processed the line l_i and obtained R_i and B_i. In the following, we describe the algorithm for processing the next line l_{i+1} and obtain the set R_{i+1} and B_{i+1}.

We first check whether l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$. If not, we simply ignore l_{i+1} and let $R_{i+1} = R_i$.

If l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$, we consider a *special case* where $l_{i+1} \cap \overline{a_{f(g_i)}b_{f(g_i)}} = b_{f(g_i)}$ and b_{i+1} is $b_{f(g_i)}$ (e.g., see Fig. 5). If this case happens, then we simply ignore l_{i+1} and let $R_{i+1} = R_i$. To determine whether b_{i+1} is $b_{f(g_i)}$, we check whether we will go outside $P(C)$ after we cross $b_{f(g_i)}$ if we move on l_{i+1} rightwards. Since $b_{f(g_i)}$ is known, we can determine whether this special case happens in $O(1)$ time.

If l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ and the special case does not happen (e.g., see Figure 6), then we proceed to compute the point b_{i+1} as follows.

As $f(g_i) \leq i < i+1$, a_{i+1} is below $a_{f(g_i)}$. Since l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ and the special case does not happen, if q is the intersection of l_{i+1} and the vertical line through $b_{f(g_i)}$, then q must be above $b_{f(g_i)}$. Imagine that we move q rightwards on l_{i+1}. As we defined earlier, b_{i+1} is the first point of $P(C)$ we encounter after which q will move out of $P(c)$ (e.g., see Figure 6). As the special case does not happen, $\lambda(b_{i+1}) > \lambda(b_{f(g_i)})$ holds. To find b_{i+1}, we simply move q rightwards on C until we meet an intersection between l_{i+1} and an edge of C.

Next we compute the upper hull of (the convex hull of) $C[b_{f(g_i)}, b_{i+1}]$ and store it in a linked list $L(b_{f(g_i)}, b_{i+1})$. The list $L(b_{f(g_i)}, b_{i+1})$ can be constructed when we compute b_{i+1} by moving q from $b_{f(g_i)}$ to b_{i+1}. Since C is λ-monotone, $L(b_{f(g_i)}, b_{i+1})$ can be constructed in linear time in the number of vertices of $C[b_{f(g_i)}, b_{i+1}]$ (e.g., by Graham's scan).

Finally, we determine the set R_{i+1} as follows. We consider the lines of R_i in the reverse order of their indices. Consider $l_{f(g_i)}$ first. If l_{i+1} does not intersect the line segment $\overline{a_{f(g_i-1)}b_{f(g_i-1)}}$ of $l_{f(g_i-1)}$, we stop the procedure with $R_{i+1} = R_i \cup \{l_{i+1}\}$.

Otherwise, there are further two subcases. We check whether l_{i+1} is above $C[b_{f(g_i-1)}, b_{f(g_i)}]$. To this end, observe that l_{i+1} is above $C[b_{f(g_i-1)}, b_{f(g_i)}]$ if and only if l_{i+1} is above the upper hull of $C[b_{f(g_i-1)}, b_{f(g_i)}]$, which is stored in the list $L(b_{f(g_i-1)}, b_{f(g_i)})$. As we will formalize later in Lemma 9, we can use a *upper hull walking procedure* to efficiently determine whether l_{i+1} is above the upper hull of $C[b_{f(g_i-1)}, b_{f(g_i)}]$.

If l_{i+1} is not above $C[b_{f(g_i-1)}, b_{f(g_i)}]$, then we stop the procedure with $R_{i+1} = R_i \cup \{l_{i+1}\}$. Otherwise, we remove $l_{f(g_i)}$ from R_i and proceed on considering the next line $l_{f(g_i-1)}$. In addition, we perform a *upper hull merge procedure* to merge the two lists $L(b_{f(g_i-1)}, b_{f(g_i)})$ and $L(b_{f(g_i)}, b_{i+1})$ to obtain a single list $L(b_{f(g_i-1)}, b_{i+1})$, representing the upper hull of $C[b_{f(g_i-1)}, b_{i+1}]$. As formalized later in Lemma 9, the merge procedure can be efficiently implemented.

The above processes the line $l_{f(g_i)}$. Processing the next line $l_{f(g_i-1)}$ (and other lines) is done similarly, and we omit the details. Refer to Figure 6 for an example.

The above algorithm may remove some lines from R_i. For ease of reference, we let R'_i be the remaining R_i after the above algorithm and we still use R_i to refer to the original set. After the above algorithm, we have $R_{i+1} = R'_i \cup \{l_{i+1}\}$.

The algorithm finishes once l_{n+1} is processed, after which we will obtain the set R_{n+1}. In the full version of this paper, we show that $R_{n+1} \setminus \{l_0, l_{n+1}\}$ is the optimal solution set R for the problem Dec-k-RMS and the whole algorithm runs in $O(n+m)$ time (excluding the preprocessing). The subsequent lemma state that the upper hull merge and walking procedures can be efficiently implemented. See the full version for details of these two procedures. Note that the efficiency of Lemma 9 relies on that the slopes of the lines of D in their index order are sorted increasingly.

▶ **Lemma 9.** *We can implement the upper hull merge procedure and the upper hull walking procedure such that the total time of the procedure in the entire algorithm is $O(m+n)$.*

By the similar binary search approach as in Section 3.1.2 with D-Greedy-k as the decision procedure instead, we can obtain an approximation algorithm for k-RMS. We refer to this algorithm as A-Greedy-k, whose performance is summarized below.

▶ **Theorem 10.** *For any $\epsilon > 0$, A-Greedy-k can find an ϵ-approximation for k-RMS in $O((n+m)\log(1/\epsilon) + n\log n + m\log^{1+\delta} n)$ time.*

3.3 Optimization Algorithms

In this section, we solve the optimization problems RMS and k-RMS. As in Section 3.1.2, the idea is to perform binary search on the candidate values of the optimal θ, with the regret ratio $1-\theta$. Unlike the algorithm there that only gives approximating result, here we present two exact algorithms. The first algorithm (Section 3.3.1) determines all candidate values implicitly (there are too many such values so we cannot afford to compute them explicitly), and performs binary search on them to find an optimal solution for k-RMS. The second algorithm (Section 3.3.2) exploits the convexity of θ-LS$_1$ and performs randomized binary search over the candidate values, and it works quite efficiently but only on RMS.

3.3.1 An Exact Algorithm for k-RMS

▶ **Lemma 11.** *The following statements are equivalent:*
1. *A set R covers θ-LS$_k$ for all $\lambda \in [0,1]$.*
2. *A set R covers θ-LS$_k$ for all $\lambda \in X(D) := \{0,1\} \cup \{\lambda \mid \exists\ l_1, l_2 \in D, l_1(\lambda) = l_2(\lambda)\}$.*

Obviously statement (1) implies (2). On the other hand, note that both θ-LS$_k$ and the upper envelop of R are piecewise linear, with all breaking points contained in $X(D)$. Therefore if (2) holds, the upper envelop of R must be above θ-LS$_k$. Hence, statement (2) implies (1) as well. The lemma thus follows.

The following lemma is a consequence of Lemma 11.

▶ **Lemma 12.** *For k-RMS, the optimal θ is $0, 1$ or in*

$$\mathsf{Cand}(D) := \left\{ \frac{l(\lambda)}{\mathsf{LS}_k(\lambda)} \middle| \; l \in D, \lambda \in X(D) \right\}.$$

Proof. Notice that by Lemma 11, θ is optimized so that R covers θ-LS_k within $X(D)$. This implies θ-LS_k and the upper envelop of R coincide at some $\lambda \in X(D)$, so the lemma holds. ◀

Clearly, the set $\mathsf{Cand}(D)$ consists of at most $|D| \cdot |X(D)| = O(n^3)$ values. To solve the problem k-RMS, we can call the decision algorithm D-Greedy-k to find the largest feasible $\theta \in \mathsf{Cand}(D)$. Computing the set $\mathsf{Cand}(D)$ explicitly would take $\Omega(n^3)$ time. Instead, we present an approach that only constructs $\mathsf{Cand}(D)$ implicitly.

First we compute and sort the set $X(D)$. For each $\lambda \in X(D)$, our algorithm maintains an interval of indices $I_\lambda \subseteq [1, n]$ so that if the optimal value θ is equal to the j-th largest value of $l(\lambda)/\mathsf{LS}_k(\lambda)$ for all $l \in D$, then $j \in I_\lambda$ must hold. Initially, I_λ is set to $[1, n]$ for each $\lambda \in X(D)$, and the interval will shrink during the algorithm.

The algorithm consists of multiple stages. In each stage, we use a line sweeping algorithm on λ, keeping track of the lines l of D ordered by $l(\lambda)$. For the i-th time the sweeping line hits a $\lambda_i \in X(D)$, we compute a value θ_i of θ (according to Lemma 12) determined by the line ranked at the median of the interval I_{λ_i}, and assign it a weight $w_i = |I_{\lambda_i}|$. In this way, we compute a weighted subset $S = \{\theta_i\}_i \subseteq \mathsf{Cand}(D)$ of size $O(n^2)$. Next we compute the weighted median of S: that is, a value $\theta_m \in S$ such that

$$\sum\{w_i \mid \theta_i < \theta_m\} \le \frac{1}{2}\sum_i w_i < \sum\{w_i \mid \theta_i \le \theta_m\}.$$

The weighted median can be found in $O(|S|)$ time using the linear-time selection algorithm [21]. Then we use D-Greedy-k to determine whether θ_m is feasible. If yes, we update each I_{λ_i} with $\theta_i \ge \theta_m$ to be its lower-half interval; otherwise, we update each I_{λ_i} with $\theta_i \le \theta$ to be its upper-half interval. Hence, the reduced weight of all intervals of S is $\frac{1}{2}\sum\{w_i \mid \theta_i \ge \theta\}$ or $\frac{1}{2}\sum\{w_i \mid \theta_i \le \theta\}$, which is larger than $\frac{1}{4}\sum_i w_i$ in either case. Therefore, each stage will reduce the total weight by at least $1/4$. Since the initial total weight, that is, the total size of all intervals I_λ is $O(n^3)$, we conclude that there are $O(\log n)$ stages (the algorithm stops once the remaining total weight is $O(1)$, after which we can use D-Greedy-k to find the optimal θ from the remaining $O(1)$ candidate values).

For the running time, each stage is comprised of an $O(n^2)$-time line sweeping algorithm, an $O(n^2)$-time weighted median algorithm [21], and one call of D-Greedy-k taking $O(n+m) = O(n^2)$ time. Thus, the total time of the algorithm is $O(n^2 \log n)$. We refer to the algorithm is as E-Greedy-k (for Exact Greedy Algorithm).

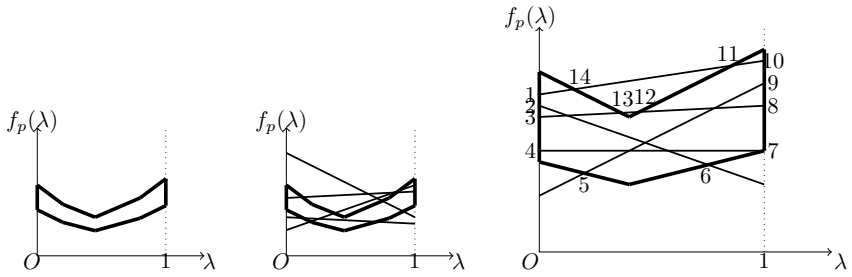
▶ **Theorem 13.** *E-Greedy-k can compute an optimal solution for the k-RMS problem in $O(n^2 \log n)$ time.*

3.3.2 An $O(n \log n)$ Time Exact Algorithm for RMS

For RMS, we derive a more efficient randomized algorithm, whose expected running time is $O(n \log n)$. Lemma 12 still applies here, but we have a stronger Lemma 14 (which is not applicable to k-RMS),

▶ **Lemma 14.** *For RMS, the optimal value θ is in the set $\mathsf{Cand}(D)$ defined as*

$$\{0, 1\} \cup \left\{ \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)} \middle| \; l \in D \text{ and } \lambda \in \{0, 1\}, \text{ or } \exists\, l' \in D, l(\lambda) = l'(\lambda) \right\}.$$



(a) An example of the band $\mathcal{B}$

(b) Each line may provide at most two segments in the band

(c) The sampling method

Figure 7 Illustration of the band, the segments in it, and the sampling method.

Proof. For the optimal θ such that a set R covers θ-LS_1, the upper envelop of R and θ-LS_1 must coincide at some point. Let $l \in R$ to be the line whose segment in the upper envelop contains such a coincidence point. Suppose two endpoints of this segment are at λ_1, λ_2. Then at least one of them is also a coincidence point, since otherwise θ-LS_1 would be strictly above the segment at one of λ_1 and λ_2, incurring contradiction. ◀

We partition $\mathsf{Cand}(D)$ into two subsets:

$$\mathsf{Cand}_1(D) = \{0,1\} \cup \left\{ \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)} \mid l \in D \text{ and } \lambda \in \{0,1\} \right\},$$

$$\mathsf{Cand}_2(D) = \left\{ \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)} \mid \exists\, l' \in D, l(\lambda) = l'(\lambda) \right\}.$$

Notice that $|\mathsf{Cand}_1(D)| \leq 2n + 2 = O(n)$. In the following, we first process $\mathsf{Cand}_2(D)$. Our approach is inspired by the random sampling technique of Matoušek [23].

We perform a randomized search over the values of $\mathsf{Cand}_2(D)$, without constructing $\mathsf{Cand}_2(D)$ explicitly. The algorithm consists of multiple rounds and each round shrinks the search range significantly. Initially, the search range of θ is $[0,1]$. In general, suppose the search range is $[\theta_0, \theta_1]$ in the current round. We define a band $\mathcal{B}$ as the region bounded by θ_0-LS_1, θ_1-LS_1, and the two vertical lines $\lambda = 0$ and $\lambda = 1$ (See Figure 7a). A candidate value $\frac{l(\lambda)}{\mathsf{LS}_1(\lambda)}$ in $[\theta_0, \theta_1]$ corresponds to an intersection $(\lambda, l(\lambda))$ of two lines of D lying in the band $\mathcal{B}$. The current round of the algorithm works as follows.

We first compute the number of line intersections in $\mathcal{B}$ and then sample at most n out of them. Note that the intersection of each line $l \in D$ and $\mathcal{B}$ consists of at most two (maximal) segments, and each segment has two endpoints on the boundary of $\mathcal{B}$. Thus there are at most 4 such endpoints on each line $l \in D$, and the total number of endpoints is $O(n)$. (See Fig. 7b). These endpoints can be calculated in $O(\log n)$ time for each line due to the convexity of LS_1. We then sort the $O(n)$ endpoints on the boundary $\mathcal{B}$ in counterclockwise order (See Fig. 7c), and for the i-th endpoint, we use s_i to denote the segment ending at it. We traverse these endpoints in order along the boundary of $\mathcal{B}$. During the traversal, we maintain an ordered list L, and a counter N. For the i-th endpoint, let i' be the index such that $s_i = s_{i'}$. If i' is not in L, then we add i to L; otherwise we delete i' from L and increase N by the size of the set $\{j \in L \mid j > i'\}$.

▶ **Lemma 15.** *After the traversal, the counter N is the number of candidates of $\theta = \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)}$ in $[\theta_0, \theta_1]$.*

Proof. For each $\lambda \in \mathsf{Cand}_2(D)$ which is determined by an intersection of two lines $l(\lambda) = l'(\lambda)$ for $l, l' \in D$, the corresponding segments in $\mathcal{B}$ must have four endpoints with indices $i' < j < i < j'$ where i and i' belong to l while j and j' belong to l'. Thus, we will increase the counter N for exactly one time, i.e., when i' is deleted from L and we find that $j < i'$. Therefore, there exists a bijection between all candidate values and all increments of the counter N. So the lemma holds. ◀

▶ **Example 16.** As an example, consider the instance presented in Figure 7c. The endpoints of segments are labeled counterclockwise. Starting from endpoint 1, we in turn add $1, 2, 3, 4, 5$ into the list L. Then for endpoint 6, since $s_2 = s_6$ the list becomes $\{1, 3, 4, 5\}$, and N increases by 3. For endpoint 7, we delete 4 from the list and increase N by 1.

The above computes the number of candidates N. We assume $N > n$. Next, we uniformly and randomly pick n candidates out of the N candidate values in $[\theta_0, \theta_1]$. To this end, we first uniformly (with replacement) pick a set S of n indices from $\{1, \ldots, N\}$. Then we compute the candidate values corresponding to these picked indices, which can be done by doing the above traversal again. Specifically, during the traversal, suppose that after processing an endpoint i, the counter N grows from N_0 to N_1. Then we add the following candidate values: for every index $k \in (N_0, N_1] \cap S$, we add the candidate value determined by the intersection of the segment s_i and the segment corresponding to the $(k - N_0)$-th largest endpoint in the current ordered list L maintained during the traversal.

The above (at most) n candidate values of θ can be regarded as uniform samples from all candidate values in the search range $[\theta_0, \theta_1]$. We sort and perform a binary search on these values using the decision procedure D-Greedy-k (with $k = 1$) to find two adjacent values θ'_1 and θ'_2 in the sorted list such that the optimal θ value is in the range $(\theta'_0, \theta'_1]$. Then, we shrink the range $[\theta_0, \theta_1]$ by updating $\theta_0 = \theta'_0$ and $\theta_1 = \theta'_1$, and proceed to the next round.

We proceed as above until there are at most n candidate values in the search range (i.e., $N \leq n$). Finally, we run the following *post-processing step*. Let U be the union of the set of these at most n candidate values in the search range and $\mathsf{Cand}_1(D)$. By the above algorithm, the optimal value θ is in U. Since $|\mathsf{Cand}_1(D)| = O(n)$, $|U| = O(n)$. We sort and perform a binary search on the values of U using the decision procedure D-Greedy-k to eventually compute the optimal θ value. This finishes our algorithm.

To analyze the running time, it is easy to see that the post-processing step takes $O(n \log n)$ time. Below, we analyze the algorithm before the post-processing step.

We first consider the running time for each round. In each round, the algorithm computes and sorts the line segment endpoints on the boundary of $\mathcal{B}$, in $O(n \log n)$ time. Maintaining the list L for a traversal of $O(n)$ endpoints can be done in $O(n \log n)$ time using a balanced binary search tree. Sorting and doing the binary search on the sampled n values takes $O(n \log n)$ time, since the procedure D-Greedy-k is called $O(\log n)$ times. Thus, the total running time of each round is $O(n \log n)$.

Next, we show that the algorithm has a constant $p^* > 0$ probability to proceed into post-processing within two rounds. Indeed, let $p(n_0, n_1)$ denote the probability of reducing the size of the candidate set from n_0 to at most n_1 in one round. We can obtain that $p(n_0, n_1) \geq 1 - n_0 \cdot \left(\frac{n_0 - n_1}{n_0}\right)^n$, because the size after the round is larger than n_1 only if our algorithm did not pick any indices from some interval $[i, i + n_1)$ (here the indices i refer to the θ values of the candidate set after they are sorted). For large enough n, we can see that $p^* = p(n^2, n^{3/2}) \cdot p(n^{3/2}, n) \geq (1 - n^2 e^{-\sqrt{n}})^2$ is close to 1. Suppose the algorithm terminates at round t (t is a random variable). For any $r \geq 3$, it holds that $\Pr[t \geq r] \leq (1 - p^*)^{r-2}$. Overall, the expected number of rounds is $\mathbb{E}[t] \leq 2 + \sum_{r=3}^{\infty} \Pr[t \geq r] \leq 2 + \sum_{r=3}^{\infty} (1 - p^*)^{r-2} = O(1)$.

By the above analysis, our algorithm, named E-Greedy-1, has the following performance.

▶ **Theorem 17.** *E-Greedy-1 solves RMS in $O(n \log n)$ expected time.*

4 NP-Hardness

The main results of this section are the NP-hardness results in Theorem 18 and Theorem 19.

▶ **Theorem 18.** *Dec-RMS is* NP*-hard even in $\mathbb{R}^3$.*

Note that Dec-RMS in $\mathbb{R}^d$ for $d > 3$ generalizes Dec-RMS in $\mathbb{R}^3$ (by adding a few dummy dimensions). Further, by duplicating each point k times in an Dec-RMS instance, we can create a Dec-k-RMS instance with exactly the same optimal solution as the Dec-RMS instance. This implies the following hardness result.

▶ **Theorem 19.** *Dec-k-RMS is* NP*-hard for any fixed $k \geq 1$ and $d \geq 3$.*

In the following, we will prove Theorem 18, by a reduction from the vertex cover problem on a special planar graph, defined as follows.

A *planar straight-line graph (PSLG)* [28] is a graph $G = (V, E)$ where V is a finite subset of $\mathbb{R}^2$, and E is a subset of mutually disjoint open line segments with both endpoints in V. A *face* of a PSLG is a connected component of $\mathbb{R}^2 \setminus (V \cup E)$. The unique unbounded face is called the *outer face*, and all others are called *inner faces*. Similarly, vertices on the boundary of the outer face are called *outer vertices*, and all others are called *inner vertices*. Notice that in a PSLG, every inner face is an open polygon, and thus for every inner vertex with degree t, there are t interior angles attached to it. We say that a PSLG is *convex* if every inner face is convex, and the complement of the outer face is also convex.

Given an undirected graph $G = (V, E)$, a *vertex cover* is a subset of vertices $S \subseteq V$ that cover all edges in E (i.e., every edge in E is incident on some vertex in S). The *vertex cover (VC)* problem asks for a vertex cover of minimum cardinality. Das and Goodrich [10] showed that the VC problem on a convex PSLG is NP-hard, and below we do the reduction to prove Theorem 18.

We first define an intermediate problem, called the *inner vertex cover problem* (IVC), on a class of so-called *normalized PSLG graphs*, and provide a reduction from VC convex PSLG to it in Section 4.1. Then, we further reduce the problem to Dec-RMS in Section 4.2.

4.1 IVC on Normalized PSLG

We define a PSLG to be *normalized* if it satisfies the following properties:

- (*convex*) Every inner face is convex, and the complement of the outer face is also convex;
- (*low-degree*) Every inner vertex has degree 2 or 3, and every outer vertex has degree 3.
- (*bounded-angle*) Every interior angle attached to an inner vertex of degree 3 is in the range $[\pi/4, \pi)$.
- (α*-isometric*) There exist a standard length l and a constant $\alpha \in (0, 1)$ such that every edge with at least one endpoint being inner vertex has a length in range $[l(1-\alpha), l(1+\alpha)]$, and such an edge must also have at least one endpoint with degree 2.

Given a normalized PSLG, the *inner vertex cover (IVC)* problem on PSLG asks for a vertex cover which contains all the outer vertices of G. Given an instance of convex PSLG $G_0 = (V_0, E_0)$, we first construct a normalized PSLG $G_4 = (V_4, E_4)$ through other three intermediate graphs, such that VC on G_0 is reduced to IVC on G_4. The construction can be found in the full version, and we also decide the value of the parameter $\alpha = \Omega(1/|V_0|)$ there.

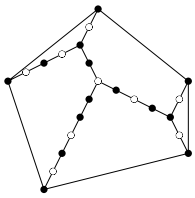


Figure 8 A normalized PSLG with an inner vertex cover labelled by black vertices.

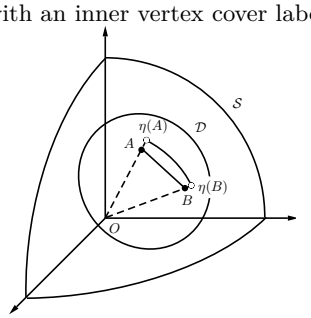


Figure 9 Sphere projection for G_4 from the disk $\mathcal{D}$.

▶ **Lemma 20.** *There is a polynomial reduction from VC on a convex PSLG to IVC on a normalized PSLG.*

4.2 Reduction to Dec-RMS

Then we construct a 3D point set D for Dec-RMS from the above IVC instance on the normalized PSLG $G_4 = (V_4, E_4)$. Consider the eighth sphere $\mathcal{S} = \{(x, y, z) \in \mathbb{R}^3_+ \mid x^2 + y^2 + z^2 = \rho^2\}$, where ρ is a constant depending on the parameters of G_4. and we draw the PSLG G_4 in the unit disk $\mathcal{D}$ inscribed in $\mathcal{S}$. From the origin O, we project the vertices and the edges $V_4 \cup E_4$ onto $\mathcal{S}$, and denote the projection mapping by η. Note that straight lines in E_4 are projected to arcs of great-circles, and thus the faces in the projection image are still convex. In fact, when ρ is large enough, the image of the graph does not deform a lot. Formally, we have Lemma 21, whose proof is omitted and can be found in the full version.

▶ **Lemma 21.** *For any two points $A, B \in \mathcal{D}$, we have*

$$\rho^2\sqrt{1-\frac{AB^2}{\rho^2-1}} \le \langle \eta(A), \eta(B)\rangle \le \rho^2\left(1-\frac{AB^2}{2\rho^2}\right).$$

Let D be the projection image of V_4. Thus, $|D| = |V_4|$, and D can be computed in polynomial time. The convexity of the sphere $\mathcal{S}$ ensures that D forms the 1-level of itself. Intuitively, the normalizing constraints on the degrees, angles, and edges of G_4 make sure that a point of D outside a subset $R \subseteq D$ could keep a small regret ratio if and only if all its neighbors are in R. We claim that by properly choosing θ in the Dec-RMS problem, a subset $R \subseteq D$ has 1-regret ratio at most $1 - \theta$ if and only if $\eta^{-1}(R)$ is an inner vertex cover of G_4, and this is implied by the following two lemmas.

▶ **Lemma 22.** *There exists a constant θ such that for any subset $R \subseteq D$, if $\eta^{-1}(R)$ is an inner vertex cover of G_4, then 1-regratio$(R) \le 1 - \theta$.*

By careful computations in the proof of Lemma 22 in the full version, we can set $\theta = \sqrt{1 - l^2(1+\alpha)^2/(\rho^2-1)}$. For carefully chosen values of l, α and ρ, we can prove:

▶ **Lemma 23.** *For any subset $R \subseteq D$, if $\eta^{-1}(R)$ is not an inner vertex cover of G_4, then 1-regratio$(R) > 1 - \theta$.*

Combining Lemmas 22 and 23 leads to Lemma 24.

▶ **Lemma 24.** *For any integer r, G_4 has an inner vertex cover of size r if and only if D has an 1-regret set of size r with ratio $1 - \theta$, where D, θ can be obtained from G_4 in polynomial time.*

Theorem 18 thus follows.

5 Algorithms in High Dimensions

5.1 The Problem RMS

The concept of ϵ-*kernel* was introduced by Agarwal et al. [2]. By showing that RMS is closely connected to ϵ-kernel, we obtain an approximation algorithm for RMS and an upper bound of the maximum regret ratio, which improves the previous result [26].

A subset R of D is an ϵ-kernel if $\frac{\max_{x\in R}\langle x,\omega\rangle - \min_{y\in R}\langle y,\omega\rangle}{\max_{x\in D}\langle x,\omega\rangle - \min_{y\in D}\langle y,\omega\rangle} \geq 1 - \epsilon$, for any non-zero real vector ω. Roughly speaking, an ϵ-kernel is a subset that approximately preserves the *width* of the data set in every direction. It is well known that an ϵ-kernel of constant size can be computed in linear time (when $d = O(1)$).

▶ **Theorem 25.** [2, 6, 35] *Given D in $\mathbb{R}^d$, one can compute an ϵ-kernel of D of size $O(\epsilon^{-(d-1)/2})$ in $O(|D| + 1/\epsilon^d)$ time.*

We reduce the RMS problem to the ϵ-kernel problem as follows. Recall that due to our assumption, all points of D are in the first orthant. We make 2^d copies of D in every orthant as follows. Define $D^{\pm} = \{(p[1]x[1], \ldots, p[d]x[d]) \mid (x[1], \ldots, x[d]) \in D, p[i] \in \{\pm 1\})\}$. Suppose we have already found an ϵ-kernel R' of $D^{\pm}$; then we can project the subset back to D by taking the absolute value in each coordinate, as follows. Define

$$\mathsf{abs}(x) := (|x[1]|, \ldots, |x[d]|), \qquad R = \mathsf{abs}(R') := \{\mathsf{abs}(x) \mid x \in R'\}.$$

▶ **Lemma 26.** *If R' is an ϵ-kernel of $D^{\pm}$, then R must have regret ratio at most ϵ in D.*

Using the above reduction and observing that $|R| \leq |R'|$, it is immediate to translate Theorem 25 to an approximation algorithm for RMS in high dimensions.

▶ **Corollary 27.** *Fixing the dimension d, one can compute a subset $R \subseteq D$ of size r with maximum regret ratio $O(r^{-2/(d-1)})$ in $O(2^d n + r^{2d/(d-1)})$ time.*

The upper bound on the regret ratio is better than the previous upper bound $O(r^{-1/(d-1)})$ given in [26].

5.2 The Problem k-RMS

Given r and k, the goal of k-RMS is to compute a set R of r points from D such that the maximum regret ratio is minimized. Let $1-\theta^*$ denote the maximum regret ratio in the optimal solution, for some $\theta^* \in [0, 1]$. Viewing each point in D as a $(d-1)$-dimensional hyperplane, Theorem 28 follows from a reduction to the set cover problem over the arrangement of these hyperplanes. See the full version for a complete proof.

▶ **Theorem 28.** *There exists a polynomial time algorithm that can compute a set R of at most $r \cdot (d \cdot \ln(2n) + 1)$ points from D with maximum regret ratio at most $1 - \theta^*$.*

6 Related Work

Due to the individual drawback of top-k queries and skyline queries, there exist a variety of ways to combine these two queries in the literature. Top-k skyline select and top-k skyline join were proposed in [15]. ϵ-skyline [33] controls the output size with respect to ϵ after the utility function specified by the user is known. However, these studies still require that the utility function should be known beforehand.

The RMS query we study in this paper has the attractive property that no information on the utility function has to be provided by the user. Since its introduction in [26], it has been extended and generalized to the interactive setting in [25] and the k-RMS problem in [8]. [27] proposed an efficient algorithm for 1-RMS. [19] generalized the notion of RMS to include nonlinear utility functions.

Computing k-level sets and obtaining tight size bounds are of fundamental importance in computational geometry. For the two-dimension case, [12] provided the best-known upper bound $O(nk^{1/3})$. However, the best-known lower bound is $\Omega(ne^{c\sqrt{\log k}})$ [32], which is still far from the upper bound, and closing the gap is an open problem for years. For algorithms that compute the k-level sets, we refer the interested readers to [5] and the references therein. Note that any improvement on computing the k-level sets may lead to improvement of the time bounds of our algorithms for k-RMS.

The notion of ϵ-kernel coreset was introduced in the seminal paper by Agarwal et al. [2]. They applied their algorithm for constructing ϵ-kernel to several shape fitting problems. Since then, the idea has been extended to many other settings such as clustering (e.g., [7, 13]), matrix approximation [11, 13] and stochastic points [17].

Acknowledgement. We are grateful to the anonymous reviewers for their constructive comments on this paper.

References

1 A greedy algorithm — the interval point cover problem. `http://www.cs.yorku.ca/~andy/courses/3101/lecture-notes/IntervalCover.html`.

2 Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

3 S Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

4 C.Y. Chan, HV Jagadish, K.L. Tan, A.K.H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.

5 Timothy M Chan. Remarks on k-level algorithms in the plane. *Manuscript, Department of Computer Science, University of Waterloo, Waterloo, Canada*, 1999.

6 Timothy M Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 152–159, 2004.

7 K. Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

8 Sean Chester, Alex Thomo, S Venkatesh, and Sue Whitesides. Computing k-regret minimizing sets. *Proceedings of VLDB*, 7(5), 2014.

9 Jan Chomicki, Paolo Ciaccia, and Niccolo' Meneghetti. Skyline queries, front and back. *SIGMOD Rec.*, 42(3):6–18, October 2013. `doi:10.1145/2536669.2536671`.

10 Gautam Das and Michael T Goodrich. On the complexity of approximating and illuminating three-dimensional convex polyhedra. In *Algorithms and Data Structures*, pages 74–85. Springer, 1995.

11 A. Deshpande, L. Rademacher, S. Vempala, and G. Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the 17th ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126, 2006.

12 Tamal K Dey. Improved bounds on planar k-sets and k-levels. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 156–161. IEEE, 1997.

13 D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 569–578, 2011.

14 P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD*, 2012.

15 M. Goncalves and M.E. Vidal. Top-k skyline: A unified approach. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 790–799. Springer, 2005.

16 John Hershberger. Finding the upper envelope of n line segments in o (n log n) time. *Information Processing Letters*, 33(4):169–174, 1989.

17 Lingxiao. Huang, Jian. Li, Jeff. Phillips, and Haitao. Wang. ϵ-kernel coresets for stochastic points. In *ESA*, 2016.

18 Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, October 2008. doi:10.1145/1391729.1391730.

19 Taylor Kessler Faulkner, Will Brackenbury, and Ashwin Lall. K-regret queries with nonlinear utilities. *Proc. VLDB Endow.*, 8(13):2098–2109, September 2015. doi:10.14778/2831360.2831364.

20 J. Lee, G. You, and S. Hwang. Personalized top-k skyline queries in high-dimensional space. *Information Systems*, 2009.

21 C.E. Leiserson, C. Stein, R. Rivest, and T.H. Cormen. *Introduction to algorithms*. MIT press, 2009.

22 X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.

23 J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters*, 39(4):183–187, 1991.

24 D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *VLDB*, 2009.

25 D. Nanongkai, A. Lall, A. D. Sarma, and K. Makino. Interactive regret minimization. In *SIGMOD*, 2012.

26 Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J Lipton, and Jun Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2):1114–1124, 2010.

27 P. Peng and R. C.-W. Wong. Geometry approach for k-regret query. In *ICDE*, 2014.

28 Franco P Preparata, Michael Ian Shamos, and Franco P Preparata. *Computational geometry: an introduction*, volume 5. Springer-Verlag New York, 1985.

29 L. Qin, J.X. Yu, and L. Chang. Diversifying top-k results. *VLDB*, 2012.

30 M.A. Soliman, I.F. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *IEEE 23rd International Conference on Data Engineering.*, pages 896–905. IEEE, 2007.

31 Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.

32 Géza Tóth. Point sets with many k-sets. *Discrete & Computational Geometry*, 26(2):187–194, 2001.

33 T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1397–1399. IEEE, 2008.

34 M.L. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. *The VLDB Journal*, 18(3):695–718, 2009.

35 Hai Yu, Pankaj K Agarwal, Raghunath Poreddy, and Kasturi R Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(3):378–402, 2008.

The Design of Arbitrage-Free Data Pricing Schemes

Shaleen Deep[1] **and Paraschos Koutris**[2]

1 **University of Wisconsin-Madison, Madison, WI, USA**
shaleen@cs.wisc.edu
2 **University of Wisconsin-Madison, Madison, WI, USA**
paris@cs.wisc.edu

Abstract

Motivated by a growing market that involves buying and selling data over the web, we study pricing schemes that assign value to queries issued over a database. Previous work studied pricing mechanisms that compute the price of a query by extending a data seller's explicit prices on certain queries, or investigated the properties that a pricing function should exhibit without detailing a generic construction. In this work, we present a formal framework for pricing queries over data that allows the construction of general families of pricing functions, with the main goal of avoiding arbitrage. We consider two types of pricing schemes: instance-independent schemes, where the price depends only on the structure of the query, and answer-dependent schemes, where the price also depends on the query output. Our main result is a complete characterization of the structure of pricing functions in both settings, by relating it to properties of a function over a lattice. We use our characterization, together with information-theoretic methods, to construct a variety of arbitrage-free pricing functions. Finally, we discuss various tradeoffs in the design space and present techniques for efficient computation of the proposed pricing functions.

1998 ACM Subject Classification H.2.4 [Systems] Relational Databases

Keywords and phrases Data pricing, Determinacy, Arbitrage

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.12

1 Introduction

The commodification of data over the last decade has created many unique research challenges, among them data privacy and pricing of data. In a broad range of application areas, data today is being collected at an unprecedented scale. This phenomenon has led to a growing market for so called big data brokers, who sell this data to buyers such as financial firms, retailers and insurance companies [6, 5].

In this paper, we investigate the problem of *query-based data pricing*, where the task is to assign prices to queries over a database, such that the price captures the amount of information revealed by asking the query. Traditionally, data pricing has been done either by allowing the buyer to access only certain queries with a fixed price set by the seller, or the buyer needs to purchase the whole dataset [20]. Although such an approach is conceptually simple, defining a large set of queries that are representative of the user's needs is a tall task for the data seller. Even if this is feasible, such a pricing scheme may allow *arbitrage*, which occurs when a data buyer can potentially buy data at a price less than what is set by the seller. It can also lead to prices that exhibit undesirable behavior.

Previous work in the area of data pricing has identified a set of *arbitrage conditions* that any reasonable pricing function should avoid. The fundamental arbitrage condition is

information arbitrage, first introduced in [16]. Intuitively, a query Q_1 that reveals a subset of the information that is revealed by another query Q_2 should be priced at most as much as Q_2. If not, an arbitrage opportunity occurs: a clever buyer can pay the price of Q_2 and then use the result of Q_2 to compute Q_1 for a lower price. A second arbitrage condition is *bundle arbitrage* [20]. Intuitively, asking simultaneously for Q_1 and Q_2 (as a bundle) should cost at most the sum of asking separately for each. Both [16, 20] propose pricing functions that avoid both arbitrage conditions. However, to the best of our knowledge, there exists no framework that supports a generic construction of pricing functions, and facilitates the analysis of the various tradeoffs in design choices.

Our Contribution. We address the question of designing arbitrage-free pricing schemes that assign prices to queries over a database. Our main result is a complete characterization of the structure of pricing functions for two pricing schemes: *answer-dependent prices* (APS), and *instance-independent prices* (QPS). We use this characterization to construct a variety of pricing functions, and also discuss the various tradeoffs involved in choosing the right pricing function. We summarize below our results in more detail.

We first study APS, where the price depends both on the query Q and on the answer of the query $E = Q(D)$. To characterize such schemes, we define the *conflict set*, which is the set of databases such that $Q(D) \neq E$. We show that any arbitrage-free pricing function is equivalent to a monotone and subadditive function over the join-semilattice defined by the conflict sets (Theorems 8 and 9). Equipped with this characterization, we present several examples of arbitrage-free functions, including the weighted coverage and the weighted set cover functions. In addition, we show that an answer-dependent pricing function with no bundle arbitrage leads to unnatural behavior: any query can cost at least half the price of the whole dataset for some databases. This suggests that there is a tradeoff that any data seller must take into account when choosing a pricing function.

Second, we examine the structure of QPS, where the pricing function depends only on the query Q. We prove that any non-trivial instance-independent pricing function must have weaker arbitrage guarantees compared to an answer-dependent function. To provide a characterize of functions in QPS, we view the query Q as a *partition* over the set of possible databases: our main results is that any arbitrage-free function is equivalent to a monotone and subadditive function over the elements of the join-semilattice formed from the partitions (Theorems 24 and 25).

To design pricing functions in QPS, we apply two methods. The first method applies an appropriate aggregate function to combine the prices of an arbitrage-free answer-dependent function (Theorem 28). The second method views the database as a random variable (with some probability distribution over the possible databases), and computes the price as the *information gain* of the data buyer after the answer has been revealed (Section 4.4). This approach is parallel to work on side-channel attacks [13], and quantitative information flow [14]. By using different entropy measures, such as *Shannon entropy*, or *min-entropy*, we obtain pricing functions that we prove to be arbitrage-free using the machinery we developed.

Third, we show how the proposed pricing functions can be computed efficiently in practical settings. We discuss two different techniques. The first method restricts the computation of a pricing function to a small set of databases (instead of all possible databases). The second method uses approximation techniques to estimate the price within a small margin of error.

Organization. Section 2 presents the key concepts, terminology and notation that we use throughout the paper. In Section 3, we study the construction and properties of pricing

functions for the answer-dependent case. Section 4 details the corresponding problem for instance-independent pricing schemes. Section 5 discusses techniques to compute a pricing function efficiently. We present the related work and conclude in Sections 6 and 7 respectively.

2 Notation and Framework

In this section, we set up the necessary notation and formally describe the pricing framework.

2.1 Preliminaries

We fix a relational schema $\mathbf{R} = (R_1, \ldots, R_k)$; we use D to denote a database instance that uses the schema. We will use $\mathcal{I}$ to denote the set of possible database instances. The set $\mathcal{I}$ encodes information about the database that is provided by the data seller, and is public information known to any data buyer. Further, we allow the set $\mathcal{I}$ to be infinite, but countable. For example, suppose that the schema consists of a single binary relation $R(A, B)$ and we know that the domain of both attributes is $[n] = \{1, \ldots, n\}$. Then, $\mathcal{I} = 2^{[n]\times[n]}$, which represents equivalently the set of all possible directed graphs on the vertex set $[n]$.

We will view a *query* Q from some query language $\mathcal{L}$ as a deterministic function that takes as input a database instance $D \in \mathcal{I}$ and returns an output $Q(D)$. In this paper, we do not impose any restriction on the query language $\mathcal{L}$, but in the examples we will use and in some of the design tradeoffs we assume Q is either a *conjunctive query (CQ)* or a *union of conjunctive queries (UCQ)*. A *query bundle* $\mathbf{Q} = (Q_1, \ldots, Q_n)$ is a finite set of queries that is asked simultaneously on the database. We denote by $B(\mathcal{L})$ the set of finite query bundles from the language $\mathcal{L}$. Given two query bundles $\mathbf{Q}_1, \mathbf{Q}_2$, we denote their union as $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$.

Queries as Partitions. It will be handy to provide an alternative viewpoint of a query bundle $\mathbf{Q}$ as a partition over the set of instances $\mathcal{I}$. A *partition* $\mathcal{P} = \{B_1, \ldots, B_k\}$ of $\mathcal{I}$ is a set of pairwise disjoint sets $B_i \subseteq \mathcal{I}$, which we call *blocks*, such that $\cup_{i=1}^{k} B_i = \mathcal{I}$. Given $\mathbf{Q} \in \mathcal{L}$, we denote by $\mathcal{P}_{\mathbf{Q}}$ the partition that is induced by the following *equivalence relation*: $D \sim D'$ iff $\mathbf{Q}(D) = \mathbf{Q}(D')$ and $\mathbf{Q} \in \mathcal{L}$. In other words, two databases belong in the same block of the partition if and only if their output for $\mathbf{Q}$ is indistinguishable. We use the standard notation $[D]_{\mathbf{Q}}$ to denote the equivalence class in which D belongs; in other words, $[D]_{\mathbf{Q}} = \{D' \in \mathcal{I} \mid \mathbf{Q}(D') = \mathbf{Q}(D)\}$. For two partitions $\mathcal{P}_1, \mathcal{P}_2$, we say that $\mathcal{P}_1$ *refines* $\mathcal{P}_2$, and write $\mathcal{P}_1 \succeq \mathcal{P}_2$, if every block of $\mathcal{P}_1$ is a subset of some block in $\mathcal{P}_2$. In other words, $\mathcal{P}_1$ is a more fine-grained partition of $\mathcal{I}$ than $\mathcal{P}_2$.

Lattices and Join-Semilattices. A *join-semilattice* $(L, \leq)$ is a partially ordered set in which every two elements in L have a unique supremum (called *join* and denoted as $\vee$). A *lattice* $(L, \leq)$ is a partially ordered set in which every two elements in L have both a unique supremum, and a unique infimum (called *meet* and denoted $\wedge$). In this paper, we will consider two different join-semilattices. The first semilattice has elements subsets of $\mathcal{I}$, which are ordered by subset inclusion $\subseteq$. The second semilattice has elements partitions of $\mathcal{I}$, which are ordered by the refinement relation $\preceq$.

Let $f : L \to \mathbb{R}$ be a function defined on the elements of the join-semilattice. We say that f is *monotone*, or *isotone*, if whenever $A \leq B$, then $f(A) \leq f(B)$. Moreover, we say that f is *subadditive* if for any two elements A, B of the semilattice we have $f(A \vee B) \leq f(A) + f(B)$.

2.2 The Pricing Framework

In our setting, a data seller offers a database instance D for sale. Data buyers can issue queries on the database in the form of query bundles $\mathbf{Q}$. For each query $\mathbf{Q}$ over the instance D, the task in hand is to assign a *price* to the query answer $\mathbf{Q}(D)$ that reflects the amount of information gained by the data buyer. When a price is assigned to a query bundle $\mathbf{Q}$, we can differentiate between three different pricing strategies, which depend on the parameters used to compute the price. There are three possible parameters we can use to determine the price of a query: the query bundle $\mathbf{Q}$, the answer of the query on the database D, denoted $E = \mathbf{Q}(D)$, and the database D itself. The price will obviously depend on which query $\mathbf{Q}$ we issue, but there is a choice of which D, E should be further used to compute the price. This choice defines three different classes of pricing schemes:

- **Instance-independent (QPS):** the price depends only on $\mathbf{Q}$, in which case the pricing function is of the form $p(\mathbf{Q})$. The price is independent of the underlying data.
- **Answer-dependent (APS):** the price depends on the answer $E = \mathbf{Q}(D)$, so the price is of the form $p(\mathbf{Q}, E)$. In this case, the price depends on the query and the query output.
- **Data-dependent (DPS):** the price depends on the underlying database D, so the pricing function is of the form $p(\mathbf{Q}, D)$.

Any instance-independent scheme can be cast as an answer-dependent scheme, and any answer-dependent scheme as a data-dependent scheme. The distinction between APS and DPS was introduced in [20], where the authors use the terminology *delayed pricing* and *up-front pricing* respectively. Notice that both in QPS and APS the prices themselves do not leak any information about the underlying data D.[1] In contrast, a data-dependent pricing scheme can leak information about the data (for more details see [20]). For this reason, in this paper we focus on the first two types of pricing schemes: QPS and APS.

The reason we consider query bundles in our setting is that in practice a data buyer will issue over time a sequence $\mathbf{Q}_1, \ldots, \mathbf{Q}_m$ of query bundles on the database. In this case, after issuing the first i queries, the data buyer should not be charged a price of $\sum_i p(\mathbf{Q}_i, D)$, but instead $p(\mathbf{Q}_1, \ldots, \mathbf{Q}_i, D)$. Notice here that, even if a user issues only single queries, we still need to be able to price a query bundle.

2.3 Arbitrage Conditions

Assigning prices to query bundles without any restrictions can lead to the occurrence of arbitrage opportunities. In [15], the authors presented a single condition that captures arbitrage. Here, we follow [20], and consider independently two different conditions where arbitrage may occur.

Information Arbitrage. The first condition captures the intuition that the price of query bundle must capture the amount of information that an answer reveals about the actual database D. In particular, if a query bundle $\mathbf{Q}_1$ reveals a subset of information than a query bundle $\mathbf{Q}_2$ reveals, the price of $\mathbf{Q}_1$ must be less than the price of $\mathbf{Q}_2$. If this condition is not satisfied, it creates an arbitrage opportunity, since a data buyer can purchase $\mathbf{Q}_2$ instead, and use it to obtain the answer of $\mathbf{Q}_1$ for a cheaper price.

[1] For the case of answer-dependent prices, we must make sure that we reveal the price only if we are certain that the buyer will be charged for the cost.

Bundle Arbitrage. The second condition regards the scenario where a data buyer that wants to obtain the answer for the bundle $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$ creates two separate accounts, and uses one to ask for $\mathbf{Q}_1$ and the other to ask for $\mathbf{Q}_2$. To avoid such an arbitrage situation, we must make sure that the price of $\mathbf{Q}$ is at most the sum of the prices for $\mathbf{Q}_1$ and $\mathbf{Q}_2$. [20] uses the terminology *separate-account arbitrage* to refer to this arbitrage condition.

We will show in the next sections how to mathematically formalize information arbitrage and bundle arbitrage for both APS and QPS.

3 Answer-Dependent Pricing

In this section, we study the design of *answer-dependent* pricing schemes. In an APS the pricing function takes the form $p(\mathbf{Q}, E)$, where $\mathbf{Q}$ is a query bundle and $E \in \{\mathbf{Q}(D) \mid D \in \mathcal{I}\}$. Throughout the section, we assume that query bundles belong to some query language $\mathcal{L}$. We first discuss how to formalize the arbitrage conditions. To formally describe information arbitrage, we use the notion of *data-dependent determinacy*.

▶ **Definition 1.** We say that $\mathbf{Q}_2$ determines $\mathbf{Q}_1$ under database D, denoted $D \vdash \mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$ if for every database D' such that $\mathbf{Q}_2(D) = \mathbf{Q}_2(D')$, we also have $\mathbf{Q}_1(D') = \mathbf{Q}_1(D)$.

The above definition of determinacy is different from *query determinacy* [23, 24], since it is defined with respect to a given database D. It is also easy to see that if $D \vdash \mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$, we also have that $D' \vdash \mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$ for any database D' such that $\mathbf{Q}_2(D) = \mathbf{Q}_2(D')$.

▶ **Definition 2** (APS Information Arbitrage)**.** Let $\mathbf{Q}_1, \mathbf{Q}_2$ be two query bundles. We say that the pricing function p has no *information arbitrage* if for every database $D \in \mathcal{I}$, $D \vdash \mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$ implies that $p(\mathbf{Q}_2, E_2) \geq p(\mathbf{Q}_1, E_1)$, where $E_i = \mathbf{Q}_i(D)$ for $i = 1, 2$.

This definition of information arbitrage captures both *post-processing arbitrage* and *serendipitous arbitrage*, as these are defined in [20]. For the case of bundle arbitrage, we formalize it as follows.

▶ **Definition 3** (APS Bundle arbitrage)**.** Let the query bundle $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$. We say that the price function p has no *bundle arbitrage* if for every database $D \in \mathcal{I}$, we have $p(\mathbf{Q}, E) \leq p(\mathbf{Q}_1, E_1) + p(\mathbf{Q}_2, E_2)$, where $E = \mathbf{Q}(D)$ and $E_i = \mathbf{Q}_i(D)$ for $i = 1, 2$.

We say that an answer-dependent pricing function is *arbitrage-free* if it has no information arbitrage and no bundle arbitrage.

3.1 How to Find a Pricing Function

In this section, we characterize the family of answer-dependent pricing functions that satisfy both arbitrage conditions. The critical component is the notion of a *conflict set*.

3.1.1 Conflict Sets

Consider a query bundle $\mathbf{Q} \in B(\mathcal{L})$, a database $D \in \mathcal{I}$ and let $E = \mathbf{Q}(D)$. We define

$$\mathcal{S}_{\mathbf{Q}}(E) = \{D' \in \mathcal{I} \mid \mathbf{Q}(D') = E\}, \qquad \overline{\mathcal{S}}_{\mathbf{Q}}(E) = \{D' \in \mathcal{I} \mid \mathbf{Q}(D') \neq E\}$$

In other words, $\mathcal{S}_{\mathbf{Q}}(E)$ computes the set of databases that "agree" with the view extension E, and $\overline{\mathcal{S}}_{\mathbf{Q}}(E)$ contains the complement set, i.e. the set of databases that "disagree" with E. Notice that $\mathcal{S}_{\mathbf{Q}}(\mathbf{Q}(D)) = [D]_{\mathbf{Q}}$. We refer to $\overline{\mathcal{S}}_{\mathbf{Q}}(E)$ as the *conflict set* for query $\mathbf{Q}$ and extension E, while we refer to $\mathcal{S}_{\mathbf{Q}}(E)$ as the *agreement set*. It is straightforward that $\overline{\mathcal{S}}_{\mathbf{Q}}(E) = \mathcal{I} \setminus \mathcal{S}_{\mathbf{Q}}(E)$.

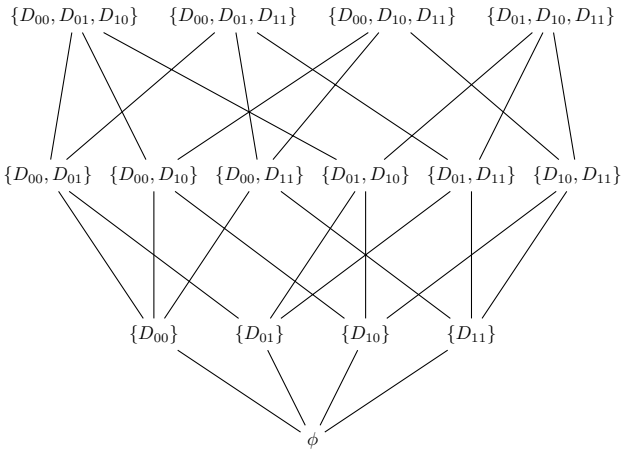


Figure 1 A simultaneous depiction of the join-semilattices for the four databases in Example 4.

▶ **Example 4.** We will use the following scenario as a running example throughout this section. Suppose that we have a binary relation $R(\underline{A}, B)$, where attribute A is the key. The values of the n keys are also publicly known $\{a_1, a_2, \ldots, a_n\}$. Moreover, assume that B can take two possible values from $\{0, 1\}$. It is easy to see that $\mathcal{I}$ consists of 2^n databases. For $n = 2$, let D_{ij} denote the database $\{(a_1, i), (a_2, j)\}$. For example $D_{01} = \{(a_1, 0), (a_2, 1)\}$.

Consider now the query $Q(x) = R(a_1, x)$, which asks for value of attribute B for the tuple with key $A = a_1$. Assume that the underlying database is D_{01}. The conflict set of Q and $E = Q(D_{01})$ consists of all databases D for which $(a_1, 1) \in D$, hence $\mathcal{S}_Q(E) = \{D_{10}, D_{11}\}$.

If $\mathbf{Q}$ returns a constant answer for every database in $\mathcal{I}$, the conflict set will be the empty set. On the other hand, if $\mathbf{Q}$ reveals the whole database D, the conflict set will be $\mathcal{I} \setminus \{D\}$. We can now define the set of all possible conflict sets for a database D and a given language $\mathcal{L}$ as $\mathcal{S}_D^{\mathcal{L}} = \{\overline{\mathcal{S}}_{\mathbf{Q}}(\mathbf{Q}(D)) \mid \mathbf{Q} \in B(\mathcal{L})\}$. The following lemma, which we prove in [8], shows that $\mathcal{S}_D^{\mathcal{L}}$ forms a *join-semilattice* under the partial order $\subseteq$, where the join operator is set union.

▶ **Lemma 5.** *Let $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$. For a database $D \in \mathcal{I}$, let $E_1 = \mathbf{Q}_1(D)$, $E_2 = \mathbf{Q}_2(D)$, and $E = \mathbf{Q}(D)$. Then, $\overline{\mathcal{S}}_{\mathbf{Q}}(E) = \overline{\mathcal{S}}_{\mathbf{Q}_1}(E_1) \cup \overline{\mathcal{S}}_{\mathbf{Q}_2}(E_2)$.*

The diagram in Figure 1 depicts simultaneously the four join-semilattices for each of the databases in Example 4. We next show in [8] the following lemma that connects the notion of a conflict set with data-dependent determinacy.

▶ **Lemma 6.** *Let $\mathbf{Q}_1, \mathbf{Q}_2$ be two query bundles, and $D \in \mathcal{I}$ be a database. Let $E_i = \mathbf{Q}_i(D)$ for $i = 1, 2$. The following two statements are equivalent:*

1. $D \vdash \mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$
2. $\overline{\mathcal{S}}_{\mathbf{Q}_2}(E_2) \supseteq \overline{\mathcal{S}}_{\mathbf{Q}_1}(E_1)$

Lemma 6 and Lemma 5 demonstrate that information and bundle arbitrage can be cast as conditions on the elements of the semilattice of conflict sets.

▶ **Example 7.** Continuing Example 4, consider the queries $Q_1(x) = R(a_1, x)$ and $Q_2() = R(x, 1)$. Let D_{00} be the underlying database. It is easy to see that $D_{00} \vdash Q_2 \twoheadrightarrow Q_1$, since after asking Q_2 we learn that the database contains no 1 values for B, and thus it must have only 0 values. The conflict sets for $E_1 = Q_1(D_{00})$, $E_2 = Q_2(D_{00})$ are $\overline{\mathcal{S}}_{Q_1}(E_1) = \{D_{11}, D_{10}\}$ and $\overline{\mathcal{S}}_{Q_2}(E_2) = \{D_{01}, D_{10}, D_{11}\}$ respectively.

3.1.2 A Characterization of Arbitrage-Free APS

We can now use the notion of a conflict set to define pricing functions of the form $p(\mathbf{Q}, E) = f(\overline{\mathcal{S}}_{\mathbf{Q}}(E))$, where $f : 2^{\mathcal{I}} \setminus \{I\} \to \mathbb{R}_+$ is a *set function*. It is straightforward to see that such a pricing function is by construction in APS, since the computation depends only on $\mathbf{Q}$ and E, and not on the database D. For example, if $\mathbf{Q}$ returns a constant answer for every database in $\mathcal{I}$, $p(\mathbf{Q}, E) = f(\emptyset)$. On the other hand, if $\mathbf{Q}$ reveals the whole database D, $p(\mathbf{Q}, E) = f(\mathcal{I} \setminus \{D\})$. We can now show a necessary and sufficient characterization of answer-dependent functions with no information arbitrage in terms of such a function f.

▶ **Theorem 8.** *Let p be an answer-dependent pricing function. The following two statements are equivalent:*
1. *p has no information arbitrage.*
2. *$p(\mathbf{Q}, E) = f(\overline{\mathcal{S}}_{\mathbf{Q}}(E))$, where f is a monotone function over every semilattice $\mathcal{S}_D^{\mathcal{L}}$.*

We have shown that in order to avoid information arbitrage it suffices to restrict the function to be monotone. We next demonstrate a similar connection of bundle arbitrage to the property of subadditivity.

▶ **Theorem 9.** *Let $p(\mathbf{Q}, E) = f(\overline{\mathcal{S}}_{\mathbf{Q}}(E))$ be a pricing function, where f is a set function. Then, the following two statements are equivalent:*
1. *p has no bundle arbitrage.*
2. *f is subadditive over every semilattice $\mathcal{S}_D^{\mathcal{L}}$.*

Both Theorem 8 and Theorem 9 are proved in the full version of the paper [8]. Observe that if a function f is monotone and subadditive over $2^{\mathcal{I}}$, it will also be monotone and subadditive over every semilattice $\mathcal{S}_D^{\mathcal{L}}$. Hence, as a corollary we can describe a general family of arbitrage-free pricing functions.

▶ **Corollary 10.** *Let f be a monotone and subadditive set function f. Then, the function $p(\mathbf{Q}, E) = f(\overline{\mathcal{S}}_{\mathbf{Q}}(E))$ is an answer-dependent pricing function that is arbitrage-free.*

3.2 Explicit Constructions of Pricing Functions

We have so far described a general class of functions that are both information and bundle arbitrage-free. Since any submodular function is also subadditive, any monotone submodular set function f will also produce a desired pricing function. We give some concrete examples of arbitrage-free pricing functions below.

▶ **Corollary 11.** *Suppose that we assign a weight of w_D to each $D \in \mathcal{I}$, such that $\sum_{D \in \mathcal{I}} w_D < \infty$. Then, the following pricing functions are arbitrage-free:*
1. *the weighted coverage function: $\sum_{D:\mathbf{Q}(D) \neq E} w_D$.*
2. *the supremum function: $\sup_{D:\mathbf{Q}(D) \neq E} w_D$.*[2]
3. *the budget-limited weighted coverage function for some $B \geq 0$: $\min\{B, \sum_{D:\mathbf{Q}(D) \neq E} w_D\}$.*

We can construct richer pricing functions by combining the weighted coverage function with a concave function g. Indeed, we can show that $p(\mathbf{Q}, E) = g(\sum_{D \in \overline{\mathcal{S}}_{\mathbf{Q}}(E)} w_D)$ is arbitrage-free for any concave function g. If $\mathcal{I}$ is finite, we can assign to each database $D \in \mathcal{I}$ an equal weight, in which case we obtain the arbitrage-free function $p(\mathbf{Q}, E) = g(|\overline{\mathcal{S}}_{\mathbf{Q}}(E)|)$.

[2] The supremum becomes equivalent to the max function if $\mathcal{I}$ is finite.

▶ **Corollary 12.** *Suppose that we assign a weight of w_D to each $D \in \mathcal{I}$, such that $\sum_{D\in\mathcal{I}} w_D < \infty$. Then, the pricing function $p(\mathbf{Q}, E) = g(\sum_{D\in\overline{\mathcal{S}}_{\mathbf{Q}}(E)} w_D)$ is arbitrage-free for any concave function g.*

Proof. We know that if $f(A)$ is a modular set function and g is concave, then $g(f(A))$ is a submodular function. Notice that $f(A) = \sum_{i\in A} w_i$ is a modular function for any choice of weights w_i. ◀

The pricing functions we have presented thus far are constructed by assigning a weight to each database in $\mathcal{I}$. Another type of construction starts by specifying a family $\mathcal{F}$ of subsets of $\mathcal{I}$. For each subset $S \in \mathcal{F}$, we assign a weight w_S. Finally, we pick some real number $B \geq \max_{S\in\mathcal{F}} w_S$. We define the *weighted set cover* function $f(A)$ as the cost of the minimum set cover for A if such a set exists, otherwise $f(A) = B$.

▶ **Lemma 13.** *The weighted set cover pricing function is arbitrage-free.*

The weighted set cover function generalizes the approach from [15], where explicit prices are specified for certain views, and the price of the query is computed as the cheapest set of views that determine the query. Indeed, if we are given explicit price points $(\mathbf{Q}_i, p_i)$ for $i = 1, \ldots, m$, we can define the following family of sets: $\mathcal{F} = \{\overline{\mathcal{S}}_{\mathbf{Q}_i}(\mathbf{Q}_i(D)) \mid i = 1, \ldots, m\}$, where each set $\overline{\mathcal{S}}_{\mathbf{Q}_i}(\mathbf{Q}_i(D))$ is assigned a weight of p_i. Since $D \vdash \mathbf{Q}_{i_1}, \ldots, \mathbf{Q}_{i_\ell} \twoheadrightarrow \mathbf{Q}$ is equivalent to saying that the union of the conflict sets of $\mathbf{Q}_{i_1}, \ldots, \mathbf{Q}_{i_\ell}$ is a superset of the conflict set of $\mathbf{Q}$, the minimum set cover for $\overline{\mathcal{S}}_{\mathbf{Q}}(E)$ corresponds to the cheapest set of views that determine $\mathbf{Q}$ under database D.

3.2.1 Information Gain as a Pricing Function

A natural mechanism for pricing is to start from a probabilistic point of view and compute the price as the reduction in uncertainty, or *information gain*, using some notion of entropy.

Formally, consider an initial probability distribution over the set $\mathcal{I}$ of possible databases: in other words, assign a probability p_D to each database $D \in \mathcal{I}$. This probability distribution may reflect public information about the database (for example some value might be more probable than some other value). Let X be a random variable such that $P(X = D) = p_D$. Given some entropy measure $H(\cdot)$ of a random variable, such as Shannon entropy or min-entropy, we can set the price as the *information gain*: the initial entropy $H(X)$ minus the entropy of the new distribution, which is now conditioned on the event $\mathbf{Q}(X) = E$. Formally, we define the price as $p(\mathbf{Q}, E) = H(X) - H(X \mid \mathbf{Q}(X) = E)$. We can now plug standard uncertainty measures to obtain a pricing function. For example, we can use the Shannon entropy $H(X) = -\sum_{D\in\mathcal{I}} p_D \log(p_D)$, or the min-entropy $H_\infty(X) = -\log(\max_D p_D)$.

▶ **Lemma 14.** *There exists a probability distribution p_D over $\mathcal{I}$ such that the answer-dependent entropy function has information-arbitrage.*

Proof. Consider two sets $B \subseteq A \subseteq \mathcal{I}$, such that $A \setminus B = \{D_0\}$. Assume that the probabilities are set as follows: for every $D \in B$ we have $p_D = \epsilon$, and $p_{D_0} = 1 - m\epsilon$, where $m = |A|$. Define now two queries $\mathbf{Q}_A$ and $\mathbf{Q}_B$ such that $\mathcal{S}_{\mathbf{Q}_A}(E) = A$ and $\mathcal{S}_{\mathbf{Q}_B}(E) = B$. In this case, we have:

$$p(\mathbf{Q}_B, E) = H(D) + \sum_{i=1}^{m} \frac{1}{m} \log(1/m) = H(D) - \log(m)$$

$$p(\mathbf{Q}_A, E) = H(D) + m\epsilon \log(\epsilon) + (1 - m\epsilon) \log(1 - m\epsilon)$$

Further, $0 < m\epsilon < 1$. To create a counterexample, we choose $m\epsilon = \frac{1}{2}$, and now we have:

$$\begin{aligned} p(\mathbf{Q}_A, E) - p(\mathbf{Q}_B, E) = \\ &= m\epsilon \log(\epsilon) + (1 - m\epsilon)\log(1 - m\epsilon) + \log(m) \\ &= \frac{1}{2}\log(\epsilon) - \frac{1}{2} + \log(m) = \frac{1}{2}\log(m) - 1 \end{aligned}$$

By picking m large enough, we can make this quantity strictly positive, hence violating the information arbitrage condition. ◀

The intuition in the above proof is the following: the result for query $\mathbf{Q}_A$ will have a somewhat small entropy, because D_0 is much more probable than the other databases. However, by asking $\mathbf{Q}_B$ we learn that D_0 cannot be the actual database, and now the probability is equally distributed among the rest of the candidates; hence, the entropy grows!

The information gain, even though it seems a natural candidate, is not a well-behaved pricing function for APS, since it exhibits both information and bundle arbitrage (see Lemma 14 for such an example of information arbitrage). As we will see in Section 4 though, we can use information gain to construct arbitrage-free functions for QPS. In the case where the probabilities p_D are all equal, the information gain based on Shannon entropy has no information arbitrage (but can still exhibit bundle arbitrage) as shown in [8].

▶ **Lemma 15.** *If the probability distribution p_D over $\mathcal{I}$ is uniform, the information gain based on Shannon entropy has no information arbitrage.*

3.3 A Tradeoff for Arbitrage-Free APS

▶ **Example 16.** Continuing Example 4, consider the query $Q(x) = R(a, x)$ and the pricing function $p_2(\mathbf{Q}, E) = \log(|\overline{\mathcal{S}}_{\mathbf{Q}}(E)|)$. Notice that, independent of the actual database D, the conflict set has always size 2^{n-1}. In this case, $p_2(Q, E) = n - 1$. Notice that the price for learning the whole database is $\log(2^n - 1)$, which means that for learning a single tuple we pay almost as much as the whole database.

We will show here that the above example is not a random occurrence, and that the requirement that a pricing function has no bundle arbitrage gives rise to the phenomenon of assigning high prices (w.r.t. to the price of the whole dataset) to queries that reveal only a small amount of information.

▶ **Lemma 17.** *Let $p(\mathbf{Q}, E) = f(\overline{\mathcal{S}}_{\mathbf{Q}}(E))$ be an answer-dependent pricing function where f is monotone and subadditive over $2^{\mathcal{I}}$. Then, for every non-constant query $\mathbf{Q} \in B(\mathcal{L})$ there exists a database $D \in \mathcal{I}$ such that $p(\mathbf{Q}, \mathbf{Q}(D))$ is at least half the price of D.*

To see that the bundle-arbitrage requirement cause the problem, consider the function $p(\mathbf{Q}, E) = \log(|\mathcal{I}|) - \log(|\mathcal{S}_{\mathbf{Q}}(E)|)$, for which we showed that it exhibits no information arbitrage, but can still have bundle arbitrage. Continuing our example, we can see that $p(Q, E) = \log(2^n) - \log(2^{n-1}) = 1$; thus, learning about one of the n tuples is priced reasonably to $1/n$ of the price of the whole database. Our analysis demonstrates an important tradeoff in the design space of answer-dependent pricing functions: *ensuring no bundle arbitrage implies that the pricing function will charge disproportionately high prices for little information.*

It is also instructive to note that while Lemma 17 guarantees that existence of database $D \in \mathcal{I}$ that behaves badly, it does not say anything about the number of such databases. In fact, for our example we can show that for query Q at least half of the databases in $\mathcal{I}$ will exhibit this undesirable behavior.

4 Instance-Independent Pricing

We study here the structure of *instance-independent* pricing schemes. In a QPS, the pricing function is of the form $p(\mathbf{Q})$, depending only on the query. We first formalize the conditions under which the pricing function has no information arbitrage and no bundle arbitrage.

▶ **Definition 18.** We say that $\mathbf{Q}_2$ determines $\mathbf{Q}_1$, denoted $\mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$, if for every database database D' and D'', $\mathbf{Q}_2(D') = \mathbf{Q}_2(D'')$ implies $\mathbf{Q}_1(D') = \mathbf{Q}_1(D'')$.

In contrast to answer-dependent pricing functions, where we used a notion of determinacy that depends on the database, here we use the standard notion of *information-theoretic determinacy*.[3] We can now describe the formal definition for information arbitrage.

▶ **Definition 19** (QPS Information Arbitrage)**.** The pricing function p has no *information arbitrage* if for any two query bundles $\mathbf{Q}_1, \mathbf{Q}_2$ such that $\mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1$, we have $p(\mathbf{Q}_2) \geq p(\mathbf{Q}_1)$.

▶ **Definition 20** (QPS Bundle arbitrage)**.** Let the query bundle $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$. We say that the pricing function p has no *bundle arbitrage* if we have $p(\mathbf{Q}) \leq p(\mathbf{Q}_1) + p(\mathbf{Q}_2)$.

4.1 Serendipitous Arbitrage

Consider two query bundles $\mathbf{Q}_1$ and $\mathbf{Q}_2$ such that $\mathbf{Q}_1 \not\twoheadrightarrow \mathbf{Q}_2$, but for some $D \in \mathcal{I}$, $D \vdash \mathbf{Q}_1 \twoheadrightarrow \mathbf{Q}_2$. For example, consider the boolean query $Q_1() = R(x, y)$ over the binary relation $R(A, B)$. Let $Q_2(x, y) = R(x, y)$. Clearly, for all databases D other than the empty database, $D \vdash Q_1 \not\twoheadrightarrow Q_2$. However, for the database $D_0 = \emptyset$, note that $D_0 \vdash Q_1 \twoheadrightarrow Q_2$. In this case, if $p(Q_1) > p(Q_2)$, the data buyer would have an arbitrage opportunity. However, this opportunity would arise by chance, since the buyer does not know the underlying database and thus does not know that asking for $\mathbf{Q}_2$ can lead to learning $\mathbf{Q}_1$ for a lower price. We call this phenomenon *serendipitous arbitrage* [20]. Our definition of QPS information arbitrage does not capture serendipitous arbitrage. The next result demonstrates a second tradeoff in the design space of pricing functions: *any non-trivial QPS will exhibit serendipitous arbitrage.* We prove in [8]

▶ **Theorem 21.** *Let $\mathcal{L} = UCQ$. If a QPS exhibits no serendipitous arbitrage, then the price of any non-constant query bundle $\mathbf{Q}$ is equal to the price of asking for the whole database.*

4.2 How to Find a Pricing Function

To characterize the structure of instance-independent pricing functions, we exploit the fact that we can equivalently view a query as a *partition* of the set of possible databases $\mathcal{I}$.

4.2.1 The Partition Lattice

Fix some query language $\mathcal{L}$. Recall that for a query bundle $\mathbf{Q} \in B(\mathcal{L})$, $\mathcal{P}_{\mathbf{Q}}$ is the partition that is induced by the following *equivalence relation*: $D \sim D'$ iff $\mathbf{Q}(D) = \mathbf{Q}(D')$.

▶ **Lemma 22.** *Let $\mathbf{Q}_1, \mathbf{Q}_2 \in \mathcal{L}$ be two query bundles. The following are equivalent:*

1. $\mathbf{Q}_1 \twoheadrightarrow \mathbf{Q}_2$
2. $\mathcal{P}_{\mathbf{Q}_1} \succeq \mathcal{P}_{\mathbf{Q}_2}$, *i.e.* $\mathcal{P}_{\mathbf{Q}_1}$ *refines* $\mathcal{P}_{\mathbf{Q}_2}$

[3] Here we should note that there exists a slight difference, since the databases we consider can come only from $\mathcal{I}$, and not be any database.

The refinement relation defines a partial order on the set $\Pi_{\mathcal{I}}^{\mathcal{L}}$ of all partitions of $\mathcal{I}$ induced by any bundle $\mathbf{Q} \in B(\mathcal{L})$. An equivalent way to define the partial order is through the *distinction set* of a partition $dit(\mathcal{P}) = \bigcup_{B,B' \in \mathcal{P}: B \neq B'} B \times B'$. Intuitively, the distinction set contains all pairs of elements that are not in the equivalence relation. It is straightforward to see that $dit(\mathcal{P}_{\mathbf{Q}}) = \{(D', D'') \in \mathcal{I} \times \mathcal{I} \mid \mathbf{Q}(D') \neq \mathbf{Q}(D'')\}$. Furthermore, $\mathcal{P}_1 \succeq \mathcal{P}_2$ if and only if $dit(\mathcal{P}_1) \supseteq dit(\mathcal{P}_2)$ and thus one can use the inclusion of the distinction sets to define a partial order on the partitions.

The partial order induced by $\succeq$ on $\Pi_{\mathcal{I}}^{\mathcal{L}}$ forms a *join-semilattice*. The bottom element of the semilattice is the partition $\{\mathcal{I}\}$, which corresponds to a query that returns a constant answer. The top element is the partition where each block is a singleton set: this corresponds to a query that informs about the whole database. The *join* $\mathcal{P}_1 \vee \mathcal{P}_2$ is a new partition whose blocks are the non-empty intersections of any two blocks from $\mathcal{P}_1, \mathcal{P}_2$. The lemma below proves that the algebraic structure we defined is indeed a semilattice.

▶ **Lemma 23.** *Let $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$, where $\mathbf{Q}_1, \mathbf{Q}_2 \in B(\mathcal{L})$. Then, $\mathcal{P}_{\mathbf{Q}} = \mathcal{P}_{\mathbf{Q}_1} \vee \mathcal{P}_{\mathbf{Q}_2}$.*

If we define the partial order as the inclusion of distinction sets, $dit(\mathcal{P}_{\mathbf{Q}}) = dit(\mathcal{P}_{\mathbf{Q}_1} \vee \mathcal{P}_{\mathbf{Q}_2}) = dit(\mathcal{P}_{\mathbf{Q}_1}) \cup dit(\mathcal{P}_{\mathbf{Q}_2})$, the join operator is simply the union of the distinction sets.

4.2.2 A Characterization of Arbitrage-Free QPS

We now consider the family of instance-independent pricing functions of the form $p(\mathbf{Q}) = f(\mathcal{P}_{\mathbf{Q}})$, where $f : \Pi_{\mathcal{I}}^{\mathcal{L}} \to \mathbb{R}_+$ is a function that maps a partition to the positive real numbers.

▶ **Theorem 24.** *Let p be an instance-independent pricing function. Then, the two statements are equivalent:*
1. *p has no information arbitrage.*
2. *$p(\mathbf{Q}) = f(\mathcal{P}_{\mathbf{Q}})$, where f is a monotone function over $\Pi_{\mathcal{I}}^{\mathcal{L}}$.*

▶ **Theorem 25.** *Let $p(\mathbf{Q}) = f(\mathcal{P}_{\mathbf{Q}})$ be an instance-independent pricing function, where f is a function over $\Pi_{\mathcal{I}}^{\mathcal{L}}$. Then, the two statements are equivalent:*
1. *p has no bundle arbitrage.*
2. *f is subadditive over $\Pi_{\mathcal{I}}^{\mathcal{L}}$.*

▶ **Corollary 26.** *Let f be a monotone and subadditive function over $\Pi_{\mathcal{I}}^{\mathcal{L}}$. Then, $p(\mathbf{Q}) = f(\mathcal{P}_{\mathbf{Q}})$ is an instance-independent pricing function that has no bundle or information arbitrage.*

Alternatively, we could also define the pricing function as $p(\mathbf{Q}) = f(dit(\mathcal{P}_{\mathbf{Q}}))$. Using the same type of arguments, we can show:

▶ **Corollary 27.** *Let f be a monotone and subadditive set function. Then, $p(\mathbf{Q}) = f(dit(\mathcal{P}_{\mathbf{Q}}))$ is an instance-independent pricing function that has no bundle or information arbitrage.*

4.3 Construction of Pricing Functions From Answer-Dependent Prices

We show first how we can design an instance-independent pricing function $p(\mathbf{Q})$ starting from an answer-dependent function $p(\mathbf{Q}, E)$. Given a query bundle $\mathbf{Q}$, the idea is to construct a vector of all prices $p(\mathbf{Q}, \mathbf{Q}(D))$ for all databases $D \in \mathcal{I}$. Formally, we define the *price vector* $\vec{p}(\mathbf{Q}) = \langle p(\mathbf{Q}, \mathbf{Q}(D)) \mid D \in \mathcal{I} \rangle$. Then we can obtain an instance-independent pricing function by computing another function $g : \mathbb{R}_+^{|\mathcal{I}|} \to \mathbb{R}_+$ over the above vector, such that $p(\mathbf{Q}) = g(\vec{p}(\mathbf{Q}))$. The next lemma describes the conditions for g under which the arbitrage-free property carries over.

▶ **Lemma 28.** *Let $p(\mathbf{Q}, E)$ be an arbitrage-free pricing function. If g is a monotone and subadditive function, then $p(\mathbf{Q}) = g(\vec{p}(\mathbf{Q}))$ is an arbitrage-free instance-independent function.*

We next present an application of Lemma 28 to obtain arbitrage-free pricing functions.

▶ **Lemma 29.** *Let f be a monotone and subadditive set function. Let w_D be a non-negative weight w_D to each $D \in \mathcal{I}$, and denote $w_B = \sum_{D \in B} w_D$. Then, the pricing functions $p_1(\mathbf{Q}) = \max_{B \in \mathcal{P}_\mathbf{Q}}\{f(\mathcal{I} \setminus B)\}$ and $p_2(\mathbf{Q}) = \sum_{B \in \mathcal{P}_\mathbf{Q}} w_B \cdot f(\mathcal{I} \setminus B)$ are arbitrage-free.*

▶ **Example 30.** Consider the function p_2 with equal weights $w_D = 1$ and the set function $f(A) = |A|$. The resulting arbitrage-free function is $p(\mathbf{Q}) = \sum_{B \in \mathcal{P}_\mathbf{Q}} |B|(|\mathcal{I}| - |B|) = |dit(\mathcal{P}_\mathbf{Q})|$, which sets the price to be the size of the distinction set.

If $\sum_D w_D = 1$ for p_2, one can interpret the weights as a probability distribution over the set of databases $\mathcal{I}$. In this case, we can write $p_2(\mathbf{Q}) = \mathbb{E}_{B \in \mathcal{P}_\mathbf{Q}}[f(\mathcal{I} \setminus B)]$, where each block B has probability w_B. In other words, the pricing function is the expected price over all answer-dependent prices. The converse of Lemma 28 does not hold: it is possible for $p(\mathbf{Q})$ to be arbitrage-free, and for some database D it may not be the case. As we will see next, this allows us to construct arbitrage-free functions that are based on measures of uncertainty.

4.4 Construction of Pricing Functions From Uncertainty Measures

In this section, we describe arbitrage-free pricing functions that do not originate from answer-dependent functions. To construct such functions, we switch to a probabilistic view of the problem and then apply information-theoretic tools that are used to measure uncertainty. For the remainder of this section, we assume that each database D is associated with a probability p_D. We denote by X the random variable such that $P(X = D) = p_D$ and let $p_E = \sum_{D:\mathbf{Q}(D)=E} p_D$. The detailed proofs in this section are presented in [8].

Shannon Entropy. The first measure of uncertainty we apply is the most commonly used form of entropy, and was proposed in [20] as a pricing function. In the answer-dependent context, we defined the price as the information gain after the output E has been revealed. Since in this setting the price is independent of the output, we define the price as the *expected information gain* over all possible outcomes. Formally:

$$p^H(\mathbf{Q}) = H(X) - \sum_E p_E \cdot H(X \mid \mathbf{Q}(X) = E) \tag{1}$$

Equivalently, we can also express the price as

$$p^H(\mathbf{Q}) = H(X) - H(X \mid \mathbf{Q}(X)) = I(X; \mathbf{Q}(X)) = H(\mathbf{Q}(X)) - H(\mathbf{Q}(X) \mid X) = H(\mathbf{Q}(X))$$

where $I(X;Y)$ is the *mutual information* between the random variables X and Y. [20] proves that p^H is both bundle and information arbitrage-free, using the subadditivity of entropy and the data-processing inequality respectively. It is instructing to write p^H as $p^H(\mathbf{Q}) = -\sum_{S \in \mathcal{P}_\mathbf{Q}} p_S \cdot \log p_S = \sum_D p_D \cdot p(\mathbf{Q}, \mathbf{Q}(D))$ where $p(\mathbf{Q}, E) = -\log(p_E)$ is now an answer-dependent pricing function. Notice that $p(\mathbf{Q}, E)$ has no information arbitrage, and thus by applying Lemma 28 we get an alternative proof that p^H is information arbitrage-free. However, $p(\mathbf{Q}, E)$ can have bundle arbitrage, and thus we cannot apply Lemma 28 to show the subadditivity property as well: entropy is subadditive only in expectation. This example demonstrates that the converse of Lemma 28 does not hold.

Tsallis Entropy. For a real number $q > 1$, the *Tsallis entropy* [27], or *q-entropy*, of a random variable X is defined as $S_q(X) = \frac{1}{q-1} \cdot \left(1 - \sum_x P(X = x)^{q-1}\right)$. Tsallis entropy is a generalization of Shannon entropy, since $\lim_{q \to 1} S_q(X) = H(X)$. We define the price as the Tsallis entropy of $\mathbf{Q}(X)$:

$$p^T(\mathbf{Q}) = S_q(\mathbf{Q}(X)) = \sum_{S \in \mathcal{P}_\mathbf{Q}} \frac{p_S}{q-1} \cdot (1 - p_S^{q-1}) \tag{2}$$

▶ **Lemma 31.** *The pricing function p^T defined in Equation (2) is arbitrage-free for $q > 1$.*

Guessing Entropy. The guessing entropy measures the average number of successive guesses required by an optimum strategy until we correctly guess the value of the random variable X (in our case the underlying database D). The guessing entropy was first introduced in [21], and subsequently used in [13] in the context of measuring leakage in side-channel attacks. To compute the guessing entropy of X, suppose that we have ordered the databases in decreasing order of their probabilities, i.e. such that $p(X = D_i) \geq p(X = D_j)$ whenever $i \leq j$. Then, we define the *guessing entropy* as $G(X) = \sum_i i \cdot p_{D_i}$. The price is now defined as the initial entropy minus the expected conditional guessing entropy $G(X \mid \mathbf{Q}(X) = E)$:

$$p^G(\mathbf{Q}) = G(X) - \sum_E p_E \cdot G(X \mid \mathbf{Q}(X) = E) \tag{3}$$

▶ **Lemma 32.** *The pricing function p^G defined in Equation (3) is arbitrage-free.*

Min-Entropy. We apply here the notion of min-entropy, as it was introduced in [26] to quantify information flow. The *min-entropy* of a random variable is $H_\infty(X) = -\log(\max_x P(X = x))$. The conditional min-entropy is defined as $H_\infty(X \mid Y) = -\log(\sum_y P(Y = y) \cdot \max_x P(X = x \mid Y = y))$. Then, we can construct the price of a query as follows:

$$p^M(\mathbf{Q}) = H_\infty(X) - H_\infty(X \mid \mathbf{Q}(X)) = -\log(\max_D p_D) + \log\left(\sum_E \max_{D:\mathbf{Q}(D)=E} p_D\right) \tag{4}$$

▶ **Lemma 33.** *The pricing function p^M defined in Equation (4) has no information arbitrage.*

The min-entropy is not in general bundle arbitrage-free, as we show in the full version of the paper [8]. However, it becomes so when the initial distribution is uniform. Let $n = |\mathcal{I}|$, in which case $p_D = 1/n$ for each database. Then, it is straightforward to see that the resulting function is the logarithm of the number of sets in the partition $\mathcal{P}_\mathbf{Q}$.

$$p^{MU}(\mathbf{Q}) = \log(n) + \log(|\mathcal{P}_\mathbf{Q}|/n) = \log(|\mathcal{P}_\mathbf{Q}|) \tag{5}$$

▶ **Lemma 34.** *The pricing function $p^{MU}(\mathbf{Q})$ defined in Equation (5) is arbitrage-free.*

β-Success Rate. This information measure, first introduced in [4], captures the expected success of guessing the database D with β tries. We will consider here only the case where the probability distribution is uniform, in which case the pricing functions becomes:

$$p^\beta(\mathbf{Q}) = \log\left(\sum_{S \in \mathcal{P}_\mathbf{Q}} \min\{\beta, |S|\}\right) \tag{6}$$

Observe that for $\beta = 1$ we have $p^\beta(\mathbf{Q}) = p^{MU}(\mathbf{Q})$, hence this generalizes uniform min-entropy.

Table 1 The price of a query bundle $\mathbf{Q}$ according to various entropy measures for the case of uniform probability distributions. We denote $n = |\mathcal{I}|$.

Shannon Entropy	$p^H(\mathbf{Q}) = \frac{1}{n}\sum_{B\in\mathcal{P}_\mathbf{Q}} \lvert B\rvert \log \lvert B\rvert$
Guessing Entropy	$p^G(\mathbf{Q}) = \frac{1}{2n}\left(n^2 - \sum_{B\in\mathcal{P}_\mathbf{Q}} \lvert B\rvert^2\right)$
Min-Entropy	$p^{MU}(\mathbf{Q}) = \log \lvert\mathcal{P}_Q\rvert$
Tsallis Entropy	$p^T(\mathbf{Q}) = \frac{1}{q-1}\left(1 - \sum_{B\in\mathcal{P}_\mathbf{Q}} (\frac{\lvert B\rvert}{n})^{q-1}\right)$
β-Success Rate	$p^\beta(\mathbf{Q}) = \log\left(\sum_{B\in\mathcal{P}_\mathbf{Q}} \min\{\beta, \lvert B\rvert\}\right)$

▶ **Lemma 35.** *The pricing function $p^\beta(\mathbf{Q})$ defined in Equation (6) is arbitrage-free.*

We should finally mention that several other entropy measures have been discussed in the broader literature. The Renyi entropy [25] is a generalization of both the Shannon entropy and the min-entropy. However, it is not subadditive, and thus not applicable as an arbitrage-free pricing function. Worst-case entropy measures [13] can also be applied to measure information leakage, but they are also prone to bundle arbitrage.

5 Computing the Pricing Function

So far we have studied how to construct pricing functions for both APS and QPS. In this section, we focus on the complexity of computing a pricing function.

5.1 Support Sets

We first start by discussing an generic approach that can construct efficiently computable arbitrage-free pricing functions for any query language $\mathcal{L}$ that can be computed efficiently. The key idea behind our construction is to define the pricing function on a smaller set $\mathcal{C} \subseteq \mathcal{I}$ of our choice, which we call *support*. The next two lemmas show that this restriction still provides arbitrage-free answer-dependent and instance-independent pricing functions.

▶ **Lemma 36.** *Let $\mathcal{C} \subseteq \mathcal{I}$. If f is a monotone and subadditive set function, the pricing function $p(\mathbf{Q}, E) = f(\overline{\mathcal{S}}_\mathbf{Q}(E) \cap \mathcal{C})$ is arbitrage-free.*

Given a partition $\mathcal{P}$ of the set $\mathcal{I}$, we define the *restriction of $\mathcal{P}$ to $\mathcal{C}$*, denoted $\mathcal{P} \cap \mathcal{C}$, as the set $\{B \cap \mathcal{C} \mid B \in \mathcal{P}, B \cap \mathcal{C} \neq \emptyset\}$.

▶ **Lemma 37.** *Let $\mathcal{C} \subseteq \mathcal{I}$. If f is a monotone and subadditive function on the partition semilattice, the pricing function $p(\mathbf{Q}) = f(\mathcal{P}_\mathbf{Q} \cap \mathcal{C})$ is arbitrage-free.*

The above results provide us with a method to design an efficient arbitrage-free pricing function for a query language $\mathcal{L}$. We start by choosing a support $\mathcal{C} \subseteq \mathcal{I}$. To compute the pricing function, we first compute $\overline{\mathcal{S}}_\mathbf{Q}(E) \cap \mathcal{C}$ for answer-dependent (or $\mathcal{P}_\mathbf{Q} \cap \mathcal{C}$ for instance-independent). The observation is that we can achieve this by evaluating the query bundle $\mathbf{Q}$ only on the databases $D \in \mathcal{C}$. Hence, the running time of computing the price does not depend on $|\mathcal{I}|$, but on $|\mathcal{C}|$ and the complexity of evaluating the query bundle $\mathbf{Q}$.

▶ **Example 38.** Consider any set $\mathcal{C} \subseteq \mathcal{I}$. Then $p(\mathbf{Q}, E) = \log |\{D \in \mathcal{C} \mid \mathbf{Q}(D) \neq E\}|$ is an arbitrage-free pricing function. Similarly, $p(\mathbf{Q}) = \log |\mathcal{P}_\mathbf{Q} \cap \mathcal{C}|$ is also arbitrage-free.

The advantage of using support sets to construct pricing functions is that they provide us with a generic method that is independent of the language $\mathcal{L}$. On the other hand, the size and choice of the support $\mathcal{C}$ is a challenging problem. We can always choose $\mathcal{C}$ to contain a single database. The evaluation of the price will be very efficient, but any query will be assigned only one of two prices, and thus the pricing function will not be very successful in measuring the value of the data. If we instead choose a very large support, this leads to expensive and impractical price computation. We leave as an open research question how to choose a good support $\mathcal{C}$ that is suitable for a practical implementation.

5.2 The Complexity of Entropy-Based Pricing

In a practical setting, the set $\mathcal{I}$ will be given implicitly. For example, $\mathcal{I}$ can be the infinite set of all databases, or the set of all subsets of a given database D_0, $\mathcal{I} = \{D \mid D \subseteq D_0\}$. One might think that since the problem of determinacy (either query or data-dependent) is hard even for the class of conjunctive queries, computing an arbitrage-free pricing function is always hard. However, as we showed in the previous section about support sets, it is always possible to construct non-trivial pricing schemes that circumvent the computation of determinacy and thus can be computed efficiently. Here we will focus on the computational complexity for the pricing functions we introduced that are based on entropy.

The task necessary to compute an answer-dependent pricing function such as $p(\mathbf{Q}, E) = \log(|\overline{\mathcal{S}}_{\mathbf{Q}}(E)|)$, or any of the instance-independent functions in Table 1 is the following: *given a view extension E and $\mathbf{Q}$, compute $|\mathcal{S}_{\mathbf{Q}}(E)|$, which is the number of databases in $\mathcal{I}$ such that $\mathbf{Q}(D) = E$.* If $\mathcal{I}$ can be succinctly expressed as $\mathcal{I} = \{D \mid D \subseteq D_0\}$, the task relates to the area of probabilistic databases. Indeed, we can view D_0 as a tuple-independent probabilistic database where each tuple has the same probability 1/2. Then, we can write $|\mathcal{S}_{\mathbf{Q}}(E)| = P(\mathbf{Q}(D_0) = E) \cdot |\mathcal{I}|$. Unfortunately, computing the probability $P(\mathbf{Q}(D_0) = E)$ is in general a $^{\#}P$-hard problem (w.r.t. the size of D_0), even for the class of conjunctive queries [7]. However, the task is known to be in polynomial time for certain types of queries. For instance, in Example 4, where Q is a selection query over a single table, the size of the conflict set can be computed exactly in polynomial time. We should note here that the problem of checking whether $\mathcal{S}_{\mathbf{Q}}(E)$ is empty or not is equivalent to the problem of *view consistency*, which is shown to be NP-hard for the class of conjunctive queries [1] when $\mathcal{I}$ ranges over all databases.

Even if $|\mathcal{S}_{\mathbf{Q}}(E)|$ can be computed exactly, the number of blocks in the partition $\mathcal{P}_{\mathbf{Q}}$ may still be exponentially large, which would make computing the Shannon or Guessing entropy intractable. In this case, we can write the information gain as $p(\mathbf{Q}) = -\sum_{D \in \mathcal{I}} \log |[D]_{\mathbf{Q}}|$, and construct an estimator of the price that samples independently m databases from $\mathcal{I}$ and outputs their average: $\tilde{p}(\mathbf{Q}) = \frac{1}{m} \sum_{i=1}^{m} \log |[D_i]_{\mathbf{Q}}|$. In [14, 3], the authors show that such an estimator can achieve an additive δ-approximation of the price with a number of samples that is polynomial in $1/\delta, \log(|\mathcal{I}|)$. We say that a pricing function is *ε-approximately arbitrage-free* if the arbitrage conditions are violated within an additive ε. It is straightforward to see that $\tilde{p}$ results in a (3δ)-approximately arbitrage-free pricing scheme. This implies that we can compute in polynomial time an approximation of the entropy function that is as close to arbitrage-free as we would like to.

6 Related Work

The problem of data pricing has been studied from a wide range of perspectives, including online markets and privacy [10, 22]. [12] examined a variety of issues involved in pricing of information products and presented an economic approach to design of optimal pricing mechanism for online services. [2] introduced the challenge of developing pricing functions in the context of cloud-based environments, where users can pay for queries without buying the entire dataset. This work also outlines various research challenges, such as enabling fine-grained pricing and developing efficient and fair pricing models for cloud-based markets.

The first formal framework for query-based data pricing was introduced by Koutris et al. [15]. The authors define the notion of arbitrage, and provide a framework that takes a set of fixed prices for views over the data identified by seller, and extends these price points to a pricing function over any query. The authors also show that evaluation of the prices can be done efficiently in polynomial time for specific classes of conjunctive queries and a restricted set of views that include only selections. Subsequently, the authors demonstrated how the framework can be implemented into a prototype pricing system called QueryMarket [16, 17]. Further work [18] discusses the pricing and complexity of pricing for the class of aggregate queries. The work by Lin and Kifer [20] proposes several possible forms of arbitrage violations and integrates them into a single framework. The authors allow the queries to be randomized, and propose two potential pricing functions that are arbitrage-free across all forms.

Data pricing is tightly connected to differential privacy [9]. Ghosh and Roth [11] study the buying and selling of data by considering privacy as an entity. Their framework compensates the seller for the loss of privacy due to selling of private data. A similar approach to pricing in the context of privacy is discussed in [19].

We should finally mention the close connection of query pricing to the measurement of information leakage in programs. In [13], the authors apply information-theoretic measures, including various entropy measures, to compute the leakage of information from a side-channel attack that attempts to gain access to secret information. [14] uses similar ideas to quantify the flow of information in programs, and proposes various approximation techniques to efficiently compute them.

7 Conclusion

In this paper, we explore in depth the design space of arbitrage-free pricing functions. We present a characterization of the structure for both answer-dependent and instance-independent pricing functions, and propose several constructions. Our work opens several exciting research questions, including testing which pricing functions behave well in practical settings, and exploring the various tradeoffs when deploying a pricing scheme.

Acknowledgements. We would like to thank Aws Albarghouthi for pointing out the close connection of our work to quantitative information flow and information leakage in side-channel attacks.

References

1 Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263. ACM Press, 1998. doi:10.1145/275487.275516, db/conf/pods/AbiteboulD98.html.

2 Magdalena Balazinska, Bill Howe, and Dan Suciu. Data markets in the cloud: An opportunity for the database community. *PVLDB*, 4(12), 2011.

3 Tugkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating the entropy. *SIAM J. Comput.*, 35(1):132–150, 2005. doi:10.1137/S0097539702403645.

4 S. Boztas. Entropies, guessing and cryptography. Technical Report 6, Department of Mathematics, Royal Melbourne Institute of Technology, 1999.

5 Federal Trade Commission et al. Data brokers: A call for transparency and accountability. *Policy Reports*, Commission and Staff Reports, May 2014.

6 Kenneth Cukier and Viktor Mayer-Schoenberger. Rise of big data: How it's changing the way we think about the world, the. *Foreign Aff.*, 92:28, 2013.

7 Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009. doi:10.1145/1538788.1538810.

8 Shaleen Deep and Paraschos Koutris. The design of arbitrage-free data pricing schemes. *arXiv preprint arXiv:1606.09376*, 2016.

9 Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, 2011. doi:10.1145/1866739.1866758.

10 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

11 Arpita Ghosh and Aaron Roth. Selling privacy at auction. *Games and Economic Behavior*, 2013.

12 Sanjay Jain and P. K. Kannan. Pricing of information products on online servers: Issues, models, and analysis. *Management Science*, 48(9):1123–1142, 2002.

13 Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *CCS*, pages 286–296. ACM, 2007.

14 Boris Köpf and Andrey Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *CSF, 2010*, pages 3–14. IEEE, July 2010.

15 Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Query-based data pricing. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *PODS*, pages 167–178. ACM, 2012. doi:10.1145/2213556.2213582.

16 Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Querymarket demonstration: Pricing for online data markets. *PVLDB*, 5(12):1962–1965, 2012.

17 Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Toward practical query pricing with querymarket. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *ACMSIGMOD 2013*, pages 613–624. ACM, 2013. URL: http://dl.acm.org/citation.cfm?id=2463676, doi:10.1145/2463676.2465335.

18 C. Li and G. Miklau. Pricing aggregate queries in a data marketplace. In *WebDB*, 2012.

19 Chao Li, Daniel Yang Li, Gerome Miklau, and Dan Suciu. A theory of pricing private data. *ACM Trans. Database Syst.*, 39(4):34:1–34:28, 2014. doi:10.1145/2691190.2691191.

20 Bing-Rong Lin and Daniel Kifer. On arbitrage-free pricing for general data queries. *PVLDB*, 7(9):757–768, 2014. URL: http://www.vldb.org/pvldb/vol7/p757-lin.pdf.

21 J.L. Massey. Guessing and entropy. In *Information Theory, 1994*, page 204, Jun 1994. doi:10.1109/ISIT.1994.394764.

22 Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *ACM SIGMOD 2009*, pages 19–30. ACM, 2009. doi:10.1145/1559845.1559850.

23 Alan Nash, Luc Segoufin, and Victor Vianu. Determinacy and rewriting of conjunctive queries using views: A progress report. In *ICDT*, pages 59–73, 2007. doi:10.1007/11965893_5.

24 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), 2010. doi:10.1145/1806907.1806913.

25 A. Renyi. On measures of information and entropy. In *Berkeley Symposium on Mathematics, Statistics and Probability*, pages 547–561, 1960. URL: http://digitalassets.lib.berkeley.edu/math/ucb/text/math_s4_v1_article-27.pdf.

26 Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *FOSSACS 2009*, LNCS, pages 288–302. Springer, 2009. doi:10.1007/978-3-642-00596-1_21.

27 Constantino Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, 52(1-2):479–487, 1988. doi:10.1007/BF01016429.

A Logic for Document Spanners*

Dominik D. Freydenberger

University of Bayreuth, Bayreuth, Germany

Abstract

Document spanners are a formal framework for information extraction that was introduced by Fagin, Kimelfeld, Reiss, and Vansummeren (PODS 2013, JACM 2015). One of the central models in this framework are core spanners, which are based on regular expressions with variables that are then extended with an algebra. As shown by Freydenberger and Holldack (ICDT 2016), there is a connection between core spanners and $\mathsf{EC}^{\mathsf{reg}}$, the existential theory of concatenation with regular constraints. The present paper further develops this connection by defining SpLog, a fragment of $\mathsf{EC}^{\mathsf{reg}}$ that has the same expressive power as core spanners. This equivalence extends beyond equivalence of expressive power, as we show the existence of polynomial time conversions between this fragment and core spanners. This even holds for variants of core spanners that are based on automata instead of regular expressions. Applications of this approach include an alternative way of defining relations for spanners, insights into the relative succinctness of various classes of spanner representations, and a pumping lemma for core spanners.

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, F.4.m Miscellaneous

Keywords and phrases Information extraction, document spanners, word equations, regex, descriptional complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.13

1 Introduction

Fagin, Kimelfeld, Reiss, and Vansummeren [11] introduced *document spanners* as a formal framework for information extraction. Document spanners formalize the query language AQL that is used in IBM's SystemT. On an intuitive level, document spanners can be understood as a generalized form of searching in a text w: In its basic form, search can be understood as taking a search term u (or a regular expression α) and a word w, and computing all intervals of positions of w that contain u (or a word from $\mathcal{L}(\alpha)$). These intervals are called *spans*. Spanners generalize searching by computing *relations over spans* of w.

In order to define spanners, [11] introduced *regex formulas*, which are regular expressions with variables. Each variable x is connected to a subexpression α, and when α matches a subword of w, the corresponding span is stored in x (this behaves like the capture groups that are often used in real world implementation of search-and-replace functionality). *Core spanners* combine these regex formulas with the algebraic operators projection, union, join (on spans), and string equality selection.

Freydenberger and Holldack [12] connected core spanners to $\mathsf{EC}^{\mathsf{reg}}$, the existential theory of concatenation with regular constraints. Described very informally, $\mathsf{EC}^{\mathsf{reg}}$ is a logic that combines equations on words (like $x\mathtt{ab}y = y\mathtt{ba}x$) with positive logical connectives, and regular languages that constrain variable replacement. In particular, [12] showed that every

* Supported by Deutsche Forschungsgemeinschaft (DFG) under grant FR 3551/1-1.

core spanner can be converted into an $\mathsf{EC}^{\mathsf{reg}}$-formula, which can then be used to decide satisfiability. Furthermore, while every $\mathsf{EC}^{\mathsf{reg}}$-formula can be converted into an equisatisfiable core spanner, the resulting spanner cannot be used to evaluate the formula (as, due to details of the encoding, the input word w of the spanner needs to encode the formula).

This paper further explores the connection of core spanners and $\mathsf{EC}^{\mathsf{reg}}$. As main conceptual contribution, we introduce SpLog (short for *spanner logic*), a natural fragment of $\mathsf{EC}^{\mathsf{reg}}$ that has the same expressive power as core spanners. In contrast to the PSPACE-complete combined complexity of $\mathsf{EC}^{\mathsf{reg}}$-evaluation, the combined complexity of SpLog-evaluation is NP-complete, and its data complexity is in NL. As main technical result, we demonstrate the existence of polynomial time conversions between SpLog and spanner representations (in both directions), even if the spanners are defined with automata instead of regex formulas.

As a consequence, SpLog can augment (or even replace) the use of regex formulas or automata in the definition of core spanners. Moreover, this shows that the PSPACE upper bounds from [12] for deciding satisfiability and hierarchicality of regex formula based spanners apply to automata based spanners as well. In addition to this, we adapt a pumping lemma for word equations to SpLog (and, hence, to core spanners). The main result also provides insights into the relative succinctness of classes of automata based spanners: While there are exponential trade-offs between various classes of automata, these differences disappear when adding the algebraic operators.

From a more general point of view, this paper can also be seen as an attempt to connect spanners to the research on equations on words and on groups (cf. Diekert [7, 6] for surveys), where $\mathsf{EC}^{\mathsf{reg}}$ has been studied as a natural extension of word equations. We shall see that SpLog is a natural fragment of $\mathsf{EC}^{\mathsf{reg}}$: On an informal level, SpLog has to express relations on a word w without using additional working space (which explains the friendlier complexity of evaluation, in comparison to $\mathsf{EC}^{\mathsf{reg}}$). This gives us reason to expect that SpLog can be applied to other models, like graph databases (as a related example of an application of $\mathsf{EC}^{\mathsf{reg}}$ for graph databases, Barceló and Muñoz [1] use a restricted class of $\mathsf{EC}^{\mathsf{reg}}$-formulas for which data complexity is also in NL).

This paper is structured as follows: Section 2 gives the definitions of spanners and of $\mathsf{EC}^{\mathsf{reg}}$, as well as a few preliminary results. Section 3 introduces SpLog and connects it to spanners. We then examine properties of SpLog: Section 4 discusses how SpLog can be used to express relations, while Section 5 adapts an EC-inexpressibility result to SpLog. Section 6 concludes the paper. Most of the proofs can be found only in the full version of the paper[1].

2 Preliminaries

Let Σ be a fixed finite alphabet of *(terminal) symbols*. Except when stated otherwise, we assume $|\Sigma| \geq 2$. Let Ξ be an infinite alphabet of *variables* that is disjoint from Σ. We use ε to denote the *empty word*. For every word w and every letter a, let $|w|$ denote the length of w, and $|w|_a$ the number of occurrences of a in w. A word x is a *subword* of a word y if there exist words u, v with $y = uxv$. We denote this by $x \sqsubseteq y$; and we write $x \not\sqsubseteq y$ if $x \sqsubseteq y$ does not hold. For words x, y, z with $x = yz$, we say that y is a *prefix* of x, and z is a *suffix* of x. A prefix or suffix y of x is *proper* if $x \neq y$. For every $k \geq 0$, a *k-ary word relation (over Σ)* is a subset of $(\Sigma^*)^k$. Given a nondeterministic finite automaton (NFA) A (or a regular expression α), we use $\mathcal{L}(A)$ (or $\mathcal{L}(\alpha)$) to denote its language. In NFAs, we allow the use of ε-transitions (this model is also called ε-NFA in literature).

[1] `http://ddfy.de/sci/splog.pdf`

The remainder of this section contains the two models that this paper connects: Document spanners in Section 2.1, and $\mathsf{EC}^{\mathsf{reg}}$ in Section 2.2.

2.1 Document Spanners

2.1.1 Primitive Spanner Representations

Let $w := a_1 a_2 \cdots a_n$ be a word over Σ, with $n \geq 0$ 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 $i, j \geq 0$. For each span $[i, j\rangle$ of w, we define $w_{[i,j\rangle} := a_i \cdots a_{j-1}$. That is, each span describes a subword of w by its bounding indices.

▶ **Example 1.** 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 $[1, 10\rangle$.

Let $V \subset \Xi$ be finite, and let $w \in \Sigma^*$. A (V, w)-*tuple* is a function μ 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*. A *spanner* is a function P that maps every $w \in \Sigma^*$ to a (V, w)-relation $P(w)$. Let V be denoted by $\mathsf{SVars}(P)$. Two spanners P_1 and P_2 are equivalent if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$, and $P_1(w) = P_2(w)$ 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. We now examine a formalism that can be used to define spanners:

▶ **Definition 2.** A *regex formula* is an extension of regular expressions to include variables. The syntax is specified with the recursive rules $\alpha := \emptyset \mid \varepsilon \mid a \mid (\alpha \vee \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha)^* \mid x\{\alpha\}$ for $a \in \Sigma$, $x \in \Xi$. We add and omit parentheses freely, as long as the meaning remains clear, and use α^+ as shorthand for $\alpha \cdot \alpha^*$, and Σ as shorthand for $\bigvee_{a \in \Sigma} a$.

Regex formulas can be interpreted as special case of so-called *regex*, which extend classical regular expressions with a repetition operator (see Section 4.3 for a brief and [12] for a more detailed discussion). This applies to syntax and semantics. In particular, both models define their syntax with parse trees, which is rather impractical for many of our proofs. Instead of using this definition, we present one that is based on *ref-words* (short for *reference words*) by Schmid [23]. A ref-word is a word over the extended alphabet $(\Sigma \cup \Gamma)$, where $\Gamma := \{\vdash_x, \dashv_x \mid x \in \Xi\}$. Intuitively, the symbols $\vdash_x$ and $\dashv_x$ mark the beginning and the end of the span that belongs to the variable x. In order to define the semantics of regex formulas, we treat them as generators of ref-languages (i. e., languages of ref-words):

▶ **Definition 3.** For every regex formula α, we define its *ref-language* $\mathcal{R}(\alpha)$ by $\mathcal{R}(\emptyset) := \emptyset$, $\mathcal{R}(a) := \{a\}$ for $a \in \Sigma \cup \{\varepsilon\}$, $\mathcal{R}(\alpha_1 \vee \alpha_2) := \mathcal{R}(\alpha_1) \cup \mathcal{R}(\alpha_2)$, $\mathcal{R}(\alpha_1 \cdot \alpha_2) := \mathcal{R}(\alpha_1) \cdot \mathcal{R}(\alpha_2)$, $\mathcal{R}(\alpha_1^*) := \mathcal{R}(\alpha_1)^*$, and $\mathcal{R}(x\{\alpha_1\}) := \vdash_x \mathcal{R}(\alpha_1) \dashv_x$.

Let $\mathsf{SVars}(\alpha)$ be the set of all $x \in \Xi$ such that $x\{\ \}$ occurs in α. A ref-word $r \in \mathcal{R}(\alpha)$ is *valid* if, for every $x \in \mathsf{SVars}(\alpha)$, $|r|_{\vdash_x} = 1$. Let $\mathsf{Ref}(\alpha) := \{r \in \mathcal{R}(\alpha) \mid r \text{ is valid}\}$. We call α *functional* if $\mathsf{Ref}(\alpha) = \mathcal{R}(\alpha)$, and denote the set of all functional regex formulas by RGX.

In other words, $\mathcal{R}(\alpha)$ treats α like a standard regular expression over the alphabet $(\Sigma \cup \Gamma)$, where $x\{\alpha_1\}$ is interpreted as $\vdash_x \alpha_1 \dashv_x$. Furthermore, $\mathsf{Ref}(\alpha)$ contains exactly those words where each variable x is opened and closed exactly once.

▶ **Example 4.** Define regex formulas $\alpha := (x\{\mathtt{a}\}y\{\mathtt{b}\}) \vee (y\{\mathtt{a}\}x\{\mathtt{b}\})$, $\beta := x\{\mathtt{a}\} \vee y\{\mathtt{a}\}$, and $\gamma := x\{a\}x\{a\}$. Then α is a functional, while β and γ are not (in fact, $\mathsf{Ref}(\alpha) = \mathsf{Ref}(\beta) = \emptyset$).

Like [11, 12], this paper only examines functional regex formulas. Hence, without loss of generality, we assume that no variable binding $x\{\ \}$ occurs under a Kleene star $*$.

The definition of $\mathcal{R}(\alpha)$ implies that every $r \in \mathsf{Ref}(\alpha)$ has a unique factorization $r = r_1 \vdash_x r_2 \dashv_x r_3$ for every $x \in \mathsf{SVars}(\alpha)$. This can be used to define $\mu(x)$ (i. e., the span that is assigned to x). To this purpose, we define a morphism $\mathsf{clr}\colon (\Sigma \cup \Gamma)^* \to \Sigma^*$ by $\mathsf{clr}(a) := a$ for all $a \in \Sigma$, and $\mathsf{clr}(g) := \varepsilon$ for all $g \in \Gamma$ (in other words, clr projects ref-words to Σ). Then $\mathsf{clr}(r_1)$ contains the part of w that precedes $\mu(x)$, and $\mathsf{clr}(r_2)$ contains $w_{\mu(x)}$.

For $\alpha \in \mathsf{RGX}$ and $w \in \Sigma^*$, let $\mathsf{Ref}(\alpha, w) := \{r \in \mathsf{Ref}(\alpha) \mid \mathsf{clr}(r) = w\}$. Then every word of $\mathsf{Ref}(\alpha, w)$ encodes one possibility of assigning variables in w that is consistent with α.

▶ **Definition 5.** Let $\alpha \in \mathsf{RGX}$, $w \in \Sigma^*$, and $V := \mathsf{SVars}(\alpha)$. Every $r \in \mathsf{Ref}(\alpha, w)$ defines a (V, w)-tuple μ^r in the following way: For every $x \in \mathsf{Vars}(\alpha)$, there exist uniquely defined r_1, r_2, r_3 with $r = r_1 \vdash_x r_2 \dashv_x r_3$. Then $\mu^r(x) := [|\mathsf{clr}(r_1)| + 1, |\mathsf{clr}(r_1 r_2)| + 1\rangle$. The function $[\![\alpha]\!]$ from words $w \in \Sigma^*$ to (V, w)-relations is defined by $[\![\alpha]\!](w) := \{\mu^r \mid r \in \mathsf{Ref}(\alpha, w)\}$.

▶ **Example 6.** Assume that $\mathtt{a}, \mathtt{b} \in \Sigma$. We define the functional 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 $([2,4\rangle, [3,3\rangle, [3,4\rangle)$, $([2,5\rangle, [3,4\rangle, [4,5\rangle)$, $([2,6\rangle, [3,5\rangle, [5,6\rangle)$, $([3,5\rangle, [4,4\rangle, [4,5\rangle)$, $([3,6\rangle, [4,5\rangle, [5,6\rangle)$.

As one example of an $r \in \mathsf{Ref}(\alpha, w)$, consider $r = \mathtt{b}\vdash_x \mathtt{a}\vdash_y \mathtt{a}\dashv_y \vdash_z \mathtt{b}\dashv_z \dashv_x \mathtt{a}$. This yields $\mu^r(x) = [2,5\rangle$, $\mu^r(y) = [3,4\rangle$, and $\mu^r(z) = [4,5\rangle$.

It is easily seen that the definition of $[\![\alpha]\!]$ with ref-words is equivalent to the definition from [11]; and so is the definition of functional regex formulas. Basing the definition of semantics on ref-words has two advantages: Firstly, treating $\mathcal{R}(\alpha)$ as a language over $(\Sigma \cup \Gamma)$ allows us to use standard techniques from automata theory, and secondly, it generalizes well to two automata models for defining spanners from [11]. We begin with the first model:

▶ **Definition 7.** Let $V \subset \Xi$ be a finite set of variables, and define $\Gamma_V := \{\vdash_x, \dashv_x \mid x \in V\}$. A *variable set automaton (vset-automaton)* over Σ with variables V is a tuple $A = (Q, q_0, q_f, \delta)$, where Q is the set of states, $q_0, q_f \in Q$ are the initial and the final state, and $\delta\colon Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma_V) \to 2^Q$ is the transition function.

We interpret A as a directed graph, where the nodes are the elements of Q, each $q \in \delta(p, a)$ is represented with an edge from p to q with label a, where $p \in Q$ and $a \in (\Sigma \cup \{\varepsilon\} \cup \Gamma_V)$. We extend δ to $\hat{\delta}\colon Q \times (\Sigma \cup \Gamma_V)^* \to 2^Q$ such that for all $p, q \in Q$ and $r \in (\Sigma \cup \Gamma_V)^*$, $q \in \hat{\delta}(p, r)$ if and only if there is a path from p to q that is labeled with r. We use this to define $\mathcal{R}(A) := \{r \in (\Sigma \cup \Gamma_V)^* \mid q_f \in \hat{\delta}(q_0, r)\}$.

Let $\mathsf{SVars}(A)$ be the set of all $x \in V$ such that $\vdash_x$ or $\dashv_x$ occurs in A. A ref-word $r \in \mathcal{R}(A)$ is *valid* if, for every $x \in \mathsf{SVars}(A)$, $|r|_{\vdash_x} = |r|_{\dashv_x} = 1$, and $\vdash_x$ occurs to the left of $\dashv_x$. Then $\mathsf{Ref}(A)$, $\mathsf{Ref}(A, w)$, and $[\![A]\!]$ are defined analogously to regex formulas.

Hence, a vset-automaton can be understood as an NFA over Σ that has additional transitions that open and close variables. When using ref-words, it is interpreted as NFA over the alphabet $(\Sigma \cup \Gamma)$, and defines the ref-language $\mathcal{R}(A)$; and $\mathsf{Ref}(A)$ is the subset of $\mathcal{R}(A)$ where each variable in V is opened and closed exactly once (and the two operations occur in the correct order). This also demonstrates why our definition is equivalent to the definition from [11] (there, the condition that every variable has to be opened and closed exactly once is realized by the definition of the successor relation for configurations). In particular, every word in $\mathsf{Ref}(A)$ encodes an accepting run of A (as defined in [11]).

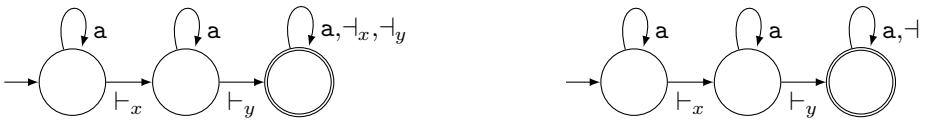


Figure 1 A vset-automaton A_{set} (left) and a vstk-automaton A_{stk} (right). Then $\mathsf{Ref}(A_{\mathsf{set}})$ consist of ref-words $r = \mathtt{a}^{i_1}\vdash_x \mathtt{a}^{i_2}\vdash_y \mathtt{a}^{i_3}\dashv_{z_1} \mathtt{a}^{i_4}\dashv_{z_2} \mathtt{a}^{i_5}$, with $i_1, \ldots, i_5 \geq 0$, $z_1, z_2 \in \{x, y\}$ and $z_1 \neq z_2$. Similarly, the ref-words from $\mathsf{Ref}(A_{\mathsf{stk}})$ are of the form $r = \mathtt{a}^{i_1}\vdash_x \mathtt{a}^{i_2}\vdash_y \mathtt{a}^{i_3}\dashv \mathtt{a}^{i_4}\dashv \mathtt{a}^{i_5}$, with $i_1, \ldots, i_5 \geq 0$. The left $\dashv$ closes y, and the right $\dashv$ closes x.

Although interpreting vset-automata as acceptors of ref-languages is often convenient, it comes with a caveat. While $\mathsf{Ref}(A_1) = \mathsf{Ref}(A_2)$ implies $[\![A_1]\!] = [\![A_2]\!]$ for all $A_1, A_2 \in \mathsf{VA}_{\mathsf{set}}$, the converse does not hold: Consider the two ref-words $r_1 := \vdash_x\vdash_y \mathtt{a}\dashv_y\dashv_x$ and $r_2 := \vdash_y\vdash_x \mathtt{a}\dashv_x\dashv_y$. Both define the same $\mathtt{a}$-tuple μ (with $\mu(x) = \mu(y) = [1, 2\rangle$), although $r_1 \neq r_2$.

Fagin et al. [11] also introduced the *variable stack automaton (vstk-automaton)*. Its definition is almost identical to vset-automata, the only difference is that instead of using a distinct symbol $\dashv_x$ for every variable x, vstk-automata have only a single closing symbol $\dashv$, which closes the variable that was opened most recently (hence the "stack" in "variable stack automaton"). From now on, assume that Γ also includes $\dashv$, and extend clr by defining $\mathsf{clr}(\dashv) := \varepsilon$. For every vstk-automaton A, $\mathcal{R}(A)$ and $\mathsf{SVars}(A)$ are defined as for vset-automata. We define $\mathsf{Ref}(A)$ as the set of all valid $r \in \mathcal{R}(A)$, where r is valid if, for each $x \in \mathsf{SVars}(A)$, $\vdash_x$ occurs exactly once in w, and is closed by a matching $\dashv$. More formally, r is valid if $|r|_\dashv = \sum_{x\in\mathsf{SVars}(A)} |r|_{\vdash_x}$, and for every $x \in \mathsf{SVars}(A)$, we have that $|r|_{\vdash_x} = 1$ and r can be uniquely factorized into $r = r_1\vdash_x r_2\dashv r_3$, with $|r_2|_\dashv = \sum_{x\in\mathsf{SVars}(A)} |r_2|_{\vdash_x}$. This unique factorization allows us to interpret every $r \in \mathsf{Ref}(A)$ as a μ^r analogously to vset-automata.

We use $\mathsf{VA}_{\mathsf{set}}$ and $\mathsf{VA}_{\mathsf{stk}}$ to denote the set of all vset-automata and all vstk-automata, respectively. We define $\mathsf{VA} := \mathsf{VA}_{\mathsf{set}} \cup \mathsf{VA}_{\mathsf{stk}}$, and refer to the elements of VA as *v-automata*. An example for each type of v-automata can be found in Figure 1.

2.1.2 Spanner Algebras

In order to construct more sophisticated spanners, we introduce spanner operators.

▶ **Definition 8.** Let P, P_1, P_2 be spanners. The algebraic operators *union*, *projection*, *natural join* and *selection* are defined as follows.

Union 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 $\mathsf{SVars}(P_1 \cup P_2) := \mathsf{SVars}(P_1)$ and $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$, $w \in \Sigma^*$.

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 $\mu \in P(w)$ 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 $\mu_1 \in P_1(w)$ and $\mu_2 \in P_2(w)$ 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 $x_1, \ldots, x_k \in \mathsf{SVars}(P)$, written as $\zeta^R_{x_1,\ldots,x_k}$. The *selection* $\zeta^R_{x_1,\ldots,x_k} P$ is defined by $\mathsf{SVars}\left(\zeta^R_{x_1,\ldots,x_k} P\right) := \mathsf{SVars}(P)$ and, for all $w \in \Sigma^*$, $\zeta^R_{x_1,\ldots,x_k} P(w)$ is the set of all $\mu \in P(w)$ for which $\left(w_{\mu(x_1)}, \ldots, w_{\mu(x_k)}\right) \in R$.

Note that join operates on spans, while selection operates on the subwords of w that are described by the spans. Like [11] (also see the brief remark on core spanners below), we mostly consider the string equality selection operator $\zeta^=$. Hence, unless otherwise noted,

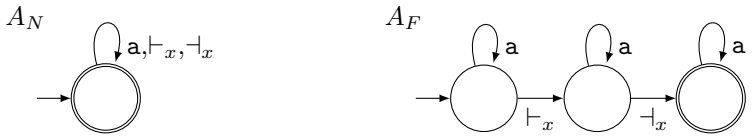


Figure 2 Two vset-automata A_N and A_F, which both define the universal spanner for the single variable x (cf. [11]) over the alphabet $\{\mathsf{a}\}$. As $\mathcal{R}(A_N)$ contains ref-words like $\mathsf{a}\dashv_x\mathsf{a}\vdash_x$ or $\mathsf{a}\vdash_x\mathsf{a}\vdash_x$, A_N is not functional. In contrast to this, A_F is functional, as it uses its three states to ensure that its ref-words contain each of $\vdash_x$ and $\dashv_x$ exactly once, and in the right order.

the term "selection" refers to selection by the k-ary string equality relation. 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. If applicable, we write $\cap$ and $\times$ instead of $\bowtie$.

We refer to regex formulas and v-automata as *primitive spanner representations*. A *spanner algebra* is a finite set of spanner operators. If O is a spanner algebra and C is a class of primitive spanner representations, then C^{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 C. 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, $[\![C^{\mathsf{O}}]\!]$ is the closure of $[\![C]\!]$ under the spanner operators in O.

Fagin et al. [11] refer to $[\![\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!]$ as the class of *core spanners*, as these capture the core of the functionality of SystemT. Following this, we define $\mathsf{core} := \{\pi, \zeta^=, \cup, \bowtie\}$. This allows us to use more compact notation, like $\mathsf{RGX}^{\mathsf{core}}$, $\mathsf{VA}^{\mathsf{core}}_{\mathsf{set}}$, $\mathsf{VA}^{\mathsf{core}}_{\mathsf{stk}}$, and $\mathsf{VA}^{\mathsf{core}}$.

2.1.3 Some Results on Automata-Based Spanners

This section develops some basic insights on aspects of v-automata, which we later use to provide further context to the main result in Section 3. While [11] defines RGX as the set of functional regex formulas, no analogous restriction is used for $\mathsf{VA}_{\mathsf{set}}$ and $\mathsf{VA}_{\mathsf{stk}}$. Using ref-word terminology, this means that for each $\alpha \in \mathsf{RGX}$, all information that is needed to determine $\mathsf{Ref}(\alpha, w)$ can be derived from $\mathcal{R}(\alpha)$. We adapt this notion to v-automata, and call $A \in \mathsf{VA}$ *functional* if $\mathsf{Ref}(A) = \mathcal{R}(A)$. Figure 2 contains examples for (non-)functional vset-automata (similar observations can be made for vstk-automata). This definition is also natural under the semantics as defined in [11]: Translated to these semantics, a v-automaton A is functional if every path from q_0 to q_f yields an accepting run of A.

While v-automata in general have to keep track of the used variables, functional v-automata store this information implicitly in their states. Hence, their evaluation problem can be solved efficiently:

▶ **Lemma 9.** *Given $w \in \Sigma^*$, a functional $A \in \mathsf{VA}$, and a $(\mathsf{SVars}(A), w)$-tuple μ, $\mu \in [\![A]\!](w)$ can be decided in polynomial time.*

With a slight modification of standard reachability techniques, we can show the following:

▶ **Proposition 10.** *Given $A \in \mathsf{VA}$, we can decide in polynomial time whether A is functional.*

In contrast to Lemma 9, even special cases of evaluating non-functional v-automata are hard:

▶ **Lemma 11.** *Given $A \in \mathsf{VA}$, deciding whether $[\![A]\!](\varepsilon) \neq \emptyset$ is NP-complete.*

The proof uses a basic reduction from the Hamiltonian path problem, which is NP-complete (cf. Garey and Johnson [13]). We discuss the matching upper bound in Section 3.

Obviously, every vset- or vstk-automaton can be transformed into an equivalent functional automaton, by intersecting with an NFA that accepts the set of all valid ref-words, using the standard constructions for NFA-intersection. Lemma 11 already suggests that this conversion is not possible in polynomial time (unless the number of variables is bounded); we also show matching exponential size bounds:

▶ **Proposition 12.** *Let* $f_{\mathsf{set}}(k) := 3^k$, $f_{\mathsf{stk}}(k) := (k+2)2^{k-1}$, *and* $s \in \{\mathsf{set}, \mathsf{stk}\}$. *For every* $A \in \mathsf{VA}_s$ *with* n *states and* k *variables, there exists an equivalent functional* $A_F \in \mathsf{VA}_s$ *with* $n \cdot f_s(k)$ *states. For every* $k \geq 1$, *there is an* $A_k \in \mathsf{VA}_s$ *with one state and* k *variables, such that every equivalent functional* $A_F \in \mathsf{VA}_s$ *has at least* $f_s(k)$ *states.*

The lower bounds are obtained by treating the v-automata as NFAs, which allows the use of a fooling set technique by Birget [2]. We briefly compare vset- and vstk-automata: As shown in [11], $[\![\mathsf{VA}_{\mathsf{stk}}]\!] \subset [\![\mathsf{VA}_{\mathsf{set}}]\!]$. The reason for this is that, as vstk-automata always close the variable that was opened most recently, they can only express hierarchical spanners (a spanner is hierarchical if its spans do not overlap – for a formal definition, see [11]). While this behavior can be simulated with vset-automata, a slight modification of the proof of Proposition 12 shows that this is not possible in an efficient manner:

▶ **Proposition 13.** *For every* $k \geq 1$, *there is a vstk-automaton* A_k *with one state and* $k+2$ *edges, such that every vset-automaton* A *with* $[\![A]\!] = [\![A_k]\!]$ *has at least* $k!$ *states.*

Hence, although vstk-automata can express strictly less than vset-automata, they may offer an exponential succinctness advantage. We revisit this in Section 3.

2.2 Word Equations and EC$^{\mathsf{reg}}$

A *pattern* is a word $\alpha \in (\Sigma \cup \Xi)^*$, and a *word equation* is a pair of patterns (η_L, η_R), which can also be written as $\eta_L = \eta_R$. A *pattern substitution* (or just *substitution)* is a morphism $\sigma\colon (\Xi \cup \Sigma)^* \to \Sigma^*$ with $\sigma(a) = a$ for all $a \in \Sigma$. Recall that a morphism from a free monoid A^* to a free monoid B^* is a function $h\colon A^* \to B^*$ such that $h(x \cdot y) = h(x) \cdot h(y)$ for all $x, y \in A^*$. Hence, in order to define h, it suffices to define $h(x)$ for all $x \in A$. Therefore, we can uniquely define a pattern substitution σ by defining $\sigma(x)$ for each $x \in \Xi$.

A substitution σ is a *solution* of a word equation (η_L, η_R) if $\sigma(\eta_L) = \sigma(\eta_R)$. The set of all variables in a pattern α is denoted by $\mathsf{var}(\alpha)$. We extend this to word equations $\eta = (\eta_L, \eta_R)$ by $\mathsf{var}(\eta) := \mathsf{var}(\eta_L) \cup \mathsf{var}(\eta_R)$.

The *existential theory of concatenation* EC is obtained by combining word equations with $\wedge$, $\vee$, and existential quantification over variables. Formally, every word equation η is an EC-formula, and $\sigma \models \eta$ if σ is a solution of η. If φ_1 and φ_2 are EC-formulas, so are $\varphi_\wedge := (\varphi_1 \wedge \varphi_2)$ and $\varphi_\vee := (\varphi_1 \vee \varphi_2)$, with $\sigma \models \varphi_\wedge$ if $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$; and $\sigma \models \varphi_\vee$ if $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$. Finally, for every EC-formula φ and every $x \in \Xi$, $\psi := (\exists x\colon \varphi)$ is an EC-formula, and $\sigma \models \psi$ if there exists a $w \in \Sigma^*$ such that $\sigma_{[x \to w]} \models \varphi$, where the substitution $\sigma_{[x \to w]}$ is defined by $\sigma_{[x \to w]}(y) := w$ if $y = x$, and $\sigma_{[x \to w]}(y) := \sigma(y)$ if $y \neq x$.

We also consider the *existential theory of concatenation with regular constraints*, EC$^{\mathsf{reg}}$. In addition to word equations, EC$^{\mathsf{reg}}$-formulas can use constraints $\mathsf{C}_A(x)$, where $x \in \Xi$ is a variable, A is an NFA, and $\sigma \models \mathsf{C}_A(x)$ if $\sigma(x) \in \mathcal{L}(A)$. As every regular expression can be directly converted into an equivalent NFA, we also allow constraints $\mathsf{C}_\alpha(x)$ that use regular expressions instead of NFAs. We freely omit parentheses, as long as the meaning of the formula

remains unambiguous. To increase readability, we allow existential quantifiers to range over multiple variables; i. e., we use $\exists x_1, x_2, \ldots, x_k \colon \varphi$ as a shorthand for $\exists x_1 \colon \exists x_2 \colon \ldots \exists x_k \colon \varphi$.

The set $\mathsf{free}(\varphi)$ of *free variables* of an $\mathsf{EC}^{\mathsf{reg}}$-formula φ is defined by $\mathsf{free}(\eta) = \mathsf{var}(\eta)$, $\mathsf{free}(\varphi_1 \wedge \varphi_2) := \mathsf{free}(\varphi_1 \vee \varphi_2) := \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2)$, and $\mathsf{free}(\exists x \colon \varphi) := \mathsf{free}(\varphi) - \{x\}$. Finally, we define $\mathsf{free}(C) = \emptyset$ for every constraint C. (While one could also argue in favor of $\mathsf{free}(C(x)) = \{x\}$, choosing $\emptyset$ simplifies the definitions in Section 3). For all $\varphi \in \mathsf{EC}^{\mathsf{reg}}$, let $[\![\varphi]\!] := \{\sigma \mid \sigma \models \varphi\}$. Two formulas $\varphi_1, \varphi_2 \in \mathsf{EC}^{\mathsf{reg}}$ are *equivalent* if $\mathsf{free}(\varphi_1) = \mathsf{free}(\varphi_2)$ and $[\![\varphi_1]\!] = [\![\varphi_2]\!]$. We write this as $\varphi_1 \equiv \varphi_2$. For increased readability, we use $\varphi(x_1, \ldots, x_k)$ to denote $\mathsf{free}(\varphi) = \{x_1, \ldots, x_k\}$. Building on this, we also use $(w_1, \ldots, w_k) \models \varphi(x_1, \ldots, x_k)$ to denote $\sigma \models \varphi$ for the substitution σ that is defined by $\sigma(x_i) := w_i$, $1 \leq i \leq k$.

▶ **Example 14.** Consider the EC-formula $\varphi_1(x, y, z) := \exists \hat{x}, \hat{y} \colon (x = z\hat{x} \wedge y = z\hat{y})$ and the $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_1(x, y, z) := \exists \hat{x}, \hat{y} \colon (x = z\hat{x} \wedge y = z\hat{y} \wedge \mathsf{C}_{\Sigma^+}(z))$. Then $\sigma \models \varphi_1$ if and only if $\sigma(x)$ and $\sigma(y)$ have $\sigma(z)$ as common prefix. If, in addition to this, $\sigma(z) \neq \varepsilon$, then $\sigma \models \varphi_2$.

Every EC-formula can be converted into a single word equation (cf. Karhumäki, Mignosi, and Plandowski [18]), and every $\mathsf{EC}^{\mathsf{reg}}$-formula into a single word equation with rational constraints (cf. Diekert [6]). For conjunctions, the construction is easily explained: Choose distinct letters $\mathtt{a}, \mathtt{b} \in \Sigma$. Hmelevskii's pattern pairing function is defined by $\langle \alpha, \beta \rangle := \alpha\mathtt{a}\beta\mathtt{a}\alpha\mathtt{b}\beta\mathtt{b}$. Then $(\alpha_L = \alpha_R) \wedge (\beta_L = \beta_R)$ holds if and only if $\langle \alpha_L, \beta_L \rangle = \langle \alpha_R, \beta_R \rangle$. The construction for disjunctions is similar, but more involved (and, in general, converting a formula with alternating disjunctions and conjunctions leads to an exponential size increase).

Satisfiability for $\mathsf{EC}^{\mathsf{reg}}$ is PSPACE-complete; but even for EC, showing the upper bound is by no means trivial (cf. [6, 8]). Note that negation is left out intentionally: Even the EC-fragment $\forall\exists^3$ (one universal over three existential variables) is undecidable (Durnev [9]).

3 SpLog: A Logic for Spanners

As shown by Freydenberger and Holldack [12], every element of $\mathsf{RGX}^{\mathsf{core}}$ can be converted into an $\mathsf{EC}^{\mathsf{reg}}$-formula, and every word equation with regular constraints (and, hence, every $\mathsf{EC}^{\mathsf{reg}}$-formula) can be converted to $\mathsf{RGX}^{\mathsf{core}}$. While the latter results in a spanner that is satisfiable if and only if the formula is satisfiable, the input word of the spanner needs to encode the whole word equation (see the comments after Example 14). Hence, the spanner can only simulate satisfiability, but not evaluation. To overcome this problem, we introduce SpLog (short for *spanner logic*), a fragment of $\mathsf{EC}^{\mathsf{reg}}$ that directly corresponds to core spanners:

▶ **Definition 15.** A formula $\varphi \in \mathsf{EC}$ is called *safe* if the following two conditions are met:
1. If $(\varphi_1 \vee \varphi_2)$ is a subformula of φ, then $\mathsf{free}(\varphi_1) = \mathsf{free}(\varphi_2)$.
2. Every constraint $\mathsf{C}_A(x)$ occurs only as part of a subformula $(\psi \wedge \mathsf{C}_A(x))$, with $x \in \mathsf{free}(\psi)$.

Let $W \in \Xi$. The set of all SpLog-*formulas with main variable* W, $\mathsf{SpLog}(\mathsf{W})$, is the set of all safe $\varphi \in \mathsf{EC}^{\mathsf{reg}}$ such that
1. all word equations in φ are of the form $\mathsf{W} = \eta_R$, with $\eta_R \in ((\Xi - \{\mathsf{W}\}) \cup \Sigma)^*$,
2. for every subformula ψ of φ, $\mathsf{W} \in \mathsf{free}(\psi)$.

We also define the set of all SpLog-formulas by $\mathsf{SpLog} := \bigcup_{W \in \Xi} \mathsf{SpLog}(\mathsf{W})$, and we use $\mathsf{SpLog}_{\mathsf{rx}}$ to denote the fragment of SpLog that exclusively defines constraints with regular expressions instead of NFAs.

Less formally, for every $\varphi \in \mathsf{SpLog}(\mathsf{W})$, the main variable W appears on the left side of every equation (and is never bound with a quantifier). The requirement that φ is safe ensures that each variable corresponds to a subword of W. When declaring the free variables

of a SpLog-formula, we slightly diverge from our convention for $\mathsf{EC}^{\mathsf{reg}}$-formulas, and write $\varphi(\mathsf{W}; x_1, \ldots, x_k)$ to denote a formula with main variable W, and $\mathsf{free}(\varphi) = \{\mathsf{W}, x_1, \ldots, x_k\}$.

▶ **Example 16.** Let $\varphi_1(\mathsf{W}; x) := \exists y, z_1, z_2 \colon (\mathsf{W} = yy \wedge \mathsf{W} = z_1 x z_2 \wedge \mathsf{C}_{\Sigma+}(x))$. Then φ_1 is a $\mathsf{SpLog}(\mathsf{W})$-formula, and $\sigma \models \varphi_1$ iff. $\sigma(\mathsf{W})$ is a square and contains $\sigma(x)$ as a nonempty subword. In contrast to this, $\varphi_2(\mathsf{W}; x, y) := (\mathsf{W} = xx \vee \mathsf{W} = yyy)$ is not a SpLog-formula, as it is not safe (intuitively, if e. g. $\sigma(\mathsf{W}) = \sigma(x)^2$, then $\sigma \models \varphi_2$, even if $\sigma(y) \not\sqsubseteq \sigma(\mathsf{W})$). Further examples for SpLog-formulas can be found in Section 4.

Before we examine conversions between SpLog and various representations of core spanners, we introduce a result that provides us with a convenient shorthand notation:

▶ **Lemma 17.** *Let $\varphi \in \mathsf{SpLog}(\mathsf{W})$, $x \in \mathsf{free}(\varphi) - \{\mathsf{W}\}$, and let $\psi \in \mathsf{SpLog}(x)$ such that W does not occur in ψ. We can compute in polynomial time a $\chi \in \mathsf{SpLog}(\mathsf{W})$ with $\chi \equiv (\varphi \wedge \psi)$.*

Proof. Let x_1, x_2 be new variables and define $\chi := \varphi \wedge \exists x_1, x_2 \colon \big((W = x_1 \cdot x \cdot x_2) \wedge \hat{\psi}\big)$, where $\hat{\psi}$ is obtained from ψ by replacing every equation $x = \eta_R$ with $W = x_1 \cdot \eta_R \cdot x_2$. Given $W = x_1 \cdot x \cdot x_2$, these equations define the same relations as the $x = \eta_R$. As W does not occur in ψ, $\chi \equiv (\varphi \wedge \psi)$ holds. ◀

This allows us to combine SpLog-formulas with different main variables.

When comparing the expressive power of spanners and SpLog, we need to address one important difference of the two models: While SpLog is defined on words, spanners are defined on spans of an input word. Apart from slight modifications to adapt it to SpLog, the following definition for the conversion of spanners to formulas was introduced in [12]:

▶ **Definition 18.** Let P be a spanner and let $\varphi \in \mathsf{SpLog}(\mathsf{W})$ with $\mathsf{free}(\varphi) = \{\mathsf{W}\} \cup \{x^P, x^C \mid x \in \mathsf{SVars}(P)\}$. We say that φ *realizes* P if, for all substitutions σ, $\sigma \models \varphi$ holds if and only if there is a $\mu \in P(\sigma(\mathsf{W}))$ such that, for each $x \in \mathsf{SVars}(P)$, $\sigma(x^P) = \sigma(\mathsf{W})_{[1,i\rangle}$ and $\sigma(x^C) = \sigma(\mathsf{W})_{[i,j\rangle}$, where $[i, j\rangle = \mu(x)$.

The intuition behind this definition is that every span $[i, j\rangle$ of w is characterized by its content $w_{[i,j\rangle}$, and by $w_{[1,i\rangle}$, the prefix of w that precedes the span. Hence, every variable x of the spanner is represented by two variables x^C and x^P, which store the content and the prefix, respectively. Moreover, the main variable of the SpLog-formula corresponds to the input word of the spanner. Next, we consider conversions in the other direction:

▶ **Definition 19.** Let $\varphi \in \mathsf{SpLog}(\mathsf{W})$. A spanner P with $\mathsf{SVars}(P) = \mathsf{free}(\varphi) - \{\mathsf{W}\}$ *realizes* φ if, for all substitutions σ, $\sigma \models \varphi$ holds if and only if there is a $\mu \in P(\sigma(\mathsf{W}))$ such that $\sigma(\mathsf{W})_{\mu(x)} = \sigma(x)$ for all $x \in \mathsf{SVars}(P)$.

Again, the main variable of the SpLog-formula corresponds to the input word of the spanner. Note that it is possible to define realizability in a stricter way: Instead of requiring that $\mu \in P(\sigma(\mathsf{W}))$ holds for *one* μ with $\sigma(\mathsf{W})_{\mu(x)} = \sigma(x)$ for all $x \in \mathsf{SVars}(P)$, we could require $\mu \in P(\sigma(\mathsf{W}))$ for *all such* μ. But such a spanner can directly be constructed from a spanner P that satisfies Definition 19, by joining P with a universal spanner (cf. [11]), and using string equality selections (for the matter of this paper, this will not affect the complexity, as consider spanners with string equality relations). ´

Let C_1 be a class of spanner representations (or SpLog-formulas), and let C_2 be a class of SpLog-formulas (or spanner representations). We say that there is a *polynomial size conversion* from C_1 to C_2 if there is an algorithm that, given a $\rho_1 \in C_1$, computes a $\rho_2 \in C_2$ such that ρ_2 realizes ρ_1, and the size of ρ_2 is polynomial in the size of ρ_1. If the algorithm also works in polynomial time, we say that there is a *polynomial time conversion*. First, we use Lemma 11 to obtain a negative result on conversions to SpLog:

▶ **Lemma 20.** $\mathsf{P} = \mathsf{NP}$*, if there is a polynomial time conversion from* $\mathsf{VA}_{\mathsf{set}}$ *or* $\mathsf{VA}_{\mathsf{stk}}$ *to* SpLog*.*

This result is less problematic than it might appear, as it can be overcome with a very minor relaxation of the definition of polynomial time conversions: We say that a SpLog-formula φ *realizes a spanner* P *modulo* ε if φ realizes a spanner $\hat{P}$ with $P(w) = \hat{P}(w)$ for all $w \in \Sigma^+$. In other words, φ realizes P on all inputs, except ε (where the behavior is undefined). Likewise, a *polynomial time conversion modulo* ε computes formulas that realize the spanners modulo ε. We now state the central result of this paper:

▶ **Theorem 21.** *There are polynomial time conversions*
1. *from* $\mathsf{RGX}^{\mathsf{core}}$ *to* $\mathsf{SpLog}_{\mathsf{rx}}$*, and from* $\mathsf{SpLog}_{\mathsf{rx}}$ *to* $\mathsf{RGX}^{\mathsf{core}}$*,*
2. *from* SpLog *to* $\mathsf{VA}^{\mathsf{core}}_{\mathsf{set}}$ *and to* $\mathsf{VA}^{\mathsf{core}}_{\mathsf{stk}}$*,*
3. *modulo* ε *from* $\mathsf{VA}^{\mathsf{core}}_{\mathsf{set}}$ *and* $\mathsf{VA}^{\mathsf{core}}_{\mathsf{stk}}$ *to* SpLog*.*

Recall that $\mathsf{SpLog}_{\mathsf{rx}}$ is the fragment of SpLog that uses only regular expressions to define constraints. The conversion from $\mathsf{RGX}^{\mathsf{core}}$ to $\mathsf{SpLog}_{\mathsf{rx}}$ is almost identical to the conversion from $\mathsf{RGX}^{\mathsf{core}}$ to $\mathsf{EC}^{\mathsf{reg}}$ that was presented in [12]. The most technically challenging part is the conversion of non-functional v-automata to SpLog, which requires a gadget that acts as a synchronization mechanism inside the formula. This is realized by sets of variables that map to either ε or the first letter of W, which is the main reason that the construction only works modulo ε. For most applications, $P(\varepsilon)$ can be considered a pathological edge case: As $P(w)$ can be understood as searching in w, $P(\varepsilon)$ corresponds to a search in ε. But even if we insist on correctness on ε, we are still able to observe polynomial size conversions:

▶ **Corollary 22.** *There are polynomial size conversions from* $\mathsf{VA}^{\mathsf{core}}$ *to* SpLog*.*

As discussed in Section 2.1.3, there are exponential blowups when moving from general to functional v-automata, as well as from vstk- to vset-automata. Another consequence of Theorem 21 is that this does not hold if we extend the automata with the core-algebra:

▶ **Corollary 23.** *Given* $\rho \in \mathsf{VA}^{\mathsf{core}}$*, we can compute an equivalent* $\rho_f \in \mathsf{VA}^{\{\pi,\zeta^=,\cup,\times\}}_{\mathsf{set}}$ *or* $\rho_f \in \mathsf{VA}^{\{\pi,\zeta^=,\cup,\times\}}_{\mathsf{stk}}$*, where* ρ_f *is of polynomial size and every v-automaton in* ρ_f *is functional.*

Again, due to Lemma 11, computing an equivalent ρ_f in polynomial time would imply $\mathsf{P} = \mathsf{NP}$; but we can compute in polynomial time a ρ_f that is equivalent modulo ε.

This also demonstrates that $\bowtie$ can be simulated by a combination of $\times$ and $\zeta^=$, in addition to showing that the algebra compensates the aforementioned disadvantages in succinctness. While we leave open whether there are polynomial size conversions from SpLog to $\mathsf{RGX}^{\mathsf{core}}$, or from $\mathsf{VA}^{\mathsf{core}}$ to $\mathsf{SpLog}_{\mathsf{rx}}$ or $\mathsf{RGX}^{\mathsf{core}}$, we observe that, due to Theorem 21, all these questions are equivalent to asking how efficiently $\mathsf{SpLog}_{\mathsf{rx}}$ can simulate NFAs.

Another question that we leave open is whether $[\![\mathsf{SpLog}]\!] = [\![\mathsf{EC}^{\mathsf{reg}}]\!]$ (see Section 4.4). But we are able to state an important difference between the two logics: While evaluation of $\mathsf{EC}^{\mathsf{reg}}$-formulas is PSPACE-hard, this does not hold for SpLog (assuming $\mathsf{NP} \neq \mathsf{PSPACE}$):

▶ **Corollary 24.** *Given* $\varphi \in \mathsf{SpLog}$ *and a substitution* σ*, deciding* $\sigma \models \varphi$ *is NP-complete. For every fixed* $\varphi \in \mathsf{SpLog}$*, given a substitution* σ*, deciding* $\sigma \models \varphi$ *is in NL.*

Finally, we remark that Theorem 21 also shows that the PSPACE upper bounds of deciding satisfiability and hierarchicality for $\mathsf{RGX}^{\mathsf{core}}$ that were observed in [12] also apply to $\mathsf{VA}^{\mathsf{core}}_{\mathsf{set}}$ and $\mathsf{VA}^{\mathsf{core}}_{\mathsf{stk}}$. The same holds for the uppers bound for combined and data complexity.

4 Expressing Relations in SpLog

This section examines how SpLog expresses relations and languages: Section 4.1 lays the formal groundwork by introducing selectability of relations in SpLog (and connecting it to core spanners), Section 4.2 contains an extended example, Section 4.3 provides an efficient conversion of a subclass of regex to SpLog, and Section 4.4 defines and applies a normal form.

4.1 Selectable Relations

One of the topics of Fagin et al. [11] is which relations can be used for selections in core spanners, without increasing the expressive power. This translates to the question which relations can be used in the definition of SpLog-formulas. For $\mathsf{EC}^{\mathsf{reg}}$, this question is simple: If, for any k-ary relation R, there is an $\mathsf{EC}^{\mathsf{reg}}$-formula φ_R such that $\vec{w} \models \varphi_R$ holds if and only if $\vec{w} \in R$, we know that we can use φ_R in the construction of $\mathsf{EC}^{\mathsf{reg}}$-formulas. In contrast to this, the special role of the main variable makes the situation a little bit more complicated for SpLog. Fortunately, [11] already introduced an appropriate concept for core spanners, that we can directly translate to SpLog: A k-ary word relation R is *selectable by core spanners* if, for every $\rho \in \mathsf{RGX}^{\mathsf{core}}$ and every sequence of variables $\vec{x} = (x_1, \dots, x_k)$ with $x_1, \dots, x_k \in \mathsf{SVars}(\rho)$, the spanner $[\![\zeta^R_{\vec{x}} \rho]\!]$ is expressible in $\mathsf{RGX}^{\mathsf{core}}$.

Analogously, we say that R is *SpLog-selectable* if for every $\varphi \in \mathsf{SpLog}$ and every sequence of variables $\vec{x} = (x_1, \dots, x_k)$ with $x_1, \dots, x_k \in \mathsf{free}(\varphi) - \{\mathsf{W}\}$, there is a $\varphi^R_{\vec{x}} \in \mathsf{SpLog}$ with $\mathsf{free}(\varphi) = \mathsf{free}(\varphi^R_{\vec{x}})$, and $\sigma \models \varphi^R_{\vec{x}}$ if and only if $\sigma \models \varphi$ and $(\sigma(x_1), \dots, \sigma(x_k)) \in R$. Before we consider some examples, we prove that these two definitions are equivalent not only to each other, but also to a more convenient third definition:

▶ **Lemma 25.** *For every relation $R \subseteq (\Sigma^*)^k$, $k \geq 1$, the following conditions are equivalent:*

1. *R is selectable by core spanners,*
2. *R is SpLog-selectable,*
3. *there is a $\varphi(W; x_1, \dots, x_k) \in \mathsf{SpLog}$ with $\sigma \models \varphi$ if and only if $(\sigma(x_1), \dots, \sigma(x_k)) \in R$.*

The equivalence of the two notions of selectability is one of the features of SpLog: When defining core spanners, one can use SpLog to define relations that are used in selections. As the proof is constructive and uses Theorem 21, this does not even affect efficiency. Before we discuss how the equivalent third condition in Lemma 25 can be used to simplify this even further, we consider a short example. As shown by Fagin et al. [11], the relation $\sqsubseteq$ is selectable by core spanners. We reprove this by showing that it is SpLog-selectable:

▶ **Example 26.** The subword relation $R_\sqsubseteq := \{(x, y) \mid x \sqsubseteq y\}$ is selected by the SpLog-formula $\varphi_\sqsubseteq(\mathsf{W}; x, y) := \exists z_1, z_2, y_1, y_2 \colon ((\mathsf{W} = z_1 y_1 x y_2 z_2) \wedge (\mathsf{W} = z_1 y z_2))$. If this is not immediately clear, note that the formula implies $z_1 y_1 x y_2 z_2 = z_1 y z_2$, which can be reduced to $y_1 x y_2 = y$.

This allows us to use $x \sqsubseteq y$ as a shorthand in SpLog-formulas. We also use $\sqsubseteq$ to address two inconveniences that arise when strictly observing the syntax of SpLog-formulas: Firstly, the need to introduce additional variables that might affect readability (like z_1, z_2 in Example 26), and, secondly, the basic form that equations have the main variable W on the left side. Together with Lemma 17 and the third condition of Lemma 25, the selectability of $\sqsubseteq$ allows us more compact definitions of SpLog-selectable relations: Instead of dealing with a single main variable, we can combine multiple SpLog-functions with different main variables. Hence, when using SpLog to define a relation over a set of variables V, we may assume that the

formula is of the form $(\bigwedge_{x\in V} x \sqsubseteq \mathsf{W}) \wedge \varphi$, and specify only φ. When the main variable is clear, we also omit it, as seen in the following examples:

▶ **Example 27.** Using the aforementioned simplifications, we can write the formula from Example 26 as $\varphi_{\sqsubseteq}(x,y) := \exists y_1, y_2 \colon (y = y_1 \cdot x \cdot y_2)$. Similarly, we can select the prefix relation with the formula $\varphi_{pref}(x,y) := \exists z \colon y = xz$. Both are shorthands for $\mathsf{SpLog}(\mathsf{W})$-formulas.

As mentioned above, this allows us to extend the syntax of SpLog with $x \sqsubseteq y$. Other extensions are $x \neq \varepsilon$ and $x \neq y$: For $x \neq \varepsilon$, we can use $\varphi_{\neq\varepsilon}(x) := (x \sqsubseteq \mathsf{W}) \wedge (\mathsf{C}_{\Sigma^+}(x))$. For the more general $x \neq y$, we consider the following $\mathsf{SpLog}(\mathsf{W})$-formula:

$$\varphi_{\neq}(x,y) := \big(\big(\exists x_2 \colon (x = yx_2) \wedge (x_2 \neq \varepsilon)\big) \vee \big(\exists y_2 \colon (y = xy_2) \wedge (y_2 \neq \varepsilon)\big)\big) \vee \Big(\bigvee_{\mathtt{a}\in\Sigma} \big(\exists z, x_2, y_2, b \colon (x = z\mathtt{a}x_2) \wedge (y = zby_2) \wedge \mathsf{C}_{\Sigma-\{\mathtt{a}\}}(b)\big)\Big)$$

The core spanner selectability of $\neq$ was already shown in [11], Proposition 5.2. Depending on personal preferences, $\varphi_{\neq}$ might be considered more readable than the spanner in that proof. A similar construction was also used in [18] to show EC-expressibility of $\neq$.

4.2 Extended Example: Relations for Approximate Matching

In this section, we examine how SpLog-formulas can be used to express relations of words that are approximately identical. In literature, this is commonly defined by the notion of an edit distance between two words. Following Navarro [21], we consider edit distances that are based on three operations: For words $u, v \in \Sigma^*$, we say that v can be obtained from u with

1. an *insertion*, if $u = u_1 \cdot u_2$ and $v = u_1 \cdot a \cdot u_2$,
2. a *deletion*, if $u = u_1 \cdot a \cdot u_2$ and $v = u_1 \cdot u_2$,
3. a *replacement*, if $u = u_1 \cdot a \cdot u_2$ and $v = u_1 \cdot b \cdot u_2$,

where $u_1, u_2 \in \Sigma^*$, $a, b \in \Sigma$. For every choice of permitted operations, a distance $d(u,v)$ is then defined as the minimal number of operations that is required to obtain v from u. One common example is the *Levenshtein-distance* d_L (also called *edit distance*), which uses insertion, deletion, and replacement. The following SpLog-formula demonstrates that, for each $k \geq 1$, the relation of all (u,v) with $d_L(u,v) \leq k$ is SpLog-selectable:

$$\varphi_{L(k)}(\mathsf{W}; x, y) := \exists x_1, \dots, x_k, y_1, \dots, y_k, z_0, \dots, z_k \colon$$
$$(x = z_0 \cdot x_1 \cdot z_1 \cdot x_2 \cdot z_2 \cdot \dots \cdot x_k \cdot z_k) \wedge (y = z_0 \cdot y_1 \cdot z_1 \cdot y_2 \cdot z_2 \cdot \dots \cdot y_k \cdot z_k) \wedge \bigwedge_{i=1}^{k} \mathsf{C}_\alpha(x_i) \wedge \bigwedge_{i=1}^{k} \mathsf{C}_\beta(y_i),$$

where $\alpha := \beta := (\Sigma \vee \varepsilon)$. Here, an insertion is expressed by assigning $x_i = \varepsilon$ and $y_i \in \Sigma$, a deletion is modeled by $x_i \in \Sigma$ and $y_i = \varepsilon$, and a replacement by $x_i, y_i \in \Sigma$. This case and $x_i = y_i = \varepsilon$ also cover cases where less than k operations are used.

Hence, by changing the constraints, this formula can also be used for the *Hamming distance* (which uses only replacements), and the *episode distance* (which uses only insertions), by defining $\alpha := \beta := \Sigma$, or $\alpha := \varepsilon$ and $\beta := \Sigma$ (respectively).

With some additional effort, we can also express the relation for the *longest common subsequence distance*, which uses only insertions and deletions. Instead of changing α or β, we need to ensure that for every i, $x_i = \varepsilon$ or $y_i = \varepsilon$ holds. We cannot directly write $((x_i = \varepsilon) \vee (y_i = \varepsilon))$, as this is not a safe formula. Instead, we extend the conjunction inside $\varphi_{L(k)}$ with $\bigwedge_{i=1}^{k}\big(((x_i = \varepsilon) \wedge (y_i \sqsubseteq \mathsf{W})) \vee ((y_i = \varepsilon) \wedge (x_i \sqsubseteq \mathsf{W}))\big)$, which is safe and equivalent to $\bigwedge_{i=1}^{k}((x_i = \varepsilon) \vee (y_i = \varepsilon))$. In other words, we use $\sqsubseteq$ to guard the x_i and y_i.

4.3 Efficient Conversion of vsf-Regex to SpLog

Most modern implementations of regular expressions contain a backreference operator that allows the definition of non-regular languages. This is formalized in *regex* (also called extended regular expressions), which extend regex formulas with variable references $\&x$ for every $x \in \Xi$. Intuitively, the semantics of $\&x$ can be understood as repeating the last value that was assigned to $x\{\ \}$, assuming that the regex is parsed left to right (for a formal definition that uses parse trees, see Freydenberger and Holldack [12]; for a definition with ref-words, see Schmid [23] or the full version of this paper). For example, $x\{\Sigma^*\} \cdot \&x \cdot \&x$ generates the language of all www with $w \in \Sigma^*$.

As shown by Fagin et al. [11], core spanners cannot define all regex languages. But [12] introduces a subclass of regex, the *vstar-free regex* (short: *vsf-regex*). A vsf-regex is a regex that does not use $x\{\ \}$ or $\&x$ inside a Kleene star *. Every vsf-regex can be converted effectively into a core spanner; but the conversion from [12] can lead to an exponential blowup. The question whether a more efficient conversion is possible was left open in [12]. Using SpLog, we answer this positively:

▶ **Theorem 28.** *Given a vsf-regex α, an equivalent $\varphi \in$ SpLog can be computed in polynomial time.*

As a consequence, it is possible to extend the syntax of $\mathsf{SpLog}_{\mathsf{rx}}$, SpLog, and $\mathsf{EC}^{\mathsf{reg}}$ by defining constraints with vsf-regex instead of classical regular expressions, without affecting the complexity of evaluation or satisfiability (and core spanner representations can also use vsf-regex). Theorem 28 also shows that, given vsf-regex $\alpha_1, \dots, \alpha_n$, one can decide in PSPACE whether $\bigcap \mathcal{L}(\alpha_i) = \emptyset$ (by converting each α_i into a formula φ_i, and deciding the satisfiability of $\bigwedge \varphi_i$). This is an interesting contrast to the full class of regex, where even the intersection emptiness problem for two languages is undecidable (cf. Carle and Narendran [3]).

4.4 A Normal Form for SpLog

Another advantage of using a logic is the existence of normal forms. In order to consider a short example of such an application, we introduce the following:

▶ **Definition 29.** A $\varphi \in$ SpLog is a *prenex conjunction* if $\varphi = \exists x_1, \dots, x_k \colon (\bigwedge_{i=1}^{m} \eta_i \wedge \bigwedge_{j=1}^{n} C_j)$, with $k, n \geq 0$, $m \geq 1$, where the η_i are word equations, and the C_j are constraints. A SpLog-formula is in *DPC-normal form (DPCNF)* if it is a disjunction of prenex conjunctions.

▶ **Lemma 30.** *Given $\varphi \in$ SpLog, we can compute $\psi \in$ SpLog in DPCNF with $\varphi \equiv \psi$.*

Fagin et al. [11] also examined $\mathsf{CRPQ}^{=}$ and $\mathsf{UCRPQ}^{=}$ (conjunctive regular path queries with string equality, and unions of these). These are existential positive queries on graphs, but when restricted to marked paths, $[\![\mathsf{UCRPQ}^{=}]\!] = [\![\mathsf{RGX}^{\mathsf{core}}]\!]$ holds (cf. [11]). Using our methods, it is easy to show that there are polynomial time transformations between $\mathsf{CRPQ}^{=}$ and SpLog prenex conjunctions, and between $\mathsf{UCRPQ}^{=}$ and DPCNF-formulas. The author conjectures that the exponential blowup from the proof of Lemma 30 is necessary. This would immediately imply that there is an exponential blowup from $\mathsf{RGX}^{\mathsf{core}}$ to $\mathsf{UCRPQ}^{=}$.

We use DPCNF to illustrate some differences between SpLog and $\mathsf{EC}^{\mathsf{reg}}$: First, consider the following: Every $\mathsf{EC}^{\mathsf{reg}}$-formula φ with $\mathsf{free}(\varphi) = \{x\}$ defines a language $\mathcal{L}(\varphi) := \{\sigma(x) \mid \sigma \models \varphi\}$ (in Section 5, we shall see that this has applications beyond the language theoretic point of view). For $\mathcal{C} \in \{\mathsf{EC}, \mathsf{EC}^{\mathsf{reg}}, \mathsf{SpLog}\}$, a language $L \subseteq \Sigma^*$ is a $\mathcal{C}$-language if there is a $\varphi \in \mathcal{C}$ with $\mathcal{L}(\varphi) = L$. We denote this by $L \in \mathcal{L}(\mathcal{C})$. For $L \subseteq \Sigma^*$ and $a \in \Sigma$, we define the *right quotient of L by a* as $L/\mathrm{a} := \{w \mid wa \in L\}$. It is easily seen that the class of $\mathsf{EC}^{\mathsf{reg}}$-languages

is closed under this operation, by using formulas like $\varphi_{/\mathtt{a}}(w) := \exists u\colon ((u = wa) \wedge \varphi(u))$. But as SpLog-variables can only contain subwords of the main variable, writing $u = wa$ is not possible in SpLog. The proof for the analogous is more involved and relies on Lemma 30.

▶ **Lemma 31.** *For every SpLog-language L and every $a \in \Sigma$, $L\,/\mathtt{a}$ is a SpLog-language.*

This allows us to use Greibach's Theorem [15] to prove the following:

▶ **Proposition 32.** *The following conditions are equivalent:*
1. $\mathcal{L}(\mathsf{EC}^{\mathrm{reg}}) = \mathcal{L}(\mathsf{SpLog})$,
2. *Given $\varphi \in \mathsf{EC}^{\mathrm{reg}}$, it is decidable whether $\mathcal{L}(\varphi) \in \mathcal{L}(\mathsf{SpLog})$,*
3. $\mathcal{L}(\mathsf{SpLog})$ *is closed under the prefix operator.*

This characterization might serve as a starting point to answer whether $[\![\mathsf{EC}^{\mathrm{reg}}]\!] = [\![\mathsf{SpLog}]\!]$, an important question that is left open in the present paper (we define $[\![\mathcal{C}]\!] := \{[\![\varphi]\!] \mid \varphi \in \mathcal{C}\}$ for $\mathcal{C} \subseteq \mathsf{EC}^{\mathrm{reg}}$). The question appears to be surprisingly complicated; even when only considering word equations. We only discuss this briefly, as a deeper examination would require considerable additional notation. In contrast to EC and $\mathsf{EC}^{\mathrm{reg}}$, SpLog can only use variables that are subwords of the main variable. Hence, one might expect that it is easy to construct an EC-formula where other variables are necessary. But as it turns out, many word equations can be rewritten to reduce the number of variables. In particular, there is a notion of word equations where the solution set can be *parameterized* (i. e., expressed with a finite number of so-called parametric words – for more details, see e. g. Czeizler [5], Karhumäki and Saarela [20]). In all cases that were considered by the author, it was possible to use these parametrizations to construct SpLog-formulas. Similarly, the solution sets of non-parametrizable equations that the author examined, like $x\mathtt{ab}y = y\mathtt{ba}x$, are self-similar in a way that allows the construction of SpLog-formulas (cf. Czeizler [5], Ilie and Plandowski [17]). On the other hand, these constructions do not appear to generalize straightforwardly to an equivalence proof.

5 Using EC-Inexpressibility to Prove Non-Selectability

While Section 4 examined various aspects of expressing relations in SpLog, the present section examines how to prove that a relation cannot be selected. As we shall see, this can often be proved by using inexpressibility of appropriate languages. To this end, general tools for language inexpressibility (like a pumping lemma) would be very convenient. Up to now, the only (somewhat) general technique for core spanner inexpressibility was given in [12], where it was observed that on unary alphabets, core spanners can only define semi-linear (and, hence, regular) languages. Due to the limited applicability of this result, having further inexpressibility techniques appears to be desirable. As SpLog is a fragment of $\mathsf{EC}^{\mathrm{reg}}$, it is natural to ask whether this connection can be used to obtain inexpressibility results.

Karhumäki et al. [18] developed multiple inexpressibility techniques for EC. Sadly, EC-inexpressibility does not imply SpLog-inexpressibility; e. g., for $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, the language $\{\mathtt{a}, \mathtt{b}\}^*$ is not EC-expressible (cf. [18]), but obviously SpLog-expressible. On the other hand, while $\mathsf{EC}^{\mathrm{reg}}$-inexpressibility results would be useful, to the author's knowledge, the only result in this direction is that every $\mathsf{EC}^{\mathrm{reg}}$-language is an EDT0L-language (cf. Ciobanu et al. [4]). While this allows the use of the EDT0L-inexpressibility results (e. g. Ehrenfeucht and Rozenberg [10]), the large expressive power of EDT0L limits the usefulness of this approach.

As we shall see, developing a sufficient criterion for EC-expressible SpLog-languages allows us to use one of the techniques from [18] for SpLog. We begin with a definition: A

language $L \subseteq \Sigma^*$ is *bounded* if there exist words $w_1, w_2, \ldots, w_n \in \Sigma^+$, $n \geq 1$, such that $L \subseteq w_1^* w_2^* \cdots w_n^*$. Combining a characterization of the class of bounded regular languages (Ginsburg and Spanier [14]) with the observations on EC from [18] yields the following:

▶ **Lemma 33.** *Every bounded regular language is an EC-language.*

▶ **Theorem 34.** *Every bounded SpLog-language is an EC-language.*

The intuition behind this is very simple: In bounded SpLog-languages, every constraint can be replaced with a bounded regular language (as this reasoning does not apply to $\mathsf{EC}^{\mathsf{reg}}$, the proof does not generalize). The EC-inexpressibility technique from [18] that we are going to use is based on the following definition by Karhumäki, Plandowski, and Rytter [19]:

▶ **Definition 35.** A word $w \in \Sigma^+$ is *imprimitive* if there exist a $u \in \Sigma^+$ and $n \geq 2$ with $w = u^n$. Otherwise, w is *primitive*. For a given primitive word Q, the $\mathcal{F}_Q$-factorization of $w \in \Sigma^*$ is the factorization $w = w_0 \cdot Q^{x_1} \cdot w_1 \cdots Q^{x_k} \cdot w_k$ that satisfies the following conditions:

1. $Q^2 \not\sqsubseteq w_i$ for all $0 \leq i \leq k$,
2. Q is a proper suffix of w_0, or $w_0 = \varepsilon$,
3. Q is a proper prefix of w_k, or $w_k = \varepsilon$,
4. Q is a proper prefix and a proper suffix of w_i for all $0 < i < k$.

Furthermore, we define $T_Q(w) := \{x \mid Q^x \text{ occurs in the } \mathcal{F}_Q\text{-factorization of } w\}$, as well as $\exp_Q(w) := \max(T_Q(w) \cup \{0\})$.

For every primitive word Q, the $\mathcal{F}_Q$-factorization of every word w (and, hence, $\exp_Q(w)$) is uniquely defined (cf. [18, 19]). We use this definition in the following pumping result:

▶ **Theorem 36** (Karhumäki et al. [18])**.** *For every EC-language L and every primitive word Q, there exists a $k \geq 0$ such that, for each $w \in L$ with $\exp_Q(w) > k$, there is a $u \in L$ with $\exp_Q(u) \leq k$ which is obtained from w by removing some occurrences of Q.*

We now consider a short example of this proof technique: As shown by Fagin et al. [11] (Theorem 4.21), $L_{\text{el}} := \{\mathtt{a}^i\mathtt{b}^i \mid i \geq 0\}$ is not expressible with core spanners (note that L_{el} is also used in [18] as an example application of Theorem 36). The length of this proof is roughly one page. Contrast this to the following: Assume that L_{el} is a SpLog-language. Then L_{el} is an EC-language, due to Theorem 34. Choose the primitive word $Q := \mathtt{a}$. Then there exists a $k \geq 0$ that satisfies Theorem 36. Choose $w := \mathtt{a}^{k+2}\mathtt{b}^{k+2}$, and observe that $\exp_Q(w) = k + 1 > k$ (due to the factorization $w = \varepsilon \cdot \mathtt{a}^{k+1} \cdot \mathtt{a}\mathtt{b}^{k+2}$). Hence there exists a $u = \mathtt{a}^{k+2-j}\mathtt{b}^{k+2}$, $j > 0$, with $u \in L_{\text{el}}$. As $k + 2 - j < k + 2$, this is a contradiction.

From the inexpressibility of L_{el}, Fagin et al. then conclude that the equal length relation $\{(u, v) \mid |u| = |v|\}$ is not selectable with core spanners (Karhumäki et al. [18] and Ilie [16] use the same approach for EC: Show the non-selectability of a relation by proving that a suitable language is not expressible). Using Theorem 36 and 34 we observe:

▶ **Proposition 37.** *For $x, y \in \Sigma^*$, x is a* scattered subword *of y if there exist a $k \geq 1$, $x_1, \ldots, x_k, y_0, \ldots y_k \in \Sigma^*$ with $x = x_1 \cdots x_k$ and $y = y_0(x_1 y_1) \cdots (x_k y_k)$. For every word $w \in \Sigma^*$, its* reversal *w^R is the word that is obtained by reading w from right to left. We define the following binary relations over Σ^*:*

$$R_{\text{scatt}} := \{(u, v) \mid u \text{ is a scattered subword of } v\}, \qquad R_{\text{rev}} := \{(u, v) \mid v = u^R\},$$
$$R_{\text{num}(a)} := \{(u, v) \mid |u|_a = |v|_a\} \text{ for } a \in \Sigma, \qquad R_< := \{(u, v) \mid |u| < |v|\},$$
$$R_{\text{permut}} := \{(u, v) \mid |u|_a = |v|_a \text{ for all } a \in \Sigma\}.$$

Each of R_{scatt}, $R_{\text{num}(a)}$, R_{permut}, R_{rev}, and $R_<$ is not SpLog-selectable.

Sadly, being limited to bounded languages also limits the applicability of this approach. For example, Ilie [16] shows that the language of square-free words over a two letter alphabet (words that contain no subword xx with $x \neq \varepsilon$) is not EC-expressible. Although one could expect that this language is not a SpLog-language, it is easily seen that every bounded subset of this language has to be finite, which means that this technique cannot be applied. Furthermore, the author conjectures that the relation $\{(x, x^n) \mid x \in \Sigma^*, n \geq 1\}$ is not SpLog-selectable, but there is no suitable bounded language that could be used to prove this.

6 Conclusions and Further Directions

As we have seen, SpLog has the same expressive power as the three classes of representations for core spanners that were introduced by Fagin et al. [11], and it is possible to convert between these models in polynomial time. As a result of this, core spanner representations can be converted to SpLog to decide satisfiability and hierarchicality, and SpLog provides a convenient way of defining core spanners, and in particular relations that are selectable by core spanners (see e. g. $\varphi_{\neq}$ in Example 27). Of course, whether one considers SpLog or one of the spanner representations more convenient mostly depends on personal preferences and the task at hand. Independent of one's opinion regarding the practical applications of SpLog, it can be used as a versatile tool for examining core spanners: For example, we used SpLog as intermediary to obtain polynomial time conversions between various subclasses of $\mathsf{VA}^{\mathsf{core}}$.

In addition to this, we defined a pumping lemma for core spanners by connecting SpLog to EC. A promising next step could be extending this to more general inexpressibility techniques that go beyond bounded SpLog languages. While the connection to word equations suggests that this line of research is difficult, one might also expect that at least some of the existing techniques for word equations can be used.

Another set of question where the comparatively simple syntax and semantics of SpLog might help is the relative succinctness of various models. For example, in order to examine the blowup from $\mathsf{VA}^{\mathsf{core}}$ to $\mathsf{RGX}^{\mathsf{core}}$, it suffices to examine the blowup from NFAs to $\mathsf{SpLog}_{\mathsf{rx}}$; and converting $\mathsf{RGX}^{\mathsf{core}}$ to $\mathsf{UCRPQ}^{=}$ has the same blowup as the transformation of SpLog-formulas to DPCNF. (Conjecture: All these blowups are exponential.)

Finally, the conversion of SpLog-formulas to spanner representations preserves many structural properties. Hence, when looking for subclasses of spanners that have certain properties (e. g., more efficient combined complexity of evaluation), the search can start with examining certain fragments of SpLog that correspond to interesting classes of spanners. One direction that seems to be promising as well as challenging is developing a notion of acyclic core spanners, which would need to account for the interplay of join and string equality (as seen in Corollary 23, every spanner representation can be rewritten into a representation that simulates $\bowtie$ with $\times$ and $\zeta^{=}$). This direction might be helped by first defining acyclicity for SpLog-formulas, which in turn could be inspired by the restrictions that are discussed in Reidenbach and Schmid [22].

A more fundamental question is whether $[\![\mathsf{EC}^{\mathsf{reg}}]\!] = [\![\mathsf{SpLog}]\!]$. In addition to our discussion in Section 4.4, a potential approach to this is examining whether every bounded $\mathsf{EC}^{\mathsf{reg}}$-language is an EC-language (as $\mathsf{EC}^{\mathsf{reg}}$ can use arbitrary variables, the reasoning from Theorem 34 does not carry over from SpLog to $\mathsf{EC}^{\mathsf{reg}}$).

Another aspect of SpLog that makes it interesting beyond its connection to core spanners is that it can be understood as the fragment of $\mathsf{EC}^{\mathsf{reg}}$ describes properties of words without using any additional space, as every variable and equation has to be a subword of the main variable (hence, the name "SpLog" can also be interpreted as "subword property logic"). One

effect of this is that evaluation of SpLog has lower upper bounds that evaluation of $\mathsf{EC}^{\mathsf{reg}}$. While we have only defined SpLog with a single main variable, a natural generalization would be allowing multiple main variables (the definition generalizes naturally, and the upper bound for evaluation remains). A potential application of SpLog with two (or more) variables is describing relations for path labels in graph databases.

Acknowledgements. The author thanks Wim Martens for helpful comments and discussions, and the anonymous reviewers for their insightful feedback.

References

1. P. Barceló and P. Muñoz. Graph logics with rational relations: the role of word combinatorics. In *Proc. CSL-LICS 2014*, 2014.
2. J.-C. Birget. Intersection and union of regular languages and state complexity. *Inform. Process. Lett.*, 43(4):185–190, 1992.
3. B. Carle and P. Narendran. On extended regular expressions. In *Proc. LATA 2009*, 2009.
4. L. Ciobanu, V. Diekert, and M. Elder. Solution sets for equations over free groups are EDT0L languages. In *Proc. ICALP 2015*, 2015.
5. Elena Czeizler. The non-parametrizability of the word equation *xyz=zvx*: A short proof. *Theor. Comput. Sci.*, 345(2-3):296–303, 2005.
6. V. Diekert. Makanin's Algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12. Cambridge University Press, 2002.
7. V. Diekert. More than 1700 years of word equations. In *Proc. CAI 2015*, 2015.
8. V. Diekert, A. Jeż, and W. Plandowski. Finding all solutions of equations in free groups and monoids with involution. In *Proc. CSR 2014*, 2014.
9. V. G. Durnev. Undecidability of the positive $\forall\exists^3$-theory of a free semigroup. *Sib. Math. J.*, 36(5):917–929, 1995.
10. A. Ehrenfeucht and G. Rozenberg. A pumping theorem for EDT0L languages. Technical report, Tech. Rep. CU-CS-047-74, University of Colorado, 1974.
11. R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.
12. D. D. Freydenberger and M. Holldack. Document spanners: From expressive power to decision problems. In *Proc. ICDT 2016*, 2016.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, 1979.
14. S. Ginsburg and E. H. Spanier. Bounded regular sets. *Proc. AMS*, 17(5):1043–1049, 1966.
15. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.
16. L. Ilie. Subwords and power-free words are not expressible by word equations. *Fundam. Inform.*, 38(1-2):109–118, 1999.
17. L. Ilie and W. Plandowski. Two-variable word equations. *ITA*, 34(6):467–501, 2000.
18. J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000.
19. J. Karhumäki, W. Plandowski, and W. Rytter. Generalized factorizations of words and their algorithmic properties. *Theor. Comput. Sci.*, 218(1):123–133, 1999.
20. J. Karhumäki and A. Saarela. An analysis and a reproof of Hmelevskii's theorem. In *Proc. DLT 2008*, 2008.
21. G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

22 D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Inform. Comput.*, 239:87–99, 2014.

23 M. L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Inform. Comput.*, 249:1–17, 2016.

Distributed Query Monitoring through Convex Analysis: Towards Composable Safe Zones

Minos Garofalakis[1] and Vasilis Samoladas[2]

1 **Technical University of Crete, Chania, Greece**
`minos@softnet.tuc.gr`
2 **Technical University of Crete, Chania, Greece**
`vsam@softnet.tuc.gr`

Abstract

Continuous tracking of complex data analytics queries over high-speed distributed streams is becoming increasingly important. Query tracking can be reduced to continuous monitoring of a condition over the global stream. Communication-efficient monitoring relies on locally processing stream data at the sites where it is generated, by deriving site-local conditions which collectively guarantee the global condition. Recently proposed geometric techniques offer a generic approach for splitting an arbitrary global condition into local geometric monitoring constraints (known as "Safe Zones"); still, their application to various problem domains has so far been based on heuristics and lacking a principled, compositional methodology. In this paper, we present the first known formal results on the difficult problem of effective Safe Zone (SZ) design for complex query monitoring over distributed streams. Exploiting tools from convex analysis, our approach relies on an algebraic representation of SZs which allows us to: (1) Formally define the notion of a "good" SZ for distributed monitoring problems; and, most importantly, (2) Tackle and solve the important problem of systematically composing SZs for monitored conditions expressed as *Boolean formulas* over simpler conditions (for which SZs are known); furthermore, we prove that, under broad assumptions, the composed SZ is good if the component SZs are good. Our results are, therefore, a first step towards a principled compositional solution to SZ design for distributed query monitoring. Finally, we discuss a number of important applications for our SZ design algorithms, also demonstrating how earlier geometric techniques can be seen as special cases of our framework.

1998 ACM Subject Classification H.2.m [Database Management] Miscellaneous, C.2.4 [Computer-Communication Networks] Distributed Systems

Keywords and phrases Distributed Data Streams, Geometric Method

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.14

1 Introduction

As we are moving from network-centric computing into the era of Internet of Things, large-scale event monitoring applications become ever more important. Such applications rely on *continuous* monitoring queries over the union of local, high-speed data streams. The scale of these applications, as well as power or bandwidth limitations, often impose critical communication contraints that prohibit the centralization of streaming data. Instead, the applications must rely on novel algorithmic paradigms for processing local streams of data *in situ* (i.e., locally at the sites where the data is observed). This obviously raises the problem of effectively decomposing the global monitoring query into "safe" local queries that can be tracked independently at each site while guaranteeing correctness for the global monitoring

operation. Such a decomposition enables truly distributed, *push-based* monitoring, where sites track their local queries and communicate (e.g., with a "coordinator" site) only when some local query constraints are violated. Still, the problem of effectively decomposing a complex (e.g., non-linear) query over the global distributed stream into such safe local constraints can be far from straightforward.

Problem Setup and the Geometric Method. In an abstract setting, our system architecture comprises a collection of k physically-distributed sites, where each site $p \in \{1, \ldots, k\}$ observes updates to the state of its local stream which is represented as a dynamic, high-dimensional vector $\boldsymbol{X}^{(p)} \in \mathbb{R}^m$. (Note that this is the standard model for general data streams used in the streaming algorithms literature, e.g., [26, 12].) The state of the global, distributed stream $\boldsymbol{X}$ is a convex combination of the local states; that is, $\boldsymbol{X} = \sum_p a_p \boldsymbol{X}^{(p)}$, with $a_p \geq 0, \sum_p a_p = 1$. Arbitrary linear combinations, e.g., summation of local frequency distribution vectors, can be captured by simply multiplying the local vectors by constant factors. In applications, these states often comprise of one or more frequency distributions of streams, or (linear) sketches thereof.

Let $F(\boldsymbol{X})$ denote a global query function, e.g., the norm or entropy of a dynamic global distribution, or the inner-product (i.e., equi-join size) of two underlying distributed streams $\boldsymbol{X_1}$, $\boldsymbol{X_2}$ (note that in this case, $\boldsymbol{X} = \boldsymbol{X_1} \oplus \boldsymbol{X_2}$, the concatenation of $\boldsymbol{X_1}, \boldsymbol{X_2}$). A natural global monitoring condition is a *threshold query* $F(\boldsymbol{X}) < T$ (or, $> T$), where T is some constant. Threshold queries can naturally express more complex monitoring tasks, including *approximate function monitoring* [13].

A general approach to tracking threshold queries over distributed streams was pioneered by Sharfman et al.'s *Geometric Method* [32, 21]. For arbitrary $F()$, it is generally impossible to relate the locally-observed values of $F(\boldsymbol{X}^{(p)})$ to the global value $F(\boldsymbol{X})$; thus, their key idea is to employ geometric arguments to monitor the *domain* (rather than the range) of the monitored function $F()$. More formally, given the threshold query $F(\boldsymbol{X}) < T$, define the set $A \triangleq \{\boldsymbol{x} \in \mathbb{R}^m | F(\boldsymbol{x}) < T\} \subseteq \mathbb{R}^m$ as the query's *admissible region*. Clearly, the condition $F(\boldsymbol{X}) < T$ is equivalent to the condition $\boldsymbol{X} \in A$, and this geometric condition in $\mathbb{R}^m$ is the one being monitored — action needs to be taken only when $\boldsymbol{X}$ leaves A.

A key concept in geometric monitoring is that of a *Safe Zone (SZ)*, which is defined as a *convex subset* of the admissible region; that is, a SZ is a convex set Z such that $Z \subseteq A$ [21]. Let $\boldsymbol{X}^{(p)}(t_0)$ be the state of site p at some initial synchronization time t_0, and let $\boldsymbol{X}_0 = \sum_p a_p \boldsymbol{X}^{(p)}(t_0) = \boldsymbol{X}(t_0)$. As updates arrive at any site p, the site maintains its local *drift vector* $\boldsymbol{u}^{(p)}(t) = \boldsymbol{X}^{(p)}(t) - \boldsymbol{X}^{(p)}(t_0) + \boldsymbol{X}_0$. It is trivial to show that, at any time t, the convex combination of the local drift vectors is exactly the state of the global stream at time t; that is, $\sum_p a_p \boldsymbol{u}^{(p)}(t) = \boldsymbol{X}(t)$ [32]. Thus, as long as at every site p, we have $\boldsymbol{u}^{(p)} \in Z$, by convexity of Z we also have $\boldsymbol{X} \in Z$ and, therefore, $\boldsymbol{X} \in A$.

The Safe Zone Design Problem. The Geometric Method has been extended and successfully applied to various monitoring problems in a number of recent papers [5, 15, 13, 20, 23, 27, 29]. A survey of this body of work reveals that a crucial, and non-trivial aspect of the technique is the issue of *Safe Zone Design:* Given a particular admissible region A, and the initial global state $\boldsymbol{X}(t_0) \in A$, define a "good" (convex) safe zone $Z \subseteq A$. As a simple example depicted in Fig. 1, if A is defined by constraint $\|\boldsymbol{X}\| \geq T$, a good SZ Z can be defined by the constraint $\boldsymbol{X} \cdot \boldsymbol{X}(t_0) \geq T\|\boldsymbol{X}(t_0)\|$.

For complex queries, safe zone design is often analytically challenging, and general methodologies are quite helpful. Simple solutions can be obtained using the original "covering spheres" method of Sharfman et al. [32], but the quality of the safe zones and the performance

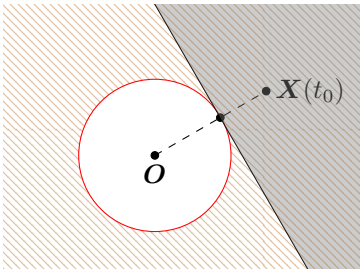


Figure 1 The admissible region (hatched) for $\|\boldsymbol{X}\| \geq T$ and a good safe zone (grayed).

obtained can be far from satisfactory. A more recent work [23] introduces the "convex decomposition" method, which addresses some of the problems of "covering spheres" and provides effective solutions for several problems, but the method is lacking a systematic foundation.

The aforementioned methods, although valuable, suffer from two important drawbacks. First, they do not provide any systematic guidance for designing safe zones of provable quality, making the evaluation of a SZ design a purely experimental task. Second, and more important, they do not provide for composable designs. To clarify our (envisioned) concept of a composable SZ, we draw an analogy to the well-known chain rule for the differential operator: $D\big(f(g(x))\big) = Df(g(x))Dg(x)$. Ideally, we would wish for a "safe zone" transform, that, similar to D, would provide safe zones for complex queries by combining safe zones for simpler queries.

Our Results. In this paper, we present the first compositional method for SZ design, where the composed safe zones inherit quality features from their components. We apply our method to the important problem of inner-product queries (tracking the inner product of two vectors), both exactly and approximately, via AMS sketches.

We formally define the notion of a *good* SZ $Z \subseteq A$, for an admissible region A and a reference point $\boldsymbol{E} \in A$ ($\boldsymbol{E}$ is usually the initial system state $\boldsymbol{X}(t_0)$). A *good* SZ has two "largeness" properties; the *maximum distance* property states that the distance of $\boldsymbol{E} \in Z$ from the boundary of Z is equal to the distance of $\boldsymbol{E}$ from the boundary of A. The *maximality* property states that there does not exist a proper superset of Z which is also a safe zone for A. (Note that the SZ defined above for $\|\boldsymbol{X}\| \geq T$ clearly satisfies both properties.)

Our compositional method is primarily applicable to query functions with separable subcomponents. The global state vector $\boldsymbol{X}$ is taken to be the concatenation of n (not necessarily equidimensional) subvectors: $\boldsymbol{X} = \boldsymbol{X}_1 \oplus \boldsymbol{X}_2 \oplus \cdots \oplus \boldsymbol{X}_n$. For given safe zones on query functions $f_i(\boldsymbol{X}_i)$, we study the design of safe zones for a query function $F(f_1(\boldsymbol{X}_1), \ldots, f_n(\boldsymbol{X}_n))$. Our compositional approach relies on expressing the constraint on F as a conjunction of separable disjunctions. In particular, we present design methods for two important cases of aggregate functions F:

Boolean Functions: Here, f_i are boolean-valued functions, $F : \{0,1\}^n \to \{0,1\}$ is a boolean function, and we monitor the condition "F is true". Given *good* safe zones for subproblems $f_i(\boldsymbol{X}_i)$, our method can compose a *good* safe zone for the overall condition.

An important application for this type of query is threshold monitoring for the median (or, other order statistics); a query of the form $\text{median}\{g_1(\boldsymbol{X}_1), \ldots, g_n(\boldsymbol{X}_n)\} \leq T$ is equivalent to the case where each $f_i(\boldsymbol{X}_i)$ equals the boolean value of condition $g_i(\boldsymbol{X}_i) \leq T$, and F is the majority function. Such order statistics queries arise often in monitoring robust estimators, distributed voting schemes, and so on.

Separable Sums: Here, f_i are real-valued functions and F is summation; that is, we are interested in the condition $f_1(\boldsymbol{X}_1) + \cdots + f_n(\boldsymbol{X}_n) \leq T$. It is easy to show (e.g., by negating the above condition) that the above condition can be written equivalently as a conjunction of separable disjunctions:

$$\forall(\tau_1, \ldots, \tau_n) \in \Sigma_T^n \ : \ \bigvee_{j=1}^{n} f_j(\boldsymbol{X}_j) \leq \tau_j, \tag{1}$$

where $\Sigma_T^n = \{(\tau_1, \ldots, \tau_n) \in \mathbb{R}^n \mid \tau_1 + \cdots + \tau_n = T\}$.
For this more complicated problem, our method's scope is more limited; it can compose a safe zone with the maximum distance property, provided maximum-distance safe zones for queries of the form $f_i(\boldsymbol{X}_i) \leq \tau$ are known, but maximality is harder to obtain in general.

A Motivating Example. The problem of estimating the inner product of two vectors is of fundamental importance for distributed stream processing. This problem abstracts the situation where the vectors capture the (dynamic) frequency distribution of values in two distinct data streams, and we wish to monitor the degree of correlation in the two streams. (Note that this is also equivalent to tracking the size of the equi-join of the two streams [1].)

The global state comprises of a pair of vectors $(\boldsymbol{X}_1, \boldsymbol{X}_2)$, whose inner product, $\boldsymbol{X}_1 \cdot \boldsymbol{X}_2$, we wish to track. Often, the dimension of the raw streaming vectors can be too large for exact tracking to be realistic, since we can only afford to maintain a *synopsis/summary* of the streaming data. This problem has been addressed via the use of AGMS sketch synopses [2, 1]. Succinctly, the AGMS sketch of the frequency vector $\boldsymbol{X}_i, i = 1, 2$ is a sequence of d l-dimensional vectors $\hat{\boldsymbol{X}}_i = (\hat{\boldsymbol{x}}_{i,1}, \ldots, \hat{\boldsymbol{x}}_{i,d})$, with $\hat{\boldsymbol{x}}_{i,j} \in \mathbb{R}^l$. Then, as shown by Alon et al. [1], the inner product $\boldsymbol{X}_1 \cdot \boldsymbol{X}_2$ can be approximated, with an accuracy of $\epsilon = \|\hat{\boldsymbol{X}}_1\|\|\hat{\boldsymbol{X}}_2\|/\sqrt{l}$, with probability at least $1 - O(1/2^d)$, by $F(\hat{\boldsymbol{X}}_1, \hat{\boldsymbol{X}}_2) = \text{median}\{\ \hat{\boldsymbol{x}}_{1,1} \cdot \hat{\boldsymbol{x}}_{2,1}, \ \ldots, \hat{\boldsymbol{x}}_{1,d} \cdot \hat{\boldsymbol{x}}_{2,d}\}$.

In a distributed stream setting, the problem has been addressed in [8, 13]. Both of the above papers rely on a safe zone approach, but these safe zones have not been proven to be either maximum-distance, or maximal. None of these, or any other known techniques, provide guarantees on communication cost. Using our techniques, we design *good* safe zones for both exact and approximate tracking (via AGMS sketches) of the inner product. For exact tracking, the inner product query can be rewritten as a separable sum and the designed safe zone is good (maximum-distance and maximal). Furthermore, the designed safe zone is composed into a design of a *good* safe zone for approximate tracking (which is just tracking the median of d inner products).

2 Safe Zone Design

In this section, we introduce some basic notation and definitions, and give the initial mathematical formulation of safe zone representation and composition. Throughout the paper, we use the notation of vector calculus; boldface letters stand for vectors. All vector spaces in this paper are Euclidean. We use the letter V to denote a vector space $\mathbb{R}^d$, equipped with the standard inner product. Vector inner product is written as $\boldsymbol{xy}$ and self-product is written as squaring, therefore $\boldsymbol{x}^2 = \|\boldsymbol{x}\|^2$. We also use $\boldsymbol{x} \oplus \boldsymbol{y}$ to denote the so-called *direct sum* of two vectors, that is, the vector resulting from the concatenation of $\boldsymbol{x}$ and $\boldsymbol{y}$.

In addition, given a Boolean predicate $\phi : V \to \{0, 1\}$, we write $\{\phi\} = \{\boldsymbol{X} \in V \mid \phi(\boldsymbol{X})\}$.

We will need some simple topological concepts. Assume some vector space $V = \mathbb{R}^d$. Let $\text{Ball}(\boldsymbol{c}, \rho)$ denote the open ball centered at c with radius ρ, i.e., $\text{Ball}(\boldsymbol{c}, \rho) = \{\boldsymbol{x} \in$

$V \mid \|\boldsymbol{x} - \boldsymbol{c}\| < \rho\}$. Given a set of points $A \subseteq V$ and $\boldsymbol{x} \in A$, we say that $\boldsymbol{x}$ is *interior* to A (denoted $\boldsymbol{x} \in \mathbf{int}\, A$) iff there exists some $\epsilon > 0$, so that $\text{Ball}(\boldsymbol{x}, \epsilon) \subseteq A$. A point $\boldsymbol{x} \in V$ is *exterior* to A (denoted $\boldsymbol{x} \in \mathbf{ext}\, A$) iff it is interior to the complement of A, $\overline{A} = V - A$. A point $\boldsymbol{x} \in V$ which is neither exterior nor interior to A is a *boundary* point for A (denoted $\boldsymbol{x} \in \mathbf{bd}\, A$). Set A is *open* iff it only contains interior points; that is, $A = \mathbf{int}\, A$. Dually, set A is closed iff its complement $\overline{A}$ is open; in this case, $A = \mathbf{int}\, A \cup \mathbf{bd}\, A$.

For completeness, we also state a few basic facts about convex sets. A set $Z \subseteq V$ is convex iff $\forall \boldsymbol{x}, \boldsymbol{y} \in Z$, $\forall t \in [0,1]$, $(1-t)\boldsymbol{x} + t\boldsymbol{y} \in Z$. Alternatively, Z is convex iff it is closed under convex combinations of its elements. An important property of convex sets is that the intersection of any collection of convex sets is convex.

A (closed) halfspace h is a *supporting halfspace* of convex set Z, iff (a) $h \supseteq Z$, and (b) the boundaries of h and Z intersect in at least one point $\boldsymbol{p}$. We say that h supports Z at $\boldsymbol{p}$. At every boundary point $\boldsymbol{p} \in \mathbf{bd}\, Z$, there is at least one halfspace h supporting Z at $\boldsymbol{p}$. Where there is exactly one, $\boldsymbol{p}$ is called *smooth* and h is called *tangent*. A well-known theorem states that any convex set Z is the intersection of its supporting halfspaces. In this paper, we shall employ a stronger, but much less known theorem:

▶ **Theorem 1** (Rockafellar [28], Thm. 18.8)**.** *A closed convex set $Z \subseteq V$ is the intersection of the closed halfspaces tangent to it.*

A function $f : V \to \mathbb{R}$ is convex iff, for every $\boldsymbol{x}, \boldsymbol{y} \in V$ and $\lambda \in [0,1]$, it is $f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$. A function f is concave iff $-f$ is convex. We will be interested primarily in concave functions. A function f is both concave and convex, iff it is *affine*, that is, $f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + a$, for some $\boldsymbol{w} \in V$ and $a \in \mathbb{R}$.

2.1 Safe Zone Specification

Consider a monitored query on V, which corresponds to an admissible region $A \subseteq V$. The problem of determining a good safe zone Z requires additional information; some *reference* must be made by the user, as to the preferred locus of the safe zone, among mutually exclusive alternatives. Motivated by previous work, we adopt the concept of a reference point $\boldsymbol{E} \in \mathbf{int}\, A$, which our safe zone must include.

Therefore, the safe zone design problem can be stated as follows: given admissible region $A \subseteq V$ and reference point $\boldsymbol{E} \in \mathbf{int}\, A$, select a safe zone $Z \subseteq A$ such that Z is convex and $\boldsymbol{E} \in \mathbf{int}\, Z$. Towards a compositional approach, we assume that our admissible region A is expressible as a set-algebraic combination (involving intersection and union) of subsets of V.

Now, consider a family of sets $A_i \subseteq V$, $i \in I$, where $A = \cap_{i \in I} A_i$, with $\boldsymbol{E} \in \mathbf{int}\, A_i$ for all $i \in I$, and let $Z_i \subseteq A_i$ be safe zones. Towards a compositional approach, it is natural to consider $Z = \cap_{i \in I} Z_i$ as a candidate safe zone for admissible region $A = \cap_{i \in I} A_i$. Indeed, Z is convex and contains $\boldsymbol{E}$.

Similarly, if $A = \cup_{i \in I} A_i$, with $\boldsymbol{E} \in \mathbf{int}\, A_i$ for some $i \in I$ (note that $\boldsymbol{E}$ need not belong to all A_i), it is natural to consider $\cup_{i \in I} Z_i$ as a candidate safe zone for A. Unfortunately, this is not valid, as $\cup_i Z_i$ is *not* convex in general. Therefore, we need to restrict to a convex subset $Z \subseteq \cup_{i \in I} Z_i$, that contains $\boldsymbol{E}$; moreover, Z should inherit good qualities of zones Z_i. To overcome the difficulties of handling unions, our method focuses on the special case of decomposition into a *separable union*. In this case, $V = \mathbb{R}^d$ is the product space $V_1 \times \cdots \times V_n$ of a collection $V_i = \mathbb{R}^{d_i}, i = 1, \ldots, n$ of vector spaces (with $\sum d_i = d$), and each $\boldsymbol{x} \in V$ is the direct sum of $\boldsymbol{x}_i \in V_i$, where each $\boldsymbol{x}_i$ is the *projection* of $\boldsymbol{x}$ on V_i. For $A_i \subseteq V_i$, the separable union $\bigvee_{i=1}^n A_i$ is simply the set $\{\boldsymbol{x}_1 \oplus \cdots \oplus \boldsymbol{x}_n \in V \mid \bigvee_{i=1}^n \boldsymbol{x}_i \in A_i\}$. Equivalently, for $n = 2$,

$A_1 \vee A_2 = (A_1 \times V_2) \cup (V_1 \times A_2)$. Separable intersection $\bigwedge_{i=1}^n A_i$, defined similarly, is just the Cartesian product $A_1 \times \cdots \times A_n$.

Given safe zones $Z_i \subseteq A_i \subseteq V_i$, the separable union $\bigvee_{i=1}^n Z_i$ is not a convex set. However, it will be shown subsequently that properly selected subsets of the separable union do inherit the good qualities of the safe zones Z_i.

2.2 Safe Zone Representation and Composition

In our method, a convex set is represented as the level set of a concave function.

▶ **Definition 2** (Level set)**.** Let $f : V \to \mathbb{R}$ be any function. The level set $L(f)$ of f is the set $L(f) = \{\boldsymbol{x} \in V \mid f(\boldsymbol{x}) \geq 0\} = \{f(\boldsymbol{x}) \geq 0\}$.

When f is concave, $L(f)$ is convex. We wish to avoid the degenerate cases where $L(f)$ is empty, or the whole space, or has empty interior. This is equivalent to the following requirement:

▶ **Definition 3** (Safe Zone function)**.** A safe zone function is a concave function $f : V \to \mathbb{R}$, which attains both a positive and a negative value over V.

Safe zone functions are positive in the interior of $L(f)$, negative on the exterior and vanish on the boundary. An important property of $L(f)$ is monotonicity with respect to pointwise dominance. Given functions $f, g : V \to \mathbb{R}$, f is dominated by g (denoted by $f \leq g$) iff $\forall \boldsymbol{x} \in V, f(\boldsymbol{x}) \leq g(\boldsymbol{x})$. Clearly, $f \leq g$ directly implies $L(f) \subseteq L(g)$.

We are interested in the compositions of safe zones for intersections and unions of safe zones. We introduce two operations that construct the composite safe zone function from component safe zone functions. The first operation, used to capture the intersection of safe zones, is the pointwise-infimum of a (possibly infinite) family of safe zone functions. Given family of safe zone functions $\zeta_i : V \to \mathbb{R}$, $i \in I$, let $\zeta(\boldsymbol{x}) = \inf_{i \in I} \zeta_i(\boldsymbol{x})$. Then, $L(\zeta) = \bigcap_{i \in I} L(\zeta_i)$. The second operation is weighted sum with non-negative weights, sometimes called conical combination. It is used to construct a convex subset of the union of a finite family of safe zones.

▶ **Theorem 4.** *For safe zone functions $\zeta_i : V \to \mathbb{R}$, $i = 1, \ldots, n$, and reals $a_i \geq 0$, not all zero, let*

$$\zeta(\boldsymbol{X}) = \sum_{i=1}^{n} a_i \zeta_i(\boldsymbol{X}). \tag{2}$$

Then, $L(\zeta)$ is convex and $\bigcap_{i=1}^n L(\zeta_i) \subseteq L(\zeta) \subseteq \bigcup_{i=1}^n L(\zeta_i)$.

Proof. It is $\bigcap_{i=1}^n L(\zeta_i) \subseteq L(\inf_i a_i \zeta_i(\boldsymbol{X}))$ and $L(\sup_i a_i \zeta_i(\boldsymbol{X})) \subseteq \bigcup_{i=1}^n L(\zeta_i)$ (with equality holding when all $a_i > 0$). Since $n \inf_i a_i \zeta_i(\boldsymbol{X}) \leq \zeta(\boldsymbol{X}) \leq n \sup_i a_i \zeta_i(\boldsymbol{X})$, the theorem follows directly from these formulas and monotonicity of L. ◀

The above theorem specializes to separable union straightforwardly. If $\zeta_i : V_i \to \mathbb{R}$ are safe zone functions, then $\zeta(\boldsymbol{x}_1 \oplus \cdots \oplus \boldsymbol{x}_n) = \sum_{i=1}^n a_i \zeta_i(\boldsymbol{x}_i)$ is a safe zone function and $\times_{i=1}^n L(\zeta_i) \subseteq L(\zeta) \subseteq \bigvee_{i=1}^n L(\zeta_i)$.

2.3 Functional Analysis For Safe Zone Functions

Having defined the two fundamental operations on safe zone functions, in the following sections we proceed to study the conditions under which these operations maintain the qualities of composed safe zones. We end this section with some foundational facts from convex analysis.

All safe zone functions over V are continuous and differentiable almost everywhere in V (that is, everywhere, except for a set of measure 0). The gradient $\nabla\zeta(\boldsymbol{x}) = (\frac{\partial\zeta}{\partial x_1}, \ldots, \frac{\partial\zeta}{\partial x_d})$ is the multi-dimensional analog of the derivative. At any point $\boldsymbol{x}$, where ζ is differentiable, this derivative is a vector $\boldsymbol{v_x}$, pointing to the direction of maximum increase of ζ, and its norm $\|\boldsymbol{v_x}\|$ is proportional to the rate of change.

Let a safe zone function ζ be differentiable at point $\boldsymbol{x_0}$. The affine function $h(\boldsymbol{x}) = \nabla\zeta(\boldsymbol{x_0})(\boldsymbol{x} - \boldsymbol{x_0}) + \zeta(\boldsymbol{x_0})$ is called the *tangent* of ζ at $\boldsymbol{x_0}$. By virtue of concavity, $\zeta \leq h$.

In general, every safe zone function ζ is the pointwise infimum of some (non-unique) family $\mathcal{H}$ of affine functions. This is denoted by $\zeta = \inf \mathcal{H}$. In particular, ζ is the pointwise infimum of all its tangents.

Given a collection ζ_i, $i \in I$, of safe zone functions, and corresponding families of affine functions $\mathcal{H}_i$, such that $\zeta_i = \inf \mathcal{H}_i$, the following two properties are very important in the rest of this paper:

1. For arbitrary I, $\inf_{i\in I} \zeta_i = \inf(\bigcup_{i\in I} \mathcal{H}_i)$, and
2. For I finite, and $a_i \geq 0$, not all 0, it is $\sum_{i\in I} a_i\zeta_i = \inf(\sum_{i\in I} a_i\mathcal{H}_i)$, where $\sum_{i\in I} a_i\mathcal{H}_i$ is the set of affine functions $\{\sum_{i\in I} a_ih_i \mid h_i \in \mathcal{H}_i, i \in I\}$.

3 Maximum Distance

Given an admissible region $A \subseteq V$ and reference point $\boldsymbol{E} \in \mathbf{int}\, A$, several safe zones containing $\boldsymbol{E}$ can be constructed. Given two such safe zones, Z and Z', one can argue that one is "better" than the other, by assuming isotropy; that is, all directions *around* the reference point are equally desirable. This assumption correlates with a monitoring situation where each coordinate of the state vector behaves (or is assumed to behave) as an IID random walk. Under this assumption, we have the following criterion:

▶ **Criterion 5.** *Safe zone Z is "better" than safe zone Z' if* $\operatorname{dist}(\boldsymbol{E}, \overline{Z}) \geq \operatorname{dist}(\boldsymbol{E}, \overline{Z'})$.

With respect to criterion 5, any safe zone containing a ball that touches the admissible region's boundary is best possible.

▶ **Definition 6** (Maximum distance)**.** Let A be an admissible region and $\boldsymbol{E} \in \mathbf{int}\, A$. A safe zone $Z \subseteq A$ has maximum distance in A with respect to $\boldsymbol{E}$, iff $\operatorname{dist}(\boldsymbol{E}, \overline{Z}) = \operatorname{dist}(\boldsymbol{E}, \overline{A})$.

3.1 Preservation Of Maximum Distance Under Composition

The intersection operation always preserves the maximum distance of its operands. Given safe zones $Z_i \subseteq A_i$, and $\boldsymbol{E} \in \mathbf{int}\, A_i$, for each i, if all Z_i are maximum distance, then $\bigcap_{i\in I} Z_i$ is also maximum distance. As a matter of fact, if $D = \operatorname{dist}(\boldsymbol{E}, \overline{A}) = \inf_{i\in I} \operatorname{dist}(\boldsymbol{E}, \overline{A_i})$, it is sufficient and necessary to have $\operatorname{dist}(\boldsymbol{E}, \overline{Z_i}) \geq D$. Consequently, the pointwise-inf operation on any family of safe zone functions preserves the maximum distance property of its operands.

The situation with separable union is much more involved. It is not sufficient for the safe zones of the operands to be maximum distance; the actual safe zone functions that describe these safe zones must carry sufficient distance information, so that some conical combination

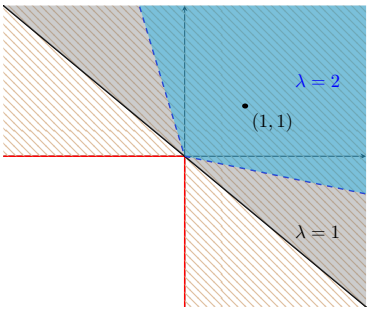


Figure 2 Example of the safe zone of union $\{x_1 \geq 0\} \vee \{x_2 \geq 0\}$, derived by with suboptimal functions. The safe zone for $\lambda = 1$, whose boundary is the solid black line, is good for reference point $(1, 1)$. The safe zone for $\lambda = 2$, whose boundary is the dashed blue line, is neither maximum-distance nor maximal.

can yield a maximum-distance subset of the union. This is best illustrated by example. With reference to Fig. 2, consider the case in $\mathbb{R}^2$, where $Z_1 = L(f(x_1))$ and $Z_2 = L(f(x_2))$, with

$$f(x) = \begin{cases} x/\lambda & \text{if } x \geq 0 \\ \lambda x & \text{if } x < 0 \end{cases} \tag{3}$$

where $\lambda \geq 1$ (so that f is concave). Note that, independently of the value of λ, $Z_i = \{x_i \geq 0\}$. The union $Z_1 \vee Z_2$ is not convex, but maximal convex subsets are all halfspaces whose boundary supports the positive quadrant $(Z_1 \times Z_2)$. Yet, none of the sets $L(a_1 f(x_1) + a_2 f(x_2))$ have maximum distance for any reference point in $Z_1 \times Z_2$, unless $\lambda = 1$. In fact, as λ grows, safe zones $L(a_1 f(x_1) + a_2 f(x_2))$ shrink towards $Z_1 \cap Z_2$, which is their lower bound.

The intuitive reason of the failure in this example is that, the shape of the region generated by Eq. (2) depends crucially on the values of f outside of $L(f)$.

To ameliorate this situation, we introduce a class of safe zone functions which contain sufficient distance information.

▶ **Definition 7** (Affine Distance Function (ADF))**.** An affine function $h : V \to \mathbb{R}$ with $h(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + a$ is an ADF iff $\|\nabla h\| = \|\boldsymbol{w}\| = 1$.

Every closed halfspace of V is equal to $L(h)$ for a unique ADF h. For each $\boldsymbol{x} \in V$, $h(\boldsymbol{x})$ is the signed distance of $\boldsymbol{x}$ from the boundary of $L(h)$, non-negative if $\boldsymbol{x} \in L(h)$ and negative if $\boldsymbol{x} \notin L(h)$.

▶ **Definition 8** (Eikonal function)**.** A safe zone function ζ is eikonal iff it is the pointwise-infimum of a collection of ADFs.

A useful alternative characterization of eikonal functions is the following:

▶ **Theorem 9** (Eikonal characterization)**.** *Let $\zeta : V \to \mathbb{R}$ be concave, and almost everywhere differentiable. Then, ζ is eikonal, if and only if, $\|\nabla\zeta\| = 1$ at every point where it is differentiable.*

(Due to space constraints, this and other ommitted proofs will appear in the full version of the paper.)

The equation $\|\nabla\zeta\| = 1$ is known as the (Euclidean) *eikonal* differential equation. As a consequence of this equation, it can be shown, using the mean-value theorem of analysis, that every eikonal function ζ is non-expansive: $|\zeta(\boldsymbol{x}) - \zeta(\boldsymbol{y})| \leq \|\boldsymbol{x} - \boldsymbol{y}\|$.

Signed Distance Functions. One important member in the family of eikonal functions is the Signed Distance Function (SDF) of a convex set Z.

$$\delta_Z(\boldsymbol{x}) = \begin{cases} \operatorname{dist}(\boldsymbol{x}, \overline{Z}) & \text{if } \boldsymbol{x} \in Z, \\ -\operatorname{dist}(\boldsymbol{x}, Z) & \text{if } \boldsymbol{x} \in \overline{Z}. \end{cases} \tag{4}$$

The SDF of a convex set is concave. Also, since $L(\delta_Z) = Z$, the SDF is positive in the interior of Z, negative on the exterior, and vanishes on the boundary; therefore, it is a safe zone function. Naturally, every ADF h is the SDF of $L(h)$. However, this is not the case for every eikonal function ζ.

Let $Z = L(\zeta)$ and let δ_Z be the SDF of $L(Z)$. Then, $\delta_Z \le \zeta$. In fact, for every $\boldsymbol{x} \in L(f)$, $\zeta(\boldsymbol{x}) = \delta_Z(\boldsymbol{x})$. But, ζ may strictly dominate δ_Z outside $L(\zeta)$.

The family of eikonal functions is well-behaved under our composition operators. The pointwise-infimum of any family of eikonal functions is also eikonal. For the case of separable union, we have the following:

▶ **Theorem 10.** *Let $\zeta_i : V_i \to \mathbb{R}$ be eikonal functions, $a_i \ge 0$, and $\zeta(\boldsymbol{x}_1 \oplus \cdots \oplus \boldsymbol{x}_n) = \sum_{i=1}^n a_i \zeta_i(\boldsymbol{x}_i)$. Then, ζ is eikonal iff $\sum_{i=1}^n a_i^2 = 1$.*

Proof. Since $\nabla\zeta_i \in V_i$ are orthogonal, $\|\nabla\zeta\|^2 = \left(\sum_{i=1}^k a_i \nabla\zeta_i\right)^2 = \sum_{i=1}^k a_i^2 \|\nabla\zeta_i\|^2 = \sum_{i=1}^k a_i^2$. ◀

Thus, any separable conical combination ζ of eikonal functions can always be scaled to an eikonal function, by dividing it by $\sqrt{\sum_i a_i^2}$, which of course does not affect the described safe zone $L(\zeta)$.

We now turn our attention to separable union. Let admissible region $A = \bigvee_{i=1}^n A_i$ and let $\boldsymbol{E} = \boldsymbol{E_1} \oplus \cdots \oplus \boldsymbol{E}_n$ be the reference point. Consider eikonal functions ζ_i such that $L(\zeta_i) \subseteq A_i$, and let $Z = \bigvee_{i=1}^n L(\zeta_i)$. What is the radius D of the largest ball centered at $\boldsymbol{E}$, that can be attained by a conical combination of ζ_i? Clearly, $D \le d_Z = \operatorname{dist}(\boldsymbol{E}, \overline{Z})$. In general, it is $d_Z \le d_A = \operatorname{dist}(\boldsymbol{E}, \overline{A})$. The following theorem specifies the conditions under which maximum distance can be achieved.

▶ **Theorem 11.** *Let $A = \bigvee_{i=1}^n A_i$, where $A_i \subseteq V_i$, and $\boldsymbol{E} = \boldsymbol{E_1} \oplus \cdots \oplus \boldsymbol{E}_n \in \mathbf{int}\, A$. Let $\zeta_i : V_i \to \mathbb{R}$ be eikonal functions, such that $L(\zeta_i) \subseteq A_i$, and let $Z = \bigvee_{i=1}^n L(\zeta_i)$. Then:*

1. *For any conical combination $\zeta = \sum a_i \zeta_i$, it is $\operatorname{dist}(\boldsymbol{E}, \overline{L(\zeta)}) = \dfrac{\sum_{i=1}^n a_i \zeta(\boldsymbol{E}_i)}{\sqrt{\sum_{i=1}^n a_i^2}} = D$.*
2. *It is $D \le \operatorname{dist}(\boldsymbol{E}, \overline{Z}) = d_Z$, with equality holding iff, for some $\lambda > 0$,*

$$a_i = \begin{cases} \lambda \zeta_i(\boldsymbol{E}_i) & \text{if } \zeta_i(\boldsymbol{E}_i) > 0, \\ 0 & \text{otherwise,} \end{cases}$$

3. *It is $d_Z \le d_A = \operatorname{dist}(\boldsymbol{E}, \overline{A})$, with equality holding iff, for every i such that $\boldsymbol{E}_i \in \mathbf{int}\, A_i$, $L(\zeta_i)$ has maximum distance in A_i w.r.t. $\boldsymbol{E}_i$.*

4 Maximality

During distributed monitoring using a safe zone $Z \subseteq A$, condition $\boldsymbol{X} \in Z$ may be violated, while $\boldsymbol{X} \in A$. We call such local violations, *false violations*. Typically, the performance of distributed monitoring depends crucially on minimizing false violations. Therefore,

▶ **Criterion 12.** *Safe zone Z is "better" that safe zone Z' if $Z \supseteq Z'$*

This criterion is a rather obviously desirable; providing a larger safe zone will tend, other things being equal, to reduce false violation.

With respect to criterion 12, the best possible for a safe zone is to be a $\subseteq$-wise maximal convex subset of the admissible region A.

▶ **Definition 13** (Maximality)**.** Let A be the admissible region. A safe zone Z is maximal in A (with respect to set containment), if and only if, no convex subset of A is a proper superset of Z.

We now develop a convenient characterization of maximality.

▶ **Definition 14** (Flats of an affine family)**.** Given a family $\mathcal{H}$ of affine functions on V, let $\Phi_{\mathcal{H}} : \mathcal{H} \to 2^V$ be

$$\Phi_{\mathcal{H}}(h) = \{\boldsymbol{p} \in V \mid h(\boldsymbol{p}) = 0 \text{ and } \forall h' \in \mathcal{H}, h' \neq h \implies h'(\boldsymbol{p}) > 0\} \tag{5}$$

Set $\Phi_{\mathcal{H}}(h)$ is the flat of h in $\mathcal{H}$.

Fix some affine family $\mathcal{H}$ and let $\zeta = \inf \mathcal{H}$ and $Z = L(\zeta)$. Let $h \in \mathcal{H}$ be an affine function with non-empty flat $\Phi(h)$. Each $\boldsymbol{p} \in \Phi(h)$ is a boundary point of Z, since $\zeta(\boldsymbol{p}) = 0$. Also, $\boldsymbol{p}$ is smooth, since ζ is differentiable at $\boldsymbol{p}$ (with $\nabla\zeta(\boldsymbol{p}) = \nabla h$). Therefore, h is a tangent halfspace to Z.

Some examples of flats; a ball has a flat for each point on its boundary. A planar triangle has three flats, each corresponding to a side minus the corners (which are not smooth). A cylinder in 3-d has two 2-d flats, its top and bottom (minus the edges) and infinite 1-d flats which are segments from top to bottom (minus the endpoints). Finally, the positive quadrant in $\mathbb{R}^2$ has two flats, the rays $(0, +\infty) \times \{0\}$ and $\{0\} \times (0, +\infty)$.

▶ **Definition 15** (Non-redundant affine family)**.** A family $\mathcal{H}$ of affine functions is non-redundant iff, for every $h \in \mathcal{H}$, $\Phi_{\mathcal{H}}(h) \neq \emptyset$.

According to the above, a non-redundant affine family $\mathcal{H}$ contains only tangent halfspaces (represented as affine functions) of $Z = L(\inf \mathcal{H})$. However, not all tangents of Z need be contained in $\mathcal{H}$. As an example, consider the planar unit disk $Z = \{x_1^2 + x_2^2 \leq 1\}$; although there is a tangent at every point of the unit circle, only a countable family of tangent halfspaces (say, those whose slope is rational) is enough to define it!

▶ **Definition 16** (Witness)**.** For admissible region $A \subseteq V$ and an affine family $\mathcal{H}$, so that $L(\inf \mathcal{H}) \subseteq A$, a point $\boldsymbol{p} \in \mathbf{bd}\, A$ is a witness iff, for some $h \in \mathcal{H}$, $\boldsymbol{p} \in \Phi_{\mathcal{H}}(h)$.

▶ **Theorem 17** (Witnessed maximality criterion)**.** *For admissible region $A \subseteq V$ and affine family $\mathcal{H}$, so that $Z = L(\inf \mathcal{H}) \subseteq A$, if every flat of $\mathcal{H}$ contains a witness, then Z is maximal in A.*

Witnessed maximality is sufficient for maximality, but not necessary. As an example, consider the 2-d case where $Z = \{y \leq 0\}$ and $A = \{y \leq 1/|x| \vee x = 0\}$. Then, Z is maximal in A, but there is no witness to the unique tangent halfspace that is the whole of Z.

The witnessed maximality criterion is handy, because it relates maximality of a set Z to a requirement on each flat of a description of Z. It is possible, but cumbersome, to extend the concept of a witness, so that a sufficient *and* necessary condition for maximality can be obtained. Because of space constraints, this extension will be presented in the full paper.

4.1 Preservation Of Maximality Under Composition

In contrast to maximum distance, intersection does not preserve maximality in general. This is quite well-known; in fact, loss of maximality under intersection is the reason for the unsatisfactory behavior of previous safe zone design approaches, such as the Covering Spheres method.

Fortunately, under suitable conditions to be explored below, separable union does preserve maximality; given maximal safe zones $L(\zeta_i) \subseteq A_i$, it is possible to select a maximal convex subset of $\bigvee L(\zeta_i)$ which is also maximal in $\bigvee A_i$. However, as in the case for maximum distance, the safe zone functions ζ_i must a special requirement, *non-redundancy*.

▶ **Definition 18** (Non-redundant safe zone function)**.** A safe zone function ζ is non-redundant iff ζ is the pointwise infimum of a non-redundant affine family; else it is redundant.

Intuitively, a non-redundant safe zone function ζ for $Z = L(\zeta)$ is one which is (pointwise) maximal, among all safe zone functions g with $L(g) = Z$; that is, if $L(f) = L(g)$ and $f < g$ (that is, $f \leq g$ and at some $\boldsymbol{x}_0$, $f(\boldsymbol{x}_0) < g(\boldsymbol{x}_0)$), then f is redundant.

An important observation pertains to eikonal non-redundant functions. Given any convex set Z, the family $\mathcal{F}$ of eikonal functions f with $L(f) = Z$ is known to contain a $\leq$-wise least element, the SDF of Z. It can be shown that it also contains a unique $\leq$-wise greatest element η_Z, which is the (unique) non-redundant function in $\mathcal{F}$. Importantly, when Z is smooth (all points of its boundary are smooth), then $\eta_Z = \delta_Z$.

We now proceed to the main result of this section, which determines maximality of safe zones for an admissible region composed as an intersection of unions. To describe the intersection of unions, we use an *antichain* on $[n] = \{1, \ldots, n\}$, i.e., a non-empty collection C of subsets of $[n]$, such that no element of C is a subset of another. A single separable union is specified $C = \{[n]\}$. A (separable) intersection on the other hand is specified as $C = \big\{\{1\}, \ldots, \{n\}\big\}$.

▶ **Theorem 19.** *Let $\zeta_i : V_i \to \mathbb{R}$, $i = 1, \ldots, n$ be non-redundant safe zone functions, and C an antichain on $[n]$. Let $\zeta = \inf_{\Gamma \in C} \sum_{i \in \Gamma} a_i(\Gamma)\zeta_i$, where $a_i(\Gamma) > 0$. Then,*

1. *ζ is non-redundant, and*
2. *if, for $A_i \subseteq V_i$, each $L(\zeta_i)$ is witnessed maximal in A_i, then $L(\zeta)$ is witnessed maximal in $A = \bigcap_{\Gamma \in C} \bigvee_{i \in \Gamma} A_i$.*

5 Safe Zones For Boolean Functions

We now apply our method to the class of (separable) boolean query functions. Let $F : \{0,1\}^n \to \{0,1\}$ be any boolean function and let $f_i : V_i \to \{0,1\}$ be predicates on V_i respectively. We are interested in a safe zone for admissible region $A = \{\boldsymbol{X} \in V \mid F(f_1(\boldsymbol{X}_1), \ldots, f_n(\boldsymbol{X}_n))\}$, containing the reference point $\boldsymbol{E} = \boldsymbol{E}_1 \oplus \cdots \oplus \boldsymbol{E}_n$, with $\boldsymbol{E} \in \mathbf{int}\, A$.

We assume that F is given to us as a conjunction of clauses, where each clause is a disjunction of literals $b_i, \overline{b_i}$, for $i = 1, \ldots, n$. A clause Γ can be represented as a subset of $\{1, \ldots, n\} \times \{+, -\}$, where pair $(j, s) \in \Gamma$ means that the clause contains b_j if $s = +$, or $\overline{b_j}$ if $s = -$.

Letting, $A_i^{(+)} = \{f_i(\boldsymbol{X}_i)\}$ and $A_i^{(-)} = V_i - A_i^{(+)}$, the admissible region A can be decomposed as $A = \bigcap_{\Gamma \in F} \bigvee_{(i,s) \in \Gamma} A_i^{(s)}$. By the observations for maximum distance, we are allowed to *reduce* F to a (stronger) boolean function $\tilde{F}$. In particular, for each $i = 1, \ldots, n$, we eliminate at least one of the literals $b_i, \overline{b_i}$ from all clauses: if $\boldsymbol{E}_i \notin \mathbf{int}\, A_i^{(-)}$, we eliminate

literal $\overline{b_i}$, and if $\boldsymbol{E}_i \notin \mathbf{int}\, A_i^{(+)}$ we eliminate b_i. This justified is because, by Thm. 11, a maximum-distance safe zone for each clause would eliminate the corresponding components. Once $\tilde{F}$ is obtained, it can be further reduced by expressing it as a conjunction of its prime implicates, so that no clause is weaker than another. This last step is needed in order to apply Thm. 19.

Then, given safe zone functions ζ_i for the remaining admissible regions on V_i, the safe zone function

$$\zeta(\boldsymbol{X}_1 \oplus \ldots \oplus \boldsymbol{X}_n) = \inf_{\Gamma \in \tilde{F}} \frac{\sum_{(i,s)\in\Gamma} \zeta_i(\boldsymbol{E}_i)\zeta_i(\boldsymbol{X}_i)}{\sqrt{\sum_{(i,s)\in\Gamma} \zeta_i^2(\boldsymbol{E}_i)}} \tag{6}$$

defines a safe zone for A. Furthermore, if, for all i, ζ_i are eikonal and $L(\zeta_i)$ are maximum distance, then $L(\zeta)$ is maximum distance and ζ is eikonal. With respect to maximality, by virtue of Thm. 19, if all ζ_i are non-redundant, and $L(\zeta_i)$ are (witnessed) maximal in A_i, then $L(\zeta)$ is also (witnessed) maximal in A.

5.1 Monitoring Quantiles

Consider the query function $Q_k(g_1(\boldsymbol{X}_1), \ldots, g_n(\boldsymbol{X}_n))$, where $Q_k : \mathbb{R}^n \to \mathbb{R}$ returns the k-th least value among its arguments. These queries arise in monitoring robust statistics of (functions of) the state, such as the median or the inter-quartile range.

Condition $Q_k() \leq T$ can be written as a boolean function $F_k(f_1(\boldsymbol{X}_1), \ldots, f_n(\boldsymbol{X}_n))$, where $f_i(\boldsymbol{X}_i)$ is the boolean value of "$g_i(\boldsymbol{X}_i) \leq T$" and F_k is true iff k or more of its inputs are true. With respect to F_k, the safe zone design of this section yields a safe zone function ζ, given safe zone functions ζ_i for each constraint $g_i(\boldsymbol{X}_i) \leq T$.

A practical point is related to the computational cost of checking membership in $L(\zeta)$, which, if done straightforwardly, requires time $O(2^n)$. For counting queries, it is possible to test condition $\zeta(\boldsymbol{X}) \geq 0$ in time $O(n)$. To see this, note that the set of clauses of $F_k(b_1, \ldots, b_n)$ is the set of all $n-k+1$-subsets of $\{b_1, \ldots, b_n\}$ (since, if every $n-k+1$-subset of literals contains a true literal, there are at most $n-k$ false literals overall, and therefore at least k true literals). Therefore, to check $\zeta(\boldsymbol{X}) \geq 0$, it is sufficient to compute the sum S of the least $n-k+1$ elements of $\{\zeta_i(\boldsymbol{E}_i)\zeta_i(\boldsymbol{X}_i) \mid i = 1, \ldots, n\}$, as it is $\zeta(\boldsymbol{X}) \geq 0$ iff $S \geq 0$.

6 Safe Zones For Separable Sums

Separable sum queries refer to conditions of the form $\sum_{i=1}^n f_i(\boldsymbol{X}_i) \leq T$, where f_i are arbitrary real functions. As shown in Eq. 1, this condition can be written as a universal quantification (which relates to conjunction) of a family of finite disjunctions. Therefore, at least in principle, our composition operators can be used to derive a safe zone formula. In this section, we demonstrate that this approach can go well beyond the principle, into an analytic method of safe zone design.

The approach is straightforward; as shown in Eq. 1, the separable sum threshold condition is rewritten as $\forall \boldsymbol{\tau} \in \Sigma_T^n$, $\bigvee_{i=1}^n f_i(\boldsymbol{X}_i) \leq \tau_i$. Let the reference point be $\boldsymbol{E} = \boldsymbol{E}_1 \oplus \cdots \oplus \boldsymbol{E}_n$. The booleanized condition decomposes as the intersection of a family $A_{\boldsymbol{\tau}} \subseteq V$, for $\boldsymbol{\tau} \in \Sigma_T^n$, of admissible regions, each corresponding to a disjunctive clause. In order to write a safe zone function for each clause, we need to have a parametrized family of safe zone functions for conditions $f_i(\boldsymbol{X}_i) \leq \tau_i$. Let $\zeta_i(\boldsymbol{X}_i; \tau_i)$, $i = 1, \ldots, n$ denote such a parameterized family. Then, a safe zone for $A_{\boldsymbol{\tau}}$ can be given by function $\zeta_{\boldsymbol{\tau}}(\boldsymbol{X}) = \sum_{i=1}^n a_i(\boldsymbol{\tau})\zeta_i(\boldsymbol{X}_i; \tau_i)$, for suitably selected $a_i(\boldsymbol{\tau})$. Finally, the overall safe zone function is $\zeta(\boldsymbol{X}) = \inf_{\tau \in \Sigma_T^n} \zeta_\tau(\boldsymbol{X})$.

As a first example, we solve a trivial problem; the linear constraint $\sum_{i=1}^{n} w_i x_i = \boldsymbol{w}\boldsymbol{x} \leq T$. Booleanization gives $\forall \boldsymbol{\tau} \in \Sigma_T^n : \bigvee_{i=1}^{n} w_i x_i \leq \tau_i$. Each constraint $w_i x_i \leq \tau_i$ has a good safe zone described by the eikonal and non-redundant affine function $\zeta_i(x_i; \tau_i) = (\tau_i - w_i x_i)/|w_i|$. Therefore, we have an overall expression of

$$\zeta(\boldsymbol{x}) = \inf_{\boldsymbol{\tau} \in \Sigma_T^n} \zeta_{\boldsymbol{\tau}}(\boldsymbol{x}) = \inf_{\boldsymbol{\tau} \in \Sigma_T^n} \sum_{i=1}^{n} a_i(\boldsymbol{\tau}) \frac{\tau_i - w_i x_i}{|w_i|}.$$

Although any choice for $a_i(\boldsymbol{\tau})$ will yield a legal safe zone, to obtain a good solution, we need to select $a_i(\boldsymbol{\tau})$ more carefully. But, notice that if we select $a_i(\boldsymbol{\tau}) = |w_i|/\|\boldsymbol{w}\|$, we obtain $\zeta_{\boldsymbol{\tau}}(\boldsymbol{x}) = \frac{T - \boldsymbol{w}\boldsymbol{x}}{\|\boldsymbol{w}\|}$, which is independent of $\boldsymbol{\tau}$. Therefore, the inf operator becomes redundant. It is easy to see that the solution obtained is best possible; it is less clear whether there is a systematic strategy for selecting $a_i(\boldsymbol{\tau})$, for more complex problems. Below we introduce two such systematic strategies.

6.1 Dominating Index

It turns out that the previous example exhibits a *dominating index*. Assume that every $\zeta_i(\boldsymbol{x}_i; \tau_i)$ is eikonal and maximum distance. Define function

$$\Delta(\boldsymbol{\tau}) = \sqrt{\sum_{i=1}^{n} \max\big(\zeta_i(\boldsymbol{E}_i; \tau_i), 0\big)^2} = \operatorname{dist}(\boldsymbol{E}, \overline{A_{\boldsymbol{\tau}}})$$

where the second equality is a consequence of Thm. 11. Assume that Δ minimizes at a unique index $\boldsymbol{\tau}^*$, where, naturally, $\Delta(\boldsymbol{\tau}^*)$ is $\operatorname{dist}(\boldsymbol{E}, \overline{A})$. We can select $a_i(\boldsymbol{\tau}^*)$ as per Thm. 11, that is, $a_i(\boldsymbol{\tau}^*) = \zeta_i(\boldsymbol{E}_i; \tau_i^*)/\Delta(\boldsymbol{\tau}^*)$, and obtain $\zeta_{\boldsymbol{\tau}^*}$. Now, if it so happens that $L(\zeta_{\boldsymbol{\tau}^*}) \subseteq A$, then, we say that $\boldsymbol{\tau}^*$ is a *dominating index*: we can select $\zeta = \zeta_{\boldsymbol{\tau}^*}$, eliminating the inf operation. The overall safe zone $L(\zeta)$ is obviously maximum distance. Also, if $L(\zeta_{\boldsymbol{\tau}^*})$ is maximal in $A_{\boldsymbol{\tau}^*}$, $L(\zeta)$ (i.e., $L(\zeta_{\boldsymbol{\tau}^*})$) is maximal in $A \subseteq A_{\boldsymbol{\tau}}$.

6.2 Alignment

Independently of the existence of a dominating index, alignment is a simple strategy for selecting weights $a_i(\boldsymbol{\tau})$, when the query $F(\boldsymbol{X}) = \sum_{i=1}^{n} f_i(\boldsymbol{X}_i)$ is differentiable.

Fix some clause $\boldsymbol{\tau}$. Assume that each $\zeta_i(\boldsymbol{X}_i; \boldsymbol{\tau})$ is witnessed maximal; for simplicity, assume that it is also eikonal. Now, consider a point $\breve{\boldsymbol{X}} = \breve{\boldsymbol{X}}_1 \oplus \cdots \oplus \breve{\boldsymbol{X}}_n$, such that, for every i, $f_i(\breve{\boldsymbol{X}}_i) = \tau_i$ and $\zeta_i(\breve{\boldsymbol{X}}_i; \tau_i) = 0$. That is, each $\breve{\boldsymbol{X}}_i$ is a maximality witness for $\zeta_i(\boldsymbol{X}_i; \tau_i)$. By smoothness of f_i, its gradient $\boldsymbol{g}_i = \nabla f_i(\breve{\boldsymbol{X}}_i)$ will be parallel to the gradient $\nabla \zeta_i(\breve{\boldsymbol{X}}_i; \tau_i)$ at this witness point. Then, we can choose $a_i(\boldsymbol{\tau})$ so as to *align* (make parallel) the gradients of F and $\zeta_{\boldsymbol{\tau}}$ at $\breve{\boldsymbol{X}}$, which is achieved by $a_i(\boldsymbol{\tau}) = \|\boldsymbol{g}_i\|$. Note also that $\sum_{i=1}^{n} \boldsymbol{g}_i^2 = \|\nabla F(\breve{\boldsymbol{X}})\|^2$, thus, a choice of $a_i = \|\boldsymbol{g}_i\|/\|\nabla F(\breve{\boldsymbol{X}})\|$ yields normalized weights.

This justifies the suitability of the choice $a_i = |w_i|/\|\boldsymbol{w}\|$ for every clause $\boldsymbol{\tau}$ in the previous example. Also, with respect to the previous section, it is easy to see that at the minimizer $\boldsymbol{\tau}^*$ of $\Delta(\boldsymbol{\tau})$, since $\breve{\boldsymbol{X}}$ is a nearest neighbor of $\boldsymbol{E}$ on the boundary of A, by smoothness of F, $\|\boldsymbol{g}_i\|$ will be proportional to $\zeta_i(\boldsymbol{E}_i; \tau_i^*)$.

Like dominating index, this method is limited to query functions that exhibit symmetry; if there are several witness points $\breve{\boldsymbol{X}}$, each giving a different value for a_i, then it is not clear what is the best choice.

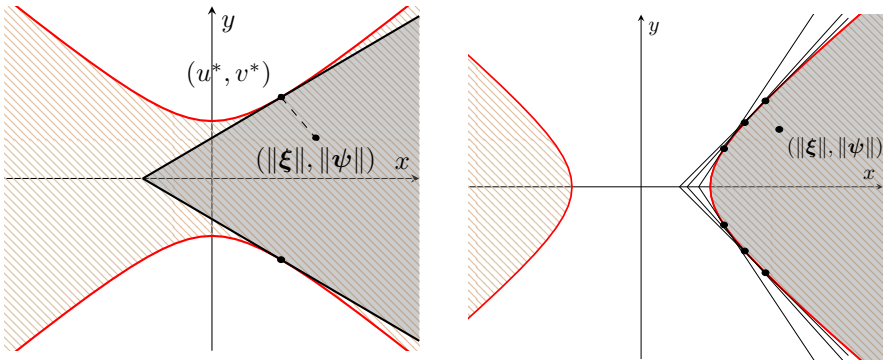


Figure 3 Safe zones (grayed) for $x^2 - y^2 \geq T$, when $T \leq 0$ (left) and $T > 0$ (right). The admissible regions are hatched. On the left, the safe zone is a single cone $u^*x - v^*|y| \geq T$, where (u^*, v^*) is the dominating index. On the right, the safe zone is defined by the intersection of all cones $ux - v|y| \geq T$ (three such cones are shown).

6.3 Safe Zones For Inner Product

We turn to the non-trivial problem of designing a good safe zone for the inner product of two vectors. Previous solutions [14, 23, 22] strove for the same qualities, but were suboptimal in terms of maximum distance [23] or maximality [14, 22].

The problem is defined by condition $\boldsymbol{XY} \geq t$. Instead of decomposing this problem on a per-dimension basis, we apply the polarization identity, by the change of variables $\boldsymbol{x} = (\boldsymbol{X} + \boldsymbol{Y})/\sqrt{2}$, and $\boldsymbol{y} = (\boldsymbol{X} - \boldsymbol{Y})/\sqrt{2}$. It is easy to see that $\boldsymbol{x}^2 - \boldsymbol{y}^2 = 2\boldsymbol{XY}$ and that the change of variables is a rotation, that is, it preserves all distances. Therefore, we focus on the (slightly more general) problem of monitoring $\boldsymbol{x}^2 - \boldsymbol{y}^2 \geq T$, with $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$. We shall design a safe zone for this problem, for reference point $\boldsymbol{E} = \boldsymbol{\xi} \oplus \boldsymbol{\psi}$, where $\boldsymbol{\xi}^2 - \boldsymbol{\psi}^2 > T$.

First note that, when $\boldsymbol{\xi}$ is 0 (which implies that $T < 0$), the constraint $\|\boldsymbol{y}\| \leq -T$ defines a good (maximum-distance and maximal) safe zone for the original problem. Therefore, we assume $\|\boldsymbol{\xi}\| \neq 0$ and we denote $\hat{\boldsymbol{\xi}} = \boldsymbol{\xi}/\|\boldsymbol{\xi}\|$.

We treat this problem as a separable sum of two functions, $f_1(\boldsymbol{x}) = \boldsymbol{x}^2$ and $f_2(\boldsymbol{y}) = -\boldsymbol{y}^2$, of which f_1 is convex and f_2 is concave (thus, $-f2$ is convex). The condition $f_1 + f_2 \geq T$ can be booleanized as

$$\forall u, v \geq 0 : u^2 - v^2 = T : \quad \boldsymbol{x}^2 \geq u^2 \vee \boldsymbol{y}^2 \leq v^2,$$

where we have applied a convenient change of variables to the index tuple $(\tau_1, \tau_2) = (u^2, -v^2)$.

The problem $\boldsymbol{x}^2 \geq u^2$ is solved optimally by the affine eikonal function $\zeta_1(\boldsymbol{x}; u) = \boldsymbol{x}\hat{\boldsymbol{\xi}} - u$. Similarly, the problem $\boldsymbol{y}^2 \leq v^2$ is solved optimally by the eikonal, non-redundant solution $\zeta_2(\boldsymbol{y}; v) = v - \|\boldsymbol{y}\|$. Putting everything together, we obtain formula

$$\zeta(\boldsymbol{x} \oplus \boldsymbol{y}) = \inf_{u^2 - v^2 = T} \zeta_{u,v}(\boldsymbol{x} \oplus \boldsymbol{y}) \qquad \text{where} \quad \zeta_{u,v}(\boldsymbol{x} \oplus \boldsymbol{y}) = \alpha(u, v)(\boldsymbol{x}\hat{\boldsymbol{\xi}} - u) + \beta(u, v)(v - \|\boldsymbol{y}\|).$$

By alignment, we select (unnormalized) $\alpha(u, v) = u$ and $\beta(u, v) = v$, and get

$$\zeta_{u,v}(\boldsymbol{x} \oplus \boldsymbol{y}) = u\boldsymbol{x}\hat{\boldsymbol{\xi}} - v\|\boldsymbol{y}\| - T.$$

which is tangent to the boundary of the admissible region at every point $u\hat{\boldsymbol{\xi}} \oplus v\hat{\boldsymbol{y}}$, where $\|\hat{\boldsymbol{y}}\| = 1$.

Instead of continuing the treatment of the problem directly, we will examine the case $n = m = 1$; that is, the equivalent planar problem with reference point $\boldsymbol{e} = (\|\boldsymbol{\xi}\|, \|\boldsymbol{\psi}\|)$. This

problem (and its solution) is depicted in Fig. 3. In fact, treating the 2-d problem is equivalent to treating the problem in any dimension, by the change of variables $x = \boldsymbol{x}\hat{\boldsymbol{\xi}}$ and $y = \|\boldsymbol{y}\|$.

Case $T \leq 0$. When $T \leq 0$, the problem admits a dominating index, (u^*, v^*), which is found by minimizing $\Delta(u, v)$ over the index set.

Case $T > 0$. In this case there is no dominating index. The safe zone is the right lobe of the hyperbola, that is $Z = \{x \geq \sqrt{y^2 + T}\}$. This set is smooth everywhere on its boundary, therefore its SDF $\delta_Z(x, y)$ is non-redundant.

Overall, we have the following:

▶ **Theorem 20.** *Given constraint $\boldsymbol{x}^2 - \boldsymbol{y}^2 \geq T$ and reference point $\boldsymbol{\xi} \oplus \boldsymbol{\psi}$ in any dimension, a good safe zone is given by $\zeta(\boldsymbol{x} \oplus \boldsymbol{y}) = \zeta_2(\boldsymbol{x}\hat{\boldsymbol{\xi}}, \|\boldsymbol{y}\|)$, where ζ_2 is the safe zone for the planar constraint $x^2 - y^2 \geq T$ with reference point $(\|\boldsymbol{\xi}\|, \|\boldsymbol{\psi}\|)$. More precisely,*
1. *if $T \leq 0$, $\zeta_2(x, y) = (u^* x - v^* |y| - T)/\sqrt{u^{*2} + v^{*2}}$, and*
2. *if $T > 0$, $\zeta_2(x, y)$ is the SDF of convex set $\{x \geq \sqrt{y^2 + T}\}$.*
Both ζ and ζ_2 are eikonal and non-redundant, and define maximum-distance and witnessed maximal safe zones.

Computation cost for monitoring the inner product. Computing the safe zone function for l-dimensional vectors, takes time $O(l)$, in order to compute $\boldsymbol{x}\hat{\boldsymbol{\xi}}$ and $\|y\|$. Then, computing the actual value can be done in time $O(1)$. When the vectors between successive computations of the safe zone function change in only $O(1)$ coordinates (which is quite standard in stream monitoring, when a stream update changes $O(1)$ locations of the state vector), it is possible simply update cached previous values of $\boldsymbol{x}\hat{\boldsymbol{\xi}}$ and $\|\boldsymbol{y}\|$ and reduce the cost to $O(1)$.

Safe zones for AGMS sketches. As discussed, monitoring the inner product of two streams by AGMS sketches, involves monitoring the median of d inner products, of dimension l each. The result's accuracy is within $O(1/\sqrt{l})$, with probability at least $1 - O(1/2^d)$. For practical purposes, d will be of the order of 10, but l of the order of 1000.

Combining our safe zones for inner product with the safe zone for the median, we obtain the first provably good safe zone for AGMS sketches. The computational cost of testing membership in the safe zone is important, as it is likely to be performed for every stream update. Using the FastAGMS sketch of [10], each stream update changes only 1 counter in each of the d vectors of a sketch. Therefore, the change to each monitored inner product can be computed in $O(1)$ time. The median's safe zone requires $O(d)$ time for testing membership. In conclusion, our safe zones can be used for membership testing with only $O(d)$ cost per update.

7 Related Work

The problem of tracking distributed streams through *in situ* constraints has attracted significant attention in recent years. Still, most existing work has focused on purpose-built solutions for specific query classes; these include, for instance, the simpler cases of thresholding *linear* functions [19, 18, 24]. top-k monitoring [4, 25], ratio threshold queries [16], and tracking polynomials of simple scalar variables [30]. All these techniques typically rely on some form of locally-installed "adaptive filters" – that is, bounds around the value of distributed variables that can grow or shrink over time (e.g., based on variability), while guaranteeing a global bound on the overall uncertainty. Similar local filtering ideas are also employed by Huang et al. [17] to monitor the eigenvalues of a network traffic matrix through

perturbation analysis, and by Wolf et al. [33] to threshold the norm of the average vector in a distributed system. Cormode and Garofalakis [7, 8] introduce the use of *sketch synopses* [9] for effectively summarizing local data streams, and propose sketch-based schemes for the communication-efficient, approximate monitoring of join aggregates over distributed streams. Finally, Cormode et al. [11] give a theoretical study of the distributed function monitoring problem focusing, in particular, on providing *communication lower bounds* for the case of various L_p-norm functions, assuming a "cash-register" (i.e., insert only) streaming model. Lower bounds for distributed norm monitoring are also given by Arackaparambil et al. [3], who demonstrate that, for general "turnstile" streams (i.e., allowing both inserts and deletes), the worst-case communication lower bounds are *linear* in the size of the stream. [1]

The Geometric Method of Sharfman et al. [31, 32] introduced the first generic approach for efficiently thresholding a *general function/query* over distributed data. Their solution relies on a function-agnostic, geometric "covering spheres" technique for breaking the global condition into safe local constraints. Extensions of the basic method as well as the more general notion of convex Safe Zones (SZs) are discussed in a later paper [21], and a broad range of applications have been explored, including distributed outlier detection [6], prediction-based distributed stream monitoring [15], sketch-based monitoring of norms and range aggregates [13], and distributed skyline tracking [27].

As demonstrated in our recent work [23], the safe zones (implicitly) defined by the "covering spheres" method are often far from optimal, and geometric convexity arguments (based on decomposing the problem into convex pieces) can give provably better results in certain important cases. Still, the methodology and results in [23] are heuristic, refer to specific classes of monitoring functions, and do not offer any hard quality guarantees for the resulting safe zones; furthermore, they do not consider the important problem of safe zone composition. Instead, our work is based on a novel functional representation of safe zones which allows us to effectively deal with general Boolean safe zone composition with provable quality guarantees. Our Boolean formalism is, in fact, much more powerful, and can easily express the methodology of [23] as a special case. The very recent work of Lazerson et al. [22] proposes another broad method based on defining "convex bounds" for the monitored function $F()$ using functional approximation techniques (assuming $F()$ is differentiable). However, no optimality properties are formally shown for the resulting SZs, and the problem of effective SZ composition is not addressed. The worst-case communication complexity of geometric techniques for distributed monitoring has not been studied, but empirical studies demonstrate significant communication gains in real problems and query workloads.

8 Conclusions

In this paper, we have presented the first formal framework for the compositional design of convex Safe Zones (SZs), for problems with separable constraints. To this end, we have introduced a functional safe zone representation that conserves, under Boolean composition, the quality guarantees of their component constraints. We have also applied our new framework to general function monitoring scenarios of practical interest.

Important problems remain open for future research, mainly relating the quality of safe zones to actual guarantees on the communication cost of monitoring, and extending our compositional approach beyond boolean, to other types of composite queries.

[1] The problem of communication lower bounds for general functions under the cash-register model remains open.

References

1 Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In *Proc. of the 18th ACM Symposium on Principles of Database Systems*, Philadeplphia, Pennsylvania, May 1999.

2 Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, Philadelphia, Pennsylvania, May 1996.

3 Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. Functional monitoring without monotonicity. In *ICALP (1)*, 2009. doi:10.1007/978-3-642-02927-1_10.

4 B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD'03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2003. ACM. doi:10.1145/872757.872764.

5 Sabbas Burdakis and Antonios Deligiannakis. "Detecting Outliers in Sensor Networks Using the Geometric Approach". In *Proc. of the 28th Intl. Conference on Data Engineering*, April 2012.

6 Sabbas Burdakis and Antonios Deligiannakis. Detecting outliers in sensor networks using the geometric approach. In *ICDE*, 2012. doi:10.1109/ICDE.2012.85.

7 Graham Cormode and Minos Garofalakis. "Sketching Streams Through the Net: Distributed Approximate Query Tracking". In *Proc. of the 31st Intl. Conference on Very Large Data Bases*, Trondheim, Norway, September 2005.

8 Graham Cormode and Minos Garofalakis. "Approximate Continuous Querying over Distributed Streams". *ACM Transactions on Database Systems*, 33(2), June 2008.

9 Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches". *Foundations and Trends in Databases*, 4(1-3), 2012.

10 Graham Cormode and Minos N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

11 Graham Cormode, S. Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. In *SODA*, 2008. doi:10.1145/1347082.1347200.

12 Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. *"Data-Stream Management – Processing High-Speed Data Streams"*. Springer-Verlag New York (Data-Centric Systems and Applications Series), 2016.

13 Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. "Sketch-based Geometric Monitoring of Distributed Stream Queries". In *Proc. of the 39th Intl. Conference on Very Large Data Bases*, Trento, Italy, August 2013.

14 Minos N. Garofalakis, Daniel Keren, and Vasilis Samoladas. Sketch-based geometric monitoring of distributed stream queries. *PVLDB*, 2013.

15 Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Izchak Sharfman, and Assaf Schuster. "Prediction-based Geometric Monitoring of Distributed Data Streams". In *Proc. of the 2012 ACM SIGMOD Intl. Conference on Management of Data*, Scottsdale, Arizona, May 2012.

16 Rajeev Gupta, Krithi Ramamritham, and Mukesh K. Mohania. "Ratio threshold queries over distributed data sources". In *Proc. of the 39th Intl. Conference on Very Large Data Bases*, Trento, Italy, August 2013.

17 Ling Huang, XuanLong Nguyen, Minos N. Garofalakis, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, and Nina Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM*, 2007. doi:10.1109/INFCOM.2007.24.

18 Srinivas R. Kashyap, Jeyashankher Ramamirtham, Rajeev Rastogi, and Pushpraj Shukla. Efficient constraint monitoring using adaptive thresholds. In *ICDE*, pages 526–535, 2008. doi:10.1109/ICDE.2008.4497461.

19 Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, 2006. doi:10.1145/1142473.1142507.

20 Daniel Keren, Guy Sagy, Amir Abboud, David Ben-David, Assaf Schuster, Izchak Sharfman, and Antonios Deligiannakis. "Geometric Monitoring of Heterogeneous Streams". *IEEE Transactions on Knowledge and Data Engineering*, 26(8), August 2014.

21 Daniel Keren, Izchak Sharfman, Assaf Schuster, and Avishay Livne. Shape sensitive geometric monitoring. *IEEE Trans. Knowl. Data Eng.*, 24(8), 2012. doi:10.1109/TKDE.2011.102.

22 Arnon Lazerson, Daniel Keren, and Assaf Schuster. Lightweight monitoring of distributed streams. In *Proc. of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'16, pages 1685–1694, New York, NY, USA, 2016. ACM. doi:10.1145/2939672.2939820.

23 Arnon Lazerson, Izchak Sharfman, Daniel Keren, Assaf Schuster, Minos Garofalakis, and Vasilis Samoladas. "Monitoring Distributed Streams using Convex Decompositions". In *Proc. of the 41st Intl. Conference on Very Large Data Bases*, August 2015.

24 Shicong Meng, Ting Wang, and Ling Liu. Monitoring continuous state violation in datacenters: Exploring the time dimension. In *ICDE*, pages 968–979, 2010. doi:10.1109/ICDE.2010.5447923.

25 Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. Klee: a framework for distributed top-k query algorithms. In *VLDB'05*. VLDB Endowment, 2005.

26 S. Muthukrishnan. "Data Streams: Algorithms and Applications". *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

27 Odysseas Papapetrou and Minos Garofalakis. "Continuous Fragmented Skylines over Distributed Streams". In *Proc. of the 30th Intl. Conference on Data Engineering*, Chicago, Illinois, April 2014.

28 R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

29 G. Sagy, D. Keren, I. Sharfman, and A. Schuster. "Distributed Threshold Querying of General Functions by a Difference of Monotonic Representation". In *Proc. of the 36th Intl. Conference on Very Large Data Bases*, August 2010.

30 Shetal Shah and Krithi Ramamritham. Handling non-linear polynomial queries over dynamic data. In *ICDE*, 2008. doi:10.1109/ICDE.2008.4497513.

31 Izchak Sharfman, Assaf Schuster, and Daniel Keren. "A geometric approach to monitoring threshold functions over distributed data streams". In *SIGMOD*, 2006. doi:10.1145/1142473.1142508.

32 Izchak Sharfman, Assaf Schuster, and Daniel Keren. "A geometric approach to monitoring threshold functions over distributed data streams". *ACM Trans. Database Syst.*, 32(4), 2007. doi:10.1145/1292609.1292613.

33 Ran Wolff, Kanishka Bhaduri, and Hillol Kargupta. A generic local algorithm for mining data streams in large distributed systems. *IEEE Trans. on Knowl. and Data Eng.*, 21(4), 2009. doi:10.1109/TKDE.2008.169.

Entropy Bounds for Conjunctive Queries with Functional Dependencies

Tomasz Gogacz[*1] **and Szymon Toruńczyk**[†2]

1 University of Edinburgh, Edinburgh, UK
2 University of Warsaw, Warsaw, Poland

Abstract

We study the problem of finding the worst-case size of the result $Q(\mathbb{D})$ of a fixed conjunctive query Q applied to a database $\mathbb{D}$ satisfying given functional dependencies. We provide a characterization of this bound in terms of entropy vectors, and in terms of finite groups. In particular, we show that an upper bound provided by Gottlob, Lee, Valiant and Valiant [9] is tight, and that a correspondence of Chan and Yeung [5] is preserved in the presence of functional dependencies. However, tightness of a weaker upper bound provided by Gottlob et al., which would have immediate applications to evaluation of join queries [11], remains open. Our result shows that the problem of computing the worst-case size bound, in the general case, is closely related to difficult problems from information theory.

1998 ACM Subject Classification H.2.4 Query Processing

Keywords and phrases database theory, conjunctive queries, size bounds, entropy, finite groups, entropy cone

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.15

1 Introduction

Given a conjunctive query Q we would like to determine the least bound $\alpha \in \mathbb{R}$ such that for every database $\mathbb{D}$, the inequality

$$|Q(\mathbb{D})| \;\leq\; c \cdot |\mathbb{D}|^{\alpha} \tag{1}$$

holds, for some multiplicative factor c depending only on Q. Above, $|\mathbb{D}|$ denotes the size of the largest table in the database $\mathbb{D}$, and $|Q(\mathbb{D})|$ denotes the size of the result of the query applied to $\mathbb{D}$. In the general problem which is the main focus of this paper, we may additionally impose some functional dependencies. Define $\alpha(Q)$ as the infimum of all values α for which there exists a multiplicative factor c so that (1) holds for all databases $\mathbb{D}$ satisfying the given functional dependencies. Note that defining $|\mathbb{D}|$ as the sum of the sizes of the tables in $\mathbb{D}$ would result in the same value of $\alpha(Q)$, since this would increase $|\mathbb{D}|$ by at most a constant factor (the number of relations in $\mathbb{D}$), and this constant can be accommodated by c in (1).

For example, if $Q_1(x, y, z) = R(x, y) \wedge S(y, z)$ and there are no functional dependencies, then it is not difficult to see that $|Q_1(\mathbb{D})| \leq |R(\mathbb{D})| \cdot |S(\mathbb{D})| \leq |\mathbb{D}|^2$, so in 1 we can take $\alpha = 2$, and it is not difficult to see that $\alpha(Q_1) = 2$. Now consider the natural join query $Q_2(x, y, z) = R(x, y) \wedge S(y, z) \wedge T(z, x)$. A trivial bound gives $\alpha(Q_2) \leq 3$. However, since

* Supported by VADA EPSRC grant M025268.
† Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ERC-2014-CoG 648276 AUTAR).

$Q_2(\mathbb{D}) \subseteq Q_1(\mathbb{D})$ for every $\mathbb{D}$, it follows that $\alpha(Q_2) \leq 2$. With some effort, one can show that in the absence of functional dependencies, $\alpha(Q_2) = 3/2$. This follows from a result due to Atserias, Grohe and Marx [1], which characterizes $\alpha(Q)$ when the query Q is a natural join query (without projections, self-joins and attribute renamings), and there are no functional dependencies, which we recall now.

Consider the hypergraph whose vertices are the variables appearing in Q, and for each relation name R in Q there is a hyperedge containing those variables which appear in R. A *fractional edge packing* of a hypergraph assigns a positive rational number to each of its vertices, so that for every hyperedge, the numbers assigned to the adjacent vertices sum up to at most 1. The total weight of a fractional edge packing is the sum of the numbers assigned to all the vertices. Define $\mathrm{AGM}(Q)$ as the largest possible total weight of a fractional edge packing of the hypergraph associated to Q. The AGM bound then states that $\mathrm{AGM}(Q) = \alpha(Q)$, in the absence of functional dependencies. For example, the hypergraph obtained from the query Q_2 is the triangle, and assigning $1/2$ to each vertex gives a fractional edge packing with total weight $3/2$, which is the largest possible. Hence, $\alpha(Q_2) = \mathrm{AGM}(Q_2) = 3/2$. Note that $\mathrm{AGM}(Q)$ can be computed using linear programming.

In this paper, we make progress towards characterizing the value $\alpha(Q)$ in the presence of functional dependencies. In particular, we show that $\alpha(Q)$ can be characterized in two ways: as an entropy bound $H(Q)$, and in terms of a number $\mathrm{GC}(Q)$ derived from systems of finite groups. The bound $\alpha(Q) \leq H(Q)$ was observed by Gottlob et al. [9]. We provide a matching lower bound $\alpha(Q) \geq H(Q)$ based on a construction using finite groups. Unfortunately, we do not know how to compute the value $\alpha(Q)$. Our results indicate that the problem is closely connected to notorious problems from information theory.

Note that the paper [9] also gives a weaker bound $\alpha(Q) \leq H(Q) \leq h(Q)$, where $h(Q)$ is obtained by relaxing the entropy cone to the polymatroid defined by Shannon inequalities (see Corollary 8). The question whether $\alpha(Q) = h(Q)$ remains open. Note that $h(Q)$ can be computed using linear programming. Moreover, $\alpha(Q) = h(Q)$ would prove optimality of an algorithm computing $Q(\mathbb{D})$, recently provided by Khamis et al. [11].

Throughout most of the paper, we focus on natural join queries. A gentle introduction to entropy and natural join queries, and the relationship between them is given in the preliminaries (Section 2). In the following section, Section 3, we state our main results and their consequences. The main result is proved by first characterizing the value $H(Q)$ (Section 4), then adapting upper bounds and the tightness proof from the work of [5] to the setting with functional dependencies (Sections 5 and 6). In Section 7 we discuss how to treat general conjunctive queries and in Section 8 we show some preliminary results concerning the computation of the result $Q(\mathbb{D})$.

Comparison with previous work. The work of Chan and Yeung [5] establishes a two-way relationship between entropy vectors and group vectors. Our construction of databases from group systems, presented in Section 5, although reminiscent of their construction of entropy vectors from group systems, was independent of that work and was motivated by the coloring construction of Gottlob et al. [9] and their example showing its suboptimality. Nevertheless, it is only fair to say that the development of Section 5 is a straightforward adaptation of a construction of Chan and Yeung [5] in the database setting. As a consequence, we extend their correspondence to a three-way correspondence between group vectors, entropy vectors, and vectors induced from databases. The easier direction, from databases to entropy vectors, was implicit in the paper by Gottlob et al. [9], however, the other direction was missing. For completeness, we present all proofs.

More importantly, however, we demonstrate that in all directions, the correspondences preserve functional dependencies. Some of these preservation results are rather obvious: if a group vector satisfies certain functional dependencies, then the corresponding entropy vector satisfies them too – this has been observed in Proposition 3 of [4] – and so does the corresponding database (see Proposition 18 below). In Section 6 we analyse the construction of group vectors from entropy vectors from [14] and demonstrate that it preserves functional dependencies. Also, to prove Theorem 5, we need to prove that functional dependencies behave well with respect to topological closure (cf. Lemma 12). Reassuming, the main contribution of this paper can be seen as a detailed study of the preservation of functional dependencies in the framework proposed by Chan and Yeung.

2 Preliminaries

To fix notation, we recall some notions concerning databases and entropy. We assume a fixed *schema* Σ, which specifies a finite set of *attributes* $\mathbf{V}(\Sigma)$, a finite set of *relation names*, and for each relation name R, a finite set $\mathbf{V}(R) \subseteq \mathbf{V}(\Sigma)$ of attributes of R. If X is a set of attributes, then a *row* with attributes X is a function r assigning to each $x \in X$ some value $r[x]$. If r is a row with attributes X and $Y \subseteq X$ then by $r[Y]$ we denote the restriction of r to Y. A *table* with attributes X is a finite set of rows with attributes X. A *database* $\mathbb{D}$ over Σ specifies for each relation name R in Σ a table with attributes $\mathbf{V}(R)$. A *natural join query* is a set Q of relation names in Σ; we denote $\mathbf{V}(Q) = \bigcup_{R \in Q} \mathbf{V}(R)$. Such a query can be applied to a database $\mathbb{D}$, yielding as result the table $Q(\mathbb{D})$ consisting of those rows r with attributes $\mathbf{V}(Q)$ such that $r[\mathbf{V}(R)] \in R(\mathbb{D})$ for every $R \in Q$.

We say that a database $\mathbb{D}$ *satisfies* a *functional dependency* $R : X \mapsto x$ – where X is a set of attributes and x is a single attribute – if for any two rows u, v of $R(\mathbb{D})$, $u[X] = v[X]$ implies $u[x] = v[x]$.

For the rest of this paper, fix a schema Σ and a set of functional dependencies $\mathcal{F}$. Every database $\mathbb{D}$ is assumed to be over this schema, and to satisfy $\mathcal{F}$. Define $\alpha(Q)$ as the smallest value α for which there exists a constant c such that (1) holds for all databases $\mathbb{D}$ over Σ which satisfy the functional dependencies in $\mathcal{F}$. For convenience, we define $|\mathbb{D}|$ to be the maximal size of a relation in $\mathbb{D}$.

▶ Remark. Observe that since Q is a natural join query and in the definition of $\alpha(Q)$ we are interested in maximizing $|Q(\mathbb{D})|$ while keeping $|\mathbb{D}|$ bounded, we may assume that Σ contains only the relation symbols which appear in Q (as $\mathbb{D}$ would have the remaining tables empty anyway) and furthermore, that if $R : X \mapsto x$ is a functional dependency in $\mathcal{F}$, then also $S : X \mapsto x$ is a functional dependency in $\mathcal{F}$, for every relation name S such that $X \subseteq \mathbf{V}(S)$. Therefore, we may simply write that $\mathcal{F}$ contains the functional dependency $X \mapsto x$ instead of writing $R : X \mapsto x$.

For a database $\mathbb{D}$ (over Σ, satisfying $\mathcal{F}$), denote

$$\alpha(Q, \mathbb{D}) = \frac{\log |Q(\mathbb{D})|}{\log |\mathbb{D}|}. \tag{α}$$

Convention. Throughout this paper we will define several real-valued parameters of the form $\gamma(Q, x)$, where Q is a query and x is some object. For a fixed query Q, we denote by $\sup_x \gamma(Q, x)$ the supremum, and by $\limsup_x \gamma(Q, x)$ the limit superior over all values x, for which the value $\gamma(Q, x)$ is defined. In particular, $\limsup_x \gamma(Q, x)$ is the smallest value s in $\mathbb{R} \cup \{-\infty, +\infty\}$ such that for every real $\varepsilon > 0$, there are only finitely many x's such that $\gamma(Q, x)$ is defined and larger than $s + \varepsilon$.

Limit superior vs. supremum. The following lemma will simplify several formulations and proofs throughout this paper.

▶ **Lemma 1.** *Let Q be a natural join query Q. Then $\alpha(Q) = \limsup_{\mathbb{D}} \alpha(Q, \mathbb{D}) = \sup_{\mathbb{D}} \alpha(Q, \mathbb{D})$.*

Proof. We omit the easy proof that $\alpha(Q) = \limsup_{\mathbb{D}}(Q, \mathbb{D})$, and prove only the second equality. To show that $\sup_{\mathbb{D}} \alpha(Q, \mathbb{D}) \leq \limsup_{\mathbb{D}} \alpha(Q, \mathbb{D})$, we use the following construction. For a database $\mathbb{D}$ and a natural number n, let $\mathbb{D}^n$ be the database defined so that the rows of $R(\mathbb{D}^n)$ are n-tuples of rows of $R(\mathbb{D})$, and for such a row $r = (r_1, \ldots, r_n)$, we define $r[x] = (r_1[x], \ldots, r_n[x])$ for an attribute $x \in \mathbf{V}(R)$. It is easy to check that $\mathbb{D}^n$ satisfies the same functional dependencies as $\mathbb{D}$, and that $|\mathbb{D}^n| = |\mathbb{D}|^n$ and $|Q(\mathbb{D})| = |Q(\mathbb{D})|^n$. In particular, $\alpha(Q, \mathbb{D}^n) = \alpha(Q, \mathbb{D})$. It follows that if $|\mathbb{D}| > 1$, then by choosing n arbitrarily large, we have arbitrarily large databases $\mathbb{D}^n$ with $\alpha(Q, \mathbb{D}^n) = \alpha(Q, \mathbb{D})$. Therefore, $\limsup_{\mathbb{D}} \alpha(Q, \mathbb{D}) \geq \sup_{\mathbb{D}} \alpha(Q, \mathbb{D})$, the other inequality being obvious. ◀

Entropy. In this paper, we only consider random variables taking finitely many values. Formally, a random variable X is a measurable function $X : \Omega \to V$ from a fixed probability space $(\Omega, \mathbb{P})$ of events to a finite set V. In this paper, however, it is not harmful to assume that Ω is a finite probability space, in which case every function $X : \Omega \to V$ is a random variable. By $\mathrm{Im}(X) \subseteq V$ we denote the set of values v such that $\mathbb{P}[X = v] > 0$, where $\mathbb{P}[X = v]$ is a shorthand for $\mathbb{P}[\{\omega \in \Omega : X(\omega) = v\}]$.

For a random variable X taking values in a finite set V, define the *entropy* of X as $H(X) = -\sum_{v \in \mathrm{Im}(X)} p_v \log p_v$, where $p_v = \mathbb{P}[X = v]$. Clearly, the entropy of X only depends on the distribution of X. Also, the maximal possible entropy of a random variable with values in a finite set V is equal to $\log |V|$, and is attained by the uniform distribution on V, as follows from Jensen's inequality applied to the convex function $-\log(x)$.

Upper bound. We recall an upper bound on $\alpha(Q, D)$, observed by Gottlob et al. [9]. Fix a natural join query Q. Let U be a random variable U taking as values rows with attributes $\mathbf{V}(\Sigma)$. For a set of attributes $X \subseteq \mathbf{V}(\Sigma)$, define $U[X]$ to be the random variable whose value is the restriction of the value of U to the set of attributes X. In particular, $U[X]$ is a random variable whose values are rows with attributes X, and $\mathrm{Im}(U[\mathbf{V}(R)])$ is a table with attributes $\mathbf{V}(R)$. We say that U satisfies a functional dependency $Y \mapsto x$ if the table $\mathrm{Im}(U)$ satisfies the functional dependency $Y \mapsto x$. We write Yx to denote the set $Y \cup \{x\}$. We say that a vector $h \in \mathbb{R}^{P(\mathbf{V}(Q))}$ satisfies a functional dependency $Y \mapsto x$ if $h(Y) = h(Yx)$. The following lemma is immediate.

▶ **Lemma 2.** *U satisfies a functional dependency $Y \mapsto x$ iff the vector h^U satisfies $Y \mapsto x$.*

We remark that in information theory, $h^U(Y) = h^U(Yx)$ can be expressed using conditional entropy as $H(\ U[x] \mid U[Y]\ ) = 0$ or $h^U(x \mid Y) = 0$.

For a random variable U which satisfies every functional dependency in $\mathcal{F}$, define

$$H(Q, U) = \frac{H(U[\mathbf{V}(Q)])}{\max_{R \in \Sigma} H(U[\mathbf{V}(R)])}. \tag{H}$$

The following is an observation from [9].

▶ **Lemma 3.** *Let Q be a natural join query. For a database $\mathbb{D}$, let $U_{\mathbb{D}}$ be a random variable which picks a row of $Q(\mathbb{D})$ uniformly at random. Then $\alpha(Q, \mathbb{D}) \leq H(Q, U_{\mathbb{D}})$.*

Proof. By definition of a natural join query, the values of $U_{\mathbb{D}}[R]$ are rows of $R(\mathbb{D})$. Then $H(Q, U_{\mathbb{D}}) \geq \alpha(Q, \mathbb{D})$ since $H(U_{\mathbb{D}}) = \log |Q(\mathbb{D})|$ and $H(U_{\mathbb{D}}[R]) \leq \log |R(\mathbb{D})|$ by the fact that the uniform distribution maximizes entropy. ◀

Entropy cone. Fix a finite set X. Let $U = (U_x)_{x \in X}$ be a family of random variables indexed by X. For $Y \subseteq X$, let $U[Y]$ denote the joint random variable $(U_y)_{y \in Y}$. The random variable $U[Y]$ can be seen as a random variable taking as values tuples indexed by Y. Consider the real-valued function h^U from subsets of X, such that $h^U(Y) = H(U[Y])$ for $Y \subseteq X$; the function h^U can be also seen as a vector in $\mathbb{R}^{P(X)}$. Vectors of the form $h^U \in \mathbb{R}^{P(X)}$, where U is a family of random variables indexed by X, are called *entropy vectors* (or *entropic vectors*) with ground set X [16].

The set of all entropy vectors with ground set X forms a subset of $\mathbb{R}^{P(X)}$, denoted Γ^*_X. Its topological closure $\overline{\Gamma^*_X}$ is a convex cone. The sets Γ^*_X and $\overline{\Gamma^*_X}$ are well studied, however, to date, they lack effective descriptions when $|X| \geq 4$. It is known that the closed cone $\overline{\Gamma^*_X}$ is a polyhedron if $|X| \leq 3$, and is not a polyhedron if $|X| > 3$ (i.e. it is not described by finitely many linear inequalities). Entropy vectors $h \in \Gamma^*_X$ (and hence, by continuity, also all $h \in \overline{\Gamma^*_X}$) satisfy the submodularity property, expressing *Shannon's inequality* for information:

$$h(Y \cup Z) + h(Y \cap Z) \leq h(Y) + h(Z) \quad \text{for } Y, Z \subseteq X. \tag{2}$$

Groups, actions and cosets. We use basic notions from algebra. We refer the reader to [12] for background. If G is a group and H is its subgroup, then a (left) *coset* of H in G is a subset of G of the form $gH = \{gh : h \in H\} \in G/H$. Let G/H denote the *coset space*, i.e., the set $G/H = \{gH : g \in G\}$ of all cosets of H in G.

For a finite group G and a set X, recall that a (left) action of G on X is a mapping $G \times X \to X$ denoted $(g, x) \mapsto g \cdot x$, such that $(g \cdot h) \cdot x = g \cdot (h \cdot x)$ for $g, h \in G$ and $x \in X$, and $e \cdot x = x$ for $x \in X$ and e the identity element of G. We say that the action is *transitive* if for every $x, y \in X$ there is $g \in G$ such that $g \cdot x = y$. The *stabiliser* of a point $x \in X$ (for a given action of G on X) is the subset $\{g \in G : g \cdot x = x\}$ of G; this is in fact a subgroup of G.

Note that if H is a subgroup of G, then G acts transitively on G/H, where the action is given by $g' \cdot gH = (g'g)H$ for $g', g \in G$; transitivity of this action follows from the fact that for $g, g' \in G$, $(g'g)^{-1}gH = g'H$. Every transitive action of G on some set X arises in this way, i.e., is isomorphic to the action of G on G/H, from some subgroup H of G (namely, one can take H as the stabilizer of any element $x \in X$).

3 Main result and consequences

In this section, we give a brief overview of the main result of the paper, and its consequences.

Main result. For a relation name R in a schema Σ, by $\mathbf{V}(R)$ we denote the set of attributes of R. For $X \subseteq \mathbf{V}(R)$ and $x \in \mathbf{V}(R)$, we write $R : X \mapsto x$ to denote the functional dependency (fd) requiring that in R, the values of attributes X determine the value of the attribute x. The value $\alpha(Q)$ is defined as in the introduction, taking into account all databases $\mathbb{D}$ over the schema Σ which satisfy a given set of functional dependencies $\mathcal{F}$. We associate another value $H(Q)$ to a query Q, as follows.

▶ **Definition 4.** Fix a schema Σ and a set of functional dependencies $\mathcal{F}$. Let Q be a natural join query with attributes X. Let $H(Q)$ be the maximal[1] value of $h(X)$, for h ranging over $\overline{\Gamma^*_X}$, satisfying:

[1] Note that the supremum of $h(X)$ is attained by some entropy vector h, since the function $h \mapsto h(X)$ is continuous, and the set of entropy vectors satisfying the given constraints is compact, as we use $\overline{\Gamma^*_X}$. This will not necessarily hold in other places throughout this paper.

$$\begin{cases} h(\mathbf{V}(R)) \leq 1 & \text{for } R \in Q, \\ h(Z \cup \{z\}) = h(Z) & \text{for every fd } R : Z \mapsto z \text{ in } \mathcal{F}. \end{cases}.$$

The main result of this paper is the following.

▶ **Theorem 5.** *Let Q be a natural join query. Then $\alpha(Q) = H(Q)$.*

▶ Remark. The inequality $\alpha(Q) \leq H(Q)$ is straightforward, and was observed in [9] (see Lemma 3). However, as we discuss in Remark 4, the inequality $\alpha(Q) \geq H(Q)$ is more involved.

Symmetric databases. The proof of the AGM bound (cf. [1] or Section 3.1 below) shows that in the absence of functional dependencies, the databases $\mathbb{D}$ for which the size-increase $\frac{\log|Q(\mathbb{D})|}{\log|\mathbb{D}|}$ achieves the bound $\alpha(Q)$ are of a very simple, specific form: each table is a full Cartesian product. It follows from [9] that in the presence of functional dependencies, this is no longer the case: databases of this form, satisfying the given functional dependencies, are arbitrarily far from reaching the value of $\alpha(Q)$.

In this paper, we improve the construction of worst-case databases in the presence of functional dependencies, by constructing databases which are arbitrarily close to achieving the bound $\alpha(Q)$. Interestingly, these databases have a very symmetric structure, and their construction uses finite groups. Our construction of databases from groups is very similar to a construction from [5].

For a fixed group G, by a *G-symmetric* database we mean a database $\mathbb{D}$ together with an action of G on the set of all values appearing in all tables of $\mathbb{D}$, such that the componentwise action of G on (the rows of) each table $R(\mathbb{D})$ is transitive (the componentwise action is given by $(g \cdot r)[x] = g \cdot (r[x])$ for $g \in G, r \in R(\mathbb{D})$ and $x \in \mathbf{V}(R)$). A *symmetric* database is a database $\mathbb{D}$ which is G-symmetric for some finite group G. Intuitively, in each table of a symmetric database, all rows are the same, up to permutation. Symmetric databases are very regular. For example, if a database $\mathbb{D}$ represents a graph G (i.e., it has one relation E, which is symmetric) and if $\mathbb{D}$ is symmetric, then the graph G is regular (all vertices have the same degree), vertex transitive (for any two vertices there is an automorphism of G mapping the first one to the second) and edge transitive (for any two edges there is an automorphism of G mapping the first one to the second).

The second main result of this paper is the following.

▶ **Theorem 6.** *Fix a schema Σ, functional dependencies $\mathcal{F}$, and a natural join query Q. Then, for each $\varepsilon > 0$ there are arbitrarily large symmetric databases $\mathbb{D}$ with $\frac{\log|Q(\mathbb{D})|}{\log|\mathbb{D}|} > \alpha(Q) - \varepsilon$.*

This can be seen as a structure result for worst-case databases. Apart from that, symmetric databases are essential in our proof of the lower bound given in Theorem 5.

Simple statistics. We state without proof a result generalizing Theorem 5, whose proof can be obtained in a similar way. In Theorem 5, intuitively, we were interested in the worst-case size of $Q(\mathbb{D})$, assuming the maximal table of $\mathbb{D}$ has a given size (which is manifested by the inequality $h(\mathbf{V}(R)) \leq 1$ for all R). Knowing the precise sizes of all tables in $\mathbb{D}$ may lead to better bounds on $Q(\mathbb{D})$, as we show below. For a database $\mathbb{D}$ over the schema of Q, its *simple log-statistics* is the vector $\mathrm{sls}(\mathbb{D}) = (\log|R(\mathbb{D})|)_{R \in Q}$.

▶ **Theorem 7.** *Fix a schema Σ and a set of functional dependencies $\mathcal{F}$. Let Q be a natural join query with attributes X, and let $s = (s_R)_{R \in Q}$ be a vector of nonnegative real numbers.*

Let $\beta(Q,s)$ be the maximal value of $h(X)$, for h ranging over $\overline{\Gamma^*_X}$ and satisfying:

$$\begin{cases} h(\mathbf{V}(R)) \leq s_R & \text{for } R \in Q, \\ h(Z \cup \{z\}) = h(Z) & \text{for every fd } R : Z \mapsto z \text{ in } \mathcal{F}. \end{cases}$$

Then $\sup_{\mathbb{D}} \log |Q(\mathbb{D})| = \limsup_{\mathbb{D}} \log |Q(\mathbb{D})| = \beta(Q,s)$, where $\mathbb{D}$ ranges over all finite databases satisfying the functional dependencies $\mathcal{F}$ and such that $\mathrm{sls}(\mathbb{D}) \leq s$ componentwise.

Theorem 5 is an immediate consequence of Theorem 7, obtained by setting $s_R = \log |\mathbb{D}|$ for each $R \in Q$, and using the fact that $\overline{\Gamma^*_X}$ is a cone, since we can divide the vector obtained by Theorem 7 by any positive number (in our case, $\log |\mathbb{D}|$) and still get a vector from $\overline{\Gamma^*_X}$. It is straightforward to verify that this new vector is a solution postulated by Theorem 5.

In this paper, we present in detail only the proof of Theorem 5. The proof of Theorem 7 proceeds similarly, and will be presented in the full version of the paper [8].

3.1 Consequences

We now show how some results known previously can be obtained as consequences of Theorem 5 (in fact, of the easier, upper bound $\alpha(Q) \leq \sup_U H(Q,U)$ presented in Section 2.) The results from Section 3.1 are not used in this paper, and are presented only for completeness.

Relaxing the condition in Definition 4 that $h \in \overline{\Gamma^*_X}$ to the condition that h is submodular gives the following upper bound on $\alpha(Q)$, which is equivalent to a bound in [9]. For a query Q and functional dependencies $\mathcal{F}$, consider the maximal[2] value of $h(X)$ for h ranging over $\mathbb{R}^{P(X)}$, satisfying:

$$h(\emptyset) = 0 \tag{3}$$
$$h(Y) \leq h(Z) \qquad \text{for } Y \subseteq Z \subseteq X, \tag{4}$$
$$h(Y \cup Z) + h(Y \cap Z) \leq h(Y) + h(Z) \qquad \text{for } Y, Z \subseteq X, \tag{5}$$
$$h(\mathbf{V}(R)) \leq 1 \qquad \text{for } R \in Q, \tag{6}$$
$$h(Z \cup \{z\}) = h(Z) \qquad \text{for every fd } R : Z \mapsto z \text{ in } \mathcal{F}. \tag{7}$$

Denote the above optimum by $h(Q)$. Since the conditions describing $h(Q)$ are a relaxation of the conditions describing $H(Q)$, from $\alpha(Q) \leq H(Q)$ we get the following.

▶ **Corollary 8** ([9]). *For a conjunctive query Q and functional dependencies $\mathcal{F}$, $\alpha(Q) \leq h(Q)$.*

Note that the bound $h(Q)$ can be computed by linear programming, where the number of variables is exponential in the size of Q. Unfortunately, the question whether $\alpha(Q) = h(Q)$ for every query Q and functional dependencies, stated in [9], remains open. Recently, Khamis, Ngo and Suciu [11] provided an algorithm for computing $Q(\mathbb{D})$ in the presence of functional dependencies, in time $\tilde{O}(|\mathbb{D}|^{h(Q)})$ (where $\tilde{O}$ hides polylogarithmic factors), which is optimal assuming the tightness of the above bound.

We now show how the AGM bound follows from Corollary 8. Note that the original proof [1] of the AGM bound used Shearer's Lemma from information theory and strong duality of linear programs. The proof presented below only uses Shannon information inequalities, manifested in Corollary 8. A function $h \in \mathbb{R}^{P(X)}$, is *modular* if it satisfies $h(Y) = \sum_{y \in Y} h(\{y\})$ for $Y \subseteq X$.

[2] The maximum is attained, as follows by a simple compactness argument.

▶ **Lemma 9.** *In absence of functional dependencies, the optimum $h(Q)$ is attained by a modular function $h \in \mathbb{R}^{P(X)}$.*

Proof. Let $h \in \mathbb{R}^{P(X)}$ be a function satisfying conditions (3)-(6), with maximal possible value of $h(X)$. In particular, h is submodular. We show that h can be in fact taken modular.

Denote by P_h the set[3] of functions $r : X \to \mathbb{R}$ satisfying $\hat{r}(Y) \leq h(Y)$ for $Y \subseteq X$, where $\hat{r}(Y)$ is shorthand for $\sum_{x \in Y} r(x)$. Consider the linear optimization problem of finding $r \in P_h$ which maximizes the value $\hat{r}(X) = \sum_{x \in X} r(x)$. It follows from general principles (see e.g. [13], Section 3) that the so-called "greedy algorithm" gives an optimal solution r to this problem, defined by

$$r(x_i) = h(\{x_1, \ldots, x_i\}) - h(\{x_1, \ldots, x_{i-1}\}) \qquad \text{for } i = 1, \ldots, n,$$

where $n = |X|$ and $x_1, x_2, \ldots, x_n$ is a fixed enumeration of the elements of X. In our special case it is sufficient and not difficult to check that conditions (3) and (5) imply $\hat{r}(Y) \leq h(Y)$ for all $Y \subseteq X$. Optimality of $\hat{r}$ then follows from the fact that $\hat{r}(X) = \sum_{i=1}^{n} r(x_i) = h(X) - h(\emptyset) = h(X)$. By modularity, $\hat{r}$ satisfies conditions (3)-(5), and it also satisfies condition (6) because $\hat{r}(Y) \leq h(Y)$ for all Y. Since $\hat{r}(X) = h(X)$, this proves the lemma. ◀

Since modular functions $h \in \mathbb{R}^{P(X)}$ are uniquely determined by their values for singletons, and automatically satisfy conditions (3)-(5), we conclude that in the absence of functional dependencies, $h(Q)$ is equal to the maximal value of $\sum_{x \in X} r(x)$ for $r \in \mathbb{R}^X$ satisfying $\sum_{x \in \mathbf{V}(R)} r(x) \leq 1$ for $R \in Q$. Such functions r are by definition real-valued edge packings of the hypergraph associated to Q, and the maximal possible total weight of such a packing is equal[4] to $\mathrm{AGM}(Q)$. Hence we get the following.

▶ **Corollary 10.** *In the absence of functional dependencies, $\alpha(Q) \leq h(Q) = \mathrm{AGM}(Q)$.*

We remark that [1] also prove a matching lower bound $\alpha(Q) \geq h(Q)$ in the absence of functional dependencies, thus proving $h(Q) = \alpha(Q) = \mathrm{AGM}(Q)$.

We remark that Theorem 7 can be used to derive the following, more precise variant of the AGM bound. A *fractional vertex covering* of a hypergraph is an assignment of rational weights to hyperedges, so that for every vertex, the sum of the weights assigned to the adjacent hyperedges is at least 1. By strong duality for linear programming, the smallest total weight of a fractional vertex covering is equal to the largest total weight of a fractional edge packing.

▶ **Corollary 11** ([1])**.** *In the absence of functional dependencies, $|Q(\mathbb{D})| \leq \prod_{R \in Q} |R(\mathbb{D})|^{w_R}$, where $(w_R)_{R \in Q}$ is any fractional vertex covering of the hypergraph associated to Q.*

To deduce the above corollary from Theorem 7, repeat the reasoning used above when deriving Corollary 10 from Theorem 5, by relaxing the condition $h \in \overline{\Gamma^*_X}$ in Theorem 7 to submodularity of h, and apply strong duality for linear programming. We leave the details to the interested reader.

[3] Called the polymatroid associated with the submodular function h.

[4] It follows from general principles of linear programming that a linear program with rational coefficients is optimized by a fractional solution.

Geometric inequalities. As noted elsewhere [15, 7, 2], Corollary 11 provides an upper bound on the size of a finite set in multi-dimensional space, in terms of the sizes of its projections. It implies the discrete versions of many inequalities from geometry and analysis, such as Hölder's inequality, Cauchy-Schwartz inequality, Loomis-Whitney inequality, Bollobás-Thomason inequality, Friedgut's inequality.

4 Characterization of $H(Q)$

In this section, we relate the value $H(Q)$ from Definition 4 in terms of $H(Q,U)$, as $\sup_U H(Q,U)$ (cf. Proposition 14). Recall that according to our convention, the supremum ranges over all random variables U which satisfy the given functional dependencies. Note that the inequality $H(Q) \geq \sup_U H(Q,U)$ follows from Definition 4 and the fact that the entropy vector h_U associated to U also satisfies the functional dependencies by Lemma 2. The remaining inequality $H(Q) \leq \sup_U H(Q,U)$ requires a more careful explanation. This is because $H(Q)$ is defined as a supremum for $h \in \overline{\Gamma^*_{\mathbf{V}(Q)}}$ which are limits of a sequence $h_1, h_2, \ldots$ of entropy vectors, and such that h (and not $h_1, h_2, \ldots$) satisfies the given functional dependencies. Therefore, to prove Proposition 14, we need the following.

▶ **Lemma 12.** *Let $h \in \overline{\Gamma^*_{\mathbf{V}(Q)}}$ be a vector satisfying functional dependencies $\mathcal{F}$. Then there exists a sequence of random variables $\{V_n\}_{n\in\mathbb{N}}$ such that $h = \lim_{n\to\infty} h^{V_n}$, and each V_n satisfies $\mathcal{F}$.*

Proof. Lemma 12 will be proven by induction on the size of $\mathcal{F}$. Let $\{U_n\}_{n\in\mathbb{N}}$ be a sequence of random variables such that $h = \lim_{n\to\infty} h^{U_n}$ satisfies a functional dependency $X \mapsto y$, and moreover each random variable U_n satisfies a set of functional dependencies $\mathcal{F}$. We construct a sequence of random variables $\{V_n\}_{n\in\mathbb{N}}$ such that each random variable V_n satisfies $\mathcal{F} \cup \{X \mapsto y\}$, and $\lim_{n\to\infty} h^{V_n} = h$.

Let us focus on a single random variable $U = U_n$. We partition the table $\mathrm{Im}(U)$ into sets $A_1, A_2, \ldots$, so that each A_i satisfies the functional dependency $X \mapsto y$, as follows. For each row $r \in \mathrm{Im}(U[X])$ proceed as follows. Choose a row $r_1 \in \mathrm{Im}(U[Xy])$ extending r, which is attained by the random variable $U[Xy]$ with the greatest probability. Add to the set A_1 all rows $r'_1 \in \mathrm{Im}(U)$ which extend r_1. After that choose a row $r_2 \in \mathrm{Im}(U[Xy])$ extending r, which is attained by the random variable $U[Xy]$ with the second greatest probability. Add to the set A_2 all rows $r'_2 \in \mathrm{Im}(U)$ which extend r_2. And so on.

At the end of this process, observe that each table A_i satisfies the dependency $X \mapsto y$.

Let A be a random variable which indicates to which set A_i the tuple from $\mathrm{Im}(U[X])$ belongs. In other words, $A = i \Leftrightarrow U \in A_i$. We omit the proof of the following lemma.

▶ **Lemma 13.** $H(A) \leq 2 \cdot H(U[y]|U[X])$

Let V be the following modification of U. Each attribute value in the set A_i gets a new prefix $A_{i\_}$ attached to its name, so the sets of possible attribute values in sets A_i and A_j are disjoint for $i \neq j$. After such a procedure, the entropy of the attributes could be increased, but not more than by $H(A)$.

We will show that setting V_n to V gives the desired result. The random variable V satisfies the dependencies $\mathcal{F} \cup \{X \mapsto y\}$ on each set A_i by definition of the set A_i. As presented during the construction, for any set of attributes Z, the entropy of $V[Z]$ can differ from $U[Z]$ at most by $2 \cdot H(U[y] \mid U[X])$. Since $\lim_{n\to\infty} H(U_n[y] \mid U_n[X]) = 0$ we get that $\lim_{n\to\infty} h^{U_n} = \lim_{n\to\infty} h^{V_n}$. ◀

Lemma 12 proves the following.

▶ **Proposition 14.** *Let Q be a natural join query. Then $H(Q) = \sup_U H(Q, U)$.*

From Lemma 3 and Proposition 14 we get the following.

▶ **Lemma 15.** *In the presence of functional dependencies, $\alpha(Q) \leq H(Q)$.*

Therefore, to prove Theorem 5, it remains to show that $\alpha(Q) \geq H(Q)$.

▶ Remark. To prove $\alpha(Q) \geq H(Q)$, it would be enough to construct, for a given random variable U a database $\mathbb{D}$ satisfying the same functional dependencies, and such that $\alpha(Q, \mathbb{D}) \geq H(Q, U)$. It is not clear how to construct such a database, even if U attains each row with rational probability (this is without loss of generality – see Lemma 22), or even with uniform distribution (this assumption would require justification, since $H(Q) \leq \alpha(Q)$ implies that for every entropy vector h there is a database $\mathbb{D}$ which achieves the same size-increase; we can then consider the uniform distribution $U_{\mathbb{D}}$ on the results $Q(\mathbb{D})$, as in Lemma 3). Our proof of Theorem 5 does not follow such a direct approach, but rather constructs a database from a system of finite groups constructed from the random variable U.

In Sections 5 and 6 we introduce a new parameter $\mathrm{GC}(Q)$ and prove the inequalities $\alpha(Q) \geq \mathrm{GC}(Q)$ and $\mathrm{GC}(Q) \geq H(Q)$. This, together with Lemma 15, will finish the proof of Theorem 5.

5 Lower bounds

In this section, we prove a lower bound $\alpha(Q) \geq \mathrm{GC}(Q)$. This bound will be obtained by a series of more and more refined constructions of databases. We start from recalling a construction using colorings due to Gottlob et al. [9]. We then improve this bound to vector space colorings, and finally, to group systems. In the entire Section 5, fix a natural join query Q over a schema Σ, and a set of functional dependencies $\mathcal{F}$.

5.1 Colorings

A *coloring* of Q is a function f assigning finite sets to $\mathbf{V}(Q)$. We say that f satisfies a functional dependency $X \mapsto x$ if $f(x) \subseteq f(X)$, where $f(X)$ denotes $\bigcup_{y \in X} f(y)$. For a coloring f of Q which satisfies all functional dependencies in $\mathcal{F}$, define

$$C(Q, f) = \frac{|f(\mathbf{V}(Q))|}{\max_{R \in \Sigma} |f(\mathbf{V}(R))|}, \tag{C}$$

and let $C(Q) = \sup_f C(Q, f)$.

The following proposition is proved in [9], and amounts to constructing a database $\mathbb{D}$ from a given coloring f. In the following section we will extend this construction to *vector space colorings*.

▶ **Proposition 16** ([9])**.** *Let Q be a natural join query. Then $\alpha(Q) \geq C(Q)$.*

It is shown in [9] that the value $C(Q)$ can be computed by a linear program. In the case without functional dependencies, this program is dual to the program for $\mathrm{AGM}(Q)$, so $C(Q) = \mathrm{AGM}(Q) = \alpha(Q)$.

5.2 Vector space colorings

In the presence of functional dependencies, there are queries Q for which $\alpha(Q) > C(Q)$, as shown in the paper [9], by elaborating an example proposed by Dániel Marx, and using Shamir's secret sharing scheme. Inspired by that construction and Blakley's secret sharing scheme, in this section we define a new parameter $\mathrm{VC}_{\mathbb{K}}(Q)$ based on vector spaces, and generalized later in Section 5.3 to $\mathrm{GC}(Q)$ based on arbitrary groups. We study the relations between these parameters and $C(Q)$. We note that the development of Section 5.2 and the inequalities involving the parameter $\mathrm{VC}_{\mathbb{K}}(Q)$ proved in Section 5.3 are not needed for proving $\alpha(Q) \geq \mathrm{GC}(Q)$ and our main result (Theorem 5), but we present them in order to relate our parameters $\mathrm{VC}_{\mathbb{K}}(Q)$ and later $\mathrm{GC}(Q)$ to the coloring number $C(Q)$, and also to provide a gradual introduction to the most general parameter $\mathrm{GC}(Q)$ defined in Section 5.3.

We consider vector spaces over a fixed finite field $\mathbb{K}$. If V is a vector space, X is a set, and V_x is a subspace of V for $x \in X$, then by $\sum_{x\in X} V_x$ we denote the smallest subspace of V containing every V_x, for $x \in X$ (we refer to [10] for background on vector spaces).

A *vector space coloring* of Q is a pair $\mathcal{V} = (V, (V_x)_{x\in\mathbf{V}(Q)})$, where V is a vector space and $(V_x)_{x\in\mathbf{V}(Q)}$ is a family of its subspaces. For such a coloring, define $V_Y = \sum_{x\in Y} V_x$ for $Y \subseteq \mathbf{V}(Q)$. We say that $\mathcal{V}$ *satisfies* a functional dependency $Y \mapsto x$ if $V_x \subseteq V_Y$. For a vector space coloring $\mathcal{V}$ satisfying all the functional dependencies in $\mathcal{F}$, define

$$\mathrm{VC}_{\mathbb{K}}(Q, \mathcal{V}) = \frac{\dim(V_{\mathbf{V}(Q)})}{\max_{R\in\Sigma} \dim(V_{\mathbf{V}(R)})}, \tag{VC}$$

Let $\mathrm{VC}_{\mathbb{K}}(Q) = \sup_{\mathcal{V}} \mathrm{VC}_{\mathbb{K}}(Q, \mathcal{V})$.

▶ **Proposition 17.** *Let Q be a natural join query. Then $\mathrm{VC}_{\mathbb{K}}(Q) \geq C(Q)$.*

Proof. The inequality $\mathrm{VC}_{\mathbb{K}}(Q) \geq C(Q)$ is obtained by defining for a coloring f a vector space coloring $\mathcal{V}$ with $V = \mathbb{K}^{Colors}$, $V_x = \mathbb{K}^{f(x)} \subseteq \mathbb{K}^{Colors}$, where $Colors = \bigcup_{x\in\mathbf{V}(Q)} f(x)$, and $\mathbb{K}^{f(x)}$ embeds into $\mathbb{K}^{Colors}$ in the natural way, by extending a vector with zeros on coordinates in $Colors - f(x)$. It is easy to see that $\mathcal{V}$ satisfies the same functional dependencies as f, and that $\mathrm{VC}_{\mathbb{K}}(Q, \mathcal{V}) = C(Q, f)$. This proves $\mathrm{VC}_{\mathbb{K}}(Q) \geq C(Q)$. ◀

5.3 Group systems

We relax the notion of a vector space coloring, by considering finite groups, as follows. Let G be a finite group and $(G_x)_{x\in\mathbf{V}(Q)}$ be a family of its subgroups. For a set of attributes $X \subseteq \mathbf{V}(Q)$, denote by G_X the group $G_X = \bigcap_{x\in X} G_x$, and by G/G_X the space of (left) cosets, $\{gG_X : g \in G\}$. We call the pair $\mathcal{G} = (G, (G_x)_{x\in X})$ a *group system* for Q, and say that it satisfies a functional dependency $X \mapsto x$ if $G_X \subseteq G_x$ (note the duality with respect to vector space colorings, with $\cup$ replaced by $\cap$ and $\subseteq$ replaced by $\supseteq$). For a group system $\mathcal{G}$ satisfying all the functional dependencies in $\mathcal{F}$, define

$$\mathrm{GC}(Q, \mathcal{G}) = \frac{\log |G/G_{\mathbf{V}(Q)}|}{\max_{R\in\Sigma}(\log |G/G_{\mathbf{V}(R)}|)}, \tag{GC}$$

and let $\mathrm{GC}(Q) = \sup_{\mathcal{G}} \mathrm{GC}(Q, \mathcal{G})$.

▶ **Proposition 18.** *Let Q be a natural join query. Then*

$$\alpha(Q) \geq \mathrm{GC}(Q) \geq \mathrm{VC}_{\mathbb{K}}(Q). \tag{8}$$

Proof. Call a group system $\mathcal{G}$ a *vector space system* if it consists of a vector space V, treated as a group with addition, and its subspaces $(V_x)_{x\in X}$, treated as subgroups of V. The second inequality in (8) is an immediate consequence of the following lemma, which is an application of vector space duality (see e.g. [10]).

▶ **Lemma 19.** *For every vector space coloring $\mathcal{V}$ there is a vector space system $\mathcal{V}^*$ satisfying the same functional dependencies as $\mathcal{V}$, and such that* $\mathrm{VC}_{\mathbb{K}}(Q,\mathcal{V}) = \mathrm{GC}(Q,\mathcal{V}^*)$.

Proof. If V is a vector space, let V^* denote its algebraic dual, i.e., the space of all linear functions from V to $\mathbb{K}$ (with addition and multiplication by scalars defined coordinatewisely). If L is a subspace of V, then let $L^\perp \subseteq V^*$ denote the set of functionals $f \in V^*$ which vanish on L, i.e., $f(L) \subseteq \{0\}$. For a vector space coloring $\mathcal{V} = (V, (V_x)_{x\in\mathbf{V}(Q)})$ let $\mathcal{V}^* = (V^*, (V_x^\perp)_{x\in\mathbf{V}(Q)})$. Using the standard facts $(L_1 \cap L_2)^\perp = L_1^\perp + L_2^\perp$ and $L_1 \subseteq L_2$ iff $L_1^\perp \supseteq L_2^\perp$, one easily checks that $\mathcal{V}^*$ satisfies the same functional dependencies as $\mathcal{V}$. Furthermore, from $\dim V = \dim V^*$ and

$$\dim(L^\perp) = \dim_V - \dim L = \log|V|/\log|\mathbb{K}| - \log|L|/\log|\mathbb{K}| = \log|V/L|/\log|\mathbb{K}|,$$

$\mathrm{VC}_{\mathbb{K}}(Q,\mathcal{V}) = \mathrm{GC}(Q,\mathcal{V}^*)$ follows easily. ◀

It remains to prove the inequality $\alpha(Q) \geq \mathrm{GC}(Q)$. To this end, from a group system $\mathcal{G}$ we construct a database $\mathbb{D}$ satisfying the same functional dependencies, and such that

$$\alpha(Q,\mathbb{D}) = \mathrm{GC}(Q,\mathcal{G}). \tag{9}$$

For a relation name R and an element $g \in G$, let r_g be the row such that $r_g[x] = gG_x \in G/G_x$ for every attribute $x \in \mathbf{V}(R)$. Define $\mathbb{D}$ by setting $R(\mathbb{D}) = \{r_g : g \in G\}$, for every relation name R. The following lemma implies immediately that the database $\mathbb{D}$ satisfies the same functional dependencies as $\mathcal{G}$.

▶ **Lemma 20.** *Fix a group G, a family $(G_x)_{x\in X}$ of subgroups of G, and a subgroup $G_0 \subseteq G$ such that $G_0 \supseteq \bigcap_{x\in X} G_x$. For any given $g \in G$, the cosets $(gG_x)_{x\in X}$ determine gG_0, i.e., for every two elements $g, h \in G$, if $gG_x = hG_x$ for all $x \in X$, then $gG_0 = hG_0$.*

Proof. If G_1 is a subgroup of G_0 then $gG_1 \subseteq gG_0$, and gG_0 is determined by gG_1 as follows: $gG_0 = \{k \cdot h : h \in gG_1, h \in G_0\}$. Applying this observation to $G_1 = \bigcap_{x\in X} G_x$ yields that gG_0 is determined by $g(\bigcap_{x\in X} G_x) = \bigcap_{x\in X}(gG_x)$. ◀

Next we show (9). For each relation name R, consider the mapping $f_R : G \to R(\mathbb{D})$, where $f_R(g) = r_g$. Clearly, the mapping is onto $R(\mathbb{D})$. To compute the size of its image, we analyse the kernel of f_R and observe that $\{h : r_g = r_h\} = gG_{\mathbf{V}(R)}$ for every $g \in G$. In particular, $|R(\mathbb{D})| = |G/G_{\mathbf{V}(R)}|$. Similarly, we verify that $|Q(\mathbb{D})| = |G/G_{\mathbf{V}(Q)}|$. Equation (9) then follows. By Lemma 1, this proves $\alpha(Q) \geq \mathrm{GC}(Q)$. ◀

▶ Remark. The database $\mathbb{D}$ constructed in the above proof is G-symmetric. Indeed, the values appearing in the database are cosets the form gG_x (where $x \in \mathbf{V}(Q)$) and G acts (from the left) on such cosets in the obvious way. Moreover, the action of G on each table $R(\mathbb{D})$ is isomorphic to the action of G on $G/G_{\mathbf{V}(R)}$, in particular, it is transitive. Also, if $\mathbb{D}$ is G-symmetric and $\mathbb{D}^n$ is defined as in the proof of Lemma 1, then $\mathbb{D}^n$ is G^n-symmetric.

6 Tightness

Lemma 15 and Proposition 18 show that $H(Q) \geq \alpha(Q) \geq \mathrm{GC}(Q)$ for every natural join query Q, in the presence of functional dependencies. In this section, we close the circle by proving the following.

▶ **Proposition 21.** *Let Q be a natural join query. Then $\mathrm{GC}(Q) \geq H(Q)$.*

Together with Proposition 18 and Lemma 15, this proves that $H(Q) = \alpha(Q) = \mathrm{GC}(Q)$. As noted in Proposition 14, this gives Theorem 5. Moreover, from the observation in Remark 5.3, it follows that the bound $\alpha(Q)$ can be approximated by (arbitrarily large) symmetric databases, proving Theorem 6.

All the necessary ideas to prove the proposition are present in the proof of the result of Chan and Yeung [5]. However, we cannot directly apply that theorem, since we need to keep track of the functional dependencies. In the rest of Section 6, we present a self-contained proof of Proposition 21, following [14].

For the rest of this section, fix a random variable U, taking values in a finite set of rows with attributes X, and satisfying the functional dependencies $\mathcal{F}$.

We say that a random variable V is *rational* if for every value $v \in \mathrm{Im}(V)$, the probability that V achieves v is a rational number. The following lemma follows easily from the density of rationals among the real numbers.

▶ **Lemma 22.** *For every number $\varepsilon > 0$ there exists a rational random variable V satisfying the same functional dependencies as U, and such that $\|h^V - h^U\| < \varepsilon$ with respect to the euclidean norm on $\mathbb{R}^{P(X)}$.*

To prove Proposition 21, we proceed as follows. For each rational random variable U satisfying the given functional dependencies, we will find a sequence of group systems $\mathcal{G}_k$ satisfying the functional dependencies $\mathcal{F}$, and such that

$$\lim_{k\to\infty} \mathrm{GC}(Q, \mathcal{G}_k) = H(Q, U). \tag{10}$$

From that, Proposition 21 follows:

$$H(Q) \overset{Prop.14}{=} \sup_U H(Q,U) \overset{Lem.22}{=} \sup_{U \text{ rational}} H(Q,U) \overset{(10)}{\leq} \sup_{\mathcal{G}} \mathrm{GC}(Q,\mathcal{G}) = \mathrm{GC}(Q).$$

From now on, let U be a rational random variable satisfying the functional dependencies $\mathcal{F}$. We will show that there exists a sequence of group systems witnessing (10).

Let $q \in \mathbb{N}$ be a natural number such that for each row $r \in \mathrm{Im}(U)$ the probability $\mathbb{P}[U = r]$ can be represented as a rational number with denominator q. For $k = q, 2q, 3q, \ldots$ let A_k be a matrix whose columns are indexed by attributes, containing exactly $k \cdot \mathbb{P}[U = r]$ copies of the row r, for every row $r \in \mathrm{Im}(U)$. Notice that $k \cdot \mathbb{P}[U = r]$ is always a natural number, and that A_k has exactly k rows in total.

Let G^k denote the group of all permutations of the rows of A_k. For a set of attributes $Y \subseteq X$, let G^k_Y denote the subgroup of G^k which stabilizes the submatrix $A_k[Y]$ of A_k, i.e.,

$$G^k_Y = \{\sigma \in G^k \mid \sigma(r)[y] = r[y] \text{ for each } r \in A_k \text{ and } y \in Y\}.$$

Denote $G^k_{\{x\}}$ by G^k_x. In particular, $G^k_Y = \bigcap_{y\in Y} G^k_y$. The following lemma is immediate.

▶ **Lemma 23.** *Suppose that U satisfies the functional dependency $Y \mapsto x$. Then $G^k_Y \subseteq G^k_x$.*

We define $\mathcal{G}_k$ as $(G^k, (G^k_x)_{x\in X})$. By Lemma 23, $\mathcal{G}_k$ is a group system satisfying the functional dependencies $\mathcal{F}$. It remains to prove (10).

Fix a set of attributes $Y \subseteq X$. For a row $r \in \mathrm{Im}(U[Y])$, let p_r denote $\mathbb{P}[U[Y] = r]$. Since r occurs exactly $k \cdot p_r$ times as a row of $A_k[Y]$, it follows that G^k_Y is isomorphic to the product of symmetric groups $\prod_{r\in\mathrm{Im}(U[Y])} S_{k\cdot p_r}$ (where $S_{k\cdot p_r}$ is the symmetric group of all permutations of the copies of the row r). In particular, $|G^k_Y| = \prod_{r\in\mathrm{Im}(U[Y])}(k\cdot p_r)!$. Using Stirling's approximation we get the following ($a_k \sim b_k$ signifies $\lim_{k\to\infty} a_k/b_k = 1$):

$$\log\left(\frac{|G^k|}{|G^k_Y|}\right) = \log\left(\frac{k!}{\prod_{r\in\mathrm{Im}(U[Y])}(k\cdot p_r)!}\right) \sim \left(k\log(k) - \sum_{r\in\mathrm{Im}(U[Y])}(k\cdot p_r)\log(k\cdot p_r)\right) \sim$$
$$-\sum_{r\in\mathrm{Im}(U[Y])}(k\cdot p_r)(\log(k\cdot p_r - \log k)) \;\sim\; (-k)\cdot\sum_{r\in\mathrm{Im}(U[Y])} p_r\log p_r \;\sim\; k\cdot H(U[Y]).$$

In particular,

$$\mathrm{GC}(Q,\mathcal{G}^k) = \frac{\log(|G^k|/|G^k_X|)}{\max_{R\in Q}\log(|G^k|/|G^k_{\mathbf{V}(R)}|)} \overset{k\to\infty}{\longrightarrow} \frac{H(U)}{\max_{R\in Q} H(U[\mathbf{V}(R)])} = H(Q,U).$$

This yields (10), proving Proposition 21, which together with Lemma 15 and Proposition 18 gives $H(Q) = \mathrm{GC}(Q) = \alpha(Q)$ and finishes the proof of Theorem 5. The more general Theorem 7 is proved similarly. Moreover, Theorem 6 follows from Remark 5.3.

7 General conjunctive queries

The previous sections concern natural join queries: conjunctive queries without existential quantifiers (or projections), in which the variables name coincides with the name of the attribute of the relation in which it appears (in particular, the same variable name cannot occur in the scope of one conjunct, and there are no equalities). In this section, we discuss how to treat arbitrary conjunctive queries.

7.1 Set semantics

Define a *natural conjunctive query* to be a query of the form $Q = \exists Y\ Q'$, where Q' is a natural join query over the schema Σ, and $Y \subseteq \mathbf{V}(\Sigma)$. The set of *free variables* of Q is $\mathbf{V}(Q) = \mathbf{V}(\Sigma) - Y$. For a database $\mathbb{D}$ over the schema Σ, define $Q(\mathbb{D})$ as $T[\mathbf{V}(Q)]$, where $T = Q'(\mathbb{D})$. In other words, $Q(\mathbb{D})$ is the table $Q'(\mathbb{D})$ restricted to the free variables of Q. This is the so-called *set-semantics*, since the result $T[\mathbf{V}(Q)]$ is a set of rows, i.e., each row occurs either 0 or 1 times.

As explained in [9] for each conjunctive query Q there exists a natural conjunctive query S, such that $\alpha(Q) = \alpha(S)$. Such S can be constructed in a purely syntactical way from Q by the chase procedure. Because of this, we only consider natural conjunctive queries.

The definition of $\alpha(Q)$ can be lifted without modification to natural conjunctive queries. The generalization of Theorem 5 has the expected form:

▶ **Theorem 24.** *Fix a relational schema Σ and a set of functional dependencies $\mathcal{F}$. Let Q be a natural conjunctive query over the schema Σ. Then $\alpha(Q)$ is equal to the maximal value of $v_{\mathbf{V}(Q)}$, for v ranging over $\overline{\Gamma^*_{\mathbf{V}(\Sigma)}}$ and satisfying:*

$$\begin{cases} v_{\mathbf{V}(R)} \le 1 & \text{for } R \in Q, \\ v_{Z\cup\{z\}} = v_Z & \text{for every fd } Z \mapsto z \text{ in } \mathcal{F}. \end{cases}$$

The proof of the lower bound is exactly the same as the proof of Theorem 5. However the proof of the upper bound has to be modified slightly, as described below.

For a query $Q = \exists Y.\ Q'$, define $H(Q, U)$ and $H(Q)$ as in Section 2. We then have the following analogue of Lemma 15, proving the upper bound.

▶ **Lemma 25.** *For a natural conjunctive query $Q = \exists Y.\ Q'$, $\alpha(Q) \leq H(Q)$.*

Proof. For a database $\mathbb{D}$, let $U_{\mathbb{D}}$ be the random variable with values in $Q'(\mathbb{D})$, described as the result r' of the following process: first choose uniformly at random a row $r \in Q(\mathbb{D})$, and then, choose uniformly at random a row $r' \in Q'(\mathbb{D})$ such that $r'[\mathbf{V}(Q)] = r$.

By definition, the random variable $U_{\mathbb{D}}[\mathbf{V}(Q)]$ is uniformly distributed on $Q(\mathbb{D})$. Now we get that:

$$\begin{aligned}\alpha(Q, \mathbb{D}) &= \frac{\log |Q(\mathbb{D})|}{\max_{R \in Q'} \log |R(\mathbb{D})|} = \frac{H(U_{\mathbb{D}}[\mathbf{V}(Q)])}{\max_{R \in Q'} \log |R(\mathbb{D})|} \\ &\leq \frac{H(U_{\mathbb{D}}[\mathbf{V}(Q)])}{\max_{R \in Q'} H(U_{\mathbb{D}}[\mathbf{V}(R)])} = H(Q, U_{\mathbb{D}}).\end{aligned}$$

By a similar argument as in the proof of Lemma 15, we derive that $\alpha(Q) \leq H(Q)$. ◀

8 Evaluation

In this section, we give some rudimentary results bounding the worst-case complexity of computing the result of a query $Q(\mathbb{D})$, for a given database $\mathbb{D}$. In the absence of functional dependencies, it was originally shown in [1] that $Q(\mathbb{D})$ can be computed from $\mathbb{D}$ in time proportional to $|\mathbb{D}|^{\alpha(Q)+1}$, which was later improved to $|\mathbb{D}|^{\alpha(Q)}$ by Ngo et al. [15]. In the bounds presented below, there is an additional factor $|\mathbb{D}|^m$, where m is a parameter depending on the functional dependencies, defined below. Recently, Khamis et al. [11] showed how to compute the result in time $\tilde{O}(|\mathbb{D}|^{h(Q)})$ in the presence of functional dependencies, i.e., in almost optimal time assuming that the bound $\alpha(Q) \leq h(Q)$ of Gottlob et al. [9] is tight.

Let $\mathcal{F}$ be a set of functional dependencies over attributes X. A *minimal component* C is an inclusion-minimal nonempty set of attributes $C \subseteq X$ with the property that whenever $Y \mapsto x$ is a functional dependency with $Y \cap C$ nonempty, then $x \in C$. For a set $X' \subseteq X$, define $\mathcal{F}[X']$ to be the set of functional dependencies over X' which consists of those functional dependencies $Y \mapsto x$ from $\mathcal{F}$, such that $Y \subseteq X'$.

We say that a set of attributes $S \subseteq X$ *spans* $\mathcal{F}$, if the smallest subset $\bar{S}$ of X containing S and closed under functional dependencies (i.e., $Y \mapsto x$ and $Y \subseteq \bar{S}$ implies $x \in \bar{S}$) is equal to X. We say that $\mathcal{F}$ has *width* m if it has a set of size at most m which spans it. We inductively define the *iterative width* by saying that $\mathcal{F}$ has iterative width m if either the set of attributes is empty, or for every its minimal component C, $\mathcal{F}[C]$ has width m, and if M is the set of attributes which belong to the minimal components, then $\mathcal{F}[X - M]$ also has iterative width m.

▶ **Example 26.** If $\mathcal{F}$ is an empty set of functional dependencies over a nonempty set of attributes, then $\mathcal{F}$ has iterative width 1. The set of dependencies $x \mapsto y, y \mapsto z, z \mapsto x$ has width 1, since it is spanned by $\{x\}$. It also has iterative width 1. Finally, let $X = \{x, y, z\}$ and $\mathcal{F}$ consist of the functional dependency $\{x, y\} \mapsto z$. Then $\mathcal{F}$ has width 2, since the set $\{x, y\}$ spans it. The only minimal component is $\{z\}$, and $\mathcal{F}[\{z\}]$ and $\mathcal{F}[\{x, y\}]$ have width 1 and 2, respectively. Hence $\mathcal{F}$ has iterative width 2.

Let $\mathbb{D}$ be a database and let C be a minimal component. For a row r over attributes X, denote by r/C the row with attributes X such that $r/C[x] = r[x]$ for $x \in X - C$ and $r/C[x] = x$ for $x \in X \cap C$. Therefore, r/C is obtained by replacing each value of an attribute in C by a placeholder, storing the name of the attribute. Denote by $\mathbb{D}/C$ the database obtained from $\mathbb{D}$ by replacing in each table R, every row r by the row r/C. Clearly, we have the following.

▶ **Lemma 27.** *The database $\mathbb{D}/C$ can be computed from $\mathbb{D}$ in linear time.*

The following lemma is immediate, by the fact that C is a minimal component.

▶ **Lemma 28.** *The database $\mathbb{D}/C$ satisfies all the functional dependencies of Q.*

Note, however, that the database $\mathbb{D}/C$ usually satisfies more functional dependencies than $\mathbb{D}$, namely, it satisfies all functional dependencies $\emptyset \mapsto x$, for $x \in C$.

▶ **Lemma 29.** *Let C be a minimal component, and suppose that $\mathcal{F}[C]$ has width m. Then, for a given database $\mathbb{D}$, the result $Q(\mathbb{D})$ can be computed from $Q(\mathbb{D}/C)$ in time $O(|Q(\mathbb{D}/C)| \cdot |\mathbb{D}|^m)$.*

Very roughly, the algorithm proceeds by computing, for all $s \in Q(\mathbb{D}/C)$ and all rows r over a spanning set for C and with admissible values, the row compatible with s and r (if it exists), and adding it to the result. We omit the details, due to lack of space.

By iteratively applying Lemma 29, we obtain the main result of this section.

▶ **Proposition 30.** *Fix a natural join query Q and functional dependencies $\mathcal{F}$ of iterative width m. There is an algorithm which for a given database $\mathbb{D}$ computes $Q(\mathbb{D})$ in time $O(|\mathbb{D}|^{\alpha(Q)+m})$.*

Observe that when the set $\mathcal{F}$ of functional dependencies is empty, Proposition 30 gives the algorithm from [1], whose running time is $O(|\mathbb{D}|^{\alpha(Q)+1})$, since then $\mathcal{F}$ has iterative width 1.

9 Conclusion and future work

We characterized the worst-case size-increase for the evaluation of conjunctive queries, in two ways: in terms of entropy and in terms of finite groups. Our construction improves a construction from [9]. We also presented a rudimentary result concerning the evaluation of natural join queries.

We see several directions of a possible future work. The first direction is to try to prove tightness of the bound $\alpha(Q) \leq h(Q)$ from Corollary 8, due to [9]. This would yield computability of $\alpha(Q)$, as $h(Q)$ can be computed using linear programming. Moreover, together with the recent results of Khamis et al. [11], this would give an almost optimal $\tilde{O}(|\mathbb{D}|^{\alpha(Q)})$ algorithm for computing the results of joins.

Computing $H(Q)$ directly looks hard, and probably would require a deeper understanding of entropy, and the entropy cone in particular. By comparison, we note that in cryptography and information theory the seemingly similar optimization problem of finding the optimal *information rate* in *access structures* [3] (a security system which can be used as a secret sharing scheme) is considered notoriously difficult [6]. This demonstrates that optimization problems over the entropy cone can be very difficult.

The fractional edge cover was useful in the analysis of the Hypercube algorithm [2], an algorithm for parallel evaluation of queries. Perhaps some ideas from the current paper can also be applied in the parallel setting.

Acknowledgements. We are grateful to Dan Suciu for introducing the problem of computing the worst-case size bound increase to us, to Adam Witkowski for fruitful discussions, and to anonymous reviewers.

References

1. Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013. doi:10.1137/110859440.
2. Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 273–284, 2013. doi:10.1145/2463664.2465224.
3. Ernest F. Brickell and Daniel M. Davenport. On the classification of ideal secret sharing schemes. *Journal of Cryptology*, 4(2):123–134, 1991. doi:10.1007/BF00196772.
4. Terence H. Chan. Group characterizable entropy functions. *CoRR*, abs/cs/0702064, 2007. URL: http://arxiv.org/abs/cs/0702064.
5. Terence H. Chan and Raymond W. Yeung. On a relation between information inequalities and group theory. *IEEE Transactions on Information Theory*, 48(7):1992–1995, 2002.
6. Oriol Farràs, Jessica Ruth Metcalf-Burton, Carles Padró, and Leonor Vázquez. On the optimization of bipartite secret sharing schemes. *Des. Codes Cryptography*, 63(2):255–271, May 2012. doi:10.1007/s10623-011-9552-7.
7. Ehud Friedgut. Hypergraphs, entropy, and inequalities. *The American Mathematical Monthly*, 111(9):749–760, 2004. URL: http://www.jstor.org/stable/4145187.
8. Tomasz Gogacz and Szymon Toruńczyk. Entropy bounds for conjunctive queries with functional dependencies. *CoRR*, abs/1512.01808, 2015. URL: http://arxiv.org/abs/1512.01808.
9. Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16, 2012. doi:10.1145/2220357.2220363.
10. Paul Halmos. *Finite-Dimensional Vector Spaces*. Springer, 1974.
11. Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 327–342. ACM, 2016. doi:10.1145/2902251.2902289.
12. Serge Lang. *Algebra*. Addison-Wesley, Menlo Park Cal, 1993. URL: http://opac.inria.fr/record=b1081613.
13. Laszló Lovász. Submodular functions and convexity. *Mathematical programming: the state of the art*, 1983. URL: http://www.cs.elte.hu/~{}lovasz/scans/submodular.pdf.
14. Desmond S. Lun. A relationship between information inequalities and group theory, 2002.
15. Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48, 2012. doi:10.1145/2213556.2213565.
16. Zhen Zhang and Raymond W. Yeung. On characterization of entropy function via information inequalities. *IEEE Transactions on Information Theory*, 44(4):1440–1452, 1998. doi:10.1109/18.681320.

On the Automated Verification of Web Applications with Embedded SQL

Shachar Itzhaky[1], Tomer Kotek[*2], Noam Rinetzky[3], Mooly Sagiv[4], Orr Tamir[5], Helmut Veith[†6], and Florian Zuleger[7]

1 **Massachusetts Institute of Technology, Cambridge, MA, USA**
2 **Vienna University of Technology, Vienna, Austria**
3 **Tel Aviv University, Tel Aviv, Israel**
4 **Tel Aviv University, Tel Aviv, Israel**
5 **Tel Aviv University, Tel Aviv, Israel**
6 **Vienna University of Technology, Vienna, Austria**
7 **Vienna University of Technology, Vienna, Austria**

Abstract

A large number of web applications is based on a relational database together with a program, typically a script, that enables the user to interact with the database through embedded SQL queries and commands. In this paper, we introduce a method for formal automated verification of such systems which connects database theory to mainstream program analysis. We identify a fragment of SQL which captures the behavior of the queries in our case studies, is algorithmically decidable, and facilitates the construction of weakest preconditions. Thus, we can integrate the analysis of SQL queries into a program analysis tool chain. To this end, we implement a new decision procedure for the SQL fragment that we introduce. We demonstrate practical applicability of our results with three case studies, a web administrator, a simple firewall, and a conference management system.

1998 ACM Subject Classification D.3.2 Database Management Languages, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases SQL, Scripting language, Web services, Program verification, Two-variable fragment of First Order logic, Decidability, Reasoning

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.16

1 Introduction

Web applications are often written in a scripting language such as PHP and store their data in a relational database which they access using SQL queries and data-manipulating commands [37]. This combination facilitates fast development of web applications, which exploit the reliability and efficiency of the underlying database engine and use the flexibility of the script language to interact with the user. While the database engine is typically a mature software product with few if any severe errors, the script with the embedded SQL statements does not meet the same standards of quality.

* Tomer Kotek, Helmut Veith, and Florian Zuleger were partially supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF). This work was done in part while Tomer Kotek was visiting the Simons Institute for the Theory of Computing.
† The tragic death of Helmut Veith prevented him from approving the final version. All faults and inaccuracies belong to his co-authors.

With a few exceptions (such as [15, 19]) the systematic analysis of programs with embedded-SQL statements has been a blind spot in both the database and the computer-aided verification community. The verification community has mostly studied the analysis of programs which fall into two classes: programs with (i) numeric variables and complex control structure, (ii) complex pointer structures and objects; however, the modeling of data and their relationships has not received the same attention. Research in the database community on the other hand has traditionally focused on correct design of databases rather than correct use of databases.

Our long-term research vision is to transfer and extend the techniques from the verification and program analysis community to the realm of programs with embedded SQL. Since the seminal papers of Hoare, the first step for developing a program analysis is a precise mathematical framework for defining programming semantics and correctness. In this paper we develop a Hoare logic for a practically useful but simple fragment of SQL, called SmpSQL, and a simple scripting language, called SmpSL, which has access to SmpSQL statements. Specifically, we describe a decidable logic for formulating specifications and develop a weakest precondition calculus for SmpSL programs; thus our Hoare logic allows to automatically discharge verification conditions. When analyzing SmpSL programs, we treat SQL as a black box library whose semantics is given by database theory. Thus we achieve verification results relative to the correctness of the underlying database engine.

We recall from Codd's theorem [13] that the core of SQL is equivalent in expressive power to first-order logic FO. Thus, it follows from Trakhtenbrot's theorem [35] that it is undecidable whether an SQL query guarantees a given post condition. We have therefore chosen our SQL fragment SmpSQL such that it captures an interesting class of SQL commands, but corresponds to a decidable fragment of first-order logic, namely $\mathrm{FO}^2_{\mathrm{BD}}$, the restriction of first-order logic in which all variables aside from two range over fixed finite domains called *bounded domains*. The decidability of the finite satisfiability problem of $\mathrm{FO}^2_{\mathrm{BD}}$ follows from that of FO^2, the fragment of first-order logic which uses only two variables. Although the decidability of FO^2 was shown by Mortimer [30] and a complexity-wise tight decision procedure was later described by Grädel, Kolaitis and Vardi [21], we provide the first efficient implementation of finite satisfiability of FO^2.

We illustrate our methodology on the example of a simple web administration tool based on [22]. The PANDA web administrator is a simple public domain web administration tool written in PHP. We describe in Section 2 how the core mailing-list administration functionality falls into the scope of SmpSL. We formulate a specification consisting of a database invariant and pre- and postconditions. Our framework allows us to automatically check the correctness of such specifications using our own $\mathrm{FO}^2_{\mathrm{BD}}$ reasoning tool.

Main contributions

1. We define SmpSQL, an SQL fragment which is contained in $\mathrm{FO}^2_{\mathrm{BD}}$.
2. We define a simple imperative script language SmpSL with embedded SmpSQL statements.
3. We give a construction for weakest preconditions in $\mathrm{FO}^2_{\mathrm{BD}}$ for SmpSL.
4. We implemented the weakest precondition computation for SmpSL.
5. We implemented a decision procedure for $\mathrm{FO}^2_{\mathrm{BD}}$. The procedure is based on the decidability and NEXPTIME completeness result for FO^2 by [21], but we use a more involved algorithm which reduces the problem to a SAT solver and is optimized for performance.

We evaluate our methodology on three applications: a web administrator, a simple firewall, and a conference management system. We compared our tool with Z3 [14], currently the most advanced general-purpose SMT solver with (limited) support for quantifiers. In general, our tool performs better than Z3 in several examples for checking the validity of verification conditions of SmpSL programs. However, our tool and Z3 have complementary advantages: Z3 does well for unsatisfiable instances while our tool performs better on satisfiable instances. We performed large experiments with custom-made blown up versions of the web administrator and the firewall examples, which suggest that our tool scales well. Moreover, we tested the scalability of our approach by comparing of our underlying FO^2 solver with three solvers on a set of benchmarks we assembled inspired by combinatorial problems. The solvers we tested against are Z3, the SMT solver CVC4 [3], and the model checker Nitpick [7]. Our solver outperformed each of these solvers on some of the benchmarks.

2 Running Example

We introduce our approach on the example of a simple web service. The example is a translation from PHP with embedded SQL commands into SmpSL of code excerpts from the Panda web-administrator. The web service provides several services implemented in dedicated functions for subscribing a user to a newsletter, deleting a newsletter, making a user an admin of a newsletter, sending emails to all subscribed users of a newsletter, etc. We illustrate our verification methodology by exposing an error in the Panda web-administrator. The verification methodology we envision in this paper consists of (1) maintaining database invariants and (2) verifying a contract specification for each function of the web service.

The database contains several tables including $NS = \mathit{NewsletterSubscription}$ with attributes *nwl*, *user*, *subscribed* and *code*. The database is a structure whose universe is partitioned into three sets: $\mathbf{dom}^{\mathrm{U}}$, $\mathbf{bool}^{\mathrm{B}}$, and $\mathbf{codes}^{\mathrm{B}}$. The attributes *nwl* and *user* range over the finite set $\mathbf{dom}^{\mathrm{U}}$, the attribute *subscribed* ranges over $\mathbf{bool}^{\mathrm{B}} = \{\mathit{true}, \mathit{false}\}$, and the attribute *code* ranges over the fixed finite set $\mathbf{codes}^{\mathrm{B}}$. The superscripts in $\mathbf{dom}^{\mathrm{U}}$, $\mathbf{bool}^{\mathrm{B}}$, and $\mathbf{codes}^{\mathrm{B}}$ serve to indicate that the domain $\mathbf{dom}^{\mathrm{U}}$ is unbounded, while the Boolean domain and the domain of codes are bounded (i.e. of fixed finite size). When $s = \mathit{true}$, $(n, u, s, c) \in NS$ signifies that the *user* u is *subscribed* to the *newsletter* n. The process of being (un)subscribed from/to a newsletter requires an intermediary confirmation step in which the confirm code c plays a role.

Figure 1 provides the functions `subscribe`, `unsubscribe`, and `confirm` translated manually into SmpSL. The comments in quotations `// "..."` originate from the PHP source code. The intended use of these functions is as follows: To subscribe a user u to a newsletter n, the function `subscribe` is called with inputs n and u (e.g. by a web interface operated by an admin or by the user). `subscribe` stores the tuple $(n, u, \mathit{false}, \mathit{new\_code})$ in NS, where $\mathit{new\_code}$ is a confirmation code which does not occur in the database, and an email containing a confirmation URL is sent to the user u. Visiting the URL triggers a call to `confirm` with input $\mathit{new\_code}$, which subscribes u to n by replacing the tuple $(n, u, \mathit{false}, \mathit{new\_code})$ of NS to with $(n, u, \mathit{true}, \mathtt{nil})$. For `unsubscribe` the process is similar, and crucially, `unsubscribe` uses the same `confirm` function. `confirm` decides between subscribe and unsubscribe according to whether n is currently subscribed to u. The `CHOOSE` command selects one row non-deterministically. The database preserves the invariant

$$\mathit{Inv} \quad = \quad \forall_{\mathbf{d}} x, y. \forall_{\mathbf{b}} s_1, s_2. \forall_{\mathbf{c}} c_1, c_2. \left((s_1 = s_2 \wedge c_1 = c_2) \vee \bigvee\nolimits_{i=1,2} \neg NS(x, y, s_i, c_i) \right) \qquad (1)$$

```
subscribe(n,u):
    A = SELECT * FROM NS WHERE user = u AND nwl = n;
    if (A != empty) exit; // "This address is already registered to this
                             newsletter."
    INSERT (n,u,false,new_code) INTO NS;
    // Send confirmation email to u

unsubscribe(n,u):
    A = SELECT * FROM NS WHERE user = u AND nwl = n;
    if (A = empty) exit; // "This address is not registered to this
                            newsletter."
    UPDATE NS SET code = new_code WHERE user = u AND nwl = n
    // Send confirmation email to u

confirm(cd):
    A = SELECT subscribe FROM NS WHERE code = cd;
    if (A = empty) exit; //"No such code"
    s1 = CHOOSE A;
    if (s1 = false)
         UPDATE NS SET subscribed = true, code = nil WHERE code = cd
    else DELETE FROM NS WHERE code = cd;
```

Figure 1 Running Example: SmpSL code.

Inv says that the pair (n, u) of newsletter and user is a key of the relation *NS*. The subscripts of the quantifiers denote the domains over which the quantified variables range. In our verification methodology we add invariants as additional conjuncts to the pre- and post-conditions of every function. In this way invariants strengthen the pre-conditions and can be used to prove the post-conditions of the functions. On the other hand, the post-conditions require to re-establish the validity of the invariants.

Figure 2 provides pre- and post-conditions $\text{pre}_{\texttt{f}}$ and $\text{post}_{\texttt{f}}$ for each of the three functions `f`. The relation names **d**, **b**, and **c** are interpreted as the sets $\mathbf{dom}^{\mathrm{U}}$, $\mathbf{bool}^{\mathrm{B}}$, and $\mathbf{codes}^{\mathrm{B}}$, respectively. Proving correctness amounts to proving the correctness of each of the *Hoare triples* $\{\text{pre}_{\texttt{f}} \wedge \mathit{Inv}\}\ \texttt{f}\ \{\text{post}_{\texttt{f}} \wedge \mathit{Inv}\}$. Each Hoare triple specifies a contract: after every execution of `f`, the condition $\text{post}_{\texttt{f}} \wedge \mathit{Inv}$ should be satisfied if $\text{pre}_{\texttt{f}} \wedge \mathit{Inv}$ was satisfied before executing `f`. $\text{pre}_{\texttt{subscribe}}$ and $\text{pre}_{\texttt{unsubscribe}}$ express that *new_code* is an unused non-nil code and that NS_{gh} is equal to *NS*. NS_{gh} is a *ghost table*, used in the post-conditions to relate the state before the execution of the function to the state after the execution. NS_{gh} does not occur in the functions and is not modified. $\text{post}_{\texttt{subscribe}}$ and $\text{post}_{\texttt{unsubscribe}}$ express that *NS* is obtained from NS_{gh} by inserting or updating a row satisfying `user` $=$ `u AND nwl` $=$ `n` whenever the `exit` command is not executed. The intended behavior of `confirm` depends on which function created *cd*. $\text{pre}_{\texttt{confirm}}$ introduces a Boolean ghost variable sub_{gh} whose value is true (respectively false) if *cd* was generated as a new code in `subscribe` (respectively `unsubscribe`). sub_{gh} does not occur in `confirm`. $\text{post}_{\texttt{confirm}}$ express that, when sub_{gh} is true, *NS* is obtained from *NS* by toggling the value of the column *subscribed* from false to true in the NS_{gh} row whose confirm code is *cd*; when sub_{gh} is false, *NS* is obtained from NS_{gh} by deleting the row with confirm code *cd*.

Let us now describe the error which prevents `confirm` from satisfying its specification. Consider the following scenario. First, `subscribe` is called and then `unsubscribe`, both with the same input n and u. Two confirm codes are created: $c_{\mathtt{s}}$ by `subscribe` and $c_{\mathtt{u}}$ by `unsubscribe`. At this point, *NS* contains a single row for the newsletter n and user u namely $(n, u, \mathit{false}, c_{\mathtt{u}})$. The user receives two confirmation emails containing the codes $c_{\mathtt{s}}$ and $c_{\mathtt{u}}$. Clicking on the confirmation URL for $c_{\mathtt{s}}$ (i.e. running $\mathtt{confirm}(c_{\mathtt{s}})$) has no effect since $c_{\mathtt{s}}$ does not occur in the database. However, clicking on the confirmation URL for $c_{\mathtt{s}}$ results in subscribing u to n. This is an error, since confirming a code created in `unsubscribe` should not lead to a subscription.

Our tool automatically checks whether the program satisfies its specification. If not, the programmer or verification engineer may try to refine the specification to adhere more closely to the intended behavior (e.g. by adding an invariant). In this case, the program is in fact incorrect, so no meaningful correct specification can be written for it.

In Section 3.3 we describe a *weakest-precondition calculus* $\mathrm{wp}[\![\cdot]\!]$ which allows us to automatically derive the weakest precondition for a post-condition with regard to a SmpSL program. For our example functions `f`, $\mathrm{wp}[\![\cdot]\!]$ allows us to automatically derive $\mathrm{wp}[\![\mathtt{f}]\!]\mathrm{post}_{\mathtt{f}}$. The basic property of the weakest precondition is that $\mathrm{post}_{\mathtt{f}}$ holds after `f` has executed iff $\mathrm{wp}[\![\mathtt{f}]\!]\mathrm{post}_{\mathtt{f}}$ held immediately at the start of the execution. It then remains to show that the pre-condition $\mathrm{pre}_{\mathtt{f}}$ implies $\mathrm{wp}[\![\mathtt{f}]\!]\mathrm{post}_{\mathtt{f}}$. This amounts to checking the validity of the verification conditions $VC_{\mathtt{f}} = \mathrm{pre}_{\mathtt{f}} \to \mathrm{wp}[\![\mathtt{f}]\!]\mathrm{post}_{\mathtt{f}}$.

Our reasoner for FO^2 sentences is the back-end for our verification tool. The specification in this example is all in $\mathrm{FO}^2_{\mathrm{BD}}$. The weakest precondition of a SmpSL program applied to a $\mathrm{FO}^2_{\mathrm{BD}}$ sentence gives again a $\mathrm{FO}^2_{\mathrm{BD}}$sentence. Hence $VC_{\mathtt{f}}$ are all in $\mathrm{FO}^2_{\mathrm{BD}}$. Automatically deciding the validity of $\mathrm{FO}^2_{\mathrm{BD}}$ sentences using our FO^2 decision procedure is described in Section 4. Recall that $\mathbf{codes}^{\mathrm{B}}$ is of fixed finite size. Here $|\mathbf{codes}^{\mathrm{B}}| = 3$ is sufficient to detect the error. Observe that the same confirm code may be reused once it is replaced with `nil` in `confirm`, so the size of the database is unbounded. The size of $\mathbf{codes}^{\mathrm{B}}$ must be chosen manually when applying our automatic tool.

A simple way to correct the error in `confirm` is by adding sub_{gh} as a second argument of `confirm` and replacing `if` $(\mathtt{s1} = \mathtt{false})$ $\cdots$ with `if` $(\mathtt{sub}_{\mathtt{gh}} = \mathtt{false})$ $\cdots$. Since s_1 is no longer used, the CHOOSE command can be deleted. The value of sub_{gh} received by `confirm` is set correctly by `subscribe` and `unsubscribe`. With these changes, the error is fixed and `confirm` satisfies its specification. In the scenario from above, the call to `confirm` with $c_{\mathtt{s}}$ and $\mathit{sub}_{gh} = \mathit{true}$ leaves the database unchanged, while the call to `confirm` with $c_{\mathtt{u}}$ and $\mathit{sub}_{gh} = \mathit{false}$ deletes the row $(n, u, \mathit{false}, c_{\mathtt{u}})$.

3 Verification of SmpSL Programs

Here we introduce our programming language and our verification methodology. We introduce the SQL fragment SmpSQL in Section 3.1 and the scripting language SmpSL in Section 3.2. In Section 3.3 we explain the weakest precondition transformer of SmpSL, and we show how discharging verification conditions of $\mathrm{FO}^2_{\mathrm{BD}}$ specification reduces to reasoning in FO^2.

3.1 The SQL fragment SmpSQL

3.1.1 Data model of SmpSQL

The data model of SmpSQL is based on the presentation of the relational model in Chapter 3.1 of [1]. We assume finite sets of $\mathbf{dom}^{\mathrm{B}}_1, \ldots, \mathbf{dom}^{\mathrm{B}}_s$ called the *bounded domains* and an infinite

$$
\begin{array}{lcl}
\mathrm{pre}_{\mathtt{g}} & = & NS = NS_{gh} \wedge \text{good-code}(new\_code) \\
\text{good-code}(c') & = & \mathbf{c}(c') \wedge (c' \neq nil) \wedge \forall_{\mathbf{d}} x, y. \forall_{\mathbf{b}} s. \neg NS(x, y, s, c') \\
\mathrm{post}_{\mathtt{g}} & = & \forall_{\mathbf{d}} x, y. \forall_{\mathbf{b}} s. \forall_{\mathbf{c}} c.\ NS(x, y, s, c) \leftrightarrow (\varphi_{\mathtt{g},1} \vee \varphi_{\mathtt{g},2}) \\
 & & \\
\varphi_{\mathtt{subscribe},1} & = & NS_{gh}(x, y, s, c) \\
\varphi_{\mathtt{subscribe},2} & = & (n = x) \wedge (u = y) \wedge (s = false) \wedge (c = new\_code) \\
 & & \wedge \neg \exists_{\mathbf{b}} s'. \exists_{\mathbf{c}} c'.\ NS_{gh}(n, u, s', c') \\
 & & \\
\varphi_{\mathtt{unsubscribe},1} & = & \neg ((n = x) \wedge (u = y)) \wedge NS_{gh}(x, y, s, c) \\
\varphi_{\mathtt{unsubscribe},2} & = & (n = x) \wedge (u = y) \wedge (c = new\_code) \wedge \exists_{\mathbf{c}} c'.\ NS_{gh}(n, u, s, c') \\
 & & \\
\mathrm{pre}_{\mathtt{confirm}} & = & NS = NS_{gh} \wedge \mathbf{b}(sub_{gh}) \\
\mathrm{post}_{\mathtt{confirm}} & = & \bigwedge_{tt \in \mathbf{b}} sub_{gh} = tt \rightarrow (\forall_{\mathbf{d}} x, y. \forall_{\mathbf{b}} s. \forall_{\mathbf{c}} c.\ NS(x, y, s, c) \leftrightarrow \psi_{tt}) \\
 & & \\
\psi_{false} & = & cd \neq c \wedge NS_{gh}(x, y, s, c) \\
\psi_{true} & = & cd \neq c \wedge NS_{gh}(x, y, s, c) \vee (c = nil \wedge s = true \wedge NS_{gh}(x, y, false, cd))
\end{array}
$$

Figure 2 Running Example: Pre- and post-conditions. `g` is either `subscribe` or `unsubscribe`.

set $\mathbf{dom}^{\mathrm{U}}$ called the *unbounded domain*. The domains are disjoint. We assume three disjoint countably infinite sets: the set of attributes **att**, the set of relation names **relnames**, and the set of variables **SQLvars**. We assume a function $\mathbf{sort} : \mathbf{att} \rightarrow \{\mathbf{dom}^{\mathrm{U}}, \mathbf{dom}^{\mathrm{B}}_1, \ldots, \mathbf{dom}^{\mathrm{B}}_s\}$. A ***table*** or a ***relation schema*** is a relation name and a finite sequence of attributes. The attributes are the names of the columns of the table. The ***arity*** $\mathrm{ar}(R)$ of a relation schema R is the number of its attributes. A ***database schema*** is a non-empty finite set of tables.

A ***database instance*** $\mathcal{I}$ of a database schema $\mathbf{R}$ is a many-sorted structure with finite domains $\mathbf{dom}_0 \subseteq \mathbf{dom}^{\mathrm{U}}$ and $\mathbf{dom}_j = \mathbf{dom}^{\mathrm{B}}_j$ for $1 \leq j \leq s$. We denote by $\mathbf{sort}_{\mathcal{I}}$ the function obtained from **sort** by setting $\mathbf{sort}_{\mathcal{I}}(att) = \mathbf{dom}_0$ whenever $\mathbf{sort}(att) = \mathbf{dom}^{\mathrm{U}}$. The relation schema $R = (relname, att_1, \ldots, att_e)$ is interpreted in $\mathcal{I}$ as a relation $R^{\mathcal{I}} \subseteq \mathbf{sort}_{\mathcal{I}}(att_1) \times \cdots \times \mathbf{sort}_{\mathcal{I}}(att_e)$. A ***row*** is a tuple in a relation $R^{\mathcal{I}}$.

A database schema $\mathbf{R}$ is **valid** for SmpSQL if for all relation schemas R with attributes $att_1, \ldots, att_e$ in $\mathbf{R}$, there are at most two attributes att_j for which $\mathbf{sort}(att_j) = \mathbf{dom}^{\mathrm{U}}$. In the sequel we assume that all database schemas are valid. The SmpSQL commands will be allowed to use variables from **SQLvars**. We denote members of **SQLvars** by p, p_1, etc.

3.1.2 Queries in SmpSQL

Given a relation schema R and attributes $att_1, \ldots, att_n$ of R, the syntax of SELECT is:

$$
\begin{array}{lcl}
\langle \mathtt{Select} \rangle & ::= & \mathtt{SELECT}\ \mathtt{att}_{\mathtt{a}_1}, \ldots, \mathtt{att}_{\mathtt{a}_i}\ \mathtt{FROM}\ \mathtt{R}\ \mathtt{WHERE}\ \langle \mathtt{Condition} \rangle \\
\langle \mathtt{Condition} \rangle & ::= & \mathtt{att}_{\mathtt{b}_1}, \ldots, \mathtt{att}_{\mathtt{b}_j}\ \mathtt{IN}\ \langle \mathtt{Select} \rangle \mid \\
 & & \langle \mathtt{Condition} \rangle\ \mathtt{AND}\ \langle \mathtt{Condition} \rangle \mid \\
 & & \langle \mathtt{Condition} \rangle\ \mathtt{OR}\ \langle \mathtt{Condition} \rangle \mid \\
 & & \mathtt{NOT}\ \langle \mathtt{Condition} \rangle \mid \\
 & & \mathtt{att}_{\mathtt{m}} = \mathtt{p}
\end{array}
$$

where p is a variable and $1 \leq m, a_1, \ldots, a_i, b_1, \ldots, b_j \leq n$. The semantics of $\langle Select \rangle$ is the set of tuples from the projection of R on $att_{a_1}, \ldots, att_{a_i}$ which satisfy $\langle Condition \rangle$. The

condition $\texttt{att}_\texttt{m} = \texttt{p}$ indicates that the set of rows of R in which the attribute att_m has value p is selected. The condition $\texttt{att}_{\texttt{b}_1}, \ldots, \texttt{att}_{\texttt{b}_i}$ IN $\langle\texttt{Select}\rangle$ selects the set of rows of R in which $att_{b_1}, \ldots, att_{b_i}$ are mapped to one of the tuples queried in the nested query $\langle Select\rangle$.

3.1.3 Data-manipulating commands in SmpSQL

SmpSQL supports the three primitive commands INSERT, UPDATE, and DELETE.

Let R be a relation schema with attributes $att_1, \ldots, att_n$. Let $p, p_1, \ldots, p_n$ be variables from **SQLvars**. The syntax of the primitive commands is:

$$\begin{aligned}
\langle\texttt{Insert}\rangle &::= \texttt{INSERT (p}_1, \ldots, \texttt{p}_\texttt{n}\texttt{) INTO R} \\
\langle\texttt{Update}\rangle &::= \texttt{UPDATE R SET att}_\texttt{m} = \texttt{p WHERE } \langle\texttt{Condition}\rangle \\
\langle\texttt{Delete}\rangle &::= \texttt{DELETE FROM R WHERE } \langle\texttt{Condition}\rangle
\end{aligned}$$

The semantics of INSERT, UPDATE and DELETE is given in the natural way. We allow update commands which set several attributes simultaneously. We assume that the data manipulating commands are used in a domain-correctness fashion, i.e. INSERT and UPDATE may only assign values from $\mathbf{sort}(att_k)$ to any attribute att_k.

3.2 The script language SmpSL

3.2.1 Data model of SmpSL

The data model of SmpSL extends that of SmpSQL with constant names and additional relation schemas. We assume a countably infinite set of constant names **connames**, which is disjoint from $\mathbf{att}, \mathbf{dom}^{\mathrm{U}}, \mathbf{dom}^{\mathrm{B}}_1, \ldots, \mathbf{dom}^{\mathrm{B}}_s, \mathbf{relnames}$ but contains **SQLvars**.

A ***state schema*** is a database schema $\mathbf{R}$ expanded with a tuple of constant names $\overline{const}$. A ***state*** interprets a state schema. It consists of a database instance $\mathcal{I}$ expanded with a tuple of universe elements $\overline{const}^{\mathcal{I}}$ interpreting $\overline{const}$. In programs, the constant names play the role of local variables, domain constants (e.g. *true* and *true*) and of inputs to the program[1].

3.2.2 SmpSL programs

The syntax of SmpSL is given by

$$\begin{aligned}
\langle\texttt{Program}\rangle &::= \langle\texttt{Command}\rangle \mid \langle\texttt{Program}\rangle\,;\langle\texttt{Command}\rangle \\
\langle\texttt{Command}\rangle &::= \langle\texttt{Insert}\rangle \mid \langle\texttt{Update}\rangle \mid \langle\texttt{Delete}\rangle \mid \texttt{R} = \langle\texttt{Select}\rangle \mid \bar{\texttt{d}} = \texttt{CHOOSE R} \mid \\
&\quad\ \texttt{if (cond)}\ \langle\texttt{Program}\rangle \mid \texttt{if (cond) exit} \mid \\
&\quad\ \texttt{if (cond)}\ \langle\texttt{Program}\rangle\ \texttt{else}\ \langle\texttt{Program}\rangle
\end{aligned}$$

Every data-manipulating command C of SmpSQL is a SmpSL command. The semantics of C in SmpSL is the same as in SmpSQL, with the caveat that the variables receive their values from their interpretations (as constant names) in the state, and C is only legal if all the variables of C indeed appear in the state schema as constant names.

The command $\texttt{R} = \langle\texttt{Select}\rangle$ assigns the result of a SmpSQL query to a relation schema $R \in \mathbf{R}$ whose arity and attribute sorts match the select query. Executing the command in a state $(\mathcal{I}, \overline{const}^{\mathcal{I}})$ sets $R^{\mathcal{I}}$ to the relation selected by S, leaving the interpretation of all other

[1] We deviate from [1] in the treatment of constants in that we do not assume that constant names are always interpreted as *distinct* members of $\mathbf{dom}^{\mathrm{U}}$. This is so since several program variables or inputs can have the same value.

names unchanged. The variables in the query receive their values from their interpretations in the state, and for the command to be legal, all variables in the query must appear in the state schema as constant names.

Given a relation schema $R \in \mathbf{R}$ with attributes $att_1, \ldots, att_n$ and a tuple $\bar{d} = (d_1, \ldots, d_n)$ of constant names from $\overline{const}$, $\bar{\mathtt{d}}$ = `CHOOSE R` is a SmpSL command. If $R^{\mathcal{I}}$ is empty, the command has no effect. If $R^{\mathcal{I}}$ is not empty, executing this command sets $(d_1^{\mathcal{I}}, \ldots, d_n^{\mathcal{I}})$ to the value of a non-deterministically selected row from $R^{\mathcal{I}}$.

The branching commands have the natural semantics. Two types of branching conditions `cond` are allowed: (R = `empty`) and (R != `empty`), which check whether $R^{\mathcal{I}}$ is the empty set, and $(c_1 = c_2)$ and $(c_1 ! = c_2)$, which check whether $c_1^{\mathcal{I}} = c_2^{\mathcal{I}}$.

3.3 Verification of SmpSL programs

3.3.1 SQL and FO

It is well-established that a core part of SQL is captured by FO by Codd's classical theorem relating the expressive power of relational algebra to relational calculus. While SQL goes beyond FO in several aspects, such as aggregation, grouping, and arithmetic operations (see [27]), these aspects are not allowed in SmpSQL. Hence, FO is especially suited for reasoning about SmpSQL and SmpSL.

The notions of state schema and state fit naturally in the syntax and semantics of FO. In the sequel, a ***vocabulary*** is a tuple of relation names and constant names. Every state schema $\mathbf{R}$ is a vocabulary. A state $(\mathcal{I}, \overline{const}^{\mathcal{I}})$ interpreting a state schema $\mathbf{R}$ and a tuple of constant names $\overline{const}$ is an $\langle \mathbf{R}, \overline{const} \rangle$-structure.

3.3.2 Hoare verification of SmpSL programs and weakest precondition

Hoare logic is a standard program verification methodology [23]. Let P be a SmpSL program and let φ_{pre} and φ_{post} be FO-sentences. A ***Hoare triple*** is of the form $\{\varphi_{pre}\}P\{\varphi_{post}\}$. A Hoare triple is a *contract* relating the state before the program is run with the state afterward. The goal of the verification process is to prove that the contract is correct.

Our method of proving that a Hoare triple is valid reduces the problem to that of finite satisfiability of a FO-sentence. We compute the ***weakest precondition*** $\mathrm{wp}[\![P]\!]\varphi_{post}$ of φ_{post} with respect to the program P. The weakest precondition transformer was introduced in Dijkstra's classic paper [17], c.f. [24]. Let $\mathcal{A}_P$ denote the state after executing P on the initial state $\mathcal{A}$. The main property of the weakest precondition is: $\mathcal{A}_P \models \varphi_{post}$ iff $\mathcal{A} \models \mathrm{wp}[\![P]\!]\varphi_{post}$. Using $\mathrm{wp}[\![\cdot]\!]$ we can rephrase the problem of whether the Hoare triple $\{\varphi_{pre}\}P\{\varphi_{post}\}$ is valid in terms of FO reasoning on finite structures: *Is the FO-sentence* $\varphi_{pre} \to \mathrm{wp}[\![P]\!]\varphi_{post}$ *a tautology?* Equivalently, *is the FO-sentence* $\varphi_{pre} \wedge \neg\mathrm{wp}[\![P]\!]\varphi_{post}$ *unsatisfiable?* Section 3.3.3 discusses the resulting FO reasoning task.

We describe the computation of the weakest precondition inductively for SmpSQL and SmpSL. The weakest precondition for SmpSQL is given in Fig. 3, and for SmpSL in Fig. 4. For SmpSQL conditions, $[\![\cdot]\!]^R$ is a formula with n free first-order variables $v_1, \ldots, v_n$ for a conditional expression in the context of relation schema R of arity n. $[\![$`SELECT` $\cdots$ `FROM R` $\cdots]\!]$ is also a formula with free variables $v_1, \ldots, v_n$ describing the rows selected by the SELECT query. The rules $\mathrm{wp}[\![s]\!]Q$ transform a (closed) formula Q, which is a postcondition of the command s, into a (closed) formula expressing the weakest precondition. The notation $\psi[t/v]$ indicates substitution of all free occurrences of the variable v in ψ by the term t.

The notation $\psi\big[\theta(\alpha_1, \ldots, \alpha_n)/R(\alpha_1, \ldots, \alpha_n)\big]$ indicates that any atomic sub-formula of ψ of the form $R(\alpha_1, \ldots, \alpha_n)$ (for any $\alpha_1, \ldots, \alpha_n$) is replaced by $\theta(\alpha_1, \ldots, \alpha_n)$ (with the same

$$
\begin{array}{lcl}
[\![\texttt{att}_\texttt{i} = \texttt{c}]\!]^R & \widehat{=} & v_i = c \\
[\![\texttt{att}_{\texttt{b}_1}, \ldots, \texttt{att}_{\texttt{b}_\texttt{j}} \texttt{ IN } \texttt{S}_1]\!]^R & \widehat{=} & [\![S_1]\!][v_{b_k}/v_k : 1 \le k \le j]
\end{array}
$$

$$
\begin{array}{lcl}
[\![\texttt{cond}_1 \texttt{ AND } \texttt{cond}_2]\!]^R & \widehat{=} & [\![cond_1]\!]^R \wedge [\![cond_2]\!]^R \\
[\![\texttt{cond}_1 \texttt{ OR } \texttt{cond}_2]\!]^R & \widehat{=} & [\![cond_1]\!]^R \vee [\![cond_2]\!]^R \\
[\![\texttt{NOT } \texttt{cond}_1]\!]^R & \widehat{=} & \neg [\![cond_1]\!]^R
\end{array}
$$

$$
\begin{array}{l}
[\![\texttt{SELECT att}_{\texttt{a}_1}, \ldots, \texttt{att}_{\texttt{a}_\texttt{i}} \texttt{ FROM R WHERE cond}]\!] \;\widehat{=} \\
\quad (\exists v_{a_{i+1}}, \ldots, v_{a_n} R(\bar{v}) \wedge [\![cond]\!]^R)[v_\ell / v_{a_\ell} : 1 \le \ell \le i] \\
\text{where } \{a_1, \ldots, a_n\} = \{1, \ldots, n\}
\end{array}
$$

$$
\text{wp}[\![\texttt{INSERT } (\texttt{c}_1, \ldots, \texttt{c}_\texttt{n}) \texttt{ INTO R}]\!]Q \;\widehat{=}\; Q\big[R(\bar{\alpha}) \vee \textstyle\bigwedge_{i=1}^n \alpha_i = c_i \;/\; R(\bar{\alpha})\big]
$$

$$
\text{wp}[\![\texttt{DELETE FROM R WHERE cond}]\!]Q \;\widehat{=}\; Q\big[R(\bar{\alpha}) \wedge \neg [\![cond]\!]^R[\alpha_i/v_i : 1 \le i \le n] \;/\; R(\bar{\alpha})\big]
$$

$$
\begin{array}{l}
\text{wp}[\![\texttt{UPDATE R SET att}_\texttt{j} = \texttt{c WHERE cond}]\!]Q \;\widehat{=} \\
\quad Q\big[R(\bar{\alpha}) \wedge \neg [\![cond]\!]^R[\alpha_i/v_i : 1 \le i \le n] \vee \\
\qquad \exists v_j R(\overline{\alpha^j}) \wedge [\![cond]\!]^R[\alpha_i^j/v_i : 1 \le i \le n] \wedge \alpha_j = c / R(\bar{\alpha})\big]
\end{array}
$$

Figure 3 Rules for weakest precondition for SmpSQL basic commands. We denote by R a relation schema with attributes $\langle att_1, \ldots, att_n \rangle$. We write α_i^j for α_i if $i \neq j$, and for v_i if $i = j$. We denote $\bar{v} = (v_1, \ldots, v_n)$, $\bar{\alpha} = (\alpha_1, \ldots, \alpha_n)$, and $\overline{\alpha^j} = (\alpha_1^j, \ldots, \alpha_n^j)$. Note that each of the last three rows $Q\big[expr(\alpha)/R(\bar{\alpha})\big]$ substitutes every occurrence of R with an updated expression *expr*.

$\alpha_1, \ldots, \alpha_n)$. The formula $\theta(v_1, \ldots, v_n)$ has n free variables, and $\theta(\alpha_1, \ldots, \alpha_n)$ is obtained by substituting each v_i into α_i. The α_i may be variables or constant names.

The weakest precondition of a SmpSL program is obtained by applying the weakest precondition of its commands.

3.3.3 The specification logic FO^2_{BD} and decidability of verification

As discussed in Section 3.3.2, using the weakest precondition, the problem of verifying Hoare triples can be reduced to the problem of checking satisfiability of a FO-sentence by a finite structure. While this problem is not decidable in general by Trakhtenbrot's theorem, it is decidable for a fragment of FO we denote FO^2_{BD}, which extends the classical *two-variable fragment* FO^2. The logic FO^2 is the set of all FO formulas which use only variables the variables x and y. The vocabularies of FO^2-sentences are not allowed function names, only relation and constant names. Note FO^2 cannot express that a relation name is interpreted as a function. FO^2 contains the equality symbol $=$. FO^2_{BD} extends FO^2 by allowing quantification on an unbounded number of variables, under the restriction that all variables besides from x and y range over the bounded domains only.

FO^2_{BD} is the language of our invariants and pre- and postconditions, see Eq. (1) and Fig. 2 in Section 2. An important property of FO^2_{BD} is that it is essentially closed under taking weakest precondition according to Figs. 3 and 4 since all relation schemas in a (valid) database schema have at most 2 attributes whose sort is $\mathbf{dom}^{\text{U}}$. We reduce the task of reasoning over FO^2_{BD} to reasoning over FO^2.

▶ **Theorem 1.** *Let $\{\varphi_{pre}\}P\{\varphi_{post}\}$ be a Hoare triple such that both φ_{pre} and φ_{post} belong to FO^2_{BD}. The problem of deciding whether $\{\varphi_{pre}\}P\{\varphi_{post}\}$ is valid is decidable.*

$$
\begin{array}{lcl}
[\![\mathtt{c}_1 = \mathtt{c}_2]\!] & \widehat{=} & c_1 = c_2 \\
[\![\mathtt{c}_1 \mathrel{!}= \mathtt{c}_2]\!] & \widehat{=} & c_1 \neq c_2 \\
\\
[\![\mathtt{R} \mathrel{!} = \mathtt{empty}]\!] & \widehat{=} & \exists v_1, \ldots, v_n R(v_1, \ldots, v_n) \\
[\![\mathtt{R} = \mathtt{empty}]\!] & \widehat{=} & \neg\exists v_1, \ldots, v_n R(v_1, \ldots, v_n) \\
\\
\mathrm{wp}[\![\mathtt{R} = \mathtt{SELECT} \ \cdots]\!]Q & \widehat{=} & Q\big[[\![\mathtt{SELECT} \ \cdots]\!](\alpha_1, \ldots, \alpha_n) \big/ R(\alpha_1, \ldots, \alpha_n)\big] \\
\mathrm{wp}[\![(\mathtt{d}_1, \ldots, \mathtt{d}_n) = \mathtt{CHOOSE\ R}]\!]Q & \widehat{=} & \forall u_1, \ldots, u_n \big(R(u_1, \ldots, u_n) \rightarrow Q[u_i/d_i : 1 \leq i \leq n]\big) \\
\mathrm{wp}[\![\mathtt{if\ cond\ s}_1 \mathtt{\ else\ s}_2]\!]Q & \widehat{=} & (\neg[\![cond]\!] \wedge \mathrm{wp}[\![s_2]\!]Q) \vee ([\![cond]\!] \wedge \mathrm{wp}[\![s_1]\!]Q)
\end{array}
$$

Figure 4 Rules for weakest precondition construction for SmpSL basic commands. The weakest precondition of `if cond exit`; $\mathtt{s}_2$ is the same as that of `if !cond` $\mathtt{s}_2$.

Proof (sketch). By Section 3.3.2, $\{\varphi_{pre}\}P\{\varphi_{post}\}$ is valid iff $\theta = \neg(\varphi_{pre} \wedge \neg \mathrm{wp}[\![P]\!]\varphi_{post})$ is satisfiable by a finite structure. We assume for simplicity that in all the tables, the sort of the first and second attributes att_1 and att_2 is $\mathbf{dom}^{\mathrm{U}}$. By Figs. 3 and 4, the only variables ranging over the unbounded domain are v_1 and v_2. Let θ' be the $\mathrm{FO}^2_{\mathrm{BD}}$ sentence obtained from θ by substituting v_1 and v_2 with x and y respectively, and restricting the range of the quantifiers appropriately: for a command manipulating or querying a table R with attributes $att_1, \ldots, att_n$ in Figs. 3 and 4, each quantifier $\forall v_k$ or $\exists v_k$ is replaced with $\forall_{\mathbf{sort}(att_k)} v_k$ or $\exists_{\mathbf{sort}(att_k)} v_k$. We compute an FO^2 sentence θ'' which is equivalent to θ' by hardcoding the bounded domains. Every table T with an attribute att with $\mathbf{sort}(att) = \mathbf{dom}^{\mathrm{B}}_j$ of size d is replaced with d tables $T_1, \ldots, T_d$ without the attribute att. This change is reflected in θ'', e.g. existential quantification is replaced with disjunction. By the decidability of finite satisfiability of FO^2-sentences, we get that The problem of deciding whether $\{\varphi_{pre}\}P\{\varphi_{post}\}$ is valid is reduced to the decidability of validity for FO^2-sentences. ◀

4 FO² Reasoning

4.1 The bounded model property of FO²

Section 4 is devoted to our algorithm for FO^2 finite satisfiability. The main ingredient for this algorithm is the *bounded model property*, which guarantees that if an $\mathrm{FO}^2(\tau)$ sentence ϕ over vocabulary τ is satisfiable by any τ-structure – finite or infinite – it is satisfiable by a finite τ-structure whose cardinality is bounded by a computable function of ϕ. Grädel, Kolaitis and Vardi [21] computed an asymptotically-tight exponential bound $bnd(\phi)$, and based on it gave a NEXPTIME algorithm. The algorithm non-deterministically guesses $t \leq bnd(\phi)$ and a τ-structure $\mathcal{A}$ with universe of size t, then checks whether $\mathcal{A}$ satisfies ϕ, and answers accordingly.

4.2 Finite satisfiability using a SAT solver

Our algorithm for FO^2 finite satisfiability reduces the problem of finding a satisfying model of cardinality bounded by bnd to the satisfiability of a propositional Boolean formula in Conjunctive Normal Form CNF, which is then solved using a SAT solver. The bound in [21] is given for formulas in Scott Normal Form (SNF) only. We use a refinement of SNF we call *Skolemized Scott Normal Form (SSNF)*. The CNF formula we generate encodes the semantics of the sentence ψ on a structure whose universe cardinality is bounded by bnd. An early precursor for the use of a SAT solver for finite satisfiability is [28].

4.2.1 Skolemized Scott Normal Form

An FO^2-sentence is in *Skolemized Scott Normal Form* if it is of the form

$$\forall x \forall y \left(\alpha(x,y) \wedge \bigwedge_{i=1}^{m} F_i(x,y) \rightarrow \beta_i(x,y) \right) \wedge \bigwedge_{i=1}^{m} \forall x \exists y \, F_i(x,y) \tag{2}$$

where α and β_i, $i = 1, \ldots, m$, are quantifier-free formulas which do not contain any F_j, $j = 1, \ldots, m$. Note that F_i are relation names.

▶ **Proposition 2.** *Let τ be a vocabulary and ϕ be a FO$^2(\tau)$-sentence. There are polynomial-time computable vocabulary $\sigma \supseteq \tau$ and FO$^2(\sigma)$-sentence ψ such that*

(a) *ψ is in SSNF;*
(b) *The set of cardinalities of the models of ϕ is equal to the corresponding set for ψ; and*
(c) *The size of ψ is linear in the size of ϕ.*

Proposition 2 follows from the discussion before Proposition 3.1 in [21], by applying an additional normalization step converting SNF sentences to SSNF sentences.[2][3]

4.2.2 The CNF formula

Given the sentence ψ in SSNF from Eq. (2) and a bound $bnd(\psi)$, we build a CNF propositional Boolean formula C_ψ which is satisfiable iff ψ is satisfiable. The formula C_ψ will serve as the input to the SAT solver. First we construct a related CNF formula B_ψ. The crucial property of B_ψ is that it is satisfiable iff ψ is satisfiable by a model *of cardinality exactly* $bnd(\psi)$.

It is convenient to assume ψ does not contain constants. If ψ did contain constants c, they could be replaced by unary relations U_c of size 1.

We start by introducing the variables and clauses which guarantee that B_ψ encodes a structure with the universe $\{1, \ldots, bnd(\psi)\}$. Later, we will add clauses to guarantee that this structure satisfies ψ. For every unary relation name U in ψ and $\ell_1 \in \{1, \ldots, bnd(\psi)\}$, let v_{U,ℓ_1} be a propositional variable. For every binary relation name R in ψ and $\ell_1, \ell_2 \in \{1, \ldots, bnd(\psi)\}$, let v_{R,ℓ_1,ℓ_2} be a propositional variable. The variables v_{U,ℓ_1} and v_{R,ℓ_1,ℓ_2} encode the interpretations of the unary and binary relation names U and R in the straightforward way (defined precisely below). Let V_ψ be the set of all variables v_{U,ℓ_1} and v_{R,ℓ_1,ℓ_2}.

Given an assignment S to the variables of V_ψ we define the unique structure $\mathcal{A}_\mathrm{S}$ as follows:

1. The universe A_S of $\mathcal{A}_\mathrm{S}$ is $\{1, \ldots, bnd(\psi)\}$;
2. An unary relation name U is interpreted as the set $\{\ell_1 \in A_\mathrm{S} \mid \mathrm{S}(v_{U,\ell_1}) = True\}$;
3. A binary relation name R is interpreted as the set $\{(\ell_1, \ell_2) \in A_\mathrm{S}^2 \mid \mathrm{S}(v_{R,\ell_1,\ell_2}) = True\}$;

For every structure $\mathcal{A}$ with universe $\{1, \ldots, bnd(\psi)\}$, there is S such that $\mathcal{A} = \mathcal{A}_\mathrm{S}$.

Before defining B_ψ precisely we can already state the crucial property of B_ψ:

▶ **Proposition 3.** *ψ is satisfiable by a structure with universe $\{1, \ldots, bnd(\psi)\}$ iff B_ψ is satisfiable.*

The formula B_ψ is the conjunction of B^{eq}, $B^{\forall\exists}$, and $B^{\forall\forall}$, described in the following.

---

[2] The word Skolemized is used in reference to the standard *Skolemization* process of eliminating existential quantifiers by introducing fresh function names called *Skolem functions*. In our case, since function names are not allowed in our fragment, we introduce the relation names F_i, to which we refer as *Skolem relations*. Moreover, we cannot eliminate the existential quantifiers entirely, but only simplify the formulas in their scope to the atoms $F_i(x, y)$.

[3] The linear size of ψ uses our relation symbols have arity at most 2 to get rid of a log factor in [21].

The equality symbol. The equality symbol requires special attention. Let

$$B^{\text{eq}} = \bigwedge_{1 \le \ell_1 \ne \ell_2 \le m} (\neg v_{=,\ell_1,\ell_2}) \wedge \bigwedge_{1 \le \ell \le m} v_{=,\ell,\ell}$$

B^{eq} enforces that the equality symbol is interpreted correctly as the equality relation on universe elements.

The $\forall\exists$-conjuncts. For every conjunct $\forall x \exists y\, F_i(x,y)$ and $1 \le \ell_1 \le bnd(\psi)$, let $B^{\forall\exists}_{i,\ell_1}$ be the clause $\bigvee_{\ell_2=1}^{bnd(\psi)} v_{F_i,\ell_1,\ell_2}$. This clause says that there is at least one universe element ℓ_2 such that $\mathcal{A}_{\text{S}} \models F(\ell_1,\ell_2)$. Let

$$B^{\forall\exists} = \bigwedge_{1 \le i \le m} \bigwedge_{1 \le \ell_1 \le bnd(\psi)} B^{\forall\exists}_{i,\ell_1}$$

For every truth-value assignment S to V_ψ, $\mathcal{A}_{\text{S}}$ satisfies $\bigwedge_{i=1}^{m} \forall x \exists y\, F_i(x,y)$ iff S satisfies $B^{\forall\exists}$.

The $\forall\forall$-conjunct. Let $\forall x \forall y\, \alpha'$ be the unique $\forall\forall$-conjunct of ψ. For every $1 \le \ell_1, \ell_2 \le bnd(\psi)$, let α''_{ℓ_1,ℓ_2} denote the propositional formula obtained from the quantifier-free FO^2 formula α' by substituting every atom a with the corresponding propositional variable for ℓ_1 and ℓ_2 as follows:

$$\begin{array}{llllllllll} U(x) & \mapsto & v_{U,\ell_1}, & & R(y,y) & \mapsto & v_{R,\ell_2,\ell_2}, & & R(x,x) & \mapsto & v_{R,\ell_1,\ell_1} \\ U(y) & \mapsto & v_{U,\ell_2}, & & R(x,y) & \mapsto & v_{R,\ell_1,\ell_2}, & & R(y,x) & \mapsto & v_{R,\ell_2,\ell_1} \end{array}$$

Let $B^{\forall\forall}_{\ell_1,\ell_2}$ be the Tseitin transformation of α''_{ℓ_1,ℓ_2} to CNF [36], see also [6, Chapter 2]. The Tseitin transformation introduces a linear number of new variables of the form $u^{\gamma}_{\ell_1,\ell_2}$, one for each sub-formula γ of α''_{ℓ_1,ℓ_2}. The transformation guarantees that, for every assignment S of V_ψ, S satisfies α''_{ℓ_1,ℓ_2} iff S can be expanded to satisfy $B^{\forall\forall}_{\ell_1,\ell_2}$. Let

$$B^{\forall\forall} = \bigwedge_{1 \le \ell_1,\ell_2 \le bnd(\psi)} B^{\forall\forall}_{\ell_1,\ell_2}(\ell_1,\ell_2)$$

Note that [21] guarantees only that $bnd(\psi)$ is an *upper bound* on the cardinality of a satisfying model. Therefore, we build a formula C_ψ based on B_ψ such that C_ψ is satisfiable iff ψ is satisfiable by a structure of cardinality *at most* $bnd(\psi)$. The algorithm for finite satisfiability of a FO^2-sentence ϕ consists of computing the SSNF ψ of ϕ and returning the result of a satisfiability check using a SAT solver on C_ψ. Both the number of variables and the number of clauses in $C_{\text{Uni}(\psi)}$ are quadratic in $bnd(\psi)$.

5 Experimental Results

5.1 Details of our tools

The verification condition generator described in Section 3.3.2 is implemented in Java, JFlex and CUP. It is employed to parse the schema, precondition and postcondition and the SmpSL programs. The tool checks that the pre and post conditions are specified in FO^2 and that the scheme is well defined. The SMT-LIB v2 [4] standard language is used as the output format of the verification condition generator. We compare the behavior of our FO^2-solver with Z3 on the verification condition generator output. The validity of the verification condition can be checked by providing its negation to the SAT solver. If the SAT solver exhibits a satisfying

Table 1 Running time comparison for example benchmarks.

	FO^2-solver	Z3
web-subscribe	0.910s	TO
web-unsubscribe	0.741s	OM
firewall	0.876s	OM
conf-bid	0.451s	0.015s
conf-assign	0.369s	0.013s
conf-display	0.992s	0.016s

incorrect

	FO^2-solver	Z3
web-subscribe	1.04s	0.02s
web-unsubscribe	1.46s	0.02s
firewall	18.50s	0.03s
conf-bid	TO	0.22s
conf-assign	1.196s	0.2s
conf-display	TO	0.16s

correct

assignment then that serves as counterexample for the correctness of the program. If no satisfying assignment exists, then the generated verification condition is valid, and therefore the program satisfies the assertions. The FO^2-solver described in Section 4 is implemented in python and uses pyparsing to parse the SMT-LIB v2 [4] file. The FO^2-solver assumes a FO^2-sentence as input and uses *Lingeling* [5] SAT solver as a base Solver.

5.2 Example applications

We tried our approach with a few programs inspired by real-life applications. The first case study is a simplified version of the newsletter functionality included in the PANDA web administrator, that was already discussed and is shown in Fig. 1.[4]

The second is an excerpt from a firewall that updates a table of which device is allowed to send packets to which other device. The third is a conference management system with a database of papers, and transactions to manage the review process: reviewers first *bid* on papers from the pool of submissions, with a policy that a users cannot bid for papers with which they are conflicted. The chair then *assigns* reviewers to papers by selecting a subset of the bids. At any time, users can ask to *display* the list of papers, with some details, but the system may hide some confidential information, in particular, users should not be able to see the status of papers before the program is made public. We show how our system detects an information flow bug in which the user might learn that some papers were accepted prematurely by examining the session assignments. This bug is based on a bug we observed in a real system. Each example comes with two specifications, one correct and the other incorrect.

The running time in seconds for all of our examples is reported in Table 1. Timeout is set to 60 minutes and denoted as TO. If the solver reaches *out of memory* we mark it as OM. On the set of correct examples, Z3 terminates within milliseconds, while FO^2-solver takes a few seconds and times out on some of them. On the set of incorrect examples, Z3 fails to answer while our solver performs well. Note that correct examples correspond to unsatisfiable FO^2-sentences, while incorrect examples correspond to satisfiable FO^2-sentences.

5.3 Examining scalability

Inflated examples. In order to evaluate scalability to large examples we inflated our base examples. For instance, while the *subscribe* example from Table 1 consisted of the subscription

[4] We omit the confirmation step due to a missing feature in the implementation of the weakest precondition, however the final version of the tool will support the code from Table 1.

of one new email to a mailing-list, Table 2 presents analogous examples based on combining the verification conditions arising from subscribing multiple emails to the mailing-list. The column *multiplier* details the number of individual subscriptions based on which the formula is constructed. The *unsubscribe* and *firewall* example programs are inflated similarly.

We have tested both our FO^2-solver and Z3 on large examples and the results reported in Table 2. The high-level of the results is similar to the case of the small examples. On the incorrect examples set Z3 continues to fail mostly due to running out of memory, though it succeeds on the subscribe example. On the correct examples set Z3 continues to outperform the FO^2-solver.

Artificial examples. In addition, we constructed a set of artificial benchmarks comprising of several families of FO^2-sentences. Each family is parameterized by a number that controls the size of the sentences (roughly corresponding to the number of quantifiers in the sentence). These problems are inspired by combinatorial problems such as graph coloring and paths. We ran experiments using the FO^2-solver and three publicly available solvers: Z3, CVC4 (which are SMT solvers), and Nitpick (a model checker). The results are collected in Table 3.

Scalability of FO^2-solver. We shall conclude that the FO^2-solver, despite being a proof-of-concept prototype implemented in Python with minimal optimizations, handles satisfiable sentences well and also scales well for them. It struggles on unsatisfiable sentences and does not scale well. SMT solvers usually find unsatisfiability proofs much faster, esp. when quantifiers are involved, because they do not have to instantiate all clauses and can terminate as soon as a core set of contradicting clauses is found. This suggests that in practice we may choose to run both FO^2-solver and Z3 in parallel and answer according the first result obtained. We also intend to explore how to improve the performance of our solver in the case of incorrect examples. By construction, whenever FO^2-solver finds a satisfying model, its size is at most 4 times that of the minimal model. (The constant 4 can be decreased or increased.)

Tools and benchmarks online.

1. *FO2Solver*:
   `http://forsyte.at/people/kotek/fo2-solver/`

2. *SmpSL Verification Conditions Generator*:
   `http://forsyte.at/people/kotek/smpsl-verification-conditions-generator/`

3. Benchmarks for FO^2:
   `http://forsyte.at/people/kotek/two-variable-fragment-benchmarks/`

6 Discussion

Related work. Verification of database-centric software systems has received increasing attention in the last decade, see for example the recent survey [15]. Below, we explain how our approach differs from the works surveyed in [15]. [15] assumes the services accessing the database to be provided *a priori* in terms of a local contract given by a pre- and post-condition (see also [31, 26]). The focus of verification then is on the verification of *global* temporal properties of the system, assuming the local contracts. While the services may be

Table 2 Running time comparison on inflated examples.

		FO^2-solver			Z3		
	multiplier	1	10	100	1	10	100
incorrect	subscribe	0.910s	3.84s	785.2s	TO	TO	TO
	unsubscribe	0.741s	1.70s	209.2s	TO	TO	TO
	firewall	0.876s	3.75s	455.7s	OM	OM	OM
correct	subscribe	1.04s	TO	TO	0.02s	0.03s	0.10s
	unsubscribe	1.46s	TO	TO	0.02s	0.03s	0.09s
	firewall	18.50s	TO	TO	0.03s	0.03s	0.11s

Table 3 Running time comparison on artificial benchmarks.

	size	status	Z3	CVC4	Nitpick	FO^2-solver
2col	3	unsat	0m0.037s	0m0.076s	TO	TO
	4	sat	TO	TO	0m7.038s	0m5.433s
	5	unsat	0m0.702s	0m0.477s	TO	TO
	6	sat	TO	TO	0m8.973s	0m9.323s
	10	sat	TO	TO	0m37.944s	0m19.580s
	11	unsat	1m32.664s	0m30.912s	TO	TO
	14	sat	TO	TO	2m13.661s	TO
	40	sat	TO	TO	TO	TO
alternating-paths	2	sat	0m0.049s	TO	0m11.144s	0m1.105s
	100	sat	TO	TO	TO	0m9.671s
alternating-simple-paths	3	sat	TO	TO	TO	0m6.754s
	4	sat	TO	TO	TO	0m10.128s
	7	sat	TO	TO	TO	TO
	10	sat	TO	TO	TO	TO
exponential	3	sat	TO	TO	0m12.255s	0m1.847s
	4	sat	TO	TO	0m15.358s	11m6.482s
one-var-alternating-sat	300	sat	0m0.037s	0m0.497s	0m11.605s	0m9.720s
one-var-alternating-unsat	5	unsat	0m0.026s	0m0.073s	0m22.537s	0m54.198s
one-var-nested-exists-sat	300	sat	0m0.031s	0m0.045s	0m7.132s	0m0.562s
one-var-nested-forall-sat	500	sat	0m0.033s	TO	0m7.183s	0m7.318s
path-unsat	2	unsat	0m0.033s	0m0.044s	TO	1m37.099s
	3	unsat	0m0.030s	0m0.062s	TO	1m35.451s
	6	unsat	0m0.037s	0m0.891s	TO	1m39.209s

automatically synthesized in some cases, e.g. [19, 20, 15, 16], they are often implemented manually (e.g. using a scripting language) and the validity of their contracts needs to be verified. This is the verification problem we target in this paper: we show how to prove the correctness of a single service with regard to its pre- and postcondition. The approaches have orthogonal strengths: The works surveyed in [15] use (modulo reductions) the existential fragment of first-order logic (∃FO) to formalize local changes to the database and allow the verification of LTL properties whose atoms are given by ∃FO-formulae. In contrast, our approach is limited to the verification of local pre- and post-conditions and system invariants. On the other hand we allow universal quantification in our specifications. It is an interesting direction for future work how to extend our approach to more general temporal properties (e.g. as considered in [15]).

Several papers use variations of FO^2 to study verification of programs that manipulate relational information. [8] presents a verification methodology based on FO^2, a description logic and a separation logic for analyzing the shapes and content of in-memory data structures. [33] develops a logic similar to FO^2 to reason about shapes. In both [8] and [33], the focus is

on analysis of dynamically-allocated memory, and databases are not studied. Furthermore, no tools based on these works are available. Our work draws inspiration from [2], which discusses the verification of evolving graph databases based on a description logic related to FO^2 and a dedicated action language. Our work and [2] exhibit some similarity on a technical level but have a different focus: [2] advocates the use of description logic, while we consider the use of a scripting language with embedded SQL to be advantageous because it does not require to learn new syntax (the identification of an appropriate language and SQL fragment is one of the contributions of this paper); establishing the precise technical relationship between our framework and [2] seems possible but requires additional work to be carried out. Further, the verification method suggested by [2] was not implemented. To our knowledge no description logic solver implements reasoning tasks for the description logic counterpart of FO^2 studied in [2], not even solvers for expressive description logics such as SROIQ. The authors of [2] extended their work to description logics with path constrains in [9].

Verification of script programs with embedded queries has revolved around security, see [18]. However, it seems no other work has been done on such programs.

Conclusion and future work. We developed a verification methodology for script programs with access to a relational database via SQL. We isolated a simple but useful fragment SmpSQL of SQL and developed a simple script programming language SmpSL on top of it. We have shown that verifying the correctness of SmpSL programs with respect to specifications in FO^2_{BD} is decidable. We implemented a solver for the FO^2 finite satisfiability problem, and, based on it, a verification tool for SmpSL programs. Our experimental results are very promising and suggest that our approach has great potential to evolve into a mainstream method for the verification of script programs with embedded SQL statements.

While we believe that many of the SQL statements that appear in real-life programs fall into our fragment SmpSQL it is evident that future tools need to consider all of database usage in real-world programs. In future work, we will explore the extension of SmpSL and SmpSQL. Our next goal is to be able to verify large, real-life script programs such as Moodle [29], whose programming language and SQL statements use e.g. some arithmetic or simple inner joins. To do so, we will adapt our approach from the custom-made syntax of SmpSL to a fragment of PHP. We will both explore decidable logics extending FO^2_{BD}, and investigate verification techniques based on undecidable logics including the use of first-order theorem provers such as Vampire [34, 25] and abstraction techniques which guarantee soundness but may result in spurious errors [12]. For dealing with queries with transitive closure, it is natural to consider fragments of Datalog [10].

A natural extension is to consider global temporal specifications in addition to local contracts. Here the goal is to verify properties of the system which can be expressed in a temporal logic such as Linear Temporal Logic LTL [32, 11]. The approach surveyed in [15], which explore global temporal specifications of services given in terms of local contracts, may be a good basis for studying global temporal specifications in our context.

Another research direction which emerges from the experiments in Section 5 is to explore how to improve the performance of our FO^2 solver on unsatisfiable inputs.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

2 Shqiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Managing change in graph-structured data using description logics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 966–973, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8238.

3 Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2032305.2032319.

4 Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard: Version 2.0. In Aarti Gupta and Daniel Kroening, editors, *Satisfiability Modulo Theories (SMT 2010)*, 2010. http://www.SMT-LIB.org.

5 Armin Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT race 2010. FMV report series technical report 10/1, Johannes Kepler University, Linz, Austria, 2010.

6 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

7 Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *International Conference on Interactive Theorem Proving*, pages 131–146. Springer, 2010.

8 Diego Calvanese, Tomer Kotek, Mantas Šimkus, Helmut Veith, and Florian Zuleger. Shape and content. In *Integrated Formal Methods*, pages 3–17. Springer, 2014.

9 Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Verification of evolving graph-structured data under expressive path constraints. In *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, pages 15:1–15:19, 2016. doi:10.4230/LIPIcs.ICDT.2016.15.

10 Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, 1989.

11 Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

12 Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 154–169, 2000. doi:10.1007/10722167_15.

13 Edgar F Codd. *Relational completeness of data base sublanguages*. IBM Corporation, 1972.

14 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

15 Alin Deutsch, Richard Hull, and Victor Vianu. Automatic verification of database-centric systems. *SIGMOD Record*, 43(3):5–17, 2014. doi:10.1145/2694428.2694430.

16 Alin Deutsch, Monica Marcus, Liying Sui, Victor Vianu, and Dayou Zhou. A verifier for interactive, data-driven web applications. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD'05, pages 539–550, New York, NY, USA, 2005. ACM. doi:10.1145/1066157.1066219.

17 Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

18 Michael Felderer, Matthias Büchler, Martin Johns, Achim D Brucker, Ruth Breu, and Alexander Pretschner. Security testing: A survey. *Advances in Computers*, 2015.

19 Mary F. Fernández, Daniela Florescu, Alon Y. Levy, and Dan Suciu. Declarative specification of web sites with Strudel. *VLDB J.*, 9(1):38–55, 2000. doi:10.1007/s007780050082.

20 Daniela Florescu, Valerie Issarny, Patrick Valduriez, and Khaled Yagoub. Weave: A data-intensive web site management system. In *In Proc. of the Conf. on Extending Database Technology (EDBT*, 2000.

21 Erich Grädel, Phokion G Kolaitis, and Moshe Y Vardi. On the decision problem for two-variable first-order logic. *Bulletin of symbolic logic*, 3(01):53–69, 1997.

22 Alessandro Grassi and Marco Nenciarini. Panda - the php-based email administrator. http://panda-admin.sourceforge.net/index.php?mode=home, 2007–2015.

23 Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

24 Ranjit Jhala and Rupak Majumdar. Software model checking. *ACM Computing Surveys (CSUR)*, 41(4):21, 2009.

25 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *Computer Aided Verification*, pages 1–35. Springer, 2013.

26 S. Kumaran, P. Nandi, T. Heath, K. Bhaskaran, and R. Das. Adoc-oriented programming. In *In Symp. on Applications and the Internet (SAINT)*, 2003.

27 Leonid Libkin. Expressive power of SQL. *Theoretical Computer Science*, 296(3):379–404, 2003.

28 William Mccune. A davis-putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, Argonne National Laboratory, 1994.

29 Moodle. http://sourceforge.net/projects/moodle/, 2001–2015.

30 Michael Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.

31 A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42, 2003.

32 Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

33 Arend Rensink. Canonical graph shapes. In David Schmidt, editor, *Programming Languages and Systems*, volume 2986 of *Lecture Notes in Computer Science*, pages 401–415. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24725-8_28.

34 Alexandre Riazanov and Andrei Voronkov. The design and implementation of Vampire. *AI communications*, 15(2, 3):91–110, 2002.

35 Boris Trakhtenbrot. The impossibility of an algorithm for the decidability problem on finite classes. In *Proceedings of the USSR Academy of Sciences*, volume 70, pages 569–572, 1950.

36 Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

37 Hugh E Williams and David Lane. *Web database applications with PHP and MySQL*. O'Reilly Media, Inc., 2004.

Detecting Ambiguity in Prioritized Database Repairing

Benny Kimelfeld[*1], **Ester Livshits**[2], **and Liat Peterfreund**[3]

1 **Technion, Haifa, Israel**
bennyk@cs.technion.ac.il
2 **Technion, Haifa, Israel**
esterliv@cs.technion.ac.il
3 **Technion, Haifa, Israel**
liatpf@cs.technion.ac.il

Abstract

In its traditional definition, a repair of an inconsistent database is a consistent database that differs from the inconsistent one in a "minimal way." Often, repairs are not equally legitimate, as it is desired to prefer one over another; for example, one fact is regarded more reliable than another, or a more recent fact should be preferred to an earlier one. Motivated by these considerations, researchers have introduced and investigated the framework of preferred repairs, in the context of denial constraints and subset repairs. There, a priority relation between facts is lifted towards a priority relation between consistent databases, and repairs are restricted to the ones that are optimal in the lifted sense. Three notions of lifting (and optimal repairs) have been proposed: Pareto, global, and completion.

In this paper we investigate the complexity of deciding whether the priority relation suffices to clean the database unambiguously, or in other words, whether there is exactly one optimal repair. We show that the different lifting semantics entail highly different complexities. Under Pareto optimality, the problem is coNP-complete, in data complexity, for every set of functional dependencies (FDs), except for the tractable case of (equivalence to) one FD per relation. Under global optimality, one FD per relation is still tractable, but we establish Π_2^p-completeness for a relation with two FDs. In contrast, under completion optimality the problem is solvable in polynomial time for every set of FDs. In fact, we present a polynomial-time algorithm for arbitrary conflict hypergraphs. We further show that under a general assumption of transitivity, this algorithm solves the problem even for global optimality. The algorithm is extremely simple, but its proof of correctness is quite intricate.

1998 ACM Subject Classification H.2.4 Systems, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Inconsistent Databases, Preferred Repairs, Data Cleaning, Functional Dependencies, Conflict Hypergraph

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.17

1 Introduction

Managing database inconsistency has received a lot of attention in the past two decades. Inconsistency arises for different reasons and in different applications. For example, in

* Benny Kimelfeld is a Taub Fellow, supported by the Taub Foundation. Kimelfeld's research is also supported by the Israeli Science Foundation, Grants #1295/15 and #1308/15, and the US-Israel Binational Science Foundation, Grant #2014391.

common applications of Big Data, information is obtained from imprecise sources (e.g., social encyclopedias or social networks) via imprecise procedures (e.g., natural-language processing). It may also arise when integrating conflicting data from different sources (each of which can be consistent). Arenas, Bertossi and Chomicki [3] introduced a principled approach to managing inconsistency, via the notions of *repairs* and *consistent query answering*. Informally, a *repair* of an inconsistent database I is a consistent database J that differs from I in a "minimal" way, where *minimality* refers to the *symmetric difference*. In the case of anti-symmetric integrity constraints (e.g., denial constraints and the special case of functional dependencies), such a repair is a *subset repair* (i.e., J is a consistent subinstance of I that is not properly contained in any consistent subinstance of I).

Various computational problems around database repairs have been extensively investigated. Most studied is the problem of computing the *consistent answers* of a query q on an inconsistent database I; these are the tuples in the intersection $\bigcap\{q(J) : J \text{ is a repair of } I\}$ [3, 27]. Hence, in this approach inconsistency is handled at query time by returning the tuples that are guaranteed to be in the result no matter which repair is selected. Another well studied question is that of *repair checking* [1]: given instances I and J, determine whether J is a repair of I. Depending on the type of repairs and integrity constraints, these problems may vary from tractable to highly intractable complexity classes [4].

In the above framework, all repairs of a given database instance are taken into account and treated on a par with each other. There are situations, however, in which it is natural to prefer one repair over another [16, 8, 34, 33]. For example, this is the case if one source is regarded more reliable than another (e.g., enterprise data vs. Internet harvesting, precise vs. imprecise sensing equipment, etc.) or if available timestamp information implies that a more recent fact should be preferred over an earlier fact. Recency may be implied not only by timestamps, but also by evolution semantics; for example, "divorced" is likely to be more updated than "single," and similarly is "Sergeant" compared to "Private." (See [15] for a comprehensive study of data quality.) Motivated by these considerations, Staworko, Chomicki and Marcinkowski [34, 33] introduced the framework of *preferred* repairs, where a *priority* relation between conflicting facts distinguishes a set of *preferred* repairs.

Specifically, the notion of *Pareto optimality* and that of *global optimality* are based on two different notions of *improvement*—the property of one consistent subinstance being preferred to another. Improvements are basically lifting of the priority relation from facts to consistent subinstances; J is an improvement of K if $J \setminus K$ contains a fact that is better than all those in $K \setminus J$ (in the Pareto semantics), or if for every fact in $K \setminus J$ there exists a better fact in $J \setminus K$ (in the global semantics). In each of the two semantics, an *optimal repair* is a repair that cannot be improved. A third semantics proposed by Staworko et al. [33] is that of a *completion-optimal* repair, which is a globally optimal repair under some extension of the priority relation into a *total* relation. In this paper, we refer to these preferred repairs as *p-repair*, *g-repair* and *c-repair*, respectively.

Fagin et al. [13] have built on the concept of preferred repairs (in conjunction with the framework of *document spanners* [14]) to devise a language for declaring *inconsistency cleaning* in text information-extraction systems. They have shown there that preferred repairs capture ad-hoc cleaning operations and strategies of some prominent existing systems for text analytics [2, 9].

Staworko et al. [33] showed several results on preferred repairs. For example, every c-repair is also a g-repair, and every g-repair is also a p-repair. They also showed that p-repair and c-repair checking are solvable in polynomial time (under data complexity) in the case of denial constraints, and that there is a set of functional dependencies (FDs) for

which g-repair checking is coNP-complete. Later, Fagin et al. [12] extended that hardness result to a full dichotomy in complexity over all sets of FDs: g-repair checking is solvable in polynomial time whenever the set of FDs is equivalent to a single FD or two key constraints per relation; in every other case, the problem is coNP-complete.

While the classic complexity problems studied in the theory of repairs include repair checking and consistent query answering, the presence of repairs gives rise to the *determinism problem*, which Staworko et al. [33] refer to as *categoricity*: determine whether the provided priority relation suffices to clean the database unambiguously, or in other words, decide whether there is exactly one optimal repair. The problem of repairing uniqueness (in a different repair semantics) is also referred to as *determinism* by Fan et al. [18]. In this paper, we study the three variants of this computational problem, under the three optimality semantics Pareto, global and completion, and denote them as *p-categoricity*, *g-categoricity* and *c-categoricity*, respectively.

It is known that under each of the three semantics there is always at least one preferred repair, and Staworko et al. [33] present a polynomial-time algorithm for finding such a repair. (We recall this algorithm in Section 3.) Hence, the categoricity problem is that of deciding whether the output of this algorithm is the only possible preferred repair. As we explain next, it turns out that each of the three variants of the problem entails quite a unique picture of complexity.

For p-categoricity, we focus on integrity constraints that are FDs, and establish the following dichotomy in data complexity. For a relational schema with a set Δ of FDs:

- If Δ associates (up to equivalence) a single FD with every relation symbol, then p-categoricity is solvable in polynomial time.
- In *any other case*, p-categoricity is coNP-complete.

For example, with the relation symbol $R(A, B, C)$ and the FD $A \to B$, p-categoricity is solvable in polynomial time; but if we add the dependency $B \to A$ then it becomes coNP-complete. While there have been several dichotomy results on the complexity of problems associated with inconsistent data [29, 12, 26, 36], to the best of our knowledge this paper is the first to establish a dichotomy result for any variant of repair uniqueness identification.

We then turn to investigating c-categoricity, and establish a far more positive picture than the one for p-categoricity. In particular, the problem is solvable in polynomial time for every set of FDs. In fact, we present an algorithm for solving c-categoricity in polynomial time, assuming that constraints are given as an input *conflict hypergraph* [10] (hence, we establish polynomial-time data complexity for various types of integrity constraints, such as *conditional FDs* [5] and *denial constraints* [19].) The algorithm is extremely simple, yet its proof of correctness is quite intricate.

Finally, we explore g-categoricity. We show that in the tractable case of p-categoricity (equivalence to a single FD per relation), g-categoricity is likewise solvable in polynomial time. For example, $R(A, B, C, D)$ with the dependency $A \to B$ has polynomial-time g-categoricity. Nevertheless, we prove that if we add $C \to D$, then g-categoricity becomes Π_2^{p}-complete. We do not complete a dichotomy as in p-categoricity, and leave that open for future work. Lastly, we ask whether *transitivity* of the preference relation makes a difference. We show that the three *semantics* of repairs remain different in the presence of transitivity, yet quite remarkably, the *problems* g-categoricity and c-categoricity are actually the same. Hence, in the presence of transitivity g-categoricity is solvable in polynomial time (even when constraints are given as a conflict hypergraph).

For lack of space, most of the proofs are excluded and will appear in the full version of the paper [23].

Related Work

We are not aware of any work on the complexity of categoricity within the prioritized repairing of Staworko et al. [34]. Fagin et al. [13] investigated a static version of categoricity in the context of text extraction, but the settings and problems are fundamentally different, and so are their complexity results (e.g., Fagin et al. [13] establish undecidability results).

In the framework of *data currency* [16], relations consist of entities with attributes, where each entity may appear in different tuples, every time with possibly different (conflicting) attribute values. A partial order of currency is provided on each attribute. A *completion* of an instance is obtained by completing the partial order on an attribute of every entity, and it defines a *current instance* where each attribute takes its most recent value. In addition, a completion needs to satisfy given (denial) constraints, which may introduce interdependencies among completions of different attributes. Fan et al. [16] have studied the problem of determining whether such a specification induces a single current instance (i.e., the corresponding version of categoricity), and showed that this problem is coNP-complete under data complexity. It is not clear how to simulate their hardness in p-categoricity or g-categoricity, since their hardness is due to the constrains on completions, and these constraints do not have correspondents in our case (beyond the partial orders). A similar argument relates our lower bounds to those in the framework of conflict resolution by Fan Geerts [15, Chapter 7.3], where the focus is on establishing a unique tuple from a collection of conflicting tuples.

Fan et al. [16] show that in the absence of constraints, their categoricity problem can be solved in polynomial time. This tractability result can be used for establishing the tractability side of Theorem 5.1 in the special case where the single FD is a key constraint. In the general case of a single FD, we need to argue about relationships among sets, and moreover, the differences among the three x-categoricity problems matter.

The work on *certain fixes* [18, 17] considers models that are substantially different from the one adopted here, where repairs are obtained by chasing update rules (rather than tuple deletion), and uniqueness applies to chase outcomes (rather than maximal subinstances w.r.t. preference lifting). The problems relevant to our categoricity are the *consistency problem* [18] (w.r.t. guarantees on the consistency of some attributes following certain patterns), and the *determinism* problem [18].

Finally, we remark that there have several dichotomy results on the complexity of problems associated with inconsistent data [29, 12, 26, 36], but to the best of our knowledge this paper is the first to establish a dichotomy result for any variant of repair uniqueness identification. A valid question for future work is whether one can use the techniques of this paper in order to establish a dichotomy in complexity in *any* of the cleaning frameworks studied in past research.

2 Preliminaries

We now present some general terminology and notation that we use throughout the paper.

Signatures and Instances

A (*relational*) *signature* is a finite set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of *relation symbols*, each with a designated positive integer as its *arity*, denoted $\text{arity}(R_i)$. We assume an infinite set Const of *constants*, used as database values. An *instance* I over a signature $\mathcal{R} = \{R_1, \ldots, R_n\}$ consists of finite relations $R_i^I \subseteq \mathsf{Const}^{\text{arity}(R_i)}$, where $R_i \in \mathcal{R}$. We write $[\![R_i]\!]$ to denote the

set $\{1, \ldots, \text{arity}(R_i)\}$, and we refer to the members of $[\![R_i]\!]$ as *attributes* of R_i. If I is an instance over $\mathcal{R}$ and $\mathbf{t}$ is a tuple in R_i^I, then we say that $R_i(\mathbf{t})$ is a *fact of* I. By a slight abuse of notation, we identify an instance I with the set of its facts. For example, $R_i(\mathbf{t}) \in I$ denotes that $R_i(\mathbf{t})$ is a fact of I. As another example, $J \subseteq I$ means that $R_i^J \subseteq R_i^I$ for every $R_i \in \mathcal{R}$; in this case, we say that J is *subinstance* of I.

We use the conventional notation R/k to denote that R is a relation symbol of arity k. In our examples we often name the attributes and refer to them by their names. For instance, in Figure 1a we refer to the relation symbol as $\mathit{CompCEO}(\text{company}, \text{ceo})$ where company and ceo refer to Attributes 1 and 2, respectively. In the case of generic relation symbols, we implicitly name their attributes by capital English letters with the corresponding numeric values; for instance, we may refer to Attributes 1, 2 and 3 of $R/3$ by A, B and C, respectively. We stress that attribute names are not part of our formal model, but are rather used for readability.

Integrity and Inconsistency

Let $\mathcal{R}$ be a signature, and I an instance over $\mathcal{R}$. In this paper we consider two representation systems for integrity constraints. The first is *functional dependencies* and the second is *conflict hypergraphs*.

Let $\mathcal{R}$ be a signature. A *Functional Dependency* (*FD* for short) over $\mathcal{R}$ is an expression of the form $R : X \to Y$, where R is a relation symbol of $\mathcal{R}$, and X and Y are subsets of $[\![R]\!]$. When R is clear from the context, we may omit it and write simply $X \to Y$. A special case of an FD is a *key constraint*, which is an FD of the form $R : X \to Y$ where $X \cup Y = [\![R]\!]$. An FD $R : X \to Y$ is *trivial* if $Y \subseteq X$; otherwise, it is *nontrivial*.

When we are using the alphabetic attribute notation, we may write X and Y by simply concatenating the attribute symbols. For example, if we have a relation symbol $R/3$, then $A \to BC$ denotes the FD $R : \{1\} \to \{2, 3\}$. An instance I over R *satisfies* an FD $R : X \to Y$ if for every two facts f and g over R, if f and g agree on (i.e., have the same values for) the attributes of X, then they also agree on the attributes of Y. We say that I satisfies a set Δ of FDs if I satisfies every FD in Δ; otherwise, we say that I *violates* Δ. Two sets Δ and Δ' of FDs are *equivalent* if for every instance I over $\mathcal{R}$ it holds that I satisfies Δ if and only if it satisfies Δ'. For example, for $R/3$ the sets $\{A \to BC, C \to A\}$ and $\{A \to C, C \to AB\}$ are equivalent.

In this work, a *schema* $\mathbf{S}$ is a pair $(\mathcal{R}, \Delta)$, where $\mathcal{R}$ is a signature and Δ is a set of FDs over $\mathcal{R}$. If $\mathbf{S} = (\mathcal{R}, \Delta)$ and $R \in \mathcal{R}$, then we denote by $\Delta_{|R}$ the restriction of Δ to the FDs $R : X \to Y$ over R.

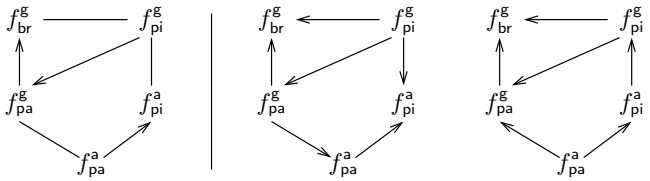
▶ **Example 2.1.** In our first running example, we use the schema $\mathbf{S} = (\mathcal{R}, \Delta)$, defined as follows. The signature $\mathcal{R}$ consists of a single relation $\mathit{CompCEO}(\text{company}, \text{ceo})$, associating companies with their Chief Executive Officers (CEO). Figure 1a depicts an instance I over $\mathcal{R}$. We define Δ to be $\{\text{company} \to \text{ceo}\,,\ \text{ceo} \to \text{company}\}$, stating that in *CompCEO* each company has a single CEO and each CEO manages a single company. Observe that I violates Δ. For example, Google has three CEOs, Alphabet has two CEOs, and each of Pichai and Page is the CEO of two companies.

While FDs define integrity logically, at the level of the signature, a *conflict hypergraph* [10] provides a direct specification of inconsistencies at the instance level, by explicitly stating sets of facts that cannot co-exist. In the case of FDs, the conflict hypergraph is a graph that has an edge between every two facts that violate an FD. Formally, for an instance I over a signature $\mathcal{R}$, a *conflict hypergraph* $\mathcal{H}$ (*for* I) is a hypergraph that has the facts of I as its

CompCEO

	company	ceo
$f^{\mathsf{g}}_{\mathsf{pi}}$	Google	Pichai
$f^{\mathsf{g}}_{\mathsf{pa}}$	Google	Page
$f^{\mathsf{g}}_{\mathsf{br}}$	Google	Brin
$f^{\mathsf{a}}_{\mathsf{pa}}$	Alphabet	Page
$f^{\mathsf{a}}_{\mathsf{pi}}$	Alphabet	Pichai

(a) Inconsistent database of the company-CEO running example



(b) The conflict graph $\mathcal{H}^I_{\mathbf{S}}$ and the priority relation $\succ$ for the company-CEO running example (left), and two completions of $\succ$ (middle and right)

Figure 1

node set. A subinstance J of I is *consistent* with respect to (w.r.t.) $\mathcal{H}$ if J is an *independent set* of $\mathcal{H}$; that is, no hyperedge of $\mathcal{H}$ is a subset of J. We say that J is *maximal* if $J \cup \{f\}$ is inconsistent for every $f \in I \setminus J$. When all the edges of a conflict hypergraph are of size two, we may call it a *conflict graph*.

Recall that conflict hypergraphs can represent inconsistencies for various types of integrity constraints, including FDs, the more general *conditional FDs* [5], and the more general *denial constraints* [19]. In fact, every constraint that is anti-monotonic (i.e., where subsets of consistent sets are always consistent) can be represented as a conflict hypergraph. In the case of denial constraints, the translation from the logical constraints to the conflict hypergraph can be done in polynomial time under *data complexity* (i.e., when the signature and constraints are assumed to be fixed).

Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema, and let I be an instance over $\mathbf{S}$. Recall that $\mathbf{S}$ is assumed to have only FDs. We denote by $\mathcal{H}^I_{\mathbf{S}}$ the conflict graph for I that has an edge between every two facts that violate some FD of $\mathbf{S}$. Note that a subinstance J of I satisfies Δ if and only if J is consistent w.r.t. $\mathcal{H}^I_{\mathbf{S}}$. As an example, the left graph of Figure 1b depicts the graph $\mathcal{H}^I_{\mathbf{S}}$ for our running example; for now, the reader should ignore the directions on the edges, and view the graph as an undirected one. The following example involves a conflict hypergraph that is not a graph.

▶ **Example 2.2.** In our second running example, we use the toy scenario where the signature has a single relation symbol *Follows*/2, where *Follows*(x, y) means that person x follows person y (e.g., in a social network). We have two sets of people: a_i for $i = 1, 2, 3$, and b_j for $j = 1, \ldots, 5$. All facts have the form *Follows*$(\mathsf{a}_\mathsf{i}, \mathsf{b}_\mathsf{j})$, denoted f_{ij}. The instance I has the following facts: f_{11}, f_{12}, f_{21}, f_{22}, f_{23}, f_{24}, f_{31}, f_{32}, f_{34}, and f_{35}. The hypergraph $\mathcal{H}$ for I encodes the following rules: *(a)* each a_i can follow at most i people; *and (b)* each b_j can be followed by at most j people. Specifically, $\mathcal{H}$ contains the following hyperedges:

- $\{f_{11}, f_{12}\}$, $\{f_{21}, f_{22}, f_{23}\}$, $\{f_{21}, f_{22}, f_{24}\}$, $\{f_{21}, f_{23}, f_{24}\}$, $\{f_{22}, f_{23}, f_{24}\}$, $\{f_{31}, f_{32}, f_{34}, f_{35}\}$
- $\{f_{11}, f_{21}\}$, $\{f_{11}, f_{31}\}$, $\{f_{21}, f_{31}\}$, $\{f_{12}, f_{22}, f_{32}\}$

An example of a (maximal) consistent subinstance J is $\{f_{11}, f_{22}, f_{23}, f_{32}, f_{34}, f_{35}\}$.

Prioritizing Inconsistent Databases

We now recall the framework of preferred repairs by Staworko et al. [33]. Let I be an instance over a signature $\mathcal{R}$. A *priority* relation $\succ$ over I is an acyclic binary relation over the facts in I. By *acyclic* we mean that I does not contain any sequence $f_1, \ldots, f_k$ of facts with $f_i \succ f_{i+1}$

for all $i = 1, \ldots, k-1$ and $f_k \succ f_1$. If $\succ$ is a priority relation over I and K is a subinstance of I, then $\max_{\succ}(K)$ denotes the set of facts $f \in K$ such that no $g \in K$ satisfies $g \succ f$.

An *inconsistent prioritizing instance* over $\mathcal{R}$ is a triple $(I, \mathcal{H}, \succ)$, where I is an instance over $\mathcal{R}$, $\mathcal{H}$ is a conflict hypergraph for I, and $\succ$ is a priority relation over I with the following property: for every two facts f and g in I, if $f \succ g$ then f and g are neighbors in $\mathcal{H}$ (that is, f and g co-occur in some hyperedge).[1] For example, if $\mathcal{H} = \mathcal{H}^I_{\mathbf{S}}$ (where all the constraints in $\mathbf{S}$ are FDs), then $f \succ g$ implies that $\{f, g\}$ violates at least one FD.

▶ **Example 2.3.** We continue our running company-CEO example. We define a priority relation $\succ$ by $f^{\mathsf{g}}_{\mathsf{pi}} \succ f^{\mathsf{g}}_{\mathsf{pa}}$, $f^{\mathsf{g}}_{\mathsf{pa}} \succ f^{\mathsf{g}}_{\mathsf{br}}$ and $f^{\mathsf{a}}_{\mathsf{pa}} \succ f^{\mathsf{a}}_{\mathsf{pi}}$. We denote $\succ$ by corresponding arrows on the left graph of Figure 1b. (Therefore, some of the edges are directed and some are undirected.) We then get the inconsistent prioritizing instance $(I, \mathcal{H}^I_{\mathbf{S}}, \succ)$ over $\mathcal{R}$. Observe that the graph does not contain directed cycles, as required from a priority relation.

▶ **Example 2.4.** Recall that the instance I of our followers example is defined in Example 2.2. The priority relation $\succ$ is given by $f_{il} \succ f_{jk}$ if one of the following holds: *(a)* $i = j$ and $k = l + 1$, *or (b)* $j = i + 1$ and $l = k$. For example, we have $f_{11} \succ f_{12}$ and $f_{12} \succ f_{22}$. But we do not have $f_{11} \succ f_{22}$ (hence, $\succ$ is not transitive).

Let $(I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance over a signature $\mathcal{R}$. We say that $\succ$ is *total* if for every two facts f and g in I, if f and g are neighbors in $\mathcal{H}$ then either $f \succ g$ or $g \succ f$. A priority $\succ_c$ over I is a *completion* of $\succ$ (w.r.t. $\mathcal{H}$) if $\succ$ is a subset of $\succ_c$ and $\succ_c$ is total. As an example, the middle and right graphs of Figure 1b are two completions of the priority relation $\succ$ depicted on the left side. A *completion* of $(I, \mathcal{H}, \succ)$ is an inconsistent prioritizing instance $(I, \mathcal{H}, \succ_c)$ where $\succ_c$ is a completion of $\succ$.

Preferred Repairs

Let $D = (I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance over $\mathcal{R}$. As defined by Arenas et al. [3], J is a *repair* of D if J is a maximal consistent subinstance of I. Staworko et al. [33] define three different notions of *preferred* repairs: *Pareto optimal*, *globally optimal*, and *completion optimal*. The first two are based on checking whether a repair J of I can be improved by replacing a set of facts in J with a "better preferred" set of facts from I; they differ in the way "better preferred" is interpreted. The third notion is based on the concept of completion. Next we give the formal definitions.

▶ **Definition 2.5** (Improvement)**.** Let $(I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance over a signature $\mathcal{R}$, and J and J' be two distinct consistent subinstances of I.

- J is a *Pareto improvement* of J' if there exists $f \in J \setminus J'$ such that $f \succ f'$ for all $f' \in J' \setminus J$.
- J is a *global improvement* of J' if for every $f' \in J' \setminus J$ there exists $f \in J \setminus J'$ such that $f \succ f'$.

That is, J is a Pareto improvement of J' if, to obtain J from J', we insert and delete facts, and one of the inserted facts is preferred to all deleted ones. And J is a global improvement of J' if we similarly obtain J from J', but now for every deleted fact a preferred one is inserted.

[1] This requirement has been made with the introduction of the framework [33]. Obviously, the lower bounds we present hold even without this requirement. Moreover, our main upper bound, Theorem 6.1, holds as well without this requirement. We defer to future work the thorough investigation of the impact of relaxing this requirement.

▶ **Example 2.6.** We continue the company-CEO running example. We define four consistent subinstances of I: $J_1 = \{f^{\mathsf{g}}_{\mathsf{br}}, f^{\mathsf{a}}_{\mathsf{pi}}\}$, $J_2 = \{f^{\mathsf{g}}_{\mathsf{pa}}, f^{\mathsf{a}}_{\mathsf{pi}}\}$, $J_3 = \{f^{\mathsf{g}}_{\mathsf{br}}, f^{\mathsf{a}}_{\mathsf{pa}}\}$, and $J_4 = \{f^{\mathsf{g}}_{\mathsf{pi}}, f^{\mathsf{a}}_{\mathsf{pa}}\}$. Note the following. First, J_2 is a Pareto improvement of J_1, since $f^{\mathsf{g}}_{\mathsf{pa}} \in J_2 \setminus J_1$ and $f^{\mathsf{g}}_{\mathsf{pa}} \succ f$ for every fact in $J_1 \setminus J_2$ (where in this case there is only one such an f, namely $f^{\mathsf{g}}_{\mathsf{br}}$). Second, J_4 is a global improvement of J_2 because $f^{\mathsf{g}}_{\mathsf{pi}} \succ f^{\mathsf{g}}_{\mathsf{pa}}$ and $f^{\mathsf{a}}_{\mathsf{pa}} \succ f^{\mathsf{a}}_{\mathsf{pi}}$. (We refer to J_3 in later examples.)

We then get the following variants of *preferred repairs*.

▶ **Definition 2.7** (p/g/c-repair)**.** Let D be an inconsistent prioritizing instance $(I, \mathcal{H}, \succ)$, and let J be a consistent subinstance of I. Then J is a:

- *Pareto-optimal repair of* D if there is no Pareto improvement of J.
- *globally-optimal repair of* D if there is no global improvement of J.
- *completion-optimal repair of* D if there exists a completion D_c of D such that J is a globally-optimal repair of D_c.

We abbreviate "Pareto-optimal repair," "globally-optimal repair," and "completion-optimal repair" by *p-repair*, *g-repair* and *c-repair*, respectively.

We remark that in the definition of a completion-optimal repair, we could replace "globally-optimal" with "Pareto-optimal" and obtain an equivalent definition [33].

Let $D = (I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance over a signature $\mathcal{R}$. We denote the set of all the repairs, p-repairs, g-repairs and c-repairs of D by $\mathrm{Rep}(D)$, $\mathrm{PRep}(D)$, $\mathrm{GRep}(D)$ and $\mathrm{CRep}(D)$, respectively. An easy observation is that when the relation $\succ$ is empty, the four types of repairs coincide. Moreover, the following was shown by Staworko et al. [33].

▶ **Proposition 2.8** ([33])**.** *For all inconsistent prioritizing instances D we have* $\mathrm{CRep}(D) \neq \emptyset$ *and* $\mathrm{CRep}(D) \subseteq \mathrm{GRep}(D) \subseteq \mathrm{PRep}(D) \subseteq \mathrm{Rep}(D)$.

▶ **Example 2.9.** We continue our company-CEO example. Recall the instances J_i defined in Example 2.6. We showed that J_1 has a Pareto improvement, so J_1 is *not* a p-repair (although a repair in the ordinary sense). The reader can verify that J_2 has no Pareto improvements, so J_2 is a p-repair. But J_2 is not a g-repair, as J_4 is a global improvement of J_2. The reader can verify that J_3 is a g-repair (hence, a p-repair). Finally, J_4 is a g-repair w.r.t. the left completion of $\succ$ in Figure 1b (and also w.r.t. the right one). Hence, J_4 is a c-repair (and so a g-repair and a p-repair). In contrast, J_3 has a global improvement (and a Pareto improvement) in both completions; but it does not prove that J_3 is not a c-repair (since, conceptually, one needs to consider all possible completions of $\succ$).

▶ **Example 2.10.** We now continue the follower example. The inconsistent prioritizing instance $(I, \mathcal{H}, \succ)$ is defined in Examples 2.2 and 2.4. Consider the instance $J_1 = \{f_{11}, f_{22}, f_{23}, f_{32}, f_{34}, f_{35}\}$. The reader can verify that J_1 is a c-repair (e.g., by completing $\succ$ through the lexicographic order). The subinstance $J_2 = \{f_{12}, f_{21}, f_{22}, f_{34}, f_{35}\}$ is a repair but not a p-repair, since we can add f_{11} and remove both f_{12} and f_{21}, and thus obtain a Pareto improvement.

3 Categoricity

In this section we define the computational problem of *categoricity*, which is the main problem that we study in this paper. Proposition 2.8 states that, under each of the semantics of preferred repairs, at least one such a repair exists. In general, there can be many possible

preferred repairs. The problem of *categoricity* [33] is that of testing whether there is *precisely* one such a repair; that is, there do not exist two distinct preferred repairs, and therefore, the priority relation contains enough information to clean the inconsistent instance unambiguously.

▶ **Problem 3.1.** *The problems* p-categoricity*,* g-categoricity*, and* c-categoricity *are those of testing whether* $|\mathrm{PRep}(D)| = 1$*,* $|\mathrm{GRep}(D)| = 1$ *and* $|\mathrm{CRep}(D)| = 1$*, respectively, given a signature* $\mathcal{R}$ *and an inconsistent prioritizing instance* D *over* $\mathcal{R}$*.*

As defined, categoricity takes as input both the signature $\mathcal{R}$ and the inconsistent prioritizing instance D, where constraints are represented by a conflict hypergraph. We also study this problem from the perspective of *data complexity*; there, we fix a schema $\mathbf{S} = (\mathcal{R}, \Delta)$, where Δ is a set of FDs. In that case, the input consists of an instance I over $\mathcal{R}$ and a priority relation $\prec$ over I. The conflict hypergraph is then implicitly assumed to be $\mathcal{H}^I_{\mathbf{S}}$. We denote the corresponding variants of the problem by p-categoricity$\langle\mathbf{S}\rangle$, g-categoricity$\langle\mathbf{S}\rangle$ and c-categoricity$\langle\mathbf{S}\rangle$, respectively.

▶ **Example 3.2.** Continuing our company-CEO example, we showed in Example 2.9 that there are at least two g-repairs and at least three p-repairs. Hence, a solver for g-categoricity$\langle\mathbf{S}\rangle$ should return false on $(I, \succ)$, and so is a solver for p-categoricity$\langle\mathbf{S}\rangle$. In contrast, we later show that there is precisely one c-repair (Example 6.2); hence, a solver for c-categoricity$\langle\mathbf{S}\rangle$ should return true on $(I, \succ)$. If, on the other hand, we replaced $\succ$ with any of the completions in Figure 1b, then there would be precisely one p-repair and one g-repair (namely, the current single c-repair). This follows from a result of Staworko et al. [33], stating that categoricity holds in the case of total priority relations.

4 Preliminary Insights

We begin with some basic insights into the different variants of the categoricity problem.

Generating an Optimal Repair

We recall an algorithm by Staworko et al. [33] for greedily constructing a c-repair. This is the algorithm FindCRep of Figure 3a. The algorithm takes as input an inconsistent prioritizing instance $(I, \mathcal{H}, \succ)$ and returns a c-repair J. It begins with an empty J, and incrementally inserts tuples to J, as follows. In each iteration of lines 3–6, the algorithm selects a fact f from $\max_{\succ}(I)$ and removes it from I. Then, f is added to J if it does not violate consistency, that is, if $\mathcal{H}$ does not contain any hyperedge e such that $e \subseteq J \cup \{f\}$. The specific way of choosing the fact f among all those in $\max_{\succ}(I)$ is (deliberately) left unspecified, and hence, different executions may result in different c-repairs. In that sense, the algorithm is nondeterministic. Staworko et al. [33] proved that the possible results of these different executions are *precisely* the c-repairs.

▶ **Theorem 4.1** ([33])**.** *Let* $(I, \mathcal{H}, \succ)$ *be an inconsistent prioritizing instance over* $\mathcal{R}$*. Let* J *be a consistent subinstance of* I*. Then* J *is a c-repair if and only if there exists an execution of* FindCRep$(I, \mathcal{H}, \succ)$ *that returns* J*.*

Theorem 4.1, combined with Proposition 2.8, has several implications for us. First, we can obtain an x-repair (where x is either p, g or c) in polynomial time. Hence, if a solver for x-categoricity determines that there is a single x-repair, then we can actually generate that x-repair in polynomial time. Second, c-categoricity is the problem of testing

whether $\mathsf{FindCRep}(I, \mathcal{H}, \succ)$ returns the same instance J on every execution. Moreover, due to Proposition 2.8, p-categoricity (resp. g-categoricity) is the problem of testing whether every p-repair (resp. g-repair) is equal to the one that is obtained by some execution of the algorithm.

▶ **Example 4.2.** We consider the application of the algorithm FindCRep to the instance of our company-CEO example (where $\mathcal{H} = \mathcal{H}^I_{\mathbf{S}}$). The following are two different executions: *(1)* $+f^{\mathsf{g}}_{\mathsf{pi}}$, $-f^{\mathsf{g}}_{\mathsf{pa}}$, $-f^{\mathsf{g}}_{\mathsf{br}}$, $+f^{\mathsf{a}}_{\mathsf{pa}}$, $-f^{\mathsf{a}}_{\mathsf{pi}}$, *(2)* $+f^{\mathsf{a}}_{\mathsf{pa}}$, $-f^{\mathsf{a}}_{\mathsf{pi}}$, $+f^{\mathsf{g}}_{\mathsf{pi}}$, $-f^{\mathsf{g}}_{\mathsf{pa}}$, $-f^{\mathsf{g}}_{\mathsf{br}}$. Here, we denote inclusion in J (i.e., the condition of line 5 is true) by plus and exclusion from J by minus. Observe that both executions return $J_4 = \{f^{\mathsf{g}}_{\mathsf{pi}}, f^{\mathsf{a}}_{\mathsf{pa}}\}$. This is on a par with the statement in Example 3.2 that in this running example there is a single c-repair.

Complexity Insights

Our goal is to study the complexity of x-categoricity (where x is g, p and c). This problem is related to that of *x-repair checking*, namely, given $D = (I, \mathcal{H}, \succ)$ and J, determine whether J is an x-repair of D. The following is known about this problem.

▶ **Theorem 4.3** ([33, 12])**.** *The following hold.*

- *p-repair checking and c-repair checking are solvable in polynomial time; g-repair checking is in coNP [33].*
- *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a fixed schema. If $\Delta_{|R}$ is equivalent to either a single FD or two key constraints for every $R \in \mathcal{R}$, then g-repair checking over $\mathbf{S}$ is solvable in polynomial time; otherwise, g-repair checking over $\mathbf{S}$ is coNP-complete [12].*

Recall from Proposition 2.8 that there is always at least one x-repair. Therefore, given $(I, \mathcal{H}, \succ)$ we can solve the problem x-categoricity using a coNP algorithm with an oracle to x-repair checking: for all two distinct subinstances J_1 and J_2, either J_1 or J_2 is not an x-repair. Therefore, from Theorem 4.3 we conclude the following.

▶ **Corollary 4.4.** *The following hold.*

- *p-categoricity and c-categoricity are in coNP, and g-categoricity is in Π^{p}_2.*
- *For all fixed schemas $\mathbf{S} = (\mathcal{R}, \Delta)$, g-categoricity$\langle\mathbf{S}\rangle$ is in Π^{p}_2, and moreover, if $\Delta_{|R}$ is equivalent to either a single FD or two key constraints for every $R \in \mathcal{R}$ then g-categoricity$\langle\mathbf{S}\rangle$ is in coNP.*

We stress here that if x-categoricity is solvable in polynomial time, then x-categoricity$\langle\mathbf{S}\rangle$ is solvable in polynomial time for *all* schemas $\mathbf{S}$; this is true since for every fixed schema $\mathbf{S}$ the hypergraph $\mathcal{H}^I_{\mathbf{S}}$ can be constructed in polynomial time, given I. Similarly, if x-categoricity$\langle\mathbf{S}\rangle$ is coNP-hard (resp. Π^{p}_2-hard) for *at least one* $\mathbf{S}$, then x-categoricity is coNP-hard (resp. Π^{p}_2-hard).

When we are considering x-categoricity$\langle\mathbf{S}\rangle$, we assume that all the integrity constraints are FDs. Therefore, unlike the general problem of x-categoricity, in x-categoricity$\langle\mathbf{S}\rangle$ conflicting facts always belong to the same relation. It thus follows that our analysis for x-categoricity$\langle\mathbf{S}\rangle$ can be restricted to single-relation schemas. Formally, we have the following.

▶ **Proposition 4.5.** *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema and x be one of p, g and c. For each relation $R \in \mathcal{R}$, let $\mathbf{S}_{|R}$ be the schema $(\{R\}, \Delta_{|R})$.*

- *If x-categoricity$\langle\mathbf{S}_{|R}\rangle$ is solvable in polynomial time for every $R \in \mathcal{R}$, then x-categoricity$\langle\mathbf{S}\rangle$ is solvable in polynomial time.*
- *If x-categoricity$\langle\mathbf{S}_{|R}\rangle$ is coNP-hard (resp. Π^{p}_2-hard) for at least one $R \in \mathcal{R}$, then x-categoricity$\langle\mathbf{S}\rangle$ is coNP-hard (resp. Π^{p}_2-hard).*

Observe that the phenomenon of Proposition 4.5 *does not* hold for general x-categoricity (where conflicts are given by a conflict hypergraph), since hyperedges may cross relations.

In the following sections we investigate each of the three variants of categoricity: p-categoricity (Section 5), c-categoricity (Section 6) and g-categoricity (Section 7).

5 p-Categoricity

In this section we prove a dichotomy in the complexity of p-categoricity$\langle \mathbf{S} \rangle$ over all schemas $\mathbf{S}$ (where Δ consists of FDs). This dichotomy states that the only tractable case is where the schema associates a single FD (which can be trivial) to each relation symbol, up to equivalence. In all other cases, p-categoricity$\langle \mathbf{S} \rangle$ is coNP-complete. Formally, we prove the following.

▶ **Theorem 5.1.** *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema. The problem p-categoricity$\langle \mathbf{S} \rangle$ can be solved in polynomial time if $\Delta|_R$ is equivalent to a single FD for every $R \in \mathcal{R}$. In every other case, p-categoricity$\langle \mathbf{S} \rangle$ is coNP-complete.*

The tractability side of Theorem 5.1 is fairly simple to prove. The proof of the hardness side is involved, and we outline it in the rest of this section. Due to Proposition 4.5, it suffices to consider schemas $\mathbf{S}$ with a single relation, which is what we do in the remainder of this section.

5.1 Proof of Hardness

Our proof is based on the concept of a *fact-wise reduction* [24], which is formally defined as follows. Let $\mathbf{S} = (\mathcal{R}, \Delta)$ and $\mathbf{S}' = (\mathcal{R}', \Delta')$ be two schemas. A *mapping* from $\mathcal{R}$ to $\mathcal{R}'$ is a function μ that maps facts over $\mathcal{R}$ to facts over $\mathcal{R}'$. We naturally extend a mapping μ to map instances J over $\mathcal{R}$ to instances over $\mathcal{R}'$ by defining $\mu(J)$ to be $\{\mu(f) \mid f \in J\}$. A *fact-wise reduction* from $\mathbf{S}$ to $\mathbf{S}'$ is a mapping Π from $\mathcal{R}$ to $\mathcal{R}'$ with the following properties: *(a)* Π is injective, that is, for all facts f and g over $\mathcal{R}$, if $\Pi(f) = \Pi(g)$ then $f = g$; *(b)* Π preserves consistency and inconsistency, that is, for every instance J over $\mathbf{S}$, the instance $\Pi(J)$ satisfies Δ' if and only if J satisfies Δ; *and (c)* Π is computable in polynomial time.

Let $\mathbf{S}$ and $\mathbf{S}'$ be two schemas, and let Π be a fact-wise reduction from $\mathbf{S}$ to $\mathbf{S}'$. Given an inconsistent instance I over $\mathbf{S}$ and a priority relation $\succ$ over I, we denote by $\Pi(\succ)$ the priority relation $\succ'$ over $\Pi(I)$ where $\Pi(f) \succ' \Pi(g)$ if and only if $f \succ g$. If D is the inconsistent prioritizing instance $(I, \mathcal{H}^I_{\mathbf{S}}, \succ)$, then we denote by $\Pi(D)$ the triple $(\Pi(I), \mathcal{H}^{\Pi(I)}_{\mathbf{S}'}, \Pi(\succ))$, which is also an inconsistent prioritizing instance. The usefulness of fact-wise reductions is due to the following proposition, which is straightforward.

▶ **Proposition 5.2.** *Let $\mathbf{S}$ and $\mathbf{S}'$ be two schemas, and suppose that Π is a fact-wise reduction from $\mathbf{S}$ to $\mathbf{S}'$. Let I be an inconsistent instance over $\mathbf{S}$, $\succ$ a priority relation over I, and D the inconsistent prioritizing instance $(I, \mathcal{H}^I_{\mathbf{S}}, \succ)$. Then there is a bijection between $\mathrm{PRep}(D)$ and $\mathrm{PRep}(\Pi(D))$.*

We then conclude the following corollary.

▶ **Corollary 5.3.** *If there is a fact-wise reduction from $\mathbf{S}$ to $\mathbf{S}'$, then there is a polynomial-time reduction from p-categoricity$\langle \mathbf{S} \rangle$ to p-categoricity$\langle \mathbf{S}' \rangle$.*

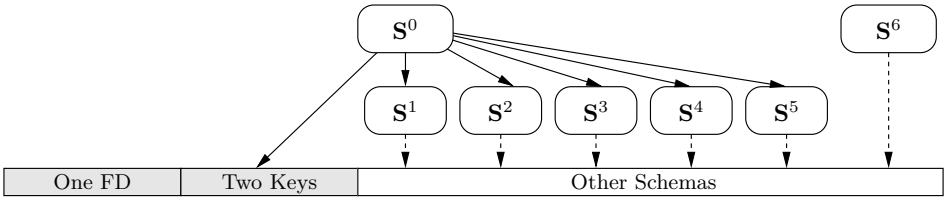


Figure 2 The structure of fact-wise reductions for proving the hardness side of the dichotomy of Theorem 5.1.

Specific Schemas

In the proof we consider seven specific schemas. The importance of these schemas will later become apparent. We denote these schemas by $\mathbf{S}^i$, for $i = 0, 1, \ldots, 6$, where each $\mathbf{S}^i$ is the schema $(\mathcal{R}^i, \Delta^i)$, and $\mathcal{R}^i$ is the singleton $\{R^i\}$. The specification of the $\mathbf{S}^i$ is as follows.

$R^0/2$ and $\Delta^0 = \{A \to B, B \to A\}$	$R^1/3$ and $\Delta^1 = \{AB \to C, BC \to A, AC \to B\}$
$R^2/3$ and $\Delta^2 = \{A \to B, B \to A\}$	$R^3/3$ and $\Delta^3 = \{AB \to C, C \to B\}$
$R^4/3$ and $\Delta^4 = \{A \to B, B \to C\}$	$R^5/3$ and $\Delta^5 = \{A \to C, B \to C\}$
$R^6/3$ and $\Delta^6 = \{\emptyset \to A, B \to C\}$	

(For $\mathbf{S}^6$, recall that $\emptyset \to A$ denotes the FD $\emptyset \to \{1\}$, that is, facts should have the same value on the first attribute.) The proof uses fact-wise reductions from the $\mathbf{S}^i$, as we explain in the next section.

Two Hard Schemas

Our proof boils down to proving coNP-hardness for two specific schemas, namely $\mathbf{S}^0$ and $\mathbf{S}^6$, and then using (known and new) fact-wise reductions in order to cover all the other schemas. For $\mathbf{S}^6$ the proof is fairly simple. However, hardness for $\mathbf{S}^0$ turned out to be highly challenging to prove, and in fact, this part is the hardest in the proof of Theorem 5.1. Note that $\mathbf{S}^0$ is the schema of our company-CEO running example (introduced in Example 2.1).

▶ **Theorem 5.4.** *The problems p-categoricity$\langle\mathbf{S}^0\rangle$ and p-categoricity$\langle\mathbf{S}^6\rangle$ are both coNP-hard.*

Applying Fact-Wise Reductions

The following has been proved by Fagin et al. [12].

▶ **Theorem 5.5** ([12])**.** *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema such that $\mathcal{R}$ consists of a single relation symbol. Suppose that Δ is equivalent to neither any single FD nor any pair of keys. Then there is a fact-wise reduction from some $\mathbf{S}^i$ to $\mathbf{S}$, where $i \in \{1, \ldots, 6\}$.*

We complete the proof using the following two lemmas, giving additional fact-wise reductions.

▶ **Lemma 5.6.** *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema such that $\mathcal{R}$ consists of a single relation symbol. Suppose that Δ is equivalent to a pair of key constraints, and Δ is not equivalent to any single FD. Then there is a fact-wise reduction from $\mathbf{S}^0$ to $\mathbf{S}$.*

▶ **Lemma 5.7.** *For all $i = 1, \ldots, 5$ there is a fact-wise reduction from $\mathbf{S}^0$ to $\mathbf{S}^i$.*

Algorithm FindCRep$(I, \mathcal{H}, \succ)$

$J := \emptyset$
while $\max_{\succ}(I) \neq \emptyset$ **do**
  choose a fact f in $\max_{\succ}(I)$
  $I := I \setminus \{f\}$
  if $J \cup \{f\}$ is consistent w.r.t. $\mathcal{H}$ **then**
    $J := J \cup \{f\}$
return J

(a) Finding a c-repair [33]

Algorithm CCategoricity$(I, \mathcal{H}, \succ)$

$i := 0$
$J := \emptyset$
while $I \neq \emptyset$ **do**
  $i := i + 1$
  $P_i := \max_{\succ^+}(I)$
  $J := J \cup P_i$
  $N_i := \{f \in I \mid \mathcal{H}$ has a hyperedge e s.t. $f \in e$, $(e \setminus \{f\}) \subseteq J$, and $(e \setminus \{f\}) \succ^+ f\}$
  $I := I \setminus (P_i \cup N_i)$
return true iff J is consistent

(b) Algorithm for c-categoricity

Figure 3 Algorithms for the completion semantics.

The structure of our fact-wise reductions is depicted in Figure 2. Dashed edges are known fact-wise reductions, while solid edges are new. Observe that each single-relation schema on the hardness side of Theorem 5.1 has an ingoing path from either $\mathbf{S}^0$ or $\mathbf{S}^6$, both shown to have coNP-hard p-categoricity (Theorem 5.4).

6 c-Categoricity

We now investigate the complexity of c-categoricity. Our main result is the following.

▶ **Theorem 6.1.** *The c-categoricity problem is solvable in polynomial time.*

In the remainder of this section we establish Theorem 6.1 by presenting a polynomial-time algorithm (Figure 3b). The algorithm is very simple, but its proof of correctness is intricate.

To present our algorithm, some notation is required. Let $(I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance. The *transitive closure* of $\succ$, denoted $\succ^+$, is the priority relation over the facts of I where for every two facts f and g it holds that $f \succ^+ g$ if and only if there exists a sequence $f_0, \ldots, f_m$ of facts, where $m > 0$, such that $f = f_0$, $f_m = g$, and $f_i \succ f_{i+1}$ for all $i = 0, \ldots, m-1$. Obviously, $\succ^+$ is acyclic (since $\succ$ is acyclic). Yet unlike $\succ$, the relation $\succ^+$ may compare between facts that are not necessarily neighbors in $\mathcal{H}$. Let $(I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance, let K be a set of facts of I, and let f be a fact of I. By $K \succ^+ f$ we denote the case where $g \succ^+ f$ for *every* fact $g \in K$.

The algorithm is depicted in Figure 3b. The input is $(I, \mathcal{H}, \succ)$, an inconsistent prioritizing instance. (The signature $\mathcal{R}$ is not needed by the algorithm.) The algorithm incrementally constructs a subinstance J of I, starting with an empty J. Later we will prove that there is a single c-repair if and only if J is consistent; and in that case, J is the single c-repair. The loop in the algorithm constructs fact sets $P_1, \ldots, P_t$ and $N_1, \ldots, N_t$ (where t is the total number of iterations). Each P_i is called a *positive stratum* and each N_i is called a *negative stratum*. Both P_i and N_i are constructed in the ith iteration. On that iteration we add to J every fact in P_i, and remove from I every fact in P_i and every fact in N_i. The sets P_i and N_i are defined as follows.

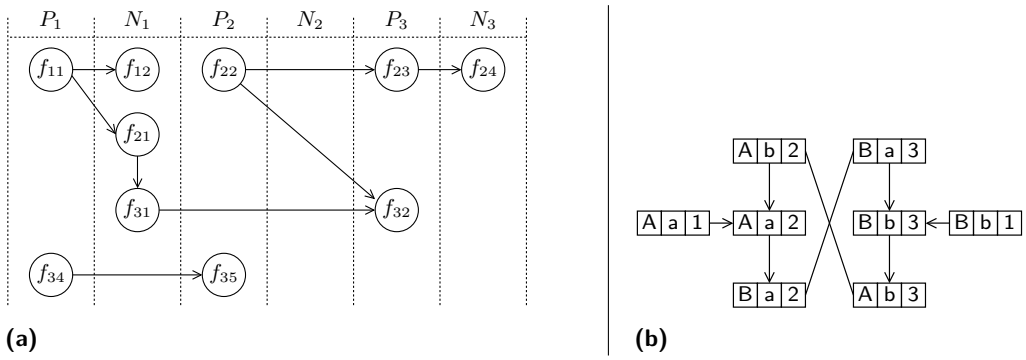


Figure 4 (a) Execution of CCategoricity on the followers example; (b) An inconsistent instance I over $\mathbf{S}^1$ with a priority relation $\succ$ over I.

- P_i consists of the maximal facts in the current I, according to $\succ^+$.
- N_i consists of all the facts f that, together with $P_1 \cup \cdots \cup P_i$, complete a hyperedge of preferred facts; that is, $\mathcal{H}$ has a hyperedge that contains f, is contained in $P_1 \cup \cdots \cup P_i \cup \{f\}$, and satisfies $g \succ^+ f$ for every incident $g \neq f$.

The algorithm continues to iterate until I gets empty. As said above, in the end the algorithm returns true if J is consistent, and otherwise false. Next, we give execution examples.

▶ **Example 6.2.** Consider $(I, \mathcal{H}, \succ)$ from our company-CEO running example, illustrated on the left side of Figure 1b. The algorithm makes a single iteration on this instance, where $P_1 = \{f^{\mathsf{g}}_{\mathsf{pi}}, f^{\mathsf{a}}_{\mathsf{pa}}\}$ and $N_1 = \{f^{\mathsf{g}}_{\mathsf{pa}}, f^{\mathsf{a}}_{\mathsf{pi}}, f^{\mathsf{g}}_{\mathsf{pa}}\}$. Both $f^{\mathsf{g}}_{\mathsf{pi}}$ and $f^{\mathsf{a}}_{\mathsf{pa}}$ are in P_1 since both are maximal. Also, each of $f^{\mathsf{g}}_{\mathsf{pa}}$, $f^{\mathsf{a}}_{\mathsf{pi}}$ and $f^{\mathsf{g}}_{\mathsf{pa}}$ is in conflict with P_1, and we have $f^{\mathsf{g}}_{\mathsf{pi}} \succ f^{\mathsf{g}}_{\mathsf{pa}}$, $f^{\mathsf{a}}_{\mathsf{pa}} \succ f^{\mathsf{a}}_{\mathsf{pi}}$, and $f^{\mathsf{g}}_{\mathsf{pi}} \succ^+ f^{\mathsf{g}}_{\mathsf{br}}$.

▶ **Example 6.3.** Now consider the inconsistent prioritizing instance $(I, \mathcal{H}, \succ)$ from our followers running example. Figure 4a illustrates the execution of the algorithm, where each column describes P_i or N_i, from left to right in the order of their construction. For convenience, the priority relation $\succ$, as defined in Example 2.4, is depicted in Figure 4a using corresponding edges between the facts.

On iteration 1, for instance, we have $P_1 = \{f_{11}, f_{34}\}$, since f_{11} and f_{34} are the facts without incoming edges on Figure 4a. Moreover, we have $N_1 = \{f_{12}, f_{21}, f_{31}\}$. The reason why N_1 contains f_{12}, for example, is that $\{f_{11}, f_{12}\}$ is a hyperedge, the fact f_{11} is in P_1, and $f_{11} \succ f_{12}$ (hence, $f_{11} \succ^+ f_{12}$). For a similar reason N_1 contains f_{21}. Fact f_{31} is in N_1 as $\{f_{11}, f_{31}\}$ is a hyperedge, and though $f_{11} \not\succ f_{31}$, we have $f_{11} \succ^+ f_{31}$. As another example, N_3 contains f_{24} since $\mathcal{H}$ has the hyperedge $\{f_{22}, f_{23}, f_{24}\}$, the set $\{f_{22}, f_{23}\}$ is contained in $P_1 \cup P_2 \cup P_3$, and $\{f_{22}, f_{23}\} \succ^+ f_{24}$.

In the end, $J = \{f_{11}, f_{22}, f_{23}, f_{32}, f_{34}, f_{35}\}$, which is also the subinstance J_1 of Example 2.10. Since J is consistent, the algorithm will determine that there is a single c-repair, and that c-repair is J.

▶ **Example 6.4.** We now give an example of an execution on a negative instance of c-categoricity. (In Section 7 we refer to this example for a different reason.) Figure 4b shows an instance I over $\mathbf{S}^1$, which is defined in Section 5.1. Recall that in this schema every two attributes form a key. Each fact $R^1(a_1, a_2, a_3)$ in I is depicted by a tuple that consists of the three values. For example, I contains the (conflicting) facts $R^1(\mathsf{A}, \mathsf{a}, 1)$ and $R^1(\mathsf{A}, \mathsf{a}, 2)$. Hereon, we write Xyi instead of $R^1(\mathsf{X}, \mathsf{y}, \mathsf{i})$. The priority relation $\succ$ is given by the directed

edges between the facts; for example, $\mathsf{Aa1} \succ \mathsf{Aa2}$. Undirected edges are between conflicting facts that are incomparable by $\succ$ (e.g., $\mathsf{Ab2}$ and $\mathsf{Ab3}$).

The execution of the algorithm on $(I, \mathcal{H}^I_{\mathbf{S}_1}, \succ)$ is as follows. On the first iteration, $P_1 = \{\mathsf{Aa1}, \mathsf{Ab2}, \mathsf{Ba3}, \mathsf{Bb1}\}$ and $N_1 = \{\mathsf{Aa2}, \mathsf{Bb3}\}$. In particular, note that N_1 does not contain $\mathsf{Ba2}$ since it conflicts only with $\mathsf{Ba3}$ in P_1, but the two are incomparable. Similarly, N_1 does not contain $\mathsf{Ab3}$ since it is incomparable with $\mathsf{Ab2}$. Consequently, in the second iteration we have $P_2 = \{\mathsf{Ba2}, \mathsf{Ab3}\}$ and $N_2 = \emptyset$. In the end, $J = P_1 \cup P_2$ is inconsistent, and therefore, the algorithm will return false. Indeed, the reader can easily verify that each of the following is a c-repair: $\{\mathsf{Aa1}, \mathsf{Ab2}, \mathsf{Ba3}, \mathsf{Bb1}\}$, $\{\mathsf{Aa1}, \mathsf{Ab2}, \mathsf{Ba2}, \mathsf{Bb1}\}$, and $\{\mathsf{Aa1}, \mathsf{Ba3}, \mathsf{Ab3}, \mathsf{Bb1}\}$.

Correctness of CCategoricity is stated in the following theorem.

▶ **Theorem 6.5.** *Let $(I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance, and let J be the subinstance of I constructed in the execution of* CCategoricity$(I, \mathcal{H}, \succ)$. *Then J is consistent if and only if there is a single c-repair. Moreover, if J is consistent then J is the single c-repair.*

Theorem 6.5, combined with the observation that the algorithm CCategoricity terminates in polynomial time, implies Theorem 6.1. As previously said, the proof of Theorem 6.5 is quite involved. The "only if" direction is that of *soundness*—if the algorithm returns true then there is precisely one c-repair. The other direction is that of *completeness*—if there is precisely one c-repair then the algorithm returns true. Soundness is the easier direction to prove, and we do not discuss the proof here. Proving completeness is more involved. We assume, by way of contradiction, that the constructed J is inconsistent. We are looking at the first positive stratum P_i such that $P_1 \cup \cdots \cup P_i$ contains a hyperedge. Then, the crux of the proof is in showing that we can then construct two c-repairs using the algorithm FindCRep: one contains some fact from P_i and another one does not contain that fact. We then establish that there are at least two c-repairs, hence a contradiction.

7 g-Categoricity

We now investigate the complexity of g-categoricity. We begin with a tractability result. Recall from Theorem 5.1 that, assuming P $\neq$ NP, the problem p-categoricity$\langle \mathbf{S} \rangle$ is solvable in polynomial time if and only if $\mathbf{S}$ consists (up to equivalence) of a single FD per relation. The proof works for g-categoricity$\langle \mathbf{S} \rangle$, so the tractable schemas of p-categoricity remain tractable for g-categoricity.

▶ **Theorem 7.1.** *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema. The problem g-categoricity$\langle \mathbf{S} \rangle$ can be solved in polynomial time if $\Delta|_R$ is equivalent to a single FD for every $R \in \mathcal{R}$.*

It is left open whether there is any schema $\mathbf{S}$ that is not as in Theorem 7.1 where g-categoricity$\langle \mathbf{S} \rangle$ is solvable in polynomial time. In the next section we give an insight into this open problem.

7.1 Intractable Schemas

Our next result shows that g-categoricity$\langle \mathbf{S} \rangle$ hits a harder complexity class than p-categoricity$\langle \mathbf{S} \rangle$. In particular, while p-categoricity$\langle \mathbf{S} \rangle$ is always in coNP (due to Corollary 4.4), we will show a schema $\mathbf{S}$ where g-categoricity$\langle \mathbf{S} \rangle$ is Π^{p}_2-complete. This schema is the schema $\mathbf{S}^6$ from Section 5.1.

▶ **Theorem 7.2.** *g-categoricity$\langle \mathbf{S}^6 \rangle$ is Π^{p}_2-complete.*

The proof of Theorem 7.2 is by a reduction from the Π_2^{p}-complete problem QCNF_2: Given a CNF formula $\psi(\mathbf{x}, \mathbf{y})$, determine whether it is the case that for every truth assignment to $\mathbf{x}$ there exists a truth assignment to $\mathbf{y}$ such that the two assignments satisfy ψ.

We can generalize Theorem 7.2 to a broad set of schemas, by using fact-wise reductions from $\mathbf{S}^6$. This is done in the following theorem.

▶ **Theorem 7.3.** *Let $\mathbf{S} = (\mathcal{R}, \Delta)$ be a schema such that $\mathcal{R}$ consists of a single relation symbol R and Δ consists of two nontrivial FDs $X \to Y$ and $W \to Z$. Suppose that each of W and Z contains an attribute that is in none of the other three sets. Then g-categoricity$\langle\mathbf{S}\rangle$ is Π_2^{p}-complete.*

As an example, recall that in $\mathbf{S}^6$ we have $\Delta = \{\emptyset \to A, B \to C\}$. This schema is a special case of Theorem 7.3, since we can use $\emptyset \to A$ as $X \to Y$ and $B \to C$ as $W \to Z$; and indeed, each of W and Z contains an attribute (namely B and C, respectively) that is not in any of the other three sets. Additional examples that satisfy the conditions of Theorem 7.3 (and hence the corresponding g-categoricity$\langle\mathbf{S}\rangle$ is Π_2^{p}-complete) are the following: $\{A \to B, C \to D\}$, $\{A \to C, AB \to CD\}$, $\{A \to B, ABC \to D\}$, and $\{A \to B, C \to ABD\}$. All of these sets are over a relation symbol $R/4$. (And in each of these sets, the first FD corresponds to $X \to Y$ and the second to $W \to Z$.)

Unlike $\mathbf{S}^6$, to this day we do not know what is the complexity of g-categoricity$\langle\mathbf{S}^i\rangle$ for any of the other $\mathbf{S}^i$ (defined in Section 5.1). This includes $\mathbf{S}^0$, for which all we know is membership in coNP (as stated in Corollary 4.4). However, except for this open problem, the proof technique of Theorem 5.1 is valid for g-categoricity$\langle\mathbf{S}\rangle$. Consequently, we can show the following.

▶ **Theorem 7.4.** *The following are equivalent.*

- *g-categoricity$\langle\mathbf{S}^0\rangle$ is coNP-hard.*
- *g-categoricity$\langle\mathbf{S}\rangle$ is coNP-hard for every schema $\mathbf{S}$ that falls outside the polynomial-time cases of Theorem 7.1.*

7.2 Transitive Priority

Let $(I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance. We say that $\succ$ is *transitive* if for every two facts f and g in I, if f and g are neighbors in $\mathcal{H}$ and $f \succ^+ g$, then $f \succ g$. Transitivity is a natural assumption when $\succ$ is interpreted as a partial order such as "is of better quality than" or "is more current than." In this section we consider g-categoricity in the presence of this assumption. The following example shows that a g-repair is not necessarily a c-repair, even if $\succ$ is transitive. This example provides an important context for the results that follow.

▶ **Example 7.5.** Consider again I and $\succ$ from Example 6.4 (depicted in Figure 4b). Observe that $\succ$ is transitive. In particular, there is no priority between $\mathsf{Ab2}$ and $\mathsf{Ba2}$, even though $\mathsf{Ab2} \succ^+ \mathsf{Ba2}$, because $\mathsf{Ab2}$ and $\mathsf{Ba2}$ are not in conflict (or, put differently, they are not neighbors in $\mathcal{H}^I_{\mathbf{S}^1}$). Consider the subinstance $J = \{\mathsf{Aa1}, \mathsf{Ba2}, \mathsf{Ab3}, \mathsf{Bb1}\}$ of I. The reader can verify that J is a g-repair, but not a c-repair (since no execution of $\mathsf{FindCRep}$ can generate J).

Example 7.5 shows that global and completion optimality are different notions, even if the priority is transitive. Yet, quite remarkably, in the presence of transitivity the two coincide on categoricity.

▶ **Theorem 7.6.** *Let $D = (I, \mathcal{H}, \succ)$ be an inconsistent prioritizing instance such that $\succ$ is transitive.* $|\mathrm{CRep}(D)| = 1$ *if and only if* $|\mathrm{GRep}(D)| = 1$.

Proof. The "if" direction follows from Proposition 2.8, since every c-repair is also a g-repair. The proof of the "only if" direction is based on the special structure of the c-repair, as established in Section 6, in the case where only one c-repair exists. Specifically, suppose that there is a single c-repair J and let $J' \neq J$ be a consistent subinstance of I. We need to show that J' has a global improvement. We claim that J is a global improvement of J'. This is clearly the case if $J' \subseteq J$. So suppose that $J' \not\subseteq J$. Let f' be a fact in $J' \setminus J$. We need to show that there is a fact $f \in J \setminus J'$ such that $f \succ f'$. We complete the proof by finding such an f.

Recall from Theorem 6.5 that J is the result of executing $\mathsf{CCategoricity}(I, \mathcal{H}, \succ)$. Consider the positive strata P_i and the negative strata N_j constructed in that execution. Since J is the union of the positive strata, we get that f' necessarily belongs to a negative stratum, say N_j. From the definition of N_j it follows that $\mathcal{H}$ has a hyperedge e such that $f' \in e$, $(e \setminus \{f'\}) \subseteq P_1 \cup \cdots \cup P_j$, and $(e \setminus \{f'\}) \succ^+ f'$. Let e be such a hyperedge. Since J' is consistent, it cannot be the case that J' contains all the facts in e. Choose a fact $f \in e$ such that $f \notin J'$. Then $f \succ^+ f'$, and since $\succ$ is transitive (and f and f' are neighbors), we have $f \succ f'$. So $f \in J \setminus J'$ and $f \succ f'$, as required. ◀

Interestingly, the proof of Theorem 7.6 is based on the correctness of the algorithm CCategoricity of Figure 3b. Combining Theorems 6.1 and 7.6, we get the following.

▶ **Corollary 7.7.** *For transitive priority relations, the problems g-categoricity and c-categoricity coincide, and in particular, g-categoricity is solvable in polynomial time.*

We conclude with two comments. First, the reader may wonder whether Theorem 7.6 and Corollary 7.7 hold for p-categoricity as well. This is not the case. Hardness of p-categoricity$\langle \mathbf{S}^6 \rangle$ is proved by a reduction to a transitive priority relation. Second, in their analysis Fagin et al. [12] have constructed various reductions for proving coNP-hardness of g-repair checking. In several of these, the priority relation is transitive. We conclude that there are schemas $\mathbf{S}$ such that, on transitive priority relations, g-repair checking is coNP-complete whereas g-categoricity is solvable in polynomial time.

8 Concluding Remarks

We investigated the complexity of the categoricity problem, which is that of determining whether the provided priority relation suffices to repair the database unambiguously, in the framework of preferred repairs [33]. In this framework, integrity constraints are anti-monotonic and repairing operations are tuple deletions (i.e., *subset repairs*). Following the three semantics of optimal repairs, we investigated the three variants of this problem: p-categoricity, g-categoricity and c-categoricity. We established a dichotomy in the data complexity of p-categoricity for the case where constraints are FDs, partitioning the cases into polynomial time and coNP-completeness. We further showed that the tractable side of p-categoricity extends to g-categoricity, but the latter can reach Π_2^{p}-completeness already for two FDs. Finally, we showed that c-categoricity is solvable in polynomial time in the general case where integrity constraints are given as a conflict hypergraph.

We did not address here any qualitative discrimination among the three notions of x-repairs. Rather, we continue the line of work [34, 13] that explores the impact of the choice on the entailed computational complexity. It has been established that, as far as repair

checking is concerned, the Pareto and the completion semantics behave much better than the global one, since g-repair checking is tractable only for a very restricted class of schemas [13]. In this work we have shown that from the viewpoint of categoricity, the Pareto semantics becomes likewise intractable (while the global semantics hits an even higher complexity class), and the completion semantics outstands so far as the most efficient option to adopt.

We complete this paper by discussing directions for future research. It would be interesting to further understand the complexity of g-categoricity, towards a dichotomy (at least for FDs). We have left open the question of whether there exists a schema with a single relation and a set of FDs, *not* equivalent to a single FD, such that g-categoricity is solvable in polynomial time. Another interesting direction is the generalization of categoricity to the problems of *counting* and *enumerating* the preferred repairs. For classical repairs (without a priority relation), Maslowski and Wijsen [29, 30] established dichotomies (FP vs. #P-completeness) in the complexity of counting in the case where constraints are primary keys. For the general case of denial constraints, counting the classical repairs reduces to the enumeration of independent sets of a hypergraph with a bounded edge size, a problem shown by Boros et al. [7] to be solvable in incremental polynomial time (and in particular polynomial input-output complexity). For a general given conflict hypergraph, repair enumeration is the well known problem of enumerating the *minimal hypergraph transversals*; whether this problem is solvable in polynomial total time is a long standing open problem [20].

A natural continuation of this work would be to chart the complexity boundaries for more general cleaning frameworks that feature preferences between repairs, including different types of integrity constraints, different cleaning operations (e.g., tuple addition and cell update [35]), and different priority specifications among repairs. The latter includes preferences by means of general scoring functions [31, 22], aggregation of scores on the individual cleaning operations [6, 18, 25, 18, 11], priorities among resolution policies [28] and preferences based on soft rules [32, 21].

Acknowledgments. The authors are very grateful to Ronald Fagin and Phokion Kolaitis for insightful discussions on the categoricity problem.

References

1 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41. ACM, 2009.

2 Douglas E. Appelt and Boyan Onyshkevych. The common pattern specification language. In *TIPSTER Text Program: Phase III*, pages 23–30. Association for Computational Linguistics, 1998.

3 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM, 1999.

4 Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

5 Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755. IEEE, 2007.

6 Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154. ACM, 2005.

7 Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Processing Letters*, 10(4):253–266, 2000.

8 Yang Cao, Wenfei Fan, and Wenyuan Yu. Determining the relative accuracy of attributes. In *SIGMOD*, pages 565–576. ACM, 2013.

9 Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *ACL*, pages 128–137, 2010.

10 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

11 Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, pages 541–552. ACM, 2013.

12 Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. Dichotomies in the complexity of preferred repairs. In *PODS*, pages 3–15. ACM, 2015.

13 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *PODS*, pages 164–175. ACM, 2014.

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

15 Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

16 Wenfei Fan, Floris Geerts, and Jef Wijsen. Determining the currency of data. *ACM Trans. Database Syst.*, 37(4):25, 2012.

17 Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2):213–238, 2012.

18 Wenfei Fan, Shuai Ma, Nan Tang, and Wenyuan Yu. Interaction between record matching and data repairing. *J. Data and Information Quality*, 4(4):16:1–16:38, 2014.

19 Terry Gaasterland, Parke Godfrey, and Jack Minker. An overview of cooperative answering. *J. Intell. Inf. Syst.*, 1(2):123–157, 1992.

20 Georg Gottlob and Enrico Malizia. Achieving new upper bounds for the hypergraph duality problem through logic. In *CSL-LICS*, pages 43:1–43:10. ACM, 2014.

21 Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. Feasibility conditions and preference criteria in querying and repairing inconsistent databases. In *DEXA*, volume 3180 of *LNCS*, pages 44–55. Springer, 2004.

22 Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. Preferred repairs for inconsistent databases. In *Encyclopedia of Database Technologies and Applications*, pages 480–485. Idea Group, 2005.

23 Benny Kimelfeld, Ester Livshits, and Liat Peterfreund. Unambiguous prioritized repairing of databases. *CoRR*, abs/1603.01820, 2016. URL: `http://arxiv.org/abs/1603.01820`.

24 Benny Kimelfeld, Jan Vondrák, and Ryan Williams. Maximizing conjunctive views in deletion propagation. *ACM Trans. Database Syst.*, 37(4):24, 2012.

25 Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 53–62. ACM, 2009.

26 Paraschos Koutris and Dan Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In *ICDT*, pages 165–176. OpenProceedings.org, 2014.

27 Paraschos Koutris and Jef Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *PODS*, pages 17–29. ACM, 2015.

28 Maria Vanina Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V. S. Subrahmanian. Inconsistency management policies. In *KR*, pages 367–377. AAAI Press, 2008.

29 Dany Maslowski and Jef Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013.

30 Dany Maslowski and Jef Wijsen. Counting database repairs that satisfy conjunctive queries with self-joins. In *ICDT*, pages 155–164. OpenProceedings.org, 2014.

31 Amihai Motro, Philipp Anokhin, and Aybar C. Acar. Utility-based resolution of data inconsistencies. In *IQIS*, pages 35–43. ACM, 2004.

32 Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. *TPLP*, 6(1-2):107–167, 2006.

33 Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.

34 Slawomir Staworko, Jan Chomicki, and Jerzy Marcinkowski. Preference-driven querying of inconsistent relational databases. In *EDBT Workshops*, volume 4254 of *LNCS*, pages 318–335. Springer, 2006.

35 Jef Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.

36 Jef Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *PODS*, pages 189–200. ACM, 2013.

Compression of Unordered XML Trees*

Markus Lohrey[1], Sebastian Maneth[2], and Carl Philipp Reh[3]

1 **University of Siegen, Siegen, Germany**
`lohrey@eti.uni-siegen.de`
2 **University of Edinburgh, Edinburgh, UK**
`smaneth@inf.ed.ac.uk`
3 **University of Siegen, Siegen, Germany**
`reh@eti.uni-siegen.de`

Abstract

Many XML documents are data-centric and do not make use of the inherent document order. Can we provide stronger compression for such documents through giving up order? We first consider compression via minimal dags (directed acyclic graphs) and study the worst case ratio of the size of the ordered dag divided by the size of the unordered dag, where the worst case is taken for all trees of size n. We prove that this worst case ratio is $n/\log n$ for the edge size and $n \log\log n/\log n$ for the node size. In experiments we compare several known compressors on the original document tree versus on a canonical version obtained by length-lexicographical sorting of subtrees. For some documents this difference is surprisingly large: reverse binary dags can be smaller by a factor of 3.7 and other compressors can be smaller by factors of up to 190.

1998 ACM Subject Classification E.4 Coding and Information Theory

Keywords and phrases tree compression, directed acyclic graphs, XML

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.18

1 Introduction

Understanding the interplay between ordered and unordered structures is an important topic of database research. For XML this interplay has received considerable attention, see, e.g., [1, 4, 20, 3, 18]. A document is deemed *document-centric*, if the order of elements matters. Examples of such documents include web pages (e.g., in XHTML). In contrast, a document is *data-centric* if the order of elements is unimportant. For instance, the order of author-, title-, and year-elements in a bibliographic entry is unimportant. Of course, there could be mixtures of both, unordered and ordered nodes. For instance, an author-node could be marked "ordered" to contain subtrees for the first author, second author, etc. JSON naturally supports ordered and unordered nodes (cf. the Conclusions).

The absence of order bears many opportunities such as query optimization and set-oriented parallel processing, cf. [1]. Unordered XML has also been studied recently with respect to schema language definitions [4], a topic already considered during the birth years of XML [16]. Here we study the question whether *tree compression* can benefit from unorderedness.

In XML compression, document trees are typically stored (and compressed) separately from the data values, see, e.g., [14]. Let us first consider a very basic tree compression technique: directed acyclic graphs (dags, for short). Let t be an ordered tree. By representing repeated occurrences of the same subtree in t only once, one obtains a unique minimal dag

* The first author was supported by the DFG via project LO 748/10-1.

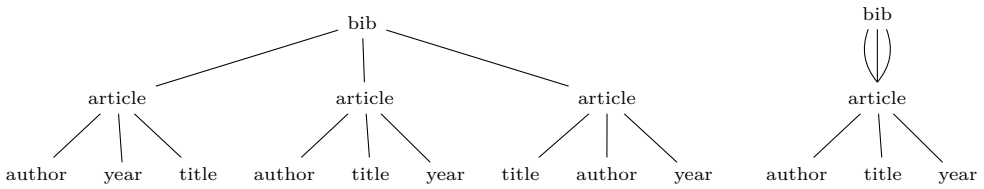


Figure 1 The tree structure of a bibliography on the left and its unordered dag on the right.

for t. In the following, we denote this minimal dag as *the dag of* t. It was observed early on that dags provide high compression ratios for common XML document trees [7] (10% on average for their documents). Moreover, the dag can be produced in linear time [9] or in amortized linear time using hashing (the well-known "hash-consing"), see, e.g., [7]. What happens if we construct the *unordered dag* of the tree t, which is the minimal dag of the unordered version of t?[1] Figure 1 shows an XML document tree consisting of 12 edges. Since there are no repeating subtrees (containing any edges), the minimal dag of this tree has 12 edges as well. In contrast, the unordered dag has only 6 edges. This raises the question how much smaller, at most, the unordered dag can be in comparison to the dag. We answer this question for two size measures: (i) the number of nodes and (ii) the number of edges. For each of these measures we study the maximum of the dag size of t divided by the unordered dag size of t, where the maximum is taken over all node-labelled trees t of size n. We denote these worst case ratios by $\alpha_N(n)$ (for the node size) and $\alpha_E(n)$ (for the edge size). Our main theoretical results provide precise growth rates (up to multiplicative constants) for these values:

(i) for the edge size we obtain $\alpha_E(n) \in \Theta(n/\log n)$ and
(ii) for the node size we obtain $\alpha_N(n) \in \Theta(n \log\log n/\log n)$.

With respect to the upper bound $\alpha_E(n) \in O(n/\log n)$ we show that for every tree t of size n, the unordered dag has at least $e/2 \cdot \ln(n/2)$ many edges (e is Euler's constant and ln is the logarithm to base e). This is shown using the well-known inequality between the geometric and arithmetic mean, see e.g., [17]. The upper bound $\alpha_N(n) \in \Theta(n \log\log n/\log n)$ uses a technique that has been applied in several related contexts, see e.g. [11]: one removes from a tree t all subtrees of size at most m, where m is logarithmic in the size of t. Then one bounds (i) the size of the remaining subtree and (ii) the number of different trees of size at most m. This yields an upper bound on the node size of the dag of t. For the lower bounds in (i) and (ii) one exploits the obvious fact that the list of subtrees of a node of rank r can be permuted in $r!$ many ways without affecting the corresponding unordered tree.

Let us also mention that the unordered dag can be computed in linear time as well. This can be done using the method for unordered tree isomorphism as given in Aho, Hopcraft, and Ullman's book [2].

In the second part of the paper, we contrast our theoretical results by experimental data for two corpora of XML trees. In addition to dags, we are interested in our experiments to gauge the impact of unorderedness of other tree compression methods. These are the dag variants introduced in [5] and the grammar-based tree compressor "TreeRePair" [15]. These are the strongest tree compressors that we are aware of. Instead of introducing (nontrivial) adaptations of each of these compressors to unordered trees, we opted for a

[1] Whenever we solely use the term dag (resp., tree), we always refer to the ordered version. If we want to speak about the unordered versions, we explicitly add the adjective "unordered".

different approach: we compress *canonical* trees. For this, we use the well-known canonization via length-lexicographical sorting of subtree lists, see., e.g., [8]. For an ordered tree t, it is easy to observe that the dag of t's canonical tree is isomorphic to the unordered dag of t. Our experimental results can be summarized as follows: for plain dags as explained above, the largest difference for any document of our collection is that the unordered dag has 60% size of the ordered dag. Then, essentially, the stronger the tree compression method as such, the larger the difference on the canonical tree. For the "hybrid dag" the largest difference is a factor of 3 smaller. For dag-plus-string-compression (called "DS" in [5]) we obtain some astonishing results: for one document tree (coming from Wikipedia data) the compression of the canonical tree is smaller by a factor of 190 than that of the original tree.

2 Preliminaries

With $e = 2,71828\cdots$ we always denote Euler's constant. With $\ln n$ (resp. $\log n$) we denote the logarithm of n to base e (resp., 2). For a positive integer k we denote by $[k]$ the set $\{1, 2, \ldots, k\}$. Let Σ be an alphabet. For a string $w = a_1 a_2 \cdots a_n \in \Sigma^*$ $(a_1, \ldots, a_n \in \Sigma)$ we denote by $\mathrm{alph}(w)$ the set $\{a_1, a_2, \ldots, a_n\}$ of symbols that appear in w. For $a \in \Sigma$ let $|w|_a$ denote the number $|\{i \in [n] \mid a_i = a\}|$ of occurrences of the symbol a in w. For $i \in [n]$ we denote the i-th letter a_i of w by $w[i]$. For $a \in \Sigma$ we denote with a^m the word $a \cdots a$ with m many occurrences of a.

3 Multi-graphs, Dags, and Trees

In this section we formally define node-labelled trees and dags in the ordered and unordered setting. Our definitions are non-standard in the sense that we define trees and dags as multi-graphs, whereas usually they are defined as ordinary graphs. Let Σ be an alphabet. A Σ-*labeled ordered dag* (or briefly dag) is a tuple $d = (V, \gamma, \lambda, v_0)$, where

- V is a finite set of nodes,
- $\gamma : V \to V^*$ assigns to each node a finite sequence of successor nodes,
- $\lambda : V \to \Sigma$ assigns to each node a label from Σ, and
- $v_0 \in V$ is the root node.

Moreover, we require that the *edge relation* $E_d := \{(u, v) \mid v \in \mathrm{alph}(\gamma(v))\}$ satisfies the following two properties:

- E_d is acyclic, i.e., there is no node v with $(v, v) \in E_d^+$ and
- every node v is reachable from v_0, i.e., $(v_0, v) \in E_d^*$.

Often we speak of the *multi-edges* of d. Formally, these are triples $(v, i, \gamma(v)[i])$ for $v \in V$ and $1 \leq i \leq |\gamma(v)|$. We use two size measures for a dag $d = (V, \gamma, \lambda, v_0)$:

- $\|d\| = \sum_{v \in V} |\gamma(v)|$ is the number of multi-edges, and
- $|d| = |V|$ is the number of nodes.

A *path* in d of length n is a sequence $v_1, k_1, v_2, k_2, \ldots, v_{n+1}$ such that (v_i, k_i, v_{i+1}) is a multi-edge of d for all $1 \leq i \leq n$. The *height* $h(d)$ of d is the length of a longest path in d. For a node $v \in V$, $\rho(v) = |\gamma(v)|$ is its rank and the rank of d is $\rho(d) = \max\{\rho(v) \mid v \in V\}$.

A Σ-*labeled ordered tree* (or briefly tree) can be defined as a dag $d = (V, \gamma, \lambda, v_0)$ such that every node $u \in V \setminus \{v_0\}$ has a unique occurrence in the set of strings $\{\gamma(v) \mid v \in V\}$. In other words: $\mathrm{alph}(\gamma(u)) \cap \mathrm{alph}(\gamma(v)) = \emptyset$ for $u \neq v$ and $|\gamma(u)|_v \leq 1$ for all $u, v \in V$. Alternatively, one can use terms over Σ to describe trees: if $t_1, \ldots, t_n$ $(n \geq 0)$ are trees and $f \in \Sigma$ then $f(t_1, \ldots, t_n)$ is also a tree. The set of all Σ-labeled ordered trees t with $\rho(t) \leq r$ is denoted as $\mathcal{T}_r(\Sigma)$. Moreover, let $\mathcal{T}_\infty(\Sigma) = \bigcup_{r \geq 1} \mathcal{T}_r(\Sigma)$. For $r \in \mathbb{N} \cup \{\infty\}$ and $k \in \mathbb{N}$ let $\mathcal{T}_{r,k} = \mathcal{T}_r([k])$. In

this notation, $\mathcal{T}_{\infty,1}$ is the set of all unlabelled trees (trees, where every node is labelled with the same symbol 1). For a tree t there is no essential difference between the size measures $\|t\|$ and $|t|$ (we have $\|t\| = |t| - 1$).

A dag $d = (V, \gamma, \lambda, v_0)$ can be unfolded into a tree $\tau(d)$. To define this tree, we associate with every node $v \in V$ the tree $\tau(v)$ inductively as follows: if $\lambda(v) = f$ and $\gamma(v) = v_1 v_2 \cdots v_n$ then $\tau(v) = f(\tau(v_1), \tau(v_2), \ldots, \tau(v_n))$. Finally, let $\tau(d) = \tau(v_0)$. For a tree t we define its *minimal dag* $\mathrm{dag}(t)$ as the smallest dag d with respect to $|d|$ such that $\tau(d) = t$. This is also the smallest dag d with respect to $\|d\|$ such that $\tau(d) = t$. The minimal dag $\mathrm{dag}(t)$ is unique up to isomorphism. It can be obtained from t by merging nodes u and v with $\lambda(u) = \lambda(v)$ and $\gamma(u) = \gamma(v)$ as long as possible. Also note that $|\mathrm{dag}(t)|$ is exactly the number of different subtrees of t. It is known that $\mathrm{dag}(t)$ can be computed in linear time [9].

There exist obvious unordered counterparts to the above definitions. A Σ-*labeled unordered dag* can be defined as a tuple $d = (V, \gamma, \lambda, v_0)$, where V, λ and v_0 have the same properties as for an ordered dag, and $\gamma : V \to \mathbb{N}^V$ assigns to each node v a V-indexed tuple of natural numbers, which can be seen as a multiset over V. Let us write $\gamma(u, v)$ instead of $(\gamma(u))(v)$, which is the number of multi-edges from u to v. It is required that the edge relation $E_d := \{(u, v) \in V \times V \mid \gamma(u, v) > 0\}$ is acyclic and that $(v_0, v) \in E_d^*$ for all $v \in V$. Unordered trees are then defined in the obvious way. As for ordered dags we define the two size measures $\|d\| = \sum_{u,v \in V} \gamma(u, v)$ and $|d| = |V|$. The unfolding of an unordered dag d is an unordered tree that we again denote with $\tau(d)$. This allows us to define the minimal dag $\mathrm{dag}(t)$ of the unordered tree t, which is an unordered dag. We make the following convention for the rest of the paper:

▶ Convention 1. Whenever we solely use the term dag (resp., tree), we always refer to the ordered version. If we want to speak about the unordered versions, we use the term unordered dag (resp., unordered tree).

For an ordered dag $d = (V, \gamma, \lambda, v_0)$, we define the *corresponding unordered dag* $d^u = (V, \gamma^u, \lambda, v_0)$, where $\gamma^u(v, w) = |\gamma(v)|_w$ is the number of occurrences of node w in the list $\gamma(v)$. For a tree t we define its *unordered minimal dag* $\mathrm{dag}^u(t)$ of t as the minimal dag of the corresponding unordered tree t^u. In the following we omit the adjective "minimal" when we speak of the minimal (unordered) dag of a tree. Figure 1 shows on the right the unordered dag of the tree on the left.

4 Sizes of Dags versus Unordered Dags

We clearly have $|\mathrm{dag}^u(t)| \leq |\mathrm{dag}(t)|$ and $\|\mathrm{dag}^u(t)\| \leq \|\mathrm{dag}(t)\|$. In this section we study the question, how much smaller the unordered dag for a given tree can be compared to its ordered dag. Formally, we study the growth of the following two worst case ratios, where $n, r, k \in \mathbb{N}$ and $r, k \leq n$:

$$\alpha_N(n, r, k) = \max\left\{ \frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \,\middle|\, t \in \mathcal{T}_{r,k}, |t| \leq n \right\},$$
$$\alpha_E(n, r, k) = \max\left\{ \frac{\|\mathrm{dag}(t)\|}{\|\mathrm{dag}^u(t)\|} \,\middle|\, t \in \mathcal{T}_{r,k}, |t| \leq n \right\}.$$

Let $x \in \{N, E\}$. Note that $\alpha_x(n, 1, k) = 1$ since for a tree $t \in \mathcal{T}_{1,k}$ (which is a linear chain) the ordered as well as the unordered dag is equal to t. Hence, we only consider the ratio $\alpha_x(n, r, k)$ for $r \geq 2$. Note that $\alpha_x(n, r, k) \leq \alpha_x(n, r', k')$ for all $r, r', k, k' \leq n$ with $r \leq r'$ and $k \leq k'$, since this implies $\mathcal{T}_{r,k} \subseteq \mathcal{T}_{r',k'}$. In the following we mainly concentrate on the extreme cases $\alpha_x(n, 2, 1)$ and $\alpha_x(n, n, n)$. We use the abbreviation $\alpha_x(n) = \alpha_x(n, n, n)$.

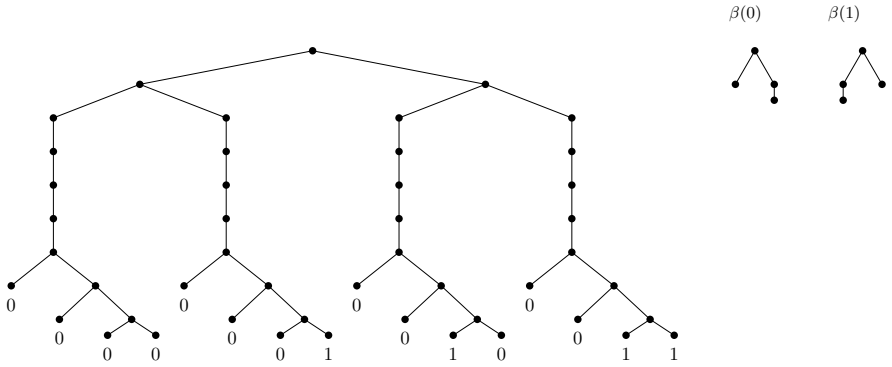


Figure 2 A possible choice for the tree t_{16} from Theorem 1 with $n = 16$, $r = h = 4$ and $k = 2$. For the 0- (resp., 1-labelled) nodes one has to substitute the tree $\beta(0)$ (resp., $\beta(1)$) on the right.

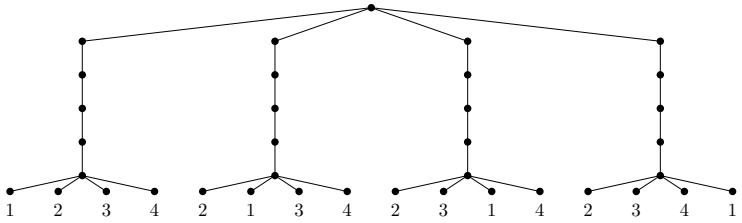


Figure 3 A possible choice for the tree t_{16} from Theorem 2 with $n = 16$, $x = \frac{1}{2}\log\log n = 1$, and $r = k = 4$.

4.1 Lower Bounds for α_E and α_N

In this section, we prove two lower bounds. In the first part we derive a lower bound of $\Omega(n/\log n)$ for $\alpha_E(n, 2, 1)$. For this, we construct a family of binary trees, where the dag achieves almost no compression, while the unordered dag achieves exponential compression ratios. Later, we show that this bound is tight by providing a matching upper bound for $\alpha_E(n)$. Note that $\alpha_N(n, 2, 1) \in \Theta(\alpha_E(n, 2, 1))$, since for a binary tree we have $|\mathrm{dag}(t)| - 1 \leq \|\mathrm{dag}(t)\| \leq 2 \cdot |\mathrm{dag}(t)|$ and analogously for $\mathrm{dag}^u(t)$. In the following theorem and its proof, we only consider trees from $\mathcal{T}_{2,1}$ (binary unlabelled trees). We denote such trees by well-parenthesized strings over (and). Formally, $() \in \mathcal{T}_{2,1}$ and if $t_1, t_2 \in \mathcal{T}_{2,1}$ then also $(t_1), (t_1t_2) \in \mathcal{T}_{2,1}$. By $B_h \in \mathcal{T}_{2,1}$ we denote the complete unlabelled binary tree with 2^h leaves and height h. This tree has size $2^{h+1} - 1$ and its dag has $2h$ multi-edges. For a tree t with k leaves and trees $t_1, \ldots, t_k$ we write $t[t_1, \ldots, t_k]$ to denote the tree obtained from t by replacing the i-th leaf (in pre-order) by t_i. For $k \geq 1$ let $c_k = (^k)^k$ denote a chain of k nodes. We encode non-empty bit strings by trees from $\mathcal{T}_{2,1}$ using the function β that is inductively defined as follows: $\beta(0) = (()(()))$, $\beta(1) = ((())())$, and $\beta(ds) = (\beta(d)\beta(s))$ for $d \in \{0, 1\}$ and $s \in \{0, 1\}^+$. The trees $\beta(0)$ and $\beta(1)$ are shown in Figure 2 on the right. Note that if $s_1, s_2 \in \{0, 1\}^+$ and $|s_1| = |s_2|$ then the unordered trees $\beta(s_1)^u$ and $\beta(s_2)^u$ are isomorphic. The construction in the proof of the following theorem is similar to a construction from [11].

▶ **Theorem 1.** *For every $n \geq 2$ there exists a tree $t_n \in \mathcal{T}_{2,1}$ with*

- $|t_n| \in \Theta(n)$
- $\|\mathrm{dag}(t_n)\| \in \Theta(n)$
- $\|\mathrm{dag}^u(t_n)\| \in \Theta(\log n)$

Hence, we have $\alpha_E(n, 2, 1) \in \Omega(n/\log n)$ and $\alpha_N(n, 2, 1) \in \Omega(n/\log n)$.

Proof. Let $n \in \mathbb{N}$ and $h = \lceil \log n \rceil$. Let $r = 2^k \in \Theta(n/\log n)$ be the smallest power of two that is at least n/h. Let $u_1, \ldots, u_r$ be r distinct bit strings of length h (note that $r \leq n \leq 2^h$). Consider the trees $s_1 = \beta(u_1), \ldots, s_r = \beta(u_r)$. We add to s_i a chain of length h and obtain the tree $s'_i = c_h[s_i]$ $(1 \leq i \leq r)$. Finally, set $t_n = B_k[s'_1, \ldots, s'_r]$. A possible choice for the tree t_{16} is shown in Figure 2.

Let us first bound the size of t_n. For B_k we have $|B_k| \in \Theta(r) = \Theta(n/\log n)$. The total size of all r copies of the chain c_h is $\Theta(r \cdot h) = \Theta(n)$. Finally, every s_i has size $\Theta(h)$; so their sizes sum up to $\Theta(r \cdot h) = \Theta(n)$. Altogether, we get $|t_n| \in \Theta(n)$.

To bound $\|\mathrm{dag}(t_n)\|$, note that the trees $s_1, \ldots, s_r$ are pairwise different (as ordered trees). This implies that in the dag of t_n, the r copies of the chains c_h are still present. Therefore, $\mathrm{dag}(t_n)$ has at least $r \cdot h \in \Theta(n)$ many nodes (and, of course, it has at most n nodes). Since every node of t_n has rank at most 2, we get $\|\mathrm{dag}(t_n)\| \in \Theta(n)$.

Finally, for the unordered dag note that the trees $s_1, \ldots, s_r$ are pairwise isomorphic when considered as unordered trees. Therefore, the copies of the chains c_h are collapsed into a single chain in $\mathrm{dag}^u(t_n)$ and also the top B_k-part is collapsed into $\Theta(\log r) = \Theta(\log n)$ many nodes. We get $\|\mathrm{dag}^u(t_n)\| \in \Theta(\log n)$ as well as $|\mathrm{dag}^u(t_n)| \in \Theta(\log n)$. ◀

In the next section, we will prove $\alpha_E(n) \in O(n/\log n)$, which yields the same upper bound for $\alpha_E(n, 2, 1)$. Moreover, also the lower bound of $\Omega(n/\log n)$ for $\alpha_N(n, 2, 1)$ turns out be sharp (see Corollary 7 for $k = 1$). On the other hand, for $\alpha_N(n) = \alpha_N(n, n, n)$ we can improve the lower bound to $\Omega(n \log\log n/\log n)$:

▶ **Theorem 2.** *Fix a constant $\delta > 1$. For every $n \geq 1$ large enough (depending on δ) there exists a tree $t_n \in \mathcal{T}_{r,k}$ with the following properties:*

- $k = \lceil \delta \cdot \log n/\log\log n \rceil$ *and* $r \in \Theta(n \cdot \log\log n/\log n)$.
- $|t_n| \in \Theta(n)$
- $|\mathrm{dag}(t_n)| \in \Theta(n)$
- $|\mathrm{dag}^u(t_n)| \in \Theta(\log n/\log\log n)$

Hence, we have $\alpha_N(n) \in \Omega(n \cdot \log\log n/\log n)$.

Proof. Fix $n \geq 1$ and let $x = \frac{1}{\delta}\log\log n$, $k = \lceil (\log n)/x \rceil = \lceil \delta \cdot \log n/\log\log n \rceil$, and $r = \lfloor n/k \rfloor \in \Theta(n \cdot \log\log n/\log n)$. Let us first show that with this choice we have $k! \geq r$. With Stirling's formula (or, more precisely, the inequality $z! \geq \sqrt{2\pi z} \cdot (z/e)^z$) we get

$$k! \geq (k/e)^k \geq (\log n/ex)^{(\log n)/x} = 2^{(\log\log n - \log(ex))(\log n)/x} = n^{(\log\log n - \log(ex))/x}.$$

Since moreover $n \geq n/k \geq r$, it suffices to show

$$n^{(\log\log(n) - \log(ex))/x} \geq n,$$

i.e., $\log\log n - \log(ex) \geq x = \frac{1}{\delta}\log\log n$, or, equivalently $(1 - \frac{1}{\delta})\log\log n \geq \log(ex) = \log(e/\delta) + \log\log\log n$, which holds for n large enough. This shows that, indeed, $k! \geq r$.

We now construct the tree $t_n \in \mathcal{T}_{r,k}$ as follows: take r many pairwise different trees $s_1, \ldots, s_r$ consisting of a root node with k many children, which are leaves. The sequence of labels of these k leaves forms a permutation of $[k]$. Since $k! \geq r$, these r pairwise different trees exist. From s_i we next construct s'_i by adding a chain of length k on top of s_i. Finally, the tree t_n is obtained by taking a new root node, whose children are the roots of the trees $s'_1, \ldots, s'_r$. A possible choice for the tree t_{16} is shown in Figure 3. Note that for n large enough we have $k \leq r$ since $k \in \Theta(\log n/\log\log n)$ and $r \in \Theta(n \cdot \log\log n/\log n)$. Hence, $t_n \in \mathcal{T}_{r,k}$.

We get $|t_n| = 1 + 2rk \in \Theta(n)$. For the node size of the dag, we obtain $|\mathrm{dag}(t_n)| = 1 + rk + k \in \Theta(n)$. Finally, for the node size of the unordered dag, note that the unordered trees corresponding to $s'_1, \ldots, s'_r$ are all isomorphic. Hence, we obtain that $|\mathrm{dag}^u(t_n)| = 1 + 2k \in \Theta(k) = \Theta(\log n / \log\log n)$, which proves the statement. ◀

4.2 Upper Bound for α_E

In this section we prove an upper bound for $\alpha_E(n)$ via a lower bound of $\Omega(\log n)$ for the function

$$\mu(n) := \min\{\|\mathrm{dag}^u(t)\| \mid t \in \mathcal{T}_{n,n}, |t| \leq n\}.$$

Thus, the unordered dag of a tree of size n has at least logarithmic size in n. Note that for binary trees (or trees of constant rank) this is obvious since the height of such a tree is at least $\log n$, which implies that also the minimal unordered dag has at least $\log n$ many edges. Also note that

$$\mu(n) = \min\{\|\mathrm{dag}(t)\| \mid t \in \mathcal{T}_{n,n}, |t| \leq n\}.$$

The reason is that for every tree t there is a tree t' with $|t| = |t'|$ and $\mathrm{dag}^u(t) = (\mathrm{dag}(t'))^u$ and thus $\|\mathrm{dag}^u(t)\| = \|\mathrm{dag}(t')\|$. Moreover, it holds that

$$\mu(n) = \min\{\|\mathrm{dag}(t)\| \mid t \in \mathcal{T}_{n,1}, |t| \leq n\},$$

i.e., it suffices to consider unlabelled trees. This is because adding labels to a tree can make the minimal dag only larger. Therefore we do not consider labels in the following and consider dags as triples (V, γ, v_0) without a labelling function λ.

Let $d = (V, \gamma, v_0)$ be such a dag. For a node $v \in V$ define $\mathrm{depth}(v)$ as the length of a *longest* path from the root v_0 to v. Thus, $\mathrm{depth}(v_0) = 0$. Note that $h(d) = \max\{\mathrm{depth}(v) \mid v \in V\}$. For $1 \leq i \leq h(d)+1$ let $V_i(d) = \{v \in V \mid \mathrm{depth}(v) = i-1\}$ be the set of nodes at depth $i-1$. Finally, let $\rho_i(d) = \sum_{v \in V_i(d)} |\gamma(v)|$ for $1 \leq i \leq h(d)$. This is the total number of multi-edges that start in a node at depth $i-1$. Every such multi-edge goes to a node at depth $j \geq i$. We write V_i and ρ_i for $V_i(d)$ and $\rho_i(d)$, respectively, if d is clear from the context.

▶ **Lemma 3.** *Let $d = (V, \gamma, v_0)$ be a dag of height $h = h(d)$. The number of leaves of the unfolding $\tau(d)$ is bounded by $\prod_{i=1}^{h} \rho_i$.*

Proof. Consider the dag $d' = (\{1, \ldots, h+1\}, \gamma', 1)$ with $\gamma'(i) = (i+1)^{\rho_i}$ (the string with ρ_i many occurrences of $i+1$) for $1 \leq i \leq h$ and $\gamma'(h+1) = \varepsilon$. It is a chain of $h+1$ nodes with ρ_i many multi-edges from node i to node $i+1$. The unfolding of d' has $\prod_{i=1}^{h} \rho_i$ many leaves. It therefore suffices to transform d into d' and show that this transformation does not reduce the number of leaves of the unfolding.

First of all, we can merge in d all nodes v with $\gamma(v) = \varepsilon$ to a single node. This does not change the unfolding, the height of the dag, and the number of multi-edges starting at depth i. Hence, V_{h+1} consists of the unique sink node of d; let us call this node s. Next, we construct from d the dag $d_1 = (V, \gamma_1, v_0)$, where γ_1 is defined as follows: we set $\gamma_1(s) = \varepsilon$. Now, let $v \in V_i$ with $1 \leq i \leq h$ and $\gamma(v) = v_1 v_2 \cdots v_r$. We set $\gamma_1(v) = v'_1 v'_2 \cdots v'_r$, where the nodes v'_j are defined as follows: if $v_j \in V_{i+1}$ then set $v'_j = v_j$. Otherwise, i.e., if $v_j \in V_k$ with $k > i+1$, then let v'_j be a node in V_{i+1} such that there exists a path from v'_j to v_j. Note that such a node v'_j exists, since every node in V_k ($k > 1$) has a predecessor in V_{k-1}. Note that the unfolding $\tau(v_j)$ is a subtree of the unfolding $\tau(v'_j)$. Therefore, $\tau(d_1)$ has indeed at least as many leaves as $\tau(d)$.

The dag d_1 has still height h and $\rho_i(d_1) = \rho_i(d)$ for $1 \leq i \leq h$. But in contrast to d, all multi-edges in d_1 go from a node in V_i to a node in V_{i+1} for some $1 \leq i \leq h$. Moreover, every node in V_i ($1 \leq i \leq h$) has at least one successor node (in V_{i+1}). If we now merge all nodes in V_i to a single node, we obtain (up to isomorphism) the dag d'. Clearly, this merging increases the number of paths from the root v_0 to the sink s. But the number of such paths is exactly the number of leaves in the unfolding. This shows the lemma. ◀

▶ **Theorem 4.** *We have* $\mu(n) \geq \frac{e}{2} \cdot \ln(n/2)$.

Proof. Let t be an arbitrary (unlabelled tree) of size n. We first transform t into a new tree t' by adding exactly one additional child node to every non-leaf of t. These new children are leaves in t'. Now t' has the property that every non-leaf node has at least two children. Note that $n \leq |t'| \leq 2n$. Moreover, for the dag we also have $\|\mathrm{dag}(t)\| \leq \|\mathrm{dag}(t')\| \leq 2 \cdot \|\mathrm{dag}(t)\|$. Intuitively, $\mathrm{dag}(t')$ is obtained from $\mathrm{dag}(t)$ by adding for every internal node v an additional multi-edge to the unique sink node of $\mathrm{dag}(t)$.

Let ℓ be the number of leaves of t'. Since every non-leaf node of t' has at least two children, we have $\ell \geq |t'|/2 \geq n/2$. Moreover, let h be the height of t' and let $\rho_i = \rho_i(\mathrm{dag}(t'))$. From Lemma 3 we obtain

$$\ell \leq \prod_{i=1}^{h} \rho_i.$$

On the other hand, we have

$$\|\mathrm{dag}(t')\| = \sum_{i=1}^{h} \rho_i.$$

The well-known inequality between the arithmetic and geometric mean states that for all $x_1, \ldots, x_m \in \mathbb{R}$,

$$\frac{1}{m} \cdot \sum_{i=1}^{m} x_i \geq \left(\prod_{i=1}^{m} x_i \right)^{1/m}.$$

Applying this to the numbers ρ_i ($1 \leq i \leq h$), we get

$$\|\mathrm{dag}(t')\| = \sum_{i=1}^{h} \rho_i \geq h \cdot \left(\prod_{i=1}^{h} \rho_i \right)^{1/h} \geq h \cdot \ell^{1/h}.$$

To further bound the term $h \cdot \ell^{1/h}$, we consider it as a function of h: let $f(x) = x \cdot \ell^{1/x}$. Its derivative is

$$f'(x) = \ell^{1/x} \left(1 - \frac{\ln(\ell)}{x} \right).$$

Therefore $f(x)$ has a minimum at $x = \ln \ell$ in the interval $(0, \infty)$, from which it follows that

$$h \cdot \ell^{1/h} \geq \ell^{1/\ln(\ell)} \cdot \ln \ell = e \cdot \ln \ell.$$

With $\ell \geq n/2$ we finally get

$$\|\mathrm{dag}(t)\| \geq \frac{1}{2} \cdot \|\mathrm{dag}(t')\| \geq \frac{e}{2} \cdot \ln \ell \geq \frac{e}{2} \cdot \ln(n/2)$$ ◀

For every tree t of size n we have $\|\mathrm{dag}(t)\| \leq n$. Moreover, by Theorem 4 it holds that $\|\mathrm{dag}^u(t)\| \geq \frac{e}{2} \cdot \ln(n/2)$. Hence, we obtain

$$\frac{\|\mathrm{dag}(t)\|}{\|\mathrm{dag}^u(t)\|} \leq \frac{2n}{e \cdot \ln(n/2)} \in \Theta(n/\log n),$$

which is stated in the next corollary.

▶ **Corollary 5.** *It holds that* $\alpha_E(n) \in O(n/\log n)$.

4.3 Upper Bound for α_N

In this section, we derive an upper bound on the node size of the minimal dag.

▶ **Theorem 6.** *For every tree* $t \in \mathcal{T}_{n,k}$ *of size* n *and height* h*, it holds that*[2]

$$|\mathrm{dag}(t)| \in O\left(\frac{n \cdot h \cdot \log(k+1)}{\log n}\right).$$

Proof. Let $t \in \mathcal{T}_{n,k}$ be a tree of size n and height h. Note that $|\mathrm{dag}(t)|$ is the number of different subtrees of t. Let t' be the tree that is obtained from t by removing all maximal subtrees of size at most

$$m := \frac{1}{2} \cdot \log_{4k} n = \frac{\log n}{2 \cdot \log 4k}.$$

Let F be the forest consisting of all these removed subtrees. Then the number of different subtrees of t (i.e., $|\mathrm{dag}(t)|$) is bounded by $|t'|$ plus the number of different subtrees in F. But the latter is bounded by the number of trees $s \in \mathcal{T}_{\infty,k}$ with $|s| \leq m$, which by [11, Lemma 2] is at most $\frac{4}{3}(4k)^m = \frac{4}{3}n^{1/2}$.

Let us now bound $|t'|$. Consider a leaf v of t'. Then, the subtree of t rooted in v must have size larger than m; otherwise v would not belong to t'. Therefore, t' has at most n/m many leaves. Clearly, if every internal node in t' would have at least two children, then we could conclude that t' has at most $2n/m$ many nodes. But t' may contain nodes with a single child. Let us call such nodes *unary*. Moreover, let ℓ be the length of a longest path in t' in which all nodes except the last one are unary. Then, we get $|t'| \leq 2(\ell+1)n/m \leq 2(h+1)n/m$. In total, we get

$$\begin{aligned} |\mathrm{dag}(t)| &\leq \frac{2(h+1)n}{m} + \frac{4}{3} \cdot n^{1/2} \\ &= \frac{4 \cdot (h+1) \cdot n \cdot \log 4k}{\log n} + \frac{4}{3} \cdot n^{1/2} \in O\left(\frac{n \cdot h \cdot \log(k+1)}{\log n}\right). \end{aligned}$$

◀

▶ **Corollary 7.** *It holds that* $\alpha_N(n,n,k) \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right)$ *and* $\alpha_N(n) \in O\left(\frac{n \cdot \log\log n}{\log n}\right)$.

Proof. Let us first show $\alpha_N(n,n,k) \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right)$. Let t be a tree of size n and height h with labels from $[k]$. By Theorem 6 we have

$$|\mathrm{dag}(t)| \in O\left(\frac{n \cdot h \cdot \log(k+1)}{\log n}\right).$$

[2] We write $\log(k+1)$ instead of $\log k$ in order to avoid $\log k = 0$ for $k = 1$.

On the other hand, we clearly have $|\mathrm{dag}^u(t)| \geq h$. Therefore, we get

$$\frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right).$$

Let us now prove $\alpha_N(n) \in O\left(\frac{n \cdot \log\log n}{\log n}\right)$. Consider an arbitrary tree t of size n with labels from $[n]$. If more than $\log n$ labels occur in t, then we clearly have $|\mathrm{dag}^u(t)| > \log n$. Since $|\mathrm{dag}(t)| \leq n$ we get (for n large enough)

$$\frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \leq \frac{n}{\log n} \leq \frac{n \cdot \log\log n}{\log n}.$$

On the other hand, if at most $\log n$ many different labels occur in t then the bound $\alpha_N(n, n, k) \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right)$ implies

$$\frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \in O\left(\frac{n \cdot \log\log n}{\log n}\right).$$

This proves the bound $\alpha_N(n) \in O\left(\frac{n \cdot \log\log n}{\log n}\right)$. ◀

4.4 Summary of the Results for α_N and α_E

The following result summarizes our theoretical bounds for the functions $\alpha_N(n, 2, 1)$, $\alpha_N(n)$, $\alpha_E(n, 2, 1)$, and $\alpha_E(n)$:

▶ **Corollary 8.** *It holds that:*

$$\alpha_N(n, 2, 1) = \Theta\left(\frac{n}{\log n}\right), \qquad \alpha_N(n) = \Theta\left(\frac{n \cdot \log\log n}{\log n}\right),$$
$$\alpha_E(n, 2, 1) = \Theta\left(\frac{n}{\log n}\right), \qquad \alpha_E(n) = \Theta\left(\frac{n}{\log n}\right).$$

5 Experimental Results

In this section we experimentally evaluate the impact of unorderedness with regards to compression of XML trees. We compute for an XML tree t its *canonical tree* $\mathsf{canon}(t)$. The tree $\mathsf{canon}(f(t_1, t_2, \ldots, t_n))$ is obtained by sorting the trees $\mathsf{canon}(t_1), \ldots, \mathsf{canon}(t_n)$ according to their size, and in case of equal sizes, according to the lexicographical order of their XML traversal strings. Clearly, for all trees s, t we have $s^u = t^u$ if and only if $\mathsf{canon}(s) = \mathsf{canon}(t)$. The minimal unordered dag can be obtained from the canonical tree by computing its minimal dag. Clearly, this is a very expensive way of obtaining the minimal unordered dag: it can be obtained in linear time by a variant of the algorithm of Aho, Hopcroft and Ullman [2], see also [19]. We have implemented that procedure and can confirm its efficiency: it runs at least as fast as our (ordered) dag programs based on hashing. The reason for computing the canonical tree is to provide a simple way of gauging how other compressors benefit from unorderedness (by imposing a canonical order). It should be understood that our experiment only gives a rough indication of the benefit of unorderedness for compressors other than the dag. We expect that a more careful adaptation of those compressors to unordered trees will provide stronger compression.

We only report number of edges, so "size" in this section always refers to number of edges.

5.1 Tree Compressors

We compare seven known tree compressors, which are considered in [5]:

1. minimal dag,
2. minimal binary dag,
3. minimal reverse binary dag,
4. minimal hybrid dag,
5. minimal reverse hybrid dag,
6. DS, and
7. TreeRePair.

We choose these compressors, since they all produce a graph-based representation of the input tree. This makes the output sizes of the compressors comparable.

It should be clear that the minimal dag of the canonical tree is isomorphic to the unordered dag of the original tree. Thus, the size of the minimal dag of a canonical tree is always smaller than or equal to the size of the minimal dag of the original tree.

The *minimal binary dag (bdag)* of an unranked tree t is the minimal dag of the "first-child/next-sibling encoding" (for short, fcns encoding) of t. The fcns encoding s is common for XML: it has the same nodes as t, a node v is the left child in s of a node u if and only if v is the first child of u in t, and, a node v is the right child in s of a node u if and only if v is the next sibling of u in t. This encoding is considered in Paragraph 2.3.2 of Knuth's first book [12]. The *minimal reverse binary dag (rbdag)* is the minimal dag of the "first-child/previous-sibling encoding" (fcps), defined in the obvious way. Binary dags and reverse binary dags share end- and begin-sequences, respectively, of subtrees. This implies that both the bdag and rbdag of a canonical tree can be *larger* than the corresponding dags of the original tree. As an example consider the following tree

$$t = f(g(c, d, b, a, h), g(c, d, b), g(b, d, c, d, b)).$$

This tree has 16 edges. Its minimal binary dag has only 14 edges, because the end-sequence of subtrees "c, d, b" occurs twice and can be shared. Similarly, the minimal reverse binary dag has size 14 (because "c, d, b" appears twice). In contrast, the canonical tree of t

$$\mathsf{canon}(t) = f(g(b, c, d), g(a, b, c, d, h), g(b, b, c, d, d))$$

has a bdag and rbdag of 16 edges. Interestingly, such scenarios where bdag and rbdag become larger for the canonical tree appear frequently in practice.

The *hybrid dag (hdag)* (and *reverse hybrid dag (rhdag)*) were introduced in [5] as data structures that are guaranteed to be smaller than or equal in size to both the dag and the bdag (rbdag) of an unranked tree. The hdag (resp., rhdag) is obtained from a dag by applying the fcns-encoding (resp., fcps-encoding) to the rules of the dag (where the dag is viewed as a regular tree grammar), and then computing the minimal dag of the resulting forest of encoded rules; see [5] for a precise definition. Similarly as with bdag and rbdag, the hdag and rhdag can be *larger* for the canonical tree than for the original one.

The acronym *DS* stands for "dag and string compression". The idea is to compute a minimal dag and to then apply a string compressor to the above mentioned rules of the dag. As in [5], we use RePair [13] as our string compressor. Finally, *TR* refers to the grammar-based tree compressor TreeRePair of [15]. The sizes are numbers of edges of the compressed structures, see [5] for details.

Table 1 Document characteristics, Edges = average number of edges in a tree, aD = average depth of a node, mD = maximum depth of a node in any tree, aR = average rank of a node, mR = maximum rank of a tree.

Corpus	Documents	Edges	aD	mD	aR	mR
I	21	$3.1 \cdot 10^6$	6.6	36	5.7	$3.9 \cdot 10^6$
II	1131	79465	7.9	65	6.0	2925

5.2 Corpora of XML Documents

We use two different Corpora of XML documents. These corpora were also used in [5]. For each document we consider the unranked tree of its element nodes, i.e., we ignore attribute and text values. *Corpus I* consists of XML documents from the web which are often used in XML compression research. Many of the files of this corpus can be downloaded from the XMLCompBench site[3] (see [5] for details). *Corpus II* is a subset of files from the *University of Amsterdam XML Web Collection*[4]. We have verified by hand that, according to the tag names, all of the documents in Corpus I appear to be order independent. By sampling Corpus II we also did not find order dependent documents.

The characteristics of the Corpora are quite different: Corpus I consists of few and very large files while Corpus II has many small files. Some characteristics are shown in Table 1. As can be seen, the average size of documents from Corpus I is about 40 times larger than that of Corpus II, and the rank (=maximum length of sibling lists) of documents from Corpus I is about 1300 times larger; this indicates that most of the documents from Corpus I are indeed very long lists of (small) subtrees.

5.3 Experimental Setup

The implementations for dag, bdag, rbdag, hdag, and DS are the same ones as used in [5]. Note that DS uses Gonzalo Navarro's implementation of RePair for strings[5]. For TreeRePair, called "TR" in what follows, we use Roy Mennicke's implementation[6]; we do not change any parameters and run it plain from the command line (thus, the maxRank parameter of TR is at its default value of 4). We do not report running times (they are provided in [5]). The canonizer was implemented from scratch in java using integer and string sorting as provided by java (this runs quite slow and can take several hours for some of the documents).

5.4 Compression of Canonical versus Original Tree

The results of applying the different compressors to the documents of Corpus I are shown in Table 2. The first line shows how the compression ratio on the canonical tree changes with respect to the compression ratio for the original tree (a percentage of more than 100% means that the compression ratio is better on the canonical tree). The second row shows the sizes of the compressed canonical trees (in number of edges). For instance, the compression ratio of the hdag of the canonical tree of document "sprot39.dat" is 67% of the ratio for the original tree. On the other hand, DS compressor over the canonical tree of document

[3] `http://xmlcompbench.sourceforge.net`
[4] `http://data.politicalmashup.nl/xmlweb`
[5] `http://www.dcc.uchile.cl/~gnavarro/software/`
[6] `http://code.google.com/p/treerepair`

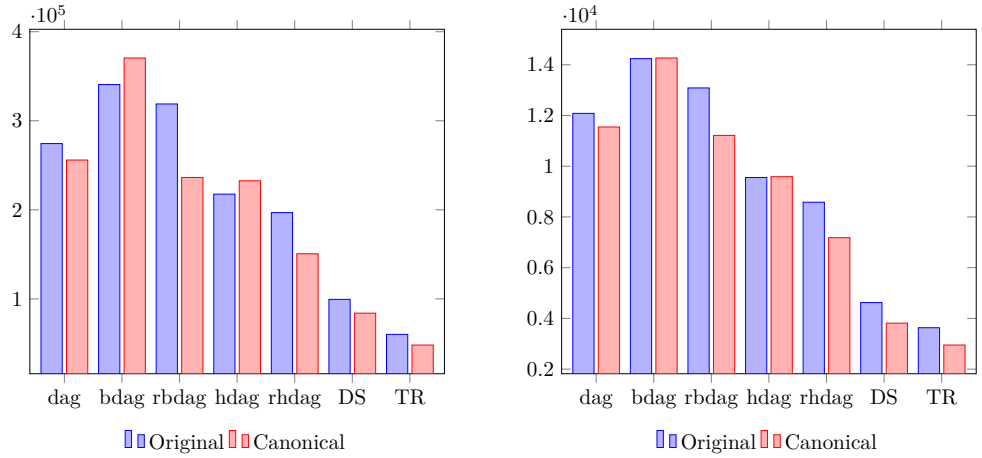


Figure 4 Comparison of average sizes of Corpus I (left) and Corpus II (right).

"EnWikTionary" has a compression that is 191-times better than the ratio for the original tree. For each document we indicate in bold the unique best increase of compression, and underline the smallest size.

Note that the dag of the canonical tree can never be larger than the dag of the original tree. Intuitively, every original repeating subtree (that gets shared in the dag) is also repeating in the canonical tree. Thus, there cannot be percentages below 100 in the column for the dag. In every other column the percent number can potentially be below 100. This is because these compressors take into account sibling sequences and hence are effected by the change of sibling orders due to canonization. In fact, this happens for the file "EXI-factbook": here *all* compression ratios (except that for the dag) become worse for the canonical tree. It means the the ordering of the canonization removes repetitions that are meaningful for the compressor. It is interesting to see that for this outlier, the strongest overall compressor TR (with respect to size) is affected the most: the compression goes down to 81% of the original; this is also the only file where this ever happens for TR. Another outlier that comes from the EXI group is EXI-weblog, where no compression ratio changes; this document is in an order that is isomorphic to that of the canonical tree.

The majority (almost one half) of documents have the strongest increase for the DS compressor. In particular, all the EnWik documents belong to this group. It is interesting to observe that for all the EnWik documents *only* DS and TR give (*massive*) compression, while for all the dag variants the compression ratio does not change. This means that after canonization there are (i) no repeating subtrees and (ii) no repeating prefixes or suffixes of sibling lists that were different before canonization. Note also that for this group,

DS achieves the smallest size values for each document. It means that there are no complex tree patterns that are repeating, and hence would be compressed by TR but not by DS; all repetition seems to be purely on the level of sibling lists. In contrast to that, observe the treebank file which features (by far) the most complex tree structure of all the documents: here the size of TR is almost twice smaller than that of DS. Curiously, the rhdag has the highest increase for this document. There is another interesting group of documents, namely those where the rbdag has the largest increase. It means that after canonization there are a lot of repeating prefixes of sibling sequences; thus, optional elements which typically appear at the end of sibling lists (the reverse dags profit from that) have, after canonization, remained to appear at the end. Apparently, this is less often the case for

Table 2 Difference (in %) of canonical versus original tree compression, and size of canonical compression output (largest in bold and smallest underlined, respectively).

document	dag	bdag	rbdag	hdag	rhdag	DS	TR
1998statistics	118%	352%	**373%**	306%	333%	239%	211%
	1164	682	632	422	373	200	238
catalog-01	146%	82%	177%	84%	**182%**	146%	117%
	5856	8514	5830	5302	3295	2994	3390
catalog-02	114%	98%	111%	98%	113%	**473%**	331%
	28496	53647	50858	27912	25823	5761	8072
dictionary-01	107%	102%	**225%**	104%	187%	130%	136%
	54575	75827	33386	45214	24960	24634	16434
dictionary-02	116%	116%	**254%**	117%	207%	138%	145%
	469915	588665	257228	353365	197197	194324	115932
EnWikiNew	100%	100%	100%	100%	100%	**2712%**	2282%
	35075	70018	70025	35057	35054	341	422
EnWikiQuote	100%	100%	100%	100%	100%	**2370%**	2091%
	23904	47692	47699	23888	23887	267	316
EnWikiVersity	100%	100%	100%	100%	100%	**2672%**	2287%
	43693	87258	87263	43676	43673	264	326
EnWikTionary	100%	100%	100%	100%	100%	**19108%**	15839%
	726221	1452273	1452279	726197	726191	428	531
EXI-Array	100%	100%	100%	100%	100%	**425%**	375%
	95584	128009	128011	95562	95563	213	267
EXI-factbook	100%	100%	91%	96%	96%	93%	81%
	4477	5090	3227	3766	2225	1937	1708
EXI-Invoice	100%	100%	100%	100%	100%	98%	**102%**
	1073	2073	2067	1071	1066	98	106
EXI-Telecomp	100%	100%	100%	100%	100%	99%	**102%**
	9933	19807	19808	9932	9931	111	137
EXI-weblog	100%	100%	100%	100%	100%	100%	100%
	8504	16997	16997	8504	8504	44	58
JST_gene.chr1	100%	99%	100%	99%	100%	**430%**	396%
	9176	14718	14103	7840	7206	917	1062
JST_snp.chr1	100%	98%	101%	97%	101%	**382%**	347%
	23509	41444	37425	22805	19111	2571	2980
medline	165%	150%	**240%**	141%	222%	145%	141%
	395754	493136	158984	326638	113932	122270	88109
NCBI_gene.chr1	100%	93%	110%	91%	116%	**148%**	137%
	16038	15504	9839	11606	5912	4237	3764
NCBI_snp.chr1	100%	100%	100%	100%	100%	100%	100%
	404704	809394	809394	404704	404704	61	83
sprot39.dat	102%	60%	**269%**	67%	237%	102%	102%
	1724689	2394532	586523	1484814	376067	328469	257376
treebank	101%	99%	106%	99%	**107%**	104%	103%
	1292198	1455300	1171666	1246195	1039933	1073301	510683

the reverse hybrid dag, i.e., after building the dag there is less profit from canonization. An interesting document that has always been challenging with respect to compression [6] is medline: with 165% it has the largest increase within the dag column. This means that many permutations of the same subtree sequences exist. This could be because these bibliography entries have been entered manually by different persons, each having their own preferences of ordering sibling lists. Observe also that every single compressor has an increase of at least 140% for the medline document. Similar to this is the 1998statistics document: here the dag only increases by 118%, but all others increase by 210% or more. Thus, there are not many subtrees with precisely the same subtrees (possibly in different orders), but, there is a large number of repetitions of subsequences of sibling lists, in particular of prefix subsequences (viz. the highest increases of rbdag and bdag).

In summary, Figure 4 shows the average sizes for the different compressors for Corpus I and Corpus II, respectively. For Corpus I, all compressors, except bdag (91%) and hdag (93%), show an improvement of the compression ratio. DS (118%) and TR (124%), which already give very high compression ratios, also have high increases. The biggest increases, however, are seen for rbdag (134%) and rhdag (130%). For Corpus II, we see that there is almost no difference in the case of bdag and hdag. Again, DS (121%) and TR (123%) improve on their already high compression ratios, while rbdag (117%) and rhdag (120%) achieve improvements as well.

Finally, we also tried a different canonizer: It assigns to every subtree t a number $i(t)$ such that for every pair of subtrees t_1, t_2 it holds that $i(t_1) = i(t_2)$ if and only if the unordered trees of t_1 and t_2 are equal. The children $t_1, \ldots, t_n$ of a subtree t are then sorted with respect to $i(t_1), \ldots, i(t_n)$. While this algorithm runs a lot faster than sorting the whole subtrees, the compression ratios only change very slightly.

6 Conclusions

In this paper we showed that the minimal *unordered dag* of a tree can be exponentially smaller than the minimal (ordered) dag of that tree. Furthermore, we proved that this difference is exponential at most, thus providing matching upper and lower bounds for the ratio of the ordered and unordered dag size of a tree. These results hold for both size measures: number of nodes and the number of multi-edges. It would be interesting in the future to investigate also the number of "collapsed" edges, where multi-edges between the same pair of nodes are collapsed to a single edge. For the unordered dag, this size measure makes sense, since one only needs to store the number of multi-edges between two nodes and these numbers can be stored succinctly in binary notation. Another interesting theoretical research problem is to compute the average size of the unordered dag, where the average is taken with respect to the uniform distribution on all trees of size n. The corresponding average for ordered dags was analysed in [10] and the asymptotic growth rate $\Theta(n/\sqrt{\log n})$ was derived using tools from analytic combinatorics, see also [5]. We conjecture that the average unordered dag size has the same asymptotic growth. Related to this, one might study the average values of the ratios α_N and α_E for which we only derived worst case bounds.

In the second part of this paper, we have experimentally evaluated the difference of sibling orders for different compressors. Within the compressors that are based on dags, the biggest impact of ignoring order is observed for the reverse binary dag (thus, the minimal dag of the first-child/*previous-sibling*-encoded tree); the biggest difference is a better compression in the unordered setting by a factor of 3.7. Within the RePair-variants DS and TreeRePair, DS features the larger difference, with a factor of 191 for one document. Such a massive

improvement due to ignoring document order could be useful in practice: it could mean that the entire tree structure of a document that is Terabytes large, can be conveniently stored (and queried) in main memory, while the data values would be stored on secondary memory.

It would be interesting to consider trees with ordered and unordered nodes. For XML documents this can be done via XML Schema Definition. JSON very naturally provides this possibility: its two primitives are lists (= ordered) and sets of key-value pairs (= unordered). We expect that canonization of unordered nodes will improve compression. In the future, we would like to find also a method that combines features of the compressors presented here, and is guaranteed to compress equally well or better than each of the the compressors.

References

1 S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. *Theory of Computing Systems*, 57(4):927–966, 2015.

2 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

3 A. Boiret, V. Hugot, J. Niehren, and R. Treinen. Logics for unordered trees with data constraints on siblings. In *Proceedings of LATA 2015*, volume 8977 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2015.

4 I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theory of Computing Systems*, 57(2):337–376, 2015.

5 M. Bousquet-Mélou, M. Lohrey, S. Maneth, and E. Noeth. XML compression via directed acyclic graphs. *Theory of Computing Systems*, 57(4):1322–1371, 2015.

6 P. Buneman. Private Communication, 2005.

7 P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proceedings of VLDB 2003*, pages 141–152. Morgan Kaufmann, 2003.

8 S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Proceedings of KGC 1997*, volume 1289 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 1997.

9 P. J. Downey, R. Sethi, and R. Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, 1980.

10 P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Proceedings of ICALP 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990.

11 M. Ganardi, D. Hucke, A. Jeż, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. Technical report, arXiv.org, 2014. `http://arxiv.org/abs/1407.4286`.

12 D. E. Knuth. *The Art of Computer Programming, Vol. I: Fundamental Algorithms*. Addison-Wesley, 1968.

13 N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proceedings of DCC 1999*, pages 296–305. IEEE Computer Society, 1999.

14 H. Liefke and D. Suciu. XMILL: an efficient compressor for XML data. In *Proceedings of ACM SIGMOD Conference 2000*, pages 153–164. ACM, 2000.

15 M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.

16 F. Neven and T. Schwentick. XML schemas without order. Unpublished manuscript, 1999.

17 J. M. Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. MAA Problem Books Series. Cambridge University Press, 2004.

18 S. Sundaram and S. Kumar Madria. A change detection system for unordered XML data using a relational model. *Data & Knowledge Engineering*, 72:257–284, 2012.

19 Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.
20 S. Zhang, Z. Du, and J. Tsong-Li Wang. New techniques for mining frequent patterns in unordered trees. *IEEE Transactions on Cybernetics*, 45(6):1113–1125, 2015.

Dynamic Complexity under Definable Changes*

Thomas Schwentick[1], Nils Vortmeier[2], and Thomas Zeume[3]

1 **TU Dortmund University, Dortmund, Germany**
`thomas.schwentick@tu-dortmund.de`
2 **TU Dortmund University, Dortmund, Germany**
`nils.vortmeier@tu-dortmund.de`
3 **TU Dortmund University, Dortmund, Germany**
`thomas.zeume@tu-dortmund.de`

Abstract

This paper studies dynamic complexity under definable change operations in the DynFO framework by Patnaik and Immerman. It is shown that for changes definable by parameter-free first-order formulas, all (uniform) AC^1 queries can be maintained by first-order dynamic programs. Furthermore, many maintenance results for single-tuple changes are extended to more powerful change operations: (1) The reachability query for undirected graphs is first-order maintainable under single tuple changes and first-order defined insertions, likewise the reachability query for directed acyclic graphs under quantifier-free insertions. (2) Context-free languages are first-order maintainable under Σ_1-defined changes. These results are complemented by several inexpressibility results, for example, that the reachability query cannot be maintained by quantifier-free programs under definable, quantifier-free deletions.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Dynamic descriptive complexity, SQL updates, dynamic programs

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.19

1 Introduction

In the setting of *Dynamic Complexity*, a database $\mathcal{D}$ is being changed and an update program $\mathcal{P}$ tries to answer a standing query q after each change. The program usually consists of logical formulas which can make use of additional, *auxiliary* relations which in turn need to be updated after each change. Dynamic Complexity can be seen as a logic-based counterpart of *Dynamic Algorithms*, where algorithms use auxiliary data structures to keep track of properties of structures like graphs under change operations. The Dynamic Complexity framework was introduced in [23] and a similar framework, FOIES, in [8].

In Dynamic Complexity, one usually allows first-order logic formulas as update mechanism for the auxiliary relations. This is in line with the database-oriented framework, since first-order logic correspond to database languages like relational algebra. Just as in Dynamic Algorithms, for most investigations the possible change operations are limited to insertions and deletions of single tuples. The class of queries maintainable in this fashion is called DynFO. This line of research has seen recent progress, particular with respect to the question whether the reachability query can be maintained in DynFO for directed graphs [2, 3].

* The authors acknowledge the financial support by DFG grant SCHW 678/6-2. The third author thanks the Simons Institute for the Theory of Computing for hosting him and providing excellent research conditions.

Although the restriction to single-tuple changes can be justified by the need to concentrate on the basic phenomena of dynamic maintainability of queries, it is also clear that from a more practical perspective one would be interested in more complex change operations at a time. One approach is to specify changes by "Δ-relations", e.g., by sets of tuples to be inserted or deleted. This is basically the viewpoint of *Incremental View Maintenance* (see for example [15]). However, it is clear that arbitrary Δ-relations can make the auxiliary relations useless.

In this work, we consider a different extension of the single-tuple-change paradigm that is inspired by SQL update queries (for a theoretical view at SQL updates we refer to [1]). We model such queries by *replacement queries* which can modify several relations at a time by first-order formulas that can use tuples of elements as parameters. Similar but slightly weaker frameworks were introduced in [17, 28], but these papers did not study maintainability under such complex changes.

Contributions. The generalized setting yields a huge range of research questions, e.g., all previously studied questions in Dynamic Complexity in combination with replacement queries of varying expressiveness, and this paper can only start to investigate a few of them.

We are mainly interested in positive results. In Section 4 we study first-order definable *insertion* queries (supplementing the single tuple changes). It turns out that the reachability query can still be maintained in DynFO for undirected graphs under first-order definable insertions (Theorem 3) and for directed acyclic graphs under quantifier-free insertions (Theorem 5). In Section 5, we investigate parameter-free replacement queries. We show that *all* queries that can be expressed in uniform AC^1 (and thus all queries that can be computed with logarithmic space) can be maintained in DynFO under first-order definable parameter-free replacement queries (Theorem 7). In Section 6, we show that many maintainability results for formal languages [23, 13] carry over to quantifier-free or Σ_1-definable replacement queries (Theorems 8 and 9).

It is notoriously difficult to prove inexpressibility results in Dynamic Complexity. One would expect that allowing more general change operations simplifies such results. In Section 7, we confirm this intuition to some extent and present cases where general replacement queries disable certain kinds of update programs to maintain queries that are maintainable under single-tuple changes.

Some proofs are omitted due to space constraints and can be found in the full version of this paper [24].

Related work. In addition to the related work mentioned already above, several other prior results for Dynamic Complexity under more general changes have been obtained. The reachability query for directed graphs has been studied under deletions of sets of edges and nodes that form an anti-chain in [5] and under insertions of sets of tuples that are cartesian-closed in [8]. Hesse observed that the maintenance procedure for this query under single tuple changes from [3] can deal with the replacement of the set of outgoing edges of a node (or, alternatively, the set of incoming edges). Edge contractions have been studied in [26]. Koch considered more general sets of changes in [19], though only for non-recursive queries.

Implementations of work on Dynamic Complexity are reported in [22] and [19].

2 Preliminaries

As much of the original motivation for the investigation of dynamic complexity came from incremental view maintenance (cf. [9, 6, 23]), it is common to consider logical structures as relational databases and to use notation from relational databases.

A *(relational) schema* τ consists of a set τ_{rel} of relation symbols, accompanied by an arity function $\text{Ar} : \tau_{\text{rel}} \to \mathbb{N}$, and a set τ_{const} of constant symbols. In this work, a *domain* is a finite set. A *database* $\mathcal{D}$ over schema τ with domain D assigns to every relation symbol $R \in \tau_{\text{rel}}$ a relation of arity $\text{Ar}(R)$ over D and to every constant symbol $c \in \tau_{\text{const}}$ an element (called *constant*) from D. A *τ-structure* $\mathcal{S}$ is a pair $(D, \mathcal{D})$ where D is a domain and $\mathcal{D}$ is a database with domain D over schema τ. By $\text{dom}(\mathcal{S})$ we refer to D. For a relation symbol $R \in \tau$ and a constant symbol $c \in \tau$ we denote by $R^{\mathcal{S}}$ and $c^{\mathcal{S}}$ the relation and constant, respectively, that are assigned to those symbols in $\mathcal{S}$. A *k-ary query* q on τ-structures is a mapping that assigns a subset of D^k to every τ-structure over domain D and is closed under isomorphisms.

We represent graphs as structures over a schema that contains a single binary relation E. The *reachability query* q_{Reach} maps graphs to their transitive closure relation.

In Section 6 we consider databases that represent words over some alphabet Σ. In a nutshell, the positions of a word correspond to elements of the domain and the letters at positions are indicated by unary relations. More formally, words are represented by databases with an immutable linear order on their domain and one unary relation R_σ for every $\sigma \in \Sigma$. For simplicity, we always assume that the domain of such a database is of the form $\{1, \dots, n\}$ and the linear order ist just the natural order. At any point in time, an element of the domain is allowed to be in at most[1] one relation R_σ. However, elements need not to be in any relation R_σ and, in this case, they do not correspond to a position with a symbol but rather to the empty word ϵ. Thus, we first associate with every position i an element $w_i \in \Sigma_\epsilon$, where by Σ_ϵ we denote the set $\Sigma \cup \{\epsilon\}$, and say that the database represents the string $w = w_1 \cdots w_n$. As an example, the database with domain $\{1, 2, 3, 4, 5\}$ and $R_a = \{2, 4\}, R_b = \{1\}$ represents the string *baa*. As a further convenience, we assume that databases have constants min and max that represent the smallest and the largest element, 1 and n, respectively.[2] We will not allow the linear order, min or max to be modified by change operations. In this paper, we will rarely distinguish between a database and the string it represents.

We use several notions from finite model theory (see, e.g., [20]). By $\text{qd}(\varphi)$, we denote the *quantifier-rank* of a first-order formula φ, that is, its maximum nesting depth of quantifiers. We denote the set of *rank-k types* of tuples of arity ℓ by $\text{FO}[k, \ell]$ (cf. [20, Definition 3.14]). The existential fragment of first-order logic is denoted by Σ_1.

3 Dynamic Programs with Complex Changes

In this section we lift the definitions from [25] to more general change operations. We first define (general) change operations, then we adapt the definition of dynamic programs presented in [25] to those more complex changes.

[1] There are ways to get rid of this requirement, but we keep it for simplicity.

[2] This assumption can be avoided by, e.g., using additional prefix and suffix relations in the proof of Theorem 8, in the spirit of [13].

Change Operations. The change operations that we consider in this paper are based on queries. In their most general form, they can modify a given database over schema τ by replacing some of its relations with the results of first-order-defined queries on the database. These queries are allowed to use parameters.

To this end, a *replacement rule* ρ_R for relation R is of the form **replace** R **by** $\mu_R(\bar{p};\bar{x})$. Here, R is a relation symbol and $\mu_R(\bar{p};\bar{x})$ is a first-order formula over τ, where the tuple $\bar{x}$ has the same arity as R and $\bar{p}$ is another tuple of variables, called the *parameter tuple*. A *replacement query* $\rho(\bar{p})$ is a set of replacement rules for distinct relations with the same parameter tuple $\bar{p}$. In the case of replacement queries ρ that consist of a single replacement rule, we usually do not distinguish between ρ and its single replacement formula μ_R.

For a database $\mathcal{D}$, a change operation $\delta = (\rho, \bar{a})$ consists of a replacement query and a tuple of elements of (the domain of) $\mathcal{D}$ with the same arity as the parameter tuple of ρ. We often use the more concise notation $\rho(\bar{a})$ and refer to change operations simply as *changes*.

The result $\delta(\mathcal{D})$ of an application of a change operation $\delta = (\rho, \bar{a})$ to a database $\mathcal{D}$ is defined in a straightforward way: each relation R in $\mathcal{D}$, for which there is a replacement rule ρ_R in ρ, is replaced by the relation resulting from evaluating μ_R, that is, by $\{\bar{b} \mid \mathcal{D} \models \mu_R(\bar{a};\bar{b})\}$.

If a replacement query has no parameters we say that it is *parameter-free*.

▶ **Example 1.**

(a) As a first example, we consider directed graph structures, that is, structures with a single binary relation E. Let, for some graph G, $\delta_1 = (\rho_1, u)$ be the change operation with replacement query $\mu_E(p; x, y) = E(x, y) \vee (x = p)$ and node u. Then, in $\delta_1(G)$, there is an edge from u to every node of G.

(b) We recall that words over the alphabet $\Sigma = \{a, b, c\}$ are represented by databases with a linear order on their domain and one unary relation R_σ for every $\sigma \in \Sigma$. Let $\mathcal{D}$ be a database representing a word w and i an element of $\mathcal{D}$. Let $\delta_2 = (\rho_2, i)$ be a change operation, where the replacement query ρ_2 consists of the rules **replace** R_a **by** $\mu_{R_a}(p; x)$ and **replace** R_b **by** $\mu_{R_b}(p; x)$ with $\mu_{R_a}(p; x) = \big((x < p) \wedge R_b(x)\big) \vee \big(\neg(x < p) \wedge R_a(x)\big)$ and $\mu_{R_b}(p; x) = \big((x < p) \wedge R_a(x)\big) \vee \big(\neg(x < p) \wedge R_b(x)\big)$. Then, $\delta_2(\mathcal{D})$ represents the word obtained from w by swapping a and b symbols on all positions before i and leaving all other positions unchanged.

Some of our investigations will focus on (syntactically) restricted replacement queries that either only remove or only insert tuples to relations. For an *insertion rule* ρ_R, the replacement formula $\mu_R(\bar{p};\bar{x})$ has the form $R(\bar{x}) \vee \varphi_R$. Similarly, *deletion rules* have replacement formulas $\mu_R(\bar{p};\bar{x})$ of the form $R(\bar{x}) \wedge \varphi_R$. In [1], the change operations *replace*, *insert*, *delete* and *modify* have been studied, in particular with respect to their expressive power. These operations are captured by our change operations[3].

Another syntactic restriction to be studied extensively in this work are the quantifier-free replacement queries, that allow only quantifier-free change formulas to be used. A special case of quantifier-free changes are the *single tuple changes*. We refer by **insert** $\bar{p}$ **into** R to the insertion query **replace** R **by** $\mu_R(\bar{p};\bar{x})$, where $\mu_E(\bar{p};\bar{x}) = R(\bar{x}) \vee (\bar{p} = \bar{x})$ and by **delete** $\bar{p}$ **from** R to the deletion query **replace** R **by** $\mu_R(\bar{p};\bar{x})$, where $\mu_R(\bar{p};\bar{x}) = R(\bar{x}) \wedge \neg(\bar{p} = \bar{x})$. As mentioned before, single tuple changes are the best studied change operations in previous work on dynamic complexity. To emphasize the difference we sometimes refer to arbitrary (not single-tuple) change operations as *complex changes*. For any schema τ we denote by Δ_τ the

[3] In [1] the domain of the database can be infinite.

set of single-tuple replacement queries for the relations (with symbols) in τ. In the case of graphs, we simply write Δ_E. In case of strings over some alphabet Σ, we write Δ_Σ.

Dynamic Programs. We now introduce dynamic programs, closely following the exposition in [25]. Inputs in dynamic complexity are represented as relational structures as defined in Section 2. The domain is fixed from the beginning, but the database in the initial structure is empty. This initially empty structure is then modified by a sequence of change operations.

The goal of a dynamic program is to answer a given query for the database that results from any change sequence. To this end, the program can use an auxiliary data structure represented by an auxiliary database over the same domain. Depending on the exact setting, the auxiliary database might be initially empty or not.

A dynamic program $\mathcal{P}$ operates on an *input database* $\mathcal{I}$ over a schema τ_{in} and updates an *auxiliary database* $\mathcal{A}$ over a schema[4] τ_{aux}, both sharing the same domain D which is fixed during a computation. We call $(D, \mathcal{I}, \mathcal{A})$ a *state* and consider it as one relational structure. The relations of $\mathcal{I}$ and $\mathcal{A}$ are called *input and auxiliary relations*, respectively.

A *dynamic program* has a set of update rules that specify how auxiliary relations are updated after a change. An *update rule* for updating an auxiliary relation T after a replacement query $\rho(\bar{p})$ is of the form **on change** $\rho(\bar{p})$ **update** $T(\bar{x})$ **as** $\varphi_T(\bar{p}, \bar{x})$ where the *update formula* φ_T is over $\tau_{\text{in}} \cup \tau_{\text{aux}}$.

The semantics of a dynamic program is as follows. When a change operation $\delta = \rho(\bar{a})$ is applied to the input database $\mathcal{I}$, then the new state $\mathcal{S}$ of $\mathcal{P}$ is obtained by replacing the input database by $\delta(\mathcal{I})$ and by defining each auxiliary relation T via $T \stackrel{\text{def}}{=} \{\bar{b} \mid (\mathcal{I}, \mathcal{A}) \models \varphi_T(\bar{a}, \bar{b})\}$. For a change operation δ we denote the updated state by $\mathcal{P}_\delta(\mathcal{S})$. For a sequence $\alpha = (\delta_1, \ldots, \delta_k)$ we write $\mathcal{P}_\alpha(\mathcal{S})$ for the state obtained after successively applying $\delta_1, \ldots, \delta_k$ to $\mathcal{S}$.

A *dynamic query* is a tuple (q, Δ) where q is a query over schema τ_{in} and Δ is a set of replacement queries. The dynamic program $\mathcal{P}$ *maintains* a dynamic query (q, Δ) with k-ary q if it has a k-ary auxiliary relation Q that, after each change sequence over Δ, contains the result of q on the current input database. More precisely, for each non-empty[5] sequence α of changes and each empty input structure $\mathcal{I}_\emptyset$, relation Q in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $q(\alpha(\mathcal{I}_\emptyset))$ coincide. Here, $\mathcal{S}_\emptyset = (\mathcal{I}_\emptyset, \mathcal{A}_\emptyset)$, where $\mathcal{A}_\emptyset$ denotes the empty auxiliary structure over the domain of $\mathcal{I}_\emptyset$.

The class of dynamic queries (q, Δ) that can be maintained by a dynamic program with update formulas from first-order logic is called DynFO. We also say that the query q can be maintained in DynFO under change operations Δ. The class of dynamic queries maintainable by quantifier-free update formulas is called DynProp.

The following very simple example shows how the transitive closure of a directed graph subject to single edge insertions can be maintained in this set-up.

▶ **Example 2.** Let q_{Reach} be the reachability query that returns all pairs (u, v) of a graph, for which there is a path from u to v. The dynamic query $(q_{\text{Reach}}, \{\textbf{insert } \bar{p} \textbf{ into } E\})$ can be maintained by a dynamic program that uses one auxiliary relation T, which always contains the transitive closure of the edge relation E. Its only update rule is given by the formula $\varphi_T(p_1, p_2; x, y) = T(x, y) \vee \big(T(x, p_1) \wedge T(p_2, y)\big)$. ◀

[4] To simplify the exposition, we will usually not mention schemas explicitly and always assume that all structures we talk about are compatible with respect to the schemas at hand.

[5] This technical restriction ensures that we can handle, e.g., Boolean queries with a yes-result on empty structures without initialization of the auxiliary relations. Alternatively, one could use an extra formula to compute the query result from the auxiliary (and input) structure.

Our general framework follows [23] and thus does not allow inserting new elements into or removing existing elements from the domain as in the FOIES framework [8]. The step from Dynamic Complexity to FOIES can be done by adding two more change operations, add(x) and remove(x). Our results of Section 4 easily carry over, and those of Section 6 carry over if, say, new elements are always added at the end of the string. Since add(x) and remove(x) have parameters, they do not quite fit into the parameter-free framework of Section 5. However, Theorem 7 survives if parameter-free remove queries are allowed.

Complex Change Operations and Initialization of Dynamic Programs. In the presence of complex replacement queries, the initialization of the auxiliary relations requires some attention. In the original setting of Patnaik and Immerman, the input database is empty at the beginning, and the auxiliary relations are initialized by first-order formulas evaluated on this (empty) initial input database. Since tuples can be inserted only one-by-one, the auxiliary relations can be adapted slowly and it can be ensured that, e.g., always a linear order [23] or arithmetic [10] on the active domain is available.

For complex changes, the situation is more challenging for a dynamic program: as an example, in the setting of strings, the first change could insert all positions of the domain into relation R_a and thus let the database represent the word a^n, if n is the size of the underlying domain. To enable the dynamic program to answer whether the string is in some language after this change, it needs some suitable (often: non-empty) initial values of the auxiliary relations. Since in this paper, we are mainly interested in the maintenance of queries and not so much in the specific complexity of the initialization, we do not define variants of DynFO with different power of initialization, but rather follow a pragmatic approach: whenever initialization is required, we say that the query can be maintained *with suitable initialization* and specify in the context what is actually needed. In all cases, it is easy to see that the initialization of the auxiliary relations can be computed in polynomial time.

An alternative approach would be to restrict the semantics of replacement queries to elements of the active domain of the current database and to allow the activation of elements only via tuple insertions.

4 Reachability and Definable Insertions

In this section, we study the impact of first-order definable complex change operations on the (binary) reachability query q_{Reach}. We present positive cases, where previous maintainability results survive under such stronger change operations. Negative results, where such operations destroy previous maintainability results, are given in Section 7.

In the classical DynFO setting with single-tuple change operations it was shown early on that q_{Reach} can be maintained in DynFO for two important graph classes: undirected graphs and directed, acyclic graphs (dags). It turns out that these results still hold in the presence of complex *insertions*: first-order insertions for undirected graphs and quantifier-free insertions for dags. In fact, in both cases basically the same auxiliary relations can be used as in the case of single-tuple changes.

We first show that for undirected graphs, the reachability query can be maintained in DynFO, for first-order insertions and the set Δ_E of single-edge insertions and deletions. We follow the convention from [14] that modifications for undirected graphs are symmetric in the sense that if an edge (a, b) is inserted then so is the edge (b, a) (and likewise for deletions).

▶ **Theorem 3.** *Let Δ be a finite set of first-order insertion queries. Then $(q_{Reach}, \Delta \cup \Delta_E)$ can be maintained in* DynFO *for undirected graphs.*

We use the approach for maintaining q_{Reach} for undirected graphs under single-edge insertions and deletions from [23, Theorem 4.1] and maintain a spanning forest and (essentially) its transitive closure relation. The crucial observation for extending this approach to first-order insertions is that, after such an insertion, between each pair of nodes in a (new) connected component, there is a connecting path that uses only a bounded number of newly inserted edges. This allows the update of the spanning forest and its transitive closure in a first-order definable way.

The observation is stated more precisely next. For two connected nodes u, v in a graph $G' = \delta(G)$ that is obtained from a graph G by an insertion δ, we define[6] the *bridge distance* $\text{bd}(u, v)$ as the minimal number d, such that there is a path from u to v in G' that uses d edges that were newly inserted by δ.

▶ **Lemma 4.** *For each first-order insertion query ρ there is a constant $m \in \mathbb{N}$ such that for each undirected graph G, each change $\delta = \rho(\bar{a})$ and all nodes u and v of G that are connected in $\delta(G)$ it holds $bd(u, v) \leq m$.*

We informally refer to this property as the *bridge boundedness property*.

Proof. The proof makes use of the result by Feferman-Vaught that the depth k first-order type of the disjoint union of two structures is determined by the depth k first-order types of these two structures [11, 12] (see also [21]).

Let $\mu(\bar{p}; \bar{x})$ be the first-order formula underlying ρ and k its quantifier-rank. Let ℓ be the arity of $\bar{p}$, m' the number of $\text{FO}[k, 1]$-types of undirected graphs and $m = \ell + m'$.

Let G be an undirected graph and let $\delta = \rho(\bar{a})$ for some tuple $\bar{a}$ of nodes of G. Let u, v be two nodes that are connected by some path π of the form $u = w_0, w_1, \ldots, w_r = v$ in $\delta(G)$ with q bridges, that is, edges that are not in G. Our goal is to show that there exists such a path with at most m bridges. Thus, if $q \leq m$, there is nothing to prove, so we assume $q > m$. It suffices to show that there is a path from u to v with fewer than q bridges. Let $(u_1, v_1), \ldots, (u_q, v_q)$ be the bridges in π. If for some i, the nodes u_i and v_i are in the same connected component of G (before the application of δ), we can replace the bridge (u_i, v_i) by a path of "old" edges resulting in an overall path with $q - 1$ bridges. Similarly, if u_i and u_j are in the same connected component of G, for some $i < j$, we can shortcut π by a path from u_i to u_j inside G. Therefore, we can assume that, for every i, the nodes u_i and v_i are in different connected components of G, and likewise u_i and u_j for $i < j$.

We show that in this case there are i, j with $i < j$ such that μ defines an edge between u_i and v_j, and therefore a path with fewer bridges can be constructed by shortcutting the path π with the edge (u_i, v_j). By the choice of m there must be two nodes u_i and u_j, with $i < j$, in distinct connected components of G that do not contain any element from $\bar{a}$, such that u_i and u_j have the same $\text{FO}[k, 1]$-type in their respective connected components. By Feferman-Vaught, it follows that $(u_i, v_j, \bar{a})$ and $(u_j, v_j, \bar{a})$ have the same $\text{FO}[k, \ell + 2]$-types and therefore, since μ defines an edge between u_j and v_j, it also defines one between u_i and v_j. ◀

Proof. Proof of Theorem 3 The dynamic program presented in [23, Theorem 4.1] maintains the transitive closure of undirected graphs under single-edge changes with the help of auxiliary relations F and PV. The binary relation F is a spanning forest of the input graph G and $(u, v, w) \in PV$ means that w is a node in the path from u to v in F. Observe that two nodes u and v are connected in an undirected graph if and only if $(u, u, v) \in PV$ holds.

[6] Since G and δ will be always clear from the context, we do not add them as parameters to this notation.

We show how to maintain the relation F and PV under FO insertions. For the moment we assume a predefined linear order $\leq$ on the domain to be present. Let ρ be an insertion query and m the bound on the number of bridges by Lemma 4. Let G be an undirected graph and $\delta = \rho(\bar{a})$ an insertion, F a spanning forest of G and PV as described above. We show how to FO-define the auxiliary relations F' and PV' for the modified graph $G' = \delta(G)$.

We first describe a strategy to define F' and then argue that it can be implemented by a first-order formula. Let C' be a (new) connected component in $\delta(G)$. We call the smallest node of C' with respect to $\leq$ the *queen* u_0 of C'. For each connected component C of G that is a subgraph of C', we define its *queen level* as the (unique) number $\mathrm{bd}(u, u_0)$, for nodes $u \in C$. A bridge in C' is inserted into F' if for a connected component C of G of some level i it is the lexicographically smallest edge with respect to $\leq$ that connects C with some component of level $i - 1$. This clearly defines a spanning forest. The chosen edges can be defined by a first-order formula because, for each number h, there are formulas $\theta_h(x, y)$ expressing that $\mathrm{bd}(x, y) \leq h$.

Since the construction of F' ensures that each path in F' from a node to the queen of its connected component only contains at most m new edges, and thus each path in F' contains at most $2m$ new edges, it is straightforward to extend the update formula for PV' from [23, Theorem 4.1].

It remains to show how the assumption of a predefined linear order can be eliminated. For a change sequence α, we denote by A_α the set of parameters used in α. When applying α to an initially empty graph, a linear order on A_α can be easily constructed as in the case of single-tuple changes [10]. The remaining nodes in $V \setminus A_\alpha$ behave very similarly. More precisely, one can show by induction on $|\alpha|$, that for all nodes $a \in V$ and $b, b' \in V - A_\alpha$ it holds $(a, b) \in E \Leftrightarrow (a, b') \in E$ (and likewise for (b, a) and (b', a)).

The dynamic program for maintaining q_{Reach} for undirected graphs now maintains the relations F and PV as described above, yet restricted to the induced (and ordered) subgraph of G on A_α. The transitive closure can be defined from those relations and the edge relation by a simple case distinction.

- Two nodes $a, a' \in A_\alpha$ are connected by a path if and only if there is a path from a to a' in F or if there are nodes $b, b' \in A_\alpha$ and a node $c \in V \setminus A_\alpha$ such that there are paths from a to b and from a' to b' in F as well as edges (b, c) and (b', c).
- Two nodes $a \in A_\alpha$ and $b \in V \setminus A$ are connected by a path if and only if there is a node $a' \in A_\alpha$ such that there is a path from a to a' in F and an edge (a', b).
- Finally, two nodes $a, a' \in V \setminus A_\alpha$ are connected if and only if there is an edge (a, a') or there is an edge (a, b) for some $b \in A_\alpha$ (and therefore also an edge (a', b)). ◀

Now we turn to the other graph class, acyclic graphs, for which DynFO maintainability under complex insertions (and single-edge deletions) is preserved; albeit (we are able to show that) only for quantifier-free insertions. In [23, Theorem 4.2], edge insertions are only allowed if they do not add cycles. Of course, given the transitive closure of the current edge relation it can be easily checked by a first-order formula (a *guard*), whether a new edge closes a cycle. We will see that this is also possible for the complex insertions we consider.

▶ **Theorem 5.** *Let Δ be a finite set of quantifier-free insertion queries. Then $(q_{Reach}, \Delta \cup \Delta_E)$ can be maintained in* DynFO *for directed, acyclic graphs. Furthermore, for each quantifier-free insertion, there is a first-order guard which checks whether the insertion destroys the acyclicity of the graph.*

As in the case of undirected graphs, the proof relies on a bridge boundedness property. This property allows extending the technique for maintaining the transitive closure relation of

acyclic graphs under single tuple changes used in [23] and [7] to quantifier-free insertions. As in [23] and [7] no further auxiliary relations besides the transitive closure relation are needed. In Section 7 we show that the transitive closure relation does not suffice for maintaining q_{Reach} for acyclic graphs subjected to Σ_1-definable insertions[7].

In the following lemma, the bridge distance bd is defined just as above. However, we can no longer assume that bridges connect (formerly) different connected components, therefore the lemma only holds for quantifier-free insertions. The proof can be found in the full version of this paper.

▶ **Lemma 6.** *For each quantifier-free insertion query ρ there is a constant $m \in \mathbb{N}$ such that for each directed, acyclic graph G and each change $\delta = \rho(\bar{a})$ it holds that $\delta(G)$ has a cycle with at most m bridges, or for all nodes u and v of G with a path from u to v in $\delta(G)$, it holds $bd(u, v) \leq m$.*

Proof. Proof of Theorem 5 In [23, Theorem 4.2] and [7, Theorem 3.3], dynamic programs are given that maintain the transitive closure of acyclic graphs under single-edge modification, using only the transitive closure as auxiliary relation. Thanks to Lemma 6, these programs can be easily extended. Indeed, since the number of bridges of cycles created by the insertion, and, if the graph remains acyclic, the bridge distance between two path-connected nodes are bounded by a constant, a guard formula and an update formula for the transitive closure can be constructed in a straightforward manner. ◀

5 Parameter-free Changes

In this section we consider replacement queries without parameters on ordered databases. It turns out that in this case a large class of queries can be maintained in DynFO: all queries that can be expressed in uniform AC^1 and thus, in particular, all queries that can be answered in logarithmic space. This result exploits the fact that for a fixed set of replacement queries without parameters there is only a constant number of possible changes to a structure.

An *ordered* database $\mathcal{D}$ contains a built-in linear order $\leq$ on its domain that is not modified by any changes. One might suspect that parameter-free replacement queries are not very powerful, especially when they are applied to the initially empty input database. However, thanks to the linear order, one can actually construct every finite graph with relatively simple replacement queries (and similarly for other kinds of databases). For instance, one can cycle through all pairs of nodes in lexicographic order. If (u, v) is the current maximal pair, operation *keep* can move to $(u, v + 1)$ (inserting it into E) while leaving (u, v) in E and *drop* can move to $(u, v + 1)$ while taking (u, v) out from E.

The update programs constructed in this section use, as additional auxiliary relation, a binary BIT-relation containing all pairs (i, j) of numbers, for which the i-th bit of the binary representation of j is 1. Here, we identify elements of an ordered database with numbers. In the following, the minimal element with respect to $\leq$ is considered as 0.

By AC^1 we denote the class of problems that can be decided by a uniform[8] family of circuits of "and", "or" and "not" gates with polynomial size, depth $\mathcal{O}(\log n)$ and unbounded fan-in. We show the following theorem.

[7] Indeed, the graphs G and G' used in the proof of Theorem 10(b) show that the following lemma fails already for Σ_1-insertions.

[8] For concreteness: first-order uniform [18].

▶ **Theorem 7.** *Let q be an* AC^1 *query over ordered databases and Δ a finite set of parameter-free first-order definable replacement queries. Then (q, Δ) is in* DynFO *with suitable initialization.*

Proof. We first explain the idea underlying the proof.

It uses the characterization of AC^1 by iterated first-order formulas. More precisely, we use the equality $\mathrm{AC}^1 = \mathrm{IND}[\log n]$ from [18, Theorem 5.22], where $\mathrm{IND}[t(n)]$ is the class of problems that can be expressed by applying a first-order formula $\mathcal{O}(t(n))$ times and n is the size of the domain[9]. We only give an example and refer to [18, Definition 4.16] for a formal definition. Consider the formula $\varphi_{\mathrm{TC}}(x,y) = (x = y) \vee E(x,y) \vee \exists z\big(R(x,z) \wedge R(z,y)\big)$. When applying the formula to a graph and an empty relation R it defines the relation R_1 of paths of length 1, applying it to $R \stackrel{\text{def}}{=} R_1$ defines the paths of length 2; in general applying the formula to $R \stackrel{\text{def}}{=} R_i$ defines the paths of length 2^i. Thus $\log n$-fold application of φ_{TC} defines the transitive closure relation of a graph with n vertices and therefore q_{Reach} is in $\mathrm{IND}[\log n]$.

Let q be a query in AC^1 and let k be such that q can be evaluated by $k \log n$ applications of a formula φ_q.

The program $\mathcal{P}$ uses a technique inspired from prefetching, which was called *squirrel technique* in [31]. At any point t in time[10], it starts a thread θ_β, for each possible future sequence β of $2 \log n$ change operations.

Within the next $\log n$ steps (i.e. changes), it compares whether the actual change sequence α is the prefix of β of length $\log n$. If not, thread θ_β is abandoned, as soon as α departs from β. For each of these $\log n$ steps, θ_β simulates two change operations of β and applies them to the graph G_t at time t, consecutively. After $\log n$ steps, that is, at time $t + \log n$, thread θ_β has computed $\beta(G_t)$.

During the next $\log n$ steps until time $t + 2 \log n$, θ_β evaluates q on $\beta(G_t)$ by repeatedly applying the formula φ_q, k times for each single step. Again, if the actual change sequence departs from β then θ_β is abandoned. However, if β is the actual change sequence from time t to $t + 2 \log n$, the thread θ_β does not stop and has the correct query result $q(\beta(G_t))$ at time $t + 2 \log n$.

We note that, although the time window in the above sketch stretches over $2 \log n$ change operations from time t to $t + 2 \log n$, the actual sequences whose effect on the current graph is precomputed are never longer than $\log n$. This is because the application of all $2 \log n$ operations of a sequence takes until time $t + \log n$ and by that time the first $\log n$ of these operations already lie in the past.

Of course, $\mathcal{P}$ uses a lot of prefetching. However, this is possible, because only a constant number, $d = |\Delta|$, of change operations is available at any time (and there are no parameters). Thus, there are only $d^{2 \log n} = 2^{2 \log d \log n} = n^{2 \log d}$ many different change sequences, each of which can be encoded by a tuple of arity $2 \log d$ over the domain.

This explains how $\mathcal{P}$ can give correct answers for all times $t \geq 2 \log n$. All previous time points have to be dealt with by the initialization. This initialization also equips the program with the BIT relation. Clearly, the initialization can be computed in AC^1, and therefore also in polynomial time. More details of this proof can be found in the full version of this paper. ◀

[9] In the setting of [18], first-order formulas may use built-in relations $\leq$ and BIT. The relation $\leq$ is also present here, the relation BIT can be generated by a suitable initialization, see [18, Exercise 4.18].

[10] We count the occurrence of one change as one time step.

6 Formal Languages and Σ_1-definable Change Operations

In this section, we consider the membership problem for formal languages and how it can be maintained, for regular and context-free languages, under certain kinds of complex changes.

The problem of maintaining formal languages dynamically has been studied intensely in the context of single insertions to and deletions from the relations R_σ (cf. Section 2). In that setting, the class of regular languages is exactly the class of languages maintainable in DynProp[11] and all context-free languages can be maintained in DynFO [13]. All regular, some context-free, and even some non-context-free languages can be maintained in DynFO with only unary auxiliary relations [16], but this is not possible for all context-free languages [30, 27].

Here, we consider the problem of maintaining formal languages under first-order definable change operations. We assume that only replacement queries are used whose application results in structures where each position is in at most one R_σ relation. For a given formal language L we denote the membership query for L as q_L.

We prove that regular and context-free languages can be maintained dynamically for large classes of change operations: all regular languages can be maintained in DynProp under quantifier-free change operations and all context-free languages can be maintained in DynFO under Σ_1-definable (and, dually, Π_1-definable) change operations. A setting, in which language membership can be maintained with respect to simple changes but not with respect to definable change operations is exhibited in Section 7. For quantifier-free change operations, the results are obtained by generalizations of the techniques of [13].

▶ **Theorem 8.** *Let L be a regular language and Δ a finite set of quantifier-free replacement queries. Then (q_L, Δ) can be maintained in* DynProp *with suitable initialization.*

Proof. Let L be a regular language of strings over alphabet Σ and $\mathcal{A} = (Q, \Sigma, \gamma, s, F)$ a corresponding deterministic finite automaton with set Q of states, transition function[12] γ, initial state s, and set F of accepting states. In [13, Proposition 3.3], the main auxiliary relations are of the form $S_{q,r}(i,j)$ where q, r are states of $\mathcal{A}$ and i, j are positions of the string under consideration. The intended meaning of $S_{q,r}$ is that $(i,j) \in S_{q,r}$ if and only if $\gamma^*(q, w_{i+1} \cdots w_{j-1}) = r$.[13] Notice that w_i and w_j are not relevant for determining whether $(i,j) \in S_{q,r}$.

In the presence of quantifier-free change operations it suffices to maintain binary auxiliary relations of the form $S^f_{q,r}$, where $f : \Sigma_\epsilon \to \Sigma_\epsilon$ is a *relabeling function*. The intended meaning is that $(i,j) \in S^f_{q,r}$ if and only if $\gamma^*(q, f(w_{i+1} \cdots w_{j-1})) = r$, where f is extended to strings in the straightforward way.[14] Clearly, $S_{q,r} = S^{\mathrm{id}}_{q,r}$.

For simplicity we show how to update $S^f_{q,r}$ for replacement queries of the form $\rho(p)$ with *one* parameter p. The general case works analogously, but is notationally more involved. A replacement query with one parameter basically consists of one quantifier free formula $\mu_\sigma(p; x)$, for each element $\sigma \in \Sigma$.

[11] So, only using quantifier-free update formulas.
[12] Since in this paper δ denotes change operations, we use γ for transition functions.
[13] The relations $S_{q,r}$ were actually named $R_{q,r}$ in [13], but we want to avoid confusion with the R_σ relations. Since [13] did not use constants min and max, it used further auxiliary relations of the form I_r and F_q that contain all positions i with $\gamma^*(s, w_1 \cdots w_{i-1}) = r$, and $\gamma^*(q, w_{i+1} \cdots w_n) \in F$, respectively.
[14] It should be noted that f need not be a homomorphism since $f(\epsilon) \neq \epsilon$ is allowed.

We show how the relations $S^f_{q,r}$ can be maintained by quantifier-free update formulas $\phi^f_{q,r}(p;x,y)$. Then the (Boolean) query relation can be updated by the formula

$$\bigvee_{q,r\in Q,r'\in F} \psi_{s,q}(\min) \wedge \phi^{\mathrm{id}}_{q,r}(p;x,y)(\min,\max) \wedge \psi_{r,r'}(\max),$$

where $\psi_{q,r}(x)$ is a formula that expresses that $\delta(q,w_x)=r$.

Intuitively, each formula $\mu_\sigma(p;x)$ determines whether position x carries σ after the change. Whether this is the case only depends on (1) the current symbol at position x, (2) the current symbol at position p, and (3) on the relative order of x and p. Thus, the impact of a change can be described as follows: some relabeling function $f_\leftarrow$ is applied at all positions $x<p$, some change might occur at position p and some relabeling function $f_\rightarrow$ is applied at all positions $x>p$. More precisely, from ρ one can derive, for each[15] $\tau\in\Sigma_\epsilon$, relabeling functions $f^\tau_\leftarrow$, $f^\tau_\rightarrow$ and a symbol $\sigma(\tau)$ such that the update formula for a relation $S^f_{q,r}$ can be described by the formula

$$\phi^f_{q,r}(p;x,y) = x<y \wedge \bigvee_{\tau\in\Sigma_\epsilon} \Big(R_\tau(p) \wedge \Big((p\le x \wedge R^{f\circ f^\tau_\rightarrow}_{q,r}(x,y)) \vee (p\ge y \wedge R^{f\circ f^\tau_\leftarrow}_{q,r}(x,y)) \vee \\ \big(x<p<y \wedge \bigvee_{q',r'} (R^{f\circ f^\tau_\leftarrow}_{q,q'}(x,p) \wedge \chi_{q',f(\sigma(\tau)),r'} \wedge R^{f\circ f^\tau_\rightarrow}_{r',r}(p,y))\big)\Big)\Big),$$

where formulas of the form $\chi_{q',a,r'}$ are defined as $\top$ if $\delta(q',a)=r'$ and $\bot$, otherwise.

The initialization of the relations $S^f_{q,r}$ is straightforward. If, for a relabeling function f, $f(\epsilon)=\sigma$ then a pair (i,j) is in $S^f_{q,r}$ if and only if $\gamma^*(q,\sigma^{j-i-1})=r$. ◀

We next turn to context-free languages. The ideas underlying the proof of Theorem 8 can be adapted to show that the result from [13], that (membership for) context-free languages can be maintained in DynFO under simple change operations, survives under quantifier-free change operations. Through some little extra effort, this can be extended to Σ_1-definable change operations (and dually, Π_1-definable change operations).

▶ **Theorem 9.** *Let L be a context-free language and Δ a finite set of Σ_1-definable replacement queries. Then (q_L,Δ) can be maintained in* DynFO *with suitable initialization.*

The proof of Theorem 9 can be found in the full version of this article. It first shows how context-free languages can be maintained under quantifier-free changes, basically combining the idea of the proof of Theorem 8 with that of [13, Theorem 4.1]. Then it shows how the case of Σ_1-definable changes can be reduced to the quantifier-free case.

7 Inexpressibility Results

We finally turn to inexpressibility results. It is notoriously difficult to show that a query *cannot* be maintained by a DynFO program. Indeed, there are no inexpressibility results for DynFO besides those that follow from the easy observation that every query that can be maintained in DynFO under single-tuple insertions is computable in polynomial time.

We expect that it should be easier to prove inexpressibility results for DynFO in the presence of first-order definable change operations. However, we have no results of this form yet. But the following results confirm that, unsurprisingly, complex change operations can

[15] Since the schema is clear from the context, we use τ here to denote a symbol from Σ.

make it harder to maintain a query. We give two examples where allowing complex changes destroy a previous maintainability result, Theorems 10 and 13, and one example, Theorem 12 where we are able to show an inexpressibility result in the presence of complex deletions but not yet for single-tuple changes.

Towards our first result, we recall that the reachability query can be maintained under single-tuple insertions with the transitive closure of the edge relation as only auxiliary relation and that this does not hold if one allows single-tuple deletions [4]. We show next that the transitive closure also does not suffice in the presence of single-tuple insertions and one complex insertion query.

For general directed graphs, a parameter-free and quantifier-free insertion query suffices, for acyclic graphs a parameter-free insertion query defined by an existential formula suffices. The latter result should be contrasted with Theorem 5.

▶ **Theorem 10.**

(a) *There is a quantifier-free and parameter-free insertion query ρ such that $(q_{Reach}, \{\textbf{\textit{insert}}\ \bar{p}\ \textbf{\textit{into}}\ E, \rho\})$ cannot be maintained in* DynFO *on ordered directed graphs, if all auxiliary relations besides the query relation and the linear order are unary.*

(b) *There exists an Σ_1-definable and parameter-free insertion query ρ', for which the above statement holds even restricted to* acyclic*, directed graphs.*

Proof. For ease of presentation, we first give a proof for unordered directed graphs.

The proof follows an approach that has been used often before and that was made precise in [30]. We say that a k-ary query q is expressed by a formula $\varphi(\bar{x})$ with *help relations of schema* τ, if, for every database $\mathcal{D}$, there is a τ-structure H over the same domain such that for every k-tuple $\bar{a}$ over the domain of $\mathcal{D}$ it holds[16]: $\bar{a} \in q(\mathcal{D})$ if and only if $(\mathcal{D}, H) \models \varphi(\bar{a})$.

The proof is by contradiction and proceeds in the same way in both cases, (a) and (b). Our goal is to show that, under the assumption that there is a dynamic program for (a) or (b), the transitive closure of path graphs, that is, graphs that consist of a single directed path, can be expressed with unary help relations, contradicting the following lemma from [30], which is not hard to prove with the help of locality arguments.

▶ **Lemma 11** ([30, Lemma 4.3.2])**.** *The transitive closure of path graphs cannot be expressed by a first-order formula with unary help relations.*

We refer to Figure 1 for an illustration of the following high-level sketch. We start from an arbitrary path graph G_0 and equip it with some unary relations C_0, C_1, C_2. From G_0, C_0, C_1, C_2 we define a graph G in a first-order fashion, whose simple directed paths have length at most 2, so the transitive closure relation TC of G is definable by a first-order formula. Finally, the crucial step happens: the change operation δ transforms G into a graph $G' = \delta(G)$ with the property that $q_{\text{Reach}}(G_0)$ can be defined from $q_{\text{Reach}}(G')$ by a first-order formula. We can conclude that $q_{\text{Reach}}(G_0)$ can be defined by a first-order formula with the help of suitable unary help relations, since all steps from G_0 to G' are first-order definable, TC is first-order definable from G, and we assume that there is a dynamic program that computes $q_{\text{Reach}}(G')$ from G, TC and some unary auxiliary relations. This contradicts Lemma 11.

[16] This notion should not be confused with definability of the query q in existential second-order logic. In the latter case, the relations can be chosen depending on $\bar{a}$, but here the relations need to "work" for all tuples $\bar{a}$.

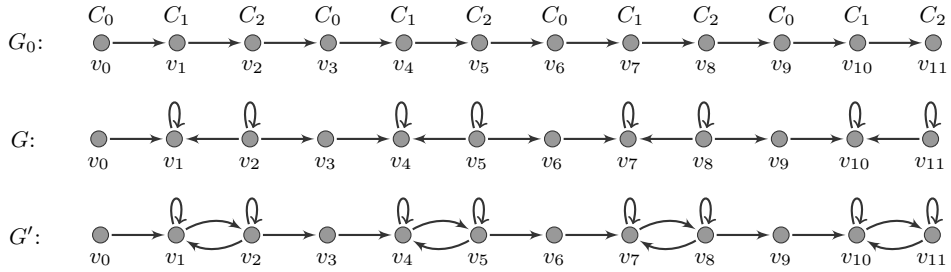


Figure 1 The graphs from the proof of Theorem 10(a).

For (a), we use the insertion query $\rho = \mu_E(x,y) \stackrel{\text{def}}{=} E(x,y) \vee \big(E(x,x) \wedge E(y,y) \wedge E(y,x)\big)$ that adds all edges (x,y) for which there is an edge (y,x) and both x and y have self-loops. We assume that there is a DynFO-program $\mathcal{P}$ that maintains the reachability query on directed graphs under insertion queries $\{\textbf{insert } \bar{p} \textbf{ into } E, \rho\}$. We further assume that $\mathcal{P}$ uses (only) unary auxiliary relations $B_1, \ldots, B_k$, for some k, besides the binary relation Q intended to store the query result. We show how to construct from $\mathcal{P}$ a first-order formula φ that expresses the reachability query q_{Reach} for simple paths with unary help relations $B_1, \ldots, B_k, C_0, C_1, C_2$, contradicting Lemma 11.

Let $G_0 = (V_0, E_0)$ be a simple path with $V_0 = \{v_0, \ldots, v_n\}$, for which we want to define q_{Reach} using unary help relations $B_1, \ldots, B_k, C_0, C_1, C_2$. Let C_0, C_1, C_2 be defined by $C_i = \{v_j \mid 0 \leq j \leq n, j \equiv_3 i\}$ where $\equiv_3$ denotes modulo 3 equivalence. From G_0 and C_0, C_1, C_2 we define the following graph G with nodes $v_0, \ldots, v_n$. The graph G has an edge from vertex v to w if one of the following cases holds:

- $v \in C_0, w \in C_1$ and (v,w) is an edge in G_0,
- $v \in C_1, w \in C_2$ and (w,v) is an edge in G_0,
- $v \in C_2, w \in C_0$ and (v,w) is an edge in G_0, or
- $v \in C_1 \cup C_2$ and $v = w$.

We observe that the graph G can be first-order defined from G_0 and C_0, C_1, C_2.

Let $\delta \stackrel{\text{def}}{=} \rho$ and[17] $G' \stackrel{\text{def}}{=} \delta(G)$. The graphs G_0, G and G' for $n = 11$ are depicted in Figure 1. By our assumption, the update formula $\psi = \phi_\delta^Q(x_1, x_2)$ of $\mathcal{P}$ for the query relation Q and operation δ defines the reachability query for $\delta(G) = G'$ with the help of suitable auxiliary relations $B_1, \ldots, B_k$ and the transitive closure TC of the edge relation of G.

Altogether, $G = f(G_0, C_0, C_1, C_2)$, for some first-order definable function f, TC is first-order definable from G, $G' = \delta(G)$, $q_{\text{Reach}}(G')$ is first-order definable from G, TC and $B_1, \ldots, B_k$, and therefore

$$q_{\text{Reach}}(G_0) = q_{\text{Reach}}(G') \setminus \{(w,v) \mid v \in C_1, w \in C_2, (v,w) \text{ is an edge in } G_0\}$$

is first-order definable from G_0, C_0, C_1, C_2, and $B_1, \ldots, B_k$, contradicting Lemma 11, as desired.

The proof for (b) and the extension to ordered graphs can be found in the full version of this paper. ◀

We now turn towards inexpressibility by quantifier-free update formulas. Very likely quantifier-free update formulas are too weak to maintain q_{Reach} even under single-tuple

[17] Since ρ is parameter-free, the insertion query ρ and its corresponding change operation δ are basically the same.

changes. Yet only restricted inexpressibility results have been obtained so far. The query q_{Reach} cannot be maintained in DynProp under single-tuple changes when the auxiliary relations are at most binary or when the initialization is severely restricted [32]. For the more general alternating reachability query quantifier-free update formulas do not suffice [13]. The next result shows that q_{Reach} cannot be maintained in DynProp, even if besides single-edge insertions only a single, very simple deletion query is allowed.

▶ **Theorem 12.** *There is a quantifier-free deletion query ρ with one parameter such that $(q_{Reach}, \Delta_E \cup \{\rho\})$ cannot be maintained in* DynProp.

Proof. For the proof, we combine the standard tool for obtaining inexpressibility results for DynProp, the Substructure Lemma [32, 13], with a combinatorial technique based on upper and lower bounds of Ramsey numbers [29].

The intuition behind the Substructure Lemma is as follows. When updating an auxiliary tuple $\bar{d}$ after a quantifier-free change parameterized by $\bar{p}$, a quantifier-free update formula only has access to $\bar{d}$ and $\bar{p}$. Thus, if a change operation changes a tuple *inside* a substructure $\mathcal{A}$ of a state $\mathcal{S}$ of a dynamic program, the auxiliary data of $\mathcal{A}$ is not affected by any information from *outside* of $\mathcal{A}$. In particular, two isomorphic substructures $\mathcal{A}$ and $\mathcal{B}$ remain isomorphic, when corresponding changes are applied to them. The Substructure Lemma is formally stated in [32, Lemma 2]. Even though the lemma is phrased for single-tuple changes only, the same proof, using the intuition explained above, extends to quantifier-free replacement queries.

For the actual proof, we assume, towards a contradiction, that there is a quantifier-free dynamic program $\mathcal{P}$ over schema τ of arity k that maintains q_{Reach} under the quantifier-free deletion $\rho(p) = \mu(p; x, y) \stackrel{\text{def}}{=} E(x, y) \wedge \neg E(p, x)$ which deletes an edge (x, y) if there is an edge (p, x). Our goal is to construct a graph G such that not all change sequences of length $k + 1$ can be maintained, no matter the initial auxiliary data.

Let n be a sufficiently large number, to be specified later. The vertex set of the graph is of the form $\{s, t\} \cup A \cup C$, for some disjoint sets A and C, with $|C| = n$. The set A contains a node for every subset of size $k + 1$ of C, that is, $A \stackrel{\text{def}}{=} \{a_X \mid X \subseteq C \text{ and } |X| = k + 1\}$. Let B be a subset of A, to be specified later.

The graph has the following edges:

(a) For each $a_X \in A$ there is an edge (s, a_X).
(b) For each $a_X \in B$ there is an edge (a_X, t).
(c) There is an edge (c, a_X) for nodes $c \in C, a_X \in A$ if $c \notin X$.

Intuitively, the nodes in C control how edges from A to t can be removed. Each node $c \in C$ is connected to a subset $A' \subseteq A$, and thus applying a change $\rho(c)$ will result in removing all edges (a_X, t) for all $a_X \in A'$. The graph is constructed in such a way that

(⋆) for a change sequence $\alpha = (\rho(c_1), \ldots, \rho(c_{k+1}))$ with $|\{c_1, \ldots, c_{k+1}\}| = k + 1$ it holds $(s, t) \in q_{\text{Reach}}(\alpha(G))$ if and only if $a_{\{c_1,\ldots,c_{k+1}\}} \in B$.

To see this, observe that after applying changes $\rho(c_1), \ldots, \rho(c_{k+1})$, all edges (a_X, t) are deleted, for which $\{c_1, \ldots, c_{k+1}\} \not\subseteq X$. Thus at most the edge $(a_{\{c_1,\ldots,c_{k+1}\}}, t)$ is still present. However, this edge was at all present in the graph if and only if $a_{\{c_1,\ldots,c_{k+1}\}} \in B$.

For choosing the size of C and the set B, we employ the combinatorial Lemma 2 from [29]. The lemma guarantees that, depending on the schema $\tau \cup \{c_s, c_t\}$, there is an n_0 such that for every $n > n_0$ there is some m such that the following holds.

(S1) For every state $\mathcal{S}$ of the dynamic program for G, and each set C with at least n vertices of G with a linear order $<$, there is a subset C' of C of size at least m such that the k-ary auxiliary data on C' is $<$-monochromatic in the structure $(\mathcal{S}, s, t)$, i.e. all $<$-ordered

k-tuples over C' have the same quantifier-free type (including their relationships to the interpretations s, t of the constants c_s, c_t).

(S2) There is a subset B of A such that for every subset $\hat{C}$ of C of size m, there are $(k+1)$-element sets $Y = \{c_1, \ldots, c_{k+1}\}, Y' = \{c'_1, \ldots, c'_{k+1}\} \subseteq \hat{C}$ with $a_Y \in B$ and $a_{Y'} \notin B$.

We outline how the graph G is used to obtain a contradiction. Let $\mathcal{S}$ be a state of the dynamic program for the graph G with $|C| = n > n_0$ and let $<$ be a linear order. Choose B as described above and a subset C' of C of size $|C'| = m$ that is $<$-monochromatic in $(\mathcal{S}, s, t)$. Choose $Y = \{c_1, \ldots, c_{k+1}\}, Y' = \{c'_1, \ldots, c'_{k+1}\} \subseteq C'$ with $a_Y \in B$ and $a_{Y'} \notin B$. By the Substructure Lemma from [32] generalized to quantifier-free changes, the dynamic program $\mathcal{P}$ yields the same result for the tuple (s, t) for the change sequences $(\rho(c_1), \ldots, \rho(c_{k+1}))$ and $(\rho(c'_1), \ldots, \rho(c'_{k+1}))$ since C' is $<$-monochromatic. Yet the result should be different due to $(\star)$ and $a_Y \in B$, $a_{Y'} \notin B$. This is a contradiction. ◀

Finally, we turn to lower bounds for the maintenance of languages. We exhibit an example that illustrates that maintaining regular languages under full first-order replacement queries might be hard: there is a regular language L that can be maintained in DynFO under single-tuple changes with nullary auxiliary relations, but there is a relatively simple replacement query, for which this no longer holds. This is no general hardness result, as we only allow very restricted auxiliary relations, but it demonstrates the barrier of our techniques. The proof of the following result can be found in the full version of this paper.

▶ **Theorem 13.** *There is a regular language L over some alphabet Σ and a replacement query ρ, such that (q_L, Δ_Σ) can be maintained in* DynFO *with nullary auxiliary relations, but not $(q_L, \Delta_\Sigma \cup \{\rho\})$.*

8 Conclusion

In this paper, we studied the maintainability of queries in the Dynamic Complexity setting under first-order defined replacement queries. The main insight of this study is that many maintainability results carry over from the single-tuple world to settings with more general change operations. We were actually quite surprised to see that so many positive results survive this transition. However, many questions remain open, for instance: To which extent can the reachability query for (undirected or acyclic) graphs be maintained under definable deletions? What about reachability for unrestricted directed graphs under definable insertions? What about other queries? Are binary auxiliary relations sufficient in Theorem 3?

We were less surprised by the fact that stronger change operations can yield inexpressibility, but even these results required some care. Our main contribution in that respect is the proof that DynProp cannot maintain the reachability query under quantifier-free replacement queries.

From Theorem 7 about parameter-free changes and its proof, we take another insight regarding inexpressibility proofs: the squirrel technique is quite powerful to prepare an update program for a non-constant (i.e., logarithmic) number of changes. Inexpressibility proofs need to take that into account and to argue "around it".

References

1 Tom J. Ameloot, Jan Van den Bussche, and Emmanuel Waller. On the expressive power of update primitives. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-*

SIGART Symposium on Principles of Database Systems (PODS), pages 139–150, 2013. doi:10.1145/2463664.2465218.

2 Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP), Proceedings, Part I*, pages 356–367, 2014. doi:10.1007/978-3-662-43948-7_30.

3 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In *Automata, Languages, and Programming - 42nd International Colloquium (ICALP), Proceedings, Part II*, pages 159–170, 2015. doi:10.1007/978-3-662-47666-6_13.

4 Guozhu Dong, Leonid Libkin, and Limsoon Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In *Proceedings of the Fifth International Workshop on Database Programming Languages (DBPL-5)*, page 7, 1995.

5 Guozhu Dong and Chaoyi Pang. Maintaining transitive closure in first order after node-set and edge-set deletions. *Inf. Process. Lett.*, 62(4):193–199, 1997. doi:10.1016/S0020-0190(97)00066-5.

6 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages (DBPL-4)*, pages 295–308, 1993.

7 Guozhu Dong and Jianwen Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Inf. Comput.*, 120(1):101–106, 1995. doi:10.1006/inco.1995.1102.

8 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995. doi:10.1007/BF01530820.

9 Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In *Proceedings of the 4th International Conference on Database Theory (ICDT)*, pages 282–296, 1992. doi:10.1007/3-540-56039-4_48.

10 Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 235–243, 1998. doi:10.1145/275487.275514.

11 Solomon Feferman. Some recent work of Ehrenfeucht and Fraïssé. In *Proc. Summer Institute of Symbolic Logic*, pages 201–209, 1957.

12 Solomon Feferman and Robert L. Vaught. The first order properties of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

13 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.

14 Erich Grädel and Sebastian Siebertz. Dynamic definability. In *15th International Conference on Database Theory (ICDT)*, pages 236–248, 2012. doi:10.1145/2274576.2274601.

15 Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 157–166, 1993. doi:10.1145/170035.170066.

16 William Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.

17 William Hesse and Neil Immerman. Complete problems for dynamic complexity classes. In *17th IEEE Symposium on Logic in Computer Science (LICS), Proceedings*, page 313, 2002. doi:10.1109/LICS.2002.1029839.

18 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

19 Christoph Koch. Incremental query evaluation in a ring of databases. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 87–98, 2010. doi:10.1145/1807085.1807100.

20 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

21 Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004. doi:10.1016/j.apal.2003.11.002.

22 Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.*, 30(3):698–721, 2005. doi:10.1145/1093382.1093384.

23 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.

24 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *CoRR*, abs/1701.02494, 2017. URL: http://arxiv.org/abs/1701.02494.

25 Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. doi:10.1145/2948896.2948899.

26 Sebastian Siebertz. Dynamic definability. Diploma thesis, RWTH Aachen, 2011.

27 Nils Vortmeier. Komplexitätstheorie verlaufsunabhängiger dynamischer Programme. Master's thesis, TU Dortmund, 2013.

28 Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007. doi:10.1007/s00224-006-1312-0.

29 Thomas Zeume. The dynamic descriptive complexity of k-clique. In *Mathematical Foundations of Computer Science (MFCS) - 39th International Symposium, Proceedings, Part I*, pages 547–558, 2014. doi:10.1007/978-3-662-44522-8_46.

30 Thomas Zeume. *Small Dynamic Complexity Classes*. PhD thesis, TU Dortmund University, 2015.

31 Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In *Proc. 17th International Conference on Database Theory (ICDT)*, pages 38–49, 2014. doi:10.5441/002/icdt.2014.08.

32 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.

Constant Delay Enumeration for FO Queries over Databases with Local Bounded Expansion

Luc Segoufin[1] **and Alexandre Vigny**[2]

1 **Inria, France; and**
ENS Cachan, Cachan, France
2 **Université Paris Diderot Paris 7, Paris, France; and**
ENS Cachan, Cachan, France

Abstract

We consider the evaluation of first-order queries over classes of databases with *local bounded expansion*. This class was introduced by Nešetřil and Ossona de Mendez and generalizes many well known classes of databases, such as bounded degree, bounded tree width or bounded expansion. It is known that over classes of databases with local bounded expansion, first-order sentences can be evaluated in pseudo-linear time (pseudo-linear time means that for all ϵ there exists an algorithm working in time $O(n^{1+\epsilon})$). Here, we investigate other scenarios, where queries are not sentences. We show that first-order queries can be enumerated with constant delay after a pseudo-linear preprocessing over any class of databases having locally bounded expansion. We also show that, in this context, counting the number of solutions can be done in pseudo-linear time.

1998 ACM Subject Classification H.2.4 [Database Management] Systems, Query Processing, H.2.3 [Database Management] Languages, Query Languages

Keywords and phrases Enumeration, First-Order queries, Local bounded expansion

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.20

1 Introduction

Query evaluation is a fundamental task in databases and a vast literature is devoted to the complexity of this problem. Given a database $\mathbf{D}$ and a query q the goal is to compute the set $q(\mathbf{D})$ of all solutions for q over $\mathbf{D}$. Unfortunately, the set $q(\mathbf{D})$ might be way bigger than the database itself as the number of solutions could be exponential in the arity of the query. It can therefore be unrealistic to compute all solutions, even for small queries. One could imagine many scenarios to overcome this situation. We could for instance only want to compute the number of solutions or just compute the k most relevant solutions relative to some ranking function.

We consider here the complexity of the enumeration of the set $q(\mathbf{D})$, i.e. generating one by one all the solutions for q over $\mathbf{D}$. In this context two parameters play an important role. The first one is the *preprocessing time*, i.e. the time it takes to produce the first solution. The second one is the *delay*, i.e. the maximum time between the output of any two consecutive solutions. An enumeration algorithm is then said to be *efficient* if these two parameters are small. For the delay, the best we can hope for is constant time: depending only on the query and independent from the size of the database. For the preprocessing time an ideal goal would be linear time: linear in the size of the database with a constant factor depending on the query. When both are achieved we say that the query can be enumerated with constant delay after linear preprocessing.

Constant delay enumeration after linear preprocessing cannot be achieved for all queries. However, for restricted classes of queries and databases several efficient enumeration algorithms have been obtained. This is the case for instance for first-order (FO) queries over databases with bounded degree [3, 11], monadic second-order (MSO) queries over databases with bounded tree-width [2, 13] and FO queries over databases with bounded expansion [12]. Bounded expansion is a large class of databases as it contains in particular all structures excluding at least one minor (planarity, bounded tree-width etc.) and all structures of bounded degree [16].

In some scenarios only pseudo-linear preprocessing time has been achieved. A query can be enumerated with constant delay after a pseudo-linear preprocessing time if for all ϵ there exists an enumeration procedure with constant delay and preprocessing time in $O(\|\mathbf{D}\|^{1+\epsilon})$. This is the case for FO queries over databases with low degree [4].

A special case of enumeration is when the query is boolean. In this case the preprocessing computes the answer to the query. In order to be able to enumerate queries of a given language efficiently, it is therefore necessary to be able to solve the boolean case efficiently.

It has been shown recently that boolean FO queries could be computed in pseudo-linear time over nowhere dense databases [9]. Nowhere dense is an important class of databases generalizing bounded expansion [16]. Amongst classes of databases closed under sub-databases, Nowhere dense is the largest possible class enjoying efficient evaluation for FO queries [14].

It's a major open problem to show that over nowhere dense databases the boolean case can be extended to a constant delay enumeration for FO queries of higher arities.

In this paper we make one step towards solving this problem, extending the bounded expansion result to databases having local bounded expansion. Local bounded expansion lies strictly between bounded expansion and nowhere dense. It requires that for all r the class of neighbors of radius r has bounded expansion. It contains for instance all databases having local bounded tree-width, or excluding locally a minor. It strictly extends bounded expansion as there exist classes of local bounded tree-width that do not have bounded expansion [7].

For FO queries over a class of databases with local bounded expansion we provide:

- an enumeration procedure with constant delay after pseudo-linear preprocessing,
- a pseudo-linear time algorithm counting the number of solutions.

Our proof for enumeration follows a classical scheme. Our first ingredient is Gaifman's theorem, decomposing a formula into local ones with distance constraints. In order to evaluate the local formulas we would need to compute local neighborhoods. However this would not be linear as each neighborhood may be of linear size and we have linearly many of them. Our second key ingredient is the result that one can compute in pseudo-linear time a representative "cover" of the database by means of neighborhoods [9]. Because these neighborhoods have bounded expansion we can use the bounded expansion case in order to evaluate the local formulas. It remains to take care of the distance constraints and this is the main technical contribution of this paper.

The paper is organized as follows. We start by giving a new proof of the boolean case in Section 5. We then extend it to constant delay enumeration in Section 6 and to counting in Section 7.

Related work. Our presentation for the model checking, Section 5, uses the same tricks that were used in [7] to lift the model checking from the bounded tree-width case to the local bounded tree-width case. The model checking results presented in Section 5 were already obtained in [5] with a very similar argument. We give the proofs again here for completeness and in order to fix the notations.

An algorithm for counting in linear time the number of solutions for FO queries over classes of databases with "nice" local bounded tree-width was presented in [6]. The restriction "nice" requires that the neighborhood cover can be computed in linear time and that one part of the cover intersects only a constant number of other parts. It is more restrictive than the one we use, given by [9], and is designed to make the counting easy with a simple exclusion/inclusion argument. This argument does not seem to extend to the cover we have and our algorithm for counting, presented in Section 7, is done by induction on the number of free variables.

In [10] a labeling scheme was presented for first-order queries over graphs with "nice" local bounded tree-width. If constant delay enumeration may be derived from the labeling scheme, this one is computed in polynomial time while we aim for pseudo-linear time. It is unclear whether this result can be generalized to classes of graphs with local bounded expansion using the tools we develop in this paper.

2 Preliminaries

For a positive integer k, $[k]$ denotes the set $\{1, \cdots, k\}$. Thereafter, ϵ will always denote an element of $\mathbb{R}^+$, p, r, s, i, j and k positive integers and f a function of $\mathbb{N} \to \mathbb{N}$.

Databases and First-Order queries. A relational signature σ is a tuple $(R_1, \ldots, R_s)$ where each R_i is a relational symbol of arity r_i. By database, we mean a finite structure over a relational signature σ, that is a tuple $\mathbf{D} = (D, R_1^{\mathbf{D}}, \ldots, R_s^{\mathbf{D}})$, where D, the domain of $\mathbf{D}$, is a finite set and for each i, $R_i^{\mathbf{D}}$ is a subset of D^{r_i}. If $\mathbf{D}$ is a database and $A \subseteq D$ a subset of its universe, we denote by $\mathbf{D}[A]$ the database given by the substructure of $\mathbf{D}$ induced by A. We fix a classical encoding of structures as input, see for example [1]. We denote by $\|\mathbf{D}\|$ the size of (the encoding of) $\mathbf{D}$. Without loss of generality we assume that the domain D comes with a linear order. If not, we arbitrarily choose one, for instance the one induced by the encoding of $\mathbf{D}$. This order induces a lexicographical order among the tuples over D.

A query is a first-order (FO) formula built from atomic formulas, "$x = y$" and $R_i(x_1, \ldots, x_{r_i})$, and closed under boolean combinations, $\wedge, \vee, \neg$, existential and universal quantifications, $\exists, \forall$. We write $q(\overline{x})$ if $\overline{x}$ are the free variables of q. The length of $\overline{x}$ is called the arity of the query. Queries of arity 0 are called sentences. The size of q is written $|q|$.

We write $\mathbf{D} \models q(\bar{a})$ to denote the fact that $\bar{a}$ is a solution for q over $\mathbf{D}$. We write $q(\mathbf{D})$ to denote the set of tuples $\overline{a}$ such that $\mathbf{D} \models q(\overline{a})$.

Given a database $\mathbf{D}$ and a sentence q, the problem of testing whether $\mathbf{D} \models q$ or not is called *the model checking problem*. It may be restricted to a class $\mathcal{C}$ of databases.

Model of computation and complexity. As usual when dealing with linear time, we use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation.

All problems encountered in this paper have two inputs, a database $\mathbf{D}$ and a query q. However they play different roles as $\|\mathbf{D}\|$ is large while $|q|$ is small. We therefore consider the data complexity point of view. We say that a problem is *linear time* if it can be solved in time $O(\|\mathbf{D}\|)$. Here, and in the rest of the paper, the constants hidden behind the "big O" depend on q. We say that a problem is *pseudo-linear time* if, for all ϵ, it can be solved in time $O(\|\mathbf{D}\|^{1+\epsilon})$. In this case the constant factor also depends on ϵ. If a subroutine of a procedure depending on ϵ produces an output of size $O(\|\mathbf{D}\|^{\epsilon})$ we will then say that the output is *pseudo-constant*.

Neighborhoods and bounded expansion. Fix a database $\mathbf{D}$ of domain D. The *Gaifman graph* of $\mathbf{D}$ is the non-directed graph which set of vertices is D and which edges are the pairs $\{a, b\}$ such that a and b occur in a tuple of some relation of $\mathbf{D}$. Given two elements a and b of D, the *distance* between a and b is the length of a shortest path between a and b in the Gaifman graph of $\mathbf{D}$. The notion of distance extends to tuples in the usual way.

Given a positive integer r, the r-neighborhood of a in $\mathbf{D}$ is the substructure of $\mathbf{D}$ induced by the elements of D at distance at most r from a. It is denoted by $N_r^{\mathbf{D}}(a)$. Similarly we define $N_r^{\mathbf{D}}(\bar{a})$ as the union of the r-neighborhoods of the elements of $\bar{a}$.

Given a graph G with a linear order on its vertices, and two of its vertices a, b, we say that b is weakly r-accessible from a if there exists a path of length at most r between a and b such that b is smaller than all vertices of the path.

A class of graphs $\mathcal{C}$ has bounded expansion if for all r, there is a constant N_r, such that for all graphs G of $\mathcal{C}$, there is a linear order on the vertices of G, such that for all vertex a of G, the number of vertices weakly r-accessible from a is bounded by N_r [16]. This is a robust class of graphs with many equivalent definitions [16]. The precise definition will not be important for this paper as we will use this notion via its known algorithmic properties, in particular the fact that constant-delay enumeration algorithms exists for any class of databases with bounded expansion, see Section 3.

It is easy to see that if $\mathcal{C}$ has bounded expansion then the class of all subgraphs of all graphs of $\mathcal{C}$ also has bounded expansion.

A class $\mathcal{C}$ of graphs has *local bounded expansion* if, for any radius r, the class $\mathcal{C}_r$ of all subgraphs of all r-neighborhoods of all graphs in $\mathcal{C}$, has bounded expansion [16].

A class $\mathcal{C}$ of databases has (local) bounded expansion if the class of their Gaifman graphs has the same property.

Normal form for FO queries. We will make use of Gaifman Normal Form and Gaifman Locality Theorem for FO queries. This is rather classical in this context.

For all r there exists FO queries $\mathrm{dist}_r(y, \bar{x})$ expressing the fact that y is at distance at most r from $\bar{x}$. A query $q(\bar{x})$ is said to be r-local if all its quantifications are relative to elements at distance at most r from one of its free variables $\bar{x}$. This can be achieved using quantifications of the form $\exists y\ \mathrm{dist}_r(y, \bar{x}) \wedge \cdots$ and $\forall y\ \mathrm{dist}_r(y, \bar{x}) \rightarrow \cdots$.

It is known as Gaifman Normal Form that for any FO query there is an r such that the query is equivalent to a boolean combination of r-local queries and sentences of the form

$$\exists x_1 \dots x_k \Big(\bigwedge_{1 \leq i \leq j \leq k} \mathrm{dist}(x_i, x_j) > 2r \wedge \bigwedge_{1 \leq i \leq k} \psi(x_i) \Big),$$

where ψ is r-local. A proof can be found for example in [15].

For r-local queries $q(\bar{x})$ it is convenient to refine this normal form in order to know which of the free variables are close together. Any r-local query $q(\overline{x})$ is equivalent to a disjunction of the form:

$$\bigvee_{(\overline{x}_1;\dots;\overline{x}_p) \in P(\overline{x})} \alpha_1(\overline{x}_1) \wedge \dots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1;\dots;\overline{x}_p), \tag{1}$$

where:

- $P(\overline{x})$ is the set of partitions of $\overline{x}$.
- $\alpha_i(\overline{x}_i)$ is r-local.
- $\tau_r(\overline{x}_1;\dots;\overline{x}_p)$ express the fact that the distance between $\overline{x}_i$ and $\overline{x}_j$ is bigger than $2r$ and that no refinement of P has this property. We will sometime refer to τ_r as a *distance type*.

Note that this implies that each α_i is $(r|\overline{x}_i|)$-local around any of its free variables, hence in particular the first one. Notice also that in (1) the disjunction is strict: no two outputs can satisfy two disjuncts. We will use this fact later to reduce our attention to a single disjunct.

Counting and enumeration. The *counting problem* is, given a database $\mathbf{D}$ and a query q, to compute the number of solutions to q over $\mathbf{D}$, i.e. the size of $q(\mathbf{D})$, noted $\#q(\mathbf{D})$.

We will now focus on the *enumeration problem*. An enumeration algorithm for a database $\mathbf{D}$ and a query q is divided into two consecutive phases:

- a preprocessing phase,
- an enumeration phase, outputting one by one and without repetitions the set $q(\mathbf{D})$.

The *preprocessing time* of the enumeration algorithm is the time taken by the preprocessing phase. Its *delay* is the maximum time between any two consecutive outputs.

One can view an enumeration algorithm as a compression algorithm computing a representation of $q(\mathbf{D})$ together with a streaming decompression algorithm. We aim for constant delay and pseudo-linear preprocessing time enumeration algorithms. By this we mean that for all ϵ, there is a preprocessing phase working in time $O(\|D\|^{1+\epsilon})$ and an enumeration phase with constant delay. Note that the multiplicative constants, for both the preprocessing phase and the delay, may depend on q and on ϵ.

All our enumeration procedures will output their tuples in lexicographical order. We will see that this is useful for queries in disjunctive normal form.

For the sake of readability, in the reminder of the paper, we only consider classes of graphs. All results and proofs can be easily adapted to the database case using standard techniques.

3 Main results

We will build on several known results over classes of databases with bounded expansion. The first is a linear time model checking algorithm for sentences:

▶ **Theorem 1** (Dvorak-Kral-Thomas [5]). *Let $\mathcal{C}$ be a class of graphs with bounded expansion. Then the model checking problem for FO queries over $\mathcal{C}$ can be solved is linear time.*

The second one solves the unary query case:

▶ **Theorem 2** (Dvorak-Kral-Thomas [5]). *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $q(x)$ be a query with one free variable. We can compute the set $q(G)$ in linear time.*

For queries with bigger arities, we cannot hope to evaluate their output in linear time anymore. A constant delay enumeration algorithm after linear preprocessing time has been obtained by Kazana and Segoufin in [12]. We present their result using a stronger statement than enumeration that will be useful for us later. Here $\geq$ is the lexicographical order on tuples over the domain. Recall that the constant factor depends on the query.

▶ **Theorem 3** (Kazana-Segoufin [12]). *Let $\mathcal{C}$ be a class of graphs with bounded expansion. Then there is an algorithm such that for all graph G in $\mathcal{C}$, and for any FO query q, after a preprocessing in linear time, on input any tuple $\overline{a}$, the algorithm computes in constant time the minimal tuple $\overline{b}$ such that:*

- $\overline{b} \geq \overline{a}$
- $G \models q(\overline{b})$

If there is no such tuple (i.e. $\overline{a}$ is bigger than all solutions), it outputs Null.

Our first result extends Theorem 3 to classes with local bounded expansion, replacing linear preprocessing time with pseudo-linear preprocessing time:

▶ **Theorem 4.** *Let $\mathcal{C}$ be a class of graphs with local bounded expansion. Then there is an algorithm such that for all graph G in $\mathcal{C}$, and for any FO query q, after a preprocessing in pseudo-linear time, on input any tuple $\overline{a}$, the algorithm computes in constant time the minimal tuple $\overline{b}$ such that:*

- $\overline{b} \geq \overline{a}$
- $G \models q(\overline{b})$

If there is no such tuple (i.e. $\overline{a}$ is bigger than all solutions), it outputs Null.

It immediately yields the constant delay enumeration after pseudo-linear preprocessing time.

▶ **Corollary 5.** *The enumeration of first-order query over class of graphs with local bounded expansion can be done with constant delay, after pseudo-linear preprocessing. Moreover the output tuples are given in lexicographiacal order.*

Our second result shows that counting the number of solutions can be done in pseudo-linear time.

▶ **Theorem 6.** *Let $\mathcal{C}$ be a class of graphs with local bounded expansion and $q(\overline{x})$ be a first-order query. Then for all graph G in $\mathcal{C}$, we can compute $\#q(G)$ in pseudo-linear time.*

Our proof works by induction on the arity of the query. It uses a partition of the database into representative neighborhoods that we describe next. It then combines this partition with the bounded expansion case.

4 Neighborhood covers and partitions

Because of the definition of local bounded expansion it is natural to examine the neighborhoods of our graphs. However, the sum of the sizes of all neighborhoods could be quadratic in the size of the input, which is too big as we aim for pseudo-linear time algorithms. To overcome this we selects some representative neighborhoods that cover the entire graph. The result presented here actually works for the more general notion of nowhere dense[1] graphs and is based on [9].

A (r, s)-*neighborhood cover* of a graph G is a set T of bags $U_1, \ldots, U_\omega$ such that:

- $\forall a \in G,\ \exists \lambda \leq \omega \quad \mathcal{N}_r^G(a) \subseteq U_\lambda$
- $\forall \lambda \leq \omega,\ \exists a \in G \quad U_\lambda \subseteq \mathcal{N}_s^G(a)$

The *size* of the cover T is the sum of the bag sizes: $\|T\| = \sum_{\lambda \leq \omega} \|U_\lambda\|$. Its *degree* is the number $\delta(T) := \max_{a \in G} |\{\lambda \leq \omega \mid a \in U_\lambda\}|$.

▶ **Theorem 7** (Grohe et al. [9])**.** *Let $\mathcal{C}$ be a nowhere-dense class of graphs. Then for all integer s and for all graph G in $\mathcal{C}$, we can compute in pseudo-linear time a $(s, 2s)$-neighborhood cover of G with a pseudo-constant degree. In particular the size of the neighborhood cover is pseudo-linear.*

[1] Nowhere dense requires that for all r, the number of weakly r-accessible nodes is pseudo-constant, instead of constant for bounded expansion.

Let A be a set of vertices of G. The *s-kernel* of A is the set $K_s(A) := \{a \in A \mid N_s^G(a) \subseteq A\}$.

We deduce from a $(s, 2s)$-neighborhood cover of G a partition of the vertices of G as follows:

$$P_\lambda := K_s^G(U_\lambda) \setminus \bigcup_{\mu<\lambda} K_s^G(U_\mu).$$

It follows from the definitions that the P_λ form a partition of the vertices of G. Moreover, modulo an extra linear preprocessing time, given an element a we have access in constant time to the unique λ such that $a \in P_\lambda$. This is a consequence of the following simple lemma.

▶ **Lemma 8.** *For all graph G, for all set A of vertices of G, and for all integer s, $K_s^G(A)$ is computable in time $O(s \cdot \|A\|)$.*

Proof. We prove the lemma by induction on s.

- If $s = 1$, let L be a list initialized empty. Then for each element a of A, we go through every neighbor of a. If we find one that is not in A, we add a to L and we go to the following element of A. At the end, we have $K_1^G(A) = A \setminus L$.
- If $s = i + 1$, from $K_{i+1}^G(A) = K_1^G(K_i^G(A))$ we get $K_s^G(A)$ is in time $O(s \cdot \|A\|)$. ◀

In the following sections, we will often say: compute $T = \{(U_1, P_1), \ldots, (U_\omega, P_\omega)\}$ that is the $(s, 2s)$-neighborhood cover paired with the s-kernel partition.

The previous observations can be synthesized in the following corollary.

▶ **Corollary 9.** *Let $\mathcal{C}$ be a class of graphs with local bounded expansion. Then for all graph G in $\mathcal{C}$ and for all integer s, we can compute in pseudo-linear time a $(s, 2s)$-neighborhood cover with pseudo-constant degree and the associated s-kernel partition.*

If a $(s, 2s)$-neighborhood cover can be computed efficiently on any nowhere dense class of graphs, a key property of the covers that works only for the local bounded expansion case is that all U_λ are in a class of graphs with bounded expansion. This is because each U_λ is included in the $2s$-neighborhood of some point and the later has bounded expansion by definition. We can therefore enumerate any FO query on each U_λ using Theorem 3 in time $O(\|U_\lambda\|)$, for a total time $O(\sum_{\lambda \leq \omega} \|U_\lambda\|)$, that is pseudo-linear. We will use this property implicitly in the rest of the paper. Note that this does not solve the general case as some solutions may have parts in different U_λ.

5 Model-Checking

Since every class of graphs with local bounded expansion is nowhere dense, we already know that the model checking problem of first-order queries over graphs with local bounded expansion can be done in pseudo-linear time [9]. Before that, another proof specific to local bounded expansion was given in [5]. In order to illustrate the tools presented in the previous sections, we give a new proof of this result. As in [5], it is based on the ideas of Grohe and Frick for graphs with local bounded tree-width [7].

▶ **Theorem 10.** *Let $\mathcal{C}$ be a class of graphs with local bounded expansion. Given a graph G in $\mathcal{C}$ and a FO sentence q, we can decide in pseudo-linear time whether $G \models q$.*

The rest of this section is devoted to the proof of Theorem 10. Fix G in $\mathcal{C}$ and a FO sentence q. In view of Gaifman Normal Form, we can assume wlog, that q is of the form:

$$q := \exists x_1 \ldots x_k \Big(\bigwedge_{1 \leq i \leq j \leq k} \mathrm{dist}(x_i, x_j) > 2r \wedge \bigwedge_{1 \leq i \leq k} \psi(x_i) \Big),$$

where ψ is r-local for some r.

Our strategy is as follows: we will first compute the set of nodes satisfying ψ and then test whether k of them are far apart from each other. The next lemma takes care of the first step.

▶ **Lemma 11.** *Let $\mathcal{C}$ be a class of graphs with local bounded expansion. For all graph G in $\mathcal{C}$, for all integer r, and for all unary and r-local FO query ψ, we can compute $\psi(G)$ in pseudo-linear time.*

Proof. Recall that $\mathcal{C}_s$ denotes the class of all subgraphs of s-neighborhoods of graphs from $\mathcal{C}$. Fix r, ψ a unary and r-local query and $G \in \mathcal{C}$.

We first compute $T = \{(U_1, P_1), \dots, (U_\omega, P_\omega)\}$, a $(2r, 4r)$-neighborhood cover paired with the $2r$-kernel partition. This can be done in pseudo linear time by Corollary 9. We can then view the P_λ as new unary predicates.

$\forall \lambda \leq \omega$, we set $\psi_\lambda(x) := \psi(x) \wedge P_\lambda(x)$. Because ψ is r-local, $\psi(G)$ is the disjoint union of all $\psi_\lambda(U_\lambda)$. By definition, $U_\lambda \in \mathcal{C}_{4r}$ which has bounded expansion. Consequently, it is possible to compute $\psi_\lambda(U_\lambda)$ in time $O(\|U_\lambda\|)$ by Theorem 2. Therefore, we are able to compute the set $\psi(G)$ in time $O\left(\sum_{\lambda=1}^{\omega} \|U_\lambda\|\right) = O(\|T\|)$ that is pseudo-linear in the size of G. ◀

Now we want to find k elements far apart in $\psi(G)$. We use a trick found in [7].

▶ **Lemma 12.** *Let $\mathcal{C}$ be a class of graphs with local bounded expansion. For all graph G in $\mathcal{C}$, for all integer r and k, and for all set A of vertices of G, we can decide in pseudo-linear time whether A contains a subset of k elements that are pairwise at distance more than $2r$.*

Proof. We proceed as follows:

We first compute $T = \{(U_1, P_1), \dots, (U_\omega, P_\omega)\}$, a $(2r, 4r)$-neighborhood cover paired with the $2r$-kernel partition as in Corollary 9.

Let L be a list, initialized as empty.

While A is not empty and $|L| < k$, we select (and remove) an element a in A.

If for all b in L we have: $(b \in P_\lambda \Rightarrow a \notin U_\lambda)$ then we add a in L. Notice that every b belongs to some P_λ, and hence $N_{2r}(b) \subseteq U_\lambda$. If furthermore $a \notin U_\lambda$ then a and b must be at distance more than $2r$.

At the end, we have tree different cases:

- 1^{st} case, $|L| = 0$. Then $A = \emptyset$.
- 2^{nd} case, $|L| = k$. Then we are done because all elements of L are far apart from each other by construction.
- 3^{rd} case, $|L| = m$, with $0 < m < k$. Let $L = \{b_1, \cdots, b_m\}$. Notice that $A \subseteq N^G_{4r}(b_1, \dots, b_m)$. We can see that $H := N^G_{4r}(b_1, \dots, b_m)$ is in $\mathcal{C}_{4rm}$. Therefore, from Theorem 1 it is possible to check in linear time if:

$$H \models \exists\, x_1, \dots, x_k \bigwedge_{1 \leq i \leq k} A(x_i) \;\wedge \bigwedge_{1 \leq i < j \leq k} \mathrm{dist}(x_i, x_j) > 2r.$$ ◀

Theorem 10 now easily follows from Lemma 11 and Lemma 12.

6 Enumeration

In this section we provide a constant delay enumeration procedure for FO queries over graphs with local bounded expansion. We actually prove a stronger result as stated in Theorem 4. Let G be a graph, q a FO query and $\overline{a}$ any tuple from G, not necessarily in $q(G)$. We denote

by $\text{FOLW}_q(\overline{a})$ the smallest tuple $\overline{b} \in q(G)$ such that $\overline{b}$ is bigger than $\overline{a}$ in the lexicographical order i.e. $\overline{b} \geq \overline{a}$. If there is no such $\overline{b}$, we say that $\text{FOLW}_q(\overline{a}) = \mathsf{Null}$.

In the rest of this section we show that for any class $\mathcal{C}$ with local bounded expansion, given $G \in \mathcal{C}$ and q, after a pseudo-linear time preprocessing, we can compute $\text{FOLW}_q(\overline{a})$ in constant time. Recall that this means that given ϵ there is a preprocessing algorithm working in time $O(\|G\|^{1+\epsilon})$ computing a structure that can then be used to compute $\text{FOLW}_q(\overline{a})$ from $\overline{a}$ in constant time. All constants depend on q and ϵ. This proves Theorem 4. We proceed by induction on the arity of the query.

▶ Remark 1. Assume q is $q_1 \vee q_2$. Then $\text{FOLW}_q(\overline{a})$ is the smallest tuple among $\text{FOLW}_{q_1}(\overline{a})$ and $\text{FOLW}_{q_2}(\overline{a})$. Hence if q is a disjunction of queries, it is enough to prove Theorem 4 on each of the disjunct to get the result for q.

Thanks to Gaifman Normal Form, we can assume that the query is a boolean combination of r-local formulas and sentences. By Theorem 10 the sentences can be precomputed during the preprocessing phase. We are then left with a r-local query (any boolean combination of r-local queries is a r-local query). Moreover, in view of Remark 1 and Gaifman Theorem for local queries, we can assume without loss of generality that our query q has the form:

$$q(\overline{x}) = \alpha_1(\overline{x}_1) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1; \ldots; \overline{x}_p)$$

where the α_i and τ_r satisfy the conditions described in Section 2.

We start with some examples in order to illustrate the difficulty of the task. Assume that the query returns the pairs of blue-red nodes that are sufficiently far apart:

$$q(x, y) := \text{dist}(x, y) > 2r \wedge B(x) \wedge R(y).$$

Given a blue node a and a node b we can compute $\text{FOLW}_q(a, b)$ as follows.

During the preprocessing phase, thanks to Corollary 9, we compute in pseudo-linear time a $(2r, 4r)$-cover with its associated partition $\{(U_1, P_1), \ldots, (U_\omega, P_\omega)\}$. Given a the λ such that $a \in P_\lambda$ can then be obtained in constant time. Assume we could compute in pseudo-linear time a structure that, given λ and b, returns in constant time the smallest red node $c \geq b$ outside of U_λ. With this, from $a \in P_\lambda$ we have that $q(a, c)$ holds and therefore $\text{FOLW}_q(a, b) \leq c$. It remains to test whether there is also a node $c' \geq b$ within U_λ that is far from a. As we are within U_λ, we can invoke Theorem 3 and compute the smallest such c' in constant time after a preprocessing linear in $\|U_\lambda\|$. As we don't know a, and therefore λ, in advance, we perform that preprocessing for all λ, for a total time linear in $O(\Sigma_\lambda \|U_\lambda\|)$, which is pseudo linear. The minimum element among c and c' is then the desired $\text{FOLW}_q(a, b)$. Unfortunately we cannot afford to construct the structure returning c from λ and b because this is a function with two parameters that can potentially have a quadratic size. We will see in the proof (this is essentially Claim 13 and Claim 14 bellow) that we can compute in pseudo-linear time a subset of this function that is sufficient for our needs.

The situation is even worse for higher arities. To see this, assume the query is now

$$q(x_1, x_2, x_3) := B(x_1) \wedge Y(x_2) \wedge R(x_3) \wedge \bigwedge_{1 \leq i < j \leq 3} \text{dist}(x_i, x_j) > 2r.$$

Given a blue node a, a yellow node a' that are both far apart, and a node b, we are looking for $\text{FOLW}_q(aa', b)$.

Let's see what happens when extending the previous reasoning. Given a and a' we derive in constant time λ and λ' such that $a \in P_\lambda$ and $a' \in P_{\lambda'}$. As above we could get in constant time the smallest red node $c \geq b$ outside of U_λ. But if this node is certainly far from a it

might be close to a'. We can then imagine precomputing the smallest red node $c \geq b$ outside of $U_\lambda \cup U_{\lambda'}$, a ternary function. Again, we will see that we can compute in pseudo-linear time a subset of this function good enough for us. It remains to test whether there is also a node $c' \geq b$ within $U_\lambda \cup U_{\lambda'}$ that is far from a and a'. We could use again Theorem 3, but we would need a preprocessing linear in $\Sigma_{\lambda,\lambda'}(\|U_\lambda \cup U_{\lambda'}\|)$, which is unfortunately quadratic. To overcome this problem we introduce an intermediate bag V_λ between P_λ and U_λ together with a more complex algorithm based on the positions of a and a' in all the bags we have, that we will describe below.

We now turn to the formal details.

Base case. Assume q is unary. Because q is also r-local, by Lemma 11 we can compute $q(G)$ during the preprocessing phase. In order to compute $\mathrm{FOLW}_q(a)$ for all $a \in G$, we go through all vertices of G starting from the maximal one. For each vertex a, if $a \in q(G)$ then we set $\mathrm{FOLW}_q(a) = a$. If $a \notin q(G)$ and a is the maximal element, we set $\mathrm{FOLW}_q(a)$ to Null. In all remaining cases, we set $\mathrm{FOLW}_a(a)$ to $\mathrm{FOLW}_q(b)$, where b is the successor of a in the linear order on the vertices.

Inductive case. Assume now that $q(\bar{x}, y)$ is an r-local query of arity $k+1$. Let $q'(\bar{x})$ be the query $\exists y\ q(\bar{x}, y)$.

We claim that, modulo a pseudo-linear preprocessing, given a tuple $\bar{a}$ such that $G \models q'(\bar{a})$ and a vertex b, we can compute in constant time the smallest $b' \geq b$ such $G \models q(\overline{a}, b)$, outputting Null if no such b' exists.

Before proving the claim we show that it implies the constant time computation of FOLW_q. Let a_{min} be the minimal element of G. Let $\overline{a}b$ be a tuple. Let $\overline{a}' = \mathrm{FOLW}_{q'}(\overline{a})$. By induction, $\overline{a}'$ can be computed in constant-time. If $\overline{a}'$ is Null we output Null and we are done. If $\overline{a}' > \overline{a}$ then we apply the claim with $\overline{a}'$ and a_{min} and we are done. If $\overline{a}' = \overline{a}$ then we apply the claim with $\overline{a}$ and b. If b' is not Null, then $\overline{a}b'$ is the desired tuple. If $b' =$ Null then let $\overline{a}'$ be the successor of $\overline{a}$ in the lexicographical order and $\overline{a}'' = \mathrm{FOLW}_{q'}(\overline{a}')$. We apply again the claim for $\overline{a}''$ and a_{min} and we are done. All this clearly takes constant time.

In the rest of this section we prove the claim. Recall that $q(\bar{x}, y)$ is of the form:

$$q(\overline{x}, y) = \alpha_1(\overline{x}_1, y) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1, y; \ldots; \overline{x}_p).$$

Let $\bar{w} = \overline{x}_2 \cup \ldots \cup \overline{x}_p$. We have:

$$q(\overline{x}, y) = q_1(\overline{x}_1, y) \wedge q_2(\bar{w}) \wedge \tau_r(\overline{x}_1, y; \ldots; \overline{x}_p).$$

We will distinguish two cases, depending whether $\overline{x}_1$ is empty or not. Let k be the arity of q.

Elements far away

We assume here that $\overline{x}_1$ is empty. By Lemma 11 we can precompute in pseudo-linear time the set L of nodes satisfying q_1. It remains to compute from $\overline{a}$, and b the smallest element of L that is at distance $2r$ from $\overline{a}$ and is greater than b.

We compute a $(4r, 8r)$-neighborhood cover and the associated $4r$-kernel partition according to Corollary 9. We then compute the $2r$-neighborhood V_λ of each P_λ. We now have $T := \{(P_1, V_1, U_1), \ldots, (P_\omega, V_\omega, U_\omega)\}$ such that $N^G_{2r}(P_\lambda) = V_\lambda$ and $N^G_{2r}(V_\lambda) \subseteq U_\lambda$.

We define for all vertex b and all set $I \subseteq \{1, \ldots, \omega\}$ such that $|I| \leq k$ the function.

$$\mathrm{NEXT}(b, I) = \min \left\{ b' \mid b' \geq b \wedge b' \notin \bigcup_{\lambda \in I} V_\lambda \wedge b' \in L \right\}$$

The domain of this function is too big (recall that ω is linear in $\|G\|$) so we cannot compute it. Fortunately, computing only a small part of it will be good enough for our needs. For each vertex b we define by induction the following set $SC(b)$ of elements $I \subseteq \{1, \dots, \omega\}$ with $|I| \leq k$:

- For all b in G and for all λ with $b \in V_\lambda$, we add $\{\lambda\}$ to $SC(b)$.
- For all b in G, for all I, and for all λ, if $|I| < k$ and $I \in SC(b)$ and $\text{NEXT}(b, I) \in V_\lambda$, then we add $\{I \cup \{\lambda\}\}$ to $SC(b)$.

Our aim is to compute all $\text{NEXT}(b, I)$ for all b and $I \in SC(b)$. We first show that it will be enough to compute in constant time $\text{NEXT}(b, I)$ for all b and I.

▶ **Claim 13.** *Given a vertex b, a set I, and $NEXT(c, J)$ for all vertices $c > b$ and sets $J \in SC(c)$, then we can compute $NEXT(b, I)$ in constant time.*

Proof.

- Case 1, $b \in L$ and $b \notin \bigcup_{\lambda \in I} V_\lambda$, then b is $\text{NEXT}(b, I)$.
- Case 2, $b \notin L$ or $b \in \bigcup_{\lambda \in I} V_\lambda$, then let c be the smallest element of L strictly bigger than b. If there is no such c then $\text{NEXT}(b, I) = \mathsf{Null}$, otherwise:
  - Case 2.1, $c \notin \bigcup_{\lambda \in I} V_\lambda$, then c is $\text{NEXT}(b, I)$.
  - Case 2.2, $c \in V_\lambda$ with $\lambda \in I$. Therefore $\{\lambda\} \in SC(c)$. Let J be a maximal (for inclusion) subset of I in $SC(c)$. Since $\{\lambda\} \in SC(c)$, we know that J is non empty. We claim that $\text{NEXT}(c, J) = \text{NEXT}(b, I)$. To see this, assume that $\text{NEXT}(c, J) \in V_\mu$ with $\mu \in I$ hence $|J| < k$, then by definition of $SC(c)$, $J \cup \{\mu\} \in SC(c)$ and J was not maximal. Moreover, by definition of $\text{NEXT}(c, J)$, every point between c and $\text{NEXT}(c, J)$ is either not in L or in one of the V we want to avoid. As all nodes between b and c are not in L, the claim follows. ◀

We now show that $SC(b)$ is small for all b and that we can compute all of $\text{NEXT}(b, I)$ for all b and $I \in SC(b)$.

▶ **Claim 14.** *For all integer b, $|SC(b)|$ is a pseudo-constant. Moreover, it is possible to compute all $NEXT(b, I)$ for all vertex b and set $I \in SC(b)$ in pseudo-linear time.*

Proof. We first prove that for all $b \in G$, $|SC(b)|$ is a pseudo-constant. Then we use Claim 13 in order to prove that we can compute these pointers by induction.

- By $SC_l(b)$ we denote the subset of $SC(b)$ of elements I with $|I| \leq l$. Let d be the degree of our cover. We have that for all $b \in G$, $|SC_1(b)| = d$. For the same reason, we have that $|SC_{l+1}(b)| = O(d \cdot |SC_l(b)|)$. Therefore, we have that for all $b \in G$, $|SC(b)| = |SC_k(b)| \leq O(d^k)$. Since d is pseudo-constant, $|SC(b)|$ is also pseudo-constant.
- We compute the pointers for b from b_{max} to b_{min} downwards, respectively the biggest and the smallest element of G. Given a b in G, assume we have computed $\text{NEXT}(c, J)$ for all $c > b$ and $J \in SC(c)$. We then compute $\text{NEXT}(b, I)$ for $I \in SC(b)$ using Claim 13. Here, every pointer was computed in constant time. Since there is only a pseudo-linear number of them, the time required to compute them all is pseudo-linear. ◀

With these two claims, we are now ready to conclude the case where $\bar{x}_1$ is empty. The preprocessing phase consists of the following steps:

- 1^{st} step: compute a $(4r, 8r)$-neighborhood cover and the associated $4r$-kernel partition according to Corollary 9.
- 2^{nd} step: compute the $2r$-neighborhood V_λ of each P_λ. We now have $T := \{(P_1, V_1, U_1), \dots, (P_\omega, V_\omega, U_\omega)\}$ such that $N^G_{2r}(P_\lambda) = V_\lambda$ and $N^G_{2r}(V_\lambda) \subseteq U_\lambda$.

- 3^{rd} step: compute $L := q_2(G)$. This can be done in pseudo-linear time by Lemma 11.
- 4^{th} step: compute $\text{NEXT}(b, I)$ for all b and $I \in SC(b)$. This can be done in pseudo-linear time by Claim 14.
- 5^{th} step: $\forall 1 \leq l \leq k,\ \forall \lambda \leq \omega$, perform on U_λ the preprocessing phase for the formula:

$$\varphi_{\lambda,l}(x_1, \ldots, x_l, y) := V_\lambda(y) \wedge L(y) \wedge \bigwedge_{i \leq l} \text{dist}(x_i, y) > 2r.$$

This can be done in time $O(\|U_\lambda\|)$ by Theorem 3 because $U_\lambda \in \mathcal{C}_{8r}$.
This is the end of the preprocessing. The total time needed is pseudo-linear.

Now we are given $(\overline{a}, a_{k+1})$ a tuple of elements of G, such that $G \models \exists y\ q(\overline{a}, y)$.

- Let $\lambda_1, \ldots, \lambda_k$ be such that $a_i \in P_{\lambda_i}$ and $I := \{\lambda_1, \ldots, \lambda_k\}$.
- Let $b_0 = \text{NEXT}(a_{k+1}, I)$.
- For all $1 \leq i \leq k$, let $\bar{c}_i$ be the elements of $\overline{a}$ that falls into U_{λ_i} and $m_i = |\bar{c}_i|$. Using Theorem 3 we compute in constant-time

$$b_i = \min\{y \mid y \in V_{\lambda_i} \wedge y \geq a_{k+1} \ \wedge\ \varphi_{\lambda_i, m_i}(\bar{c}_i, y)\}\,.$$

- We return the minimum element among the b_i, $0 \leq i \leq k$.

This is clearly constant time. To see that this is correct, let c be the correct answer.

If $c \in V_{\lambda_i}$ for some $i \in I$ then φ_{λ_i, m_i} gives us b_i that is the smallest $y \in V_{\lambda_i}$ satisfying q_2 and is at distance greater than $2r$ from the elements of $\overline{a}$ present in U_λ. Those that are not in V_{λ_i} must be at distance greater than $2r$ from b_i since $N^G_{2r}(V_{\lambda_i}) \subseteq U_{\lambda_i}$. Hence $c = b_i$.

Otherwise, i.e. $c \notin V_{\lambda_i}$ for all $i \in I$. Then the $\text{NEXT}(\,\cdot\,, I)$ pointers give us b_0 that is the smallest element satisfying q_2 that is not in one of the V_{λ_i}. Therefore $\text{dist}(b_0, \overline{a}) > 2r$. Hence $c = b_0$.

This concludes the first case.

Elements nearby

Assume now that $\overline{x}_1$ contains at least one variable, say x_1. Therefore, for all tuples $(\overline{a}, b)$ such that $G \models q(\overline{a}, b)$, we have that $\text{dist}(a_1, b) < 2kr$. This makes the second case much easier.

The preprocessing phase contains several steps.

- 1^{st} step: compute a $(4rk, 8rk)$-neighborhood cover and the associated $4rk$-kernel partition according to Corollary 9.
- 2^{nd} step: compute the $2rk$ neighborhood V_λ of each P_λ. We now have $T := \{(P_1, V_1, U_1), \ldots, (P_\omega, V_\omega, U_\omega)\}$ such that $N^G_{2rk}(P_\lambda) = V_\lambda$ and $N^G_{2rk}(V_\lambda) \subseteq U_\lambda$.
- 3^{rd} step: $\forall 1 \leq l < k,\ \forall \lambda \leq \omega$, perform the preprocessing phase on U_λ of the formula:

$$\varphi_{\lambda,l}(\bar{x}_1, y, x'_1, \ldots, x'_l) := V_\lambda(y) \ \wedge\ q_1(\bar{x}_1, y) \ \wedge \bigwedge_{i<l} \text{dist}(x'_i, y) > 2r.$$

This can be done in time $O(\|U_\lambda\|)$ by Theorem 3 because $U_\lambda \in \mathcal{C}_{8kr}$.
This is the end of the preprocessing. The total time needed is pseudo-linear.

Now we are given $(\overline{a}, a_{k+1})$ a tuple of elements of G, such that $G \models \exists y\ q(\overline{a}, y)$. Let $\overline{a}_1$ be the elements of $\overline{a}$ corresponding to $\bar{x}_1$ and a_1 the first of them.

- Let λ be such that $a_1 \in P_\lambda$.
- Let $\bar{c} := (a_j \in \bar{a} \mid a_j \in U_\lambda \wedge a_j \notin \overline{a}_1)$ and $m = |\bar{c}|$

- We return $b := \min\{b' \mid b' \in U_\lambda \wedge b' \geq a_{k+1} \ \wedge \ U_\lambda \models \varphi_{\lambda,m}(\bar{a}_1, b', \bar{c})\}$. This can be done in constant time by Theorem 3.

All this is done in constant time. It is correct because by assumption the answer b must be in V_λ, satisfy $q_1(\bar{a}_1, b)$ and be at distance greater than $2r$ from the elements of $\bar{w}$.

The elements of $\bar{w}$ that are not in U_λ must satisfy this last condition since $b \in V_\lambda$ and $N^G_{2rk}(V_\lambda)$
$\subseteq U_\lambda$. It remains to consider the elements of $\bar{w}$ that are in U_λ, that is $\bar{c}$. Then $\varphi_{\lambda,m}(\overline{a}_1, y, \bar{c})$ gives us exactly what we want.

This concludes the second case and therefore the proof.

Besides constant delay enumeration, Theorem 4 has another interesting immediate corollary. Modulo a pseudo-linear time preprocessing we can test, given a tuple $\overline{a}$ in constant time, whether it belong to $q(G)$ or not:

▶ **Corollary 15.** *Let $\mathcal{C}$ be a class of databases with local bounded expansion. Then for all graph G in $\mathcal{C}$, after a pseudo-linear time preprocessing, we can, given a tuple $\overline{a}$, decide in constant time whether it belongs to $q(G)$ or not.*

7 Counting

In this section we consider the counting problem which is to compute, given G and q, the size of $q(G)$, denoted by $\#q(G)$. We aim at computing $\#q(G)$ in time pseudo-linear.

▶ Remark 2. Assume q is $q_1 \vee q_2$ and that q_1 and q_2 have no common solution, i.e. the disjunction is strict. Then $\#q(G) = \#q_1(G) + \#q_2(G)$. Hence if q is a strict disjunction of queries, it is enough to prove Theorem 6 on each of the disjunct to get the result for q.

Again we will build on the bounded expansion case:

▶ **Theorem 16** (Kazana, Segoufin [12])**.** *Let $\mathcal{C}$ be a class of graphs with bounded expansion and $q(\overline{x})$ be a FO query. Then, for all graph G in $\mathcal{C}$, we can compute $\#q(G)$ in linear time.*

The rest of the section is dedicated to the proof of Theorem 6.

Thanks to Gaifman Normal Form, we can assume that the query is a boolean combination of r-local formulas and sentences. By Theorem 10 the sentences can be precomputed during the preprocessing phase. We are then left with a r-local query (any boolean combination of r-local queries is a r-local query). Moreover, in view of Remark 2 and Gaifman Theorem for local queries, we can assume without loss of generality that our query q has the form:

$$q(\overline{x}) = \alpha_1(\overline{x}_1) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1; \ldots; \overline{x}_p)$$

where the α_i and τ_r satisfies the conditions described in Section 2.

The proof goes by induction on p, which is the number of connected components of the distance -type τ.

We first give a small example in order to give a hint of how the induction works.

Consider again the query returning the pairs of blue-red nodes that are far apart:

$$q(x, y) := \text{dist}(x, y) > 2r \wedge B(x) \wedge R(y).$$

In this case, there are two connected components. In order to count the number of solutions, we multiply the number of blue nodes by the number of red nodes and we subtract

from the result the number of blue-red nodes that are at distance smaller than $2r$. Those three numbers correspond to the number of solutions of three queries with only one connected component in their distance type each, hence we can proceed by induction. This is essentially what we do in the general case.

We now give the details. We start with the base case followed by the inductive case.

Let $G \in \mathcal{C}$, and $q(\overline{x}) = \alpha_1(\overline{x}_1) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1; \ldots; \overline{x}_p)$

If $p = 1$. In this case, if $\overline{a} \in q(G)$, then $N_r^G(\overline{a}) \subseteq N_{2rk}^G(a_1)$.

- 1^{st} step: compute a $(2rk, 4rk)$-neighborhood cover and the associated $2rk$-kernel partition according to Corollary 9. We now have:
  $T := \{(P_1, U_1), \ldots, (P_\omega, U_\omega)\}$ such that $N_{2rk}^G(P_\lambda) \subseteq U_\lambda$.
- 2^{nd} step: for all $\lambda \leq \omega$, let $\varphi_\lambda(\overline{x}) := q(\overline{x}) \wedge x_1 \in P_\lambda$.
  We have that $q(G) = \bigcup_{1 \leq i \leq \omega} \varphi_\lambda(U_\lambda)$. Moreover, the union is disjoint. Therefore:

$$\#q(G) = \sum_{i=1}^{\omega} \#\varphi_\lambda(U_\lambda)\,.$$

  Since for all λ, $U_\lambda \in \mathcal{C}_{4kr}$, we can compute $\#\varphi_\lambda(U_\lambda)$ in time $\|U_\lambda\|$ using Theorem 16. Therefore, we can compute $\#q(G)$ in total time $O(\sum_{i=1}^{\omega}(\|U_\lambda\|)) = O(\|T\|)$, that is pseudo-linear in the size of G.

If $p > 1$. Let $\bar{w} = (\overline{x}_2, \ldots, \overline{x}_p)$. Consider the following three queries:

$$\begin{aligned} q_1(\overline{x}_1) &:= \alpha_1(\overline{x}_1) \wedge \tau_r(\overline{x}_1), \\ q_2(\bar{w}) &:= \alpha_2(\overline{x}_2) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_2; \ldots; \overline{x}_p), \\ q_3(\bar{x}, \bar{w}) &:= q_1(\overline{x}_1) \wedge q_2(\bar{w}) \wedge \mathrm{dist}(\overline{x}_1, \bar{w}) \leq 2r. \end{aligned}$$

We have that

$$G \models q(\overline{a}\overline{b}) \iff q_1(\overline{a}) \wedge q_2(\overline{b}) \wedge \mathrm{dist}(\overline{a}, \overline{b}) > 2r,$$

hence

$$q(G) = q_1(G) \times q_2(G) \setminus \{\overline{a}, \overline{b} \in G \mid q_1(\overline{a}) \wedge q_2(\overline{b}) \wedge \mathrm{dist}(\overline{a}, \overline{b}) \leq 2r\},$$

which is

$$q(G) = q_1(G) \times q_2(G) \setminus q_3(G).$$

Since

$$q_3(G) \subseteq q_1(G) \times q_2(G),$$

it follows that

$$\#q(G) = \#q_1(G) \cdot \#q_2(G) - \#q_3(G).$$

It is easy to see that both q_1 and q_2 have less than p connected components in their distance type. Therefore, by the induction assumption we can compute $\#q_1(G)$ and $\#q_2(G)$ in pseudo linear time. We now have to compute $\#q_3(G)$.

We say that $(\overline{x}'_1; \ldots; \overline{x}'_{p'}) \in \Pi(\overline{x}_1; \ldots; \overline{x}_p)$ if and only if:

- $(\overline{x}'_1, \dots, \overline{x}'_{p'})$ is a partition of $\overline{x}$ with $p' < p$,
- $\overline{x}_1 \subsetneq \overline{x}'_1$,
- $\forall 1 < j \leq p'$, there is a $i > 1$ such that $\overline{x}'_j = \overline{x}_i$.

Basically, $\overline{x}'_1$ is the collapse of $\overline{x}_1$ and at least one of the $\overline{x}_i$. The other $\overline{x}_i$ remain unaltered.

Given $(\overline{x}'_1; \dots; \overline{x}'_{p'})$, we define:
$\alpha'_1(\overline{x}'_1) = \bigwedge_{i \in I} \alpha_i(\overline{x}_i)$ where $I := \{i \leq p \mid \overline{x}_i \subset \overline{x}'_1\}$,
$\alpha'_j(\overline{x}'_j) = \alpha_i(\overline{x}_i)$ where $\overline{x}_i = \overline{x}'_j \quad \forall 1 < j \leq p'$.
It follows from those definitions that:

$$q_3(\overline{x}) = \bigvee_{(\overline{x}'_1;\dots;\overline{x}'_{p'}) \in \Pi(\overline{x}_1;\dots;\overline{x}_p)} \alpha'_1(\overline{x}_1) \wedge \dots \wedge \alpha'_{p'}(\overline{x}'_{p'}) \wedge \tau_r(\overline{x}'_1; \dots; \overline{x}'_{p'}).$$

Moreover these disjunctions are strict. Therefore, with Remark 2:

$$\#q_3(G) = \sum_{(\overline{x}'_1;\dots;\overline{x}'_{p'}) \in \Pi(\overline{x}_1;\dots;\overline{x}_p)} \#\big(\alpha'_1(\overline{x}_1) \wedge \dots \wedge \alpha'_{p'}(\overline{x}'_{p'}) \wedge \tau_r(\overline{x}'_1; \dots; \overline{x}'_{p'})\big).$$

Since every query present here has less than p connected components in its distance type, we can by induction count the number of solutions for each of them in pseudo-linear time. There is only a constant number of queries involved in this sum, therefore $\#q_3(G)$ is computable in pseudo-linear time.

As $\#q_1(G)$ and $\#q_2(G)$ are already computed, we can compute $\#q(G) = \#q_1(G) \cdot \#q_2(G) - \#q_3(G)$.

The total time needed was pseudo-linear. This concludes the proof.

8 Conclusion

We have shown how to efficiently process first-order queries over classes of graphs with locally bounded expansion. We did not explicitly mention the constant factors. These are not very good. Even in the bounded expansion case the constant factor is a tower of exponentials which height depends on the size of the query. Moreover, an elementary constant factor is not reachable (unless FPT = AW[$*$]) even for unranked trees [8].

The results state the existence of an enumeration procedure for all ϵ. A uniform version of this statement would require that the procedure is computable from ϵ. It is indeed the case if the class of local bounded expansion is "effective", see [16] for the precise definition.

An improvement of our work will be to extend the results for the counting and enumeration problems to nowhere-dense structures. On those structures, the model checking can be done in pseudo-linear time [9]. There is therefore hope to find good algorithms for the other problems. However, this remains future work.

References

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

2 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic (CSL'06)*, 2006.

3 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007. `doi:10.1145/1276920.1276923`.

4 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Symp. on Principles of Database Systems (PODS'14)*, 2014. doi:10.1145/2594538.2594539.

5 Zdenek Dvorak, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36, 2013. doi:10.1145/2499483.

6 Markus Frick. Generalized model-checking over locally tree-decomposable classes. *Theory Comput. Syst.*, 37(1):157–191, 2004. doi:10.1007/s00224-003-1111-9.

7 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

8 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004. doi:10.1016/j.apal.2004.01.007.

9 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Symp. on Theory of Computing (STOC)*, 2014. doi:10.1145/2591796.2591851.

10 Mamadou Moustapha Kanté. *Graph Structurings: Some Algorithmic Applications. (Structurations de Graphes: Quelques Applications Algorithmiques)*. PhD thesis, University of Bordeaux, France, 2008. URL: https://tel.archives-ouvertes.fr/tel-00419301.

11 Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011. doi:10.2168/LMCS-7(2:20)2011.

12 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Symp. on Principles of Database Systems (PODS)*, 2013. doi:10.1145/2463664.2463667.

13 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4), 2013. doi:10.1145/2528928.

14 Stephan Kreutzer and Anuj Dawar. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009. URL: http://eccc.hpi-web.de/report/2009/131.

15 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.

16 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *Eur. J. Comb.*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.

m-tables: Representing Missing Data

Bruhathi Sundarmurthy[1], Paraschos Koutris[1], Willis Lang[3], Jeffrey Naughton[4], and Val Tannen[5]

1 **University of Wisconsin-Madison, Madison, WI, USA**
2 **University of Wisconsin-Madison, Madison, WI, USA**
3 **Microsoft Gray Systems Lab, Madison, WI, USA**
4 **University of Wisconsin-Madison, Madison, WI, USA**
5 **University of Pennsylvania, Pennsylvania, PA, USA**

Abstract

Representation systems have been widely used to capture different forms of incomplete data in various settings. However, existing representation systems are not expressive enough to handle the more complex scenarios of missing data that can occur in practice: these could vary from missing attribute values, missing a known number of tuples, or even missing an unknown number of tuples. In this work, we propose a new representation system called m-tables, that can represent many different types of missing data. We show that m-tables form a *closed*, *complete* and *strong* representation system under both set and bag semantics and are strictly more expressive than conditional tables under both the closed and open world assumptions. We further study the complexity of computing certain and possible answers in m-tables. Finally, we discuss how to "interpret" m-tables through a novel labeling scheme that marks a type of generalized tuples as certain or possible.

1998 ACM Subject Classification H.2.4 [Database Management] Systems

Keywords and phrases missing values, incomplete data, c tables, representation systems

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.21

1 Introduction

Advances in technology have resulted in a large amount of data being collected and integrated from numerous data sources. A popular way to store such massive data-sets is by sharding over multiple independent relational database instances. When a user given query spans over such a collection, the complete data required by the query may not always be available. For instance, due to inconsistencies and incompleteness during data integration pipelines, data-sets are frequently incomplete and missing data [7, 11, 21]. As another example, when a query spans a large collection of shards, it becomes likely that some nodes will be unavailable for query processing due to outages, misconfigurations, or network congestion. This scenario of missing data due to node failures has also received attention in the applied literature, with a recent work [19] advocating an interesting solution: to "ignore" failures during query evaluation, and to inform the user of problems that may exist by labeling the result with possible errors.

Missing data in such scenarios can be of varied types: a database could be missing attribute values, a set of tuples with partially known values along with bounds on the cardinality of missing tuples, or even an unknown number of tuples. Current techniques that process queries over incomplete databases cannot handle the numerous kinds of missing data described above. For this reason, we propose in this paper a new representation system called m-tables to represent and operate on incomplete databases.

Table 1 Table R.

Name	DOB	State	Salary	
William Smith	6/5/1976	CA	100000	certain tuple
John Doe	?	?	?	two records may be missing
?	?	WI	?	up to 50 employee records may be missing
?	?	PA	?	unknown number of employee records may be missing

Querying over incomplete databases has been widely studied in the literature [1, 22]. The standard approach is to model an incomplete database using a formal representation system, M, that can succinctly describe the incomplete database, and then query over M. For example, conditional tables (c-tables) [16], a widely used representation system, are expressive enough to efficiently capture the result of any relational algebra query over the incomplete database represented by the c-table (such a representation system is called *strong* for relational algebra). However, c-tables can only represent missing attribute values under the so-called closed world assumption, and under the open world assumption (where they can represent an unknown number of missing tuples, but in a very limited manner) they are not closed for the selection operator. Other representation systems, such as v-tables or Codd-tables are even less expressive than c-tables. We note here that, to the best of our knowledge, none of the existing representation systems can represent incomplete databases consisting of zero tuples (the zero information incomplete database), or tuples with cardinality constraints (including possibly infinite cardinality). A different approach than using representation systems is to compute *certain* answers: a tuple is considered to be a certain answer if it is in the result of executing a given query over every possible instance of the incomplete database. However, such an approach loses information about the incomplete database.

To give a more concrete example, consider a practical scenario of a user running analytical queries over a column store database that consists of information about a company's employees in various states. Due to missing values in the database and due to node failures during query execution, the data required to process the analytical queries may not be completely available. A toy version of a resulting incomplete database is shown in Table 1. The last column in table R indicates whether the tuple is certainly part of the table or not and if not, what constraints are placed on the missing data. The first tuple is definitely part of the dataset. The second tuple indicates that we may be missing two employees, both with name 'John Doe'. We could be missing up to fifty employees from the state of Wisconsin as indicated by the third tuple, and the fourth tuple indicates that we could be missing an unknown number of tuples from the state of Pennsylvania. None of the existing representation systems can handle the last three types of missing data.

Contributions. We summarize below the contributions of this work:

1. We present in detail a new representation system called m-tables (Section 3). To construct our new representation we use several ideas, among which is the use of annotations in the form of polynomials (similar to provenance polynomials [13]). We give several examples of how m-tables can be used to express various types of missing data that can arise in practice (Section 3.3).
2. We show (in Section 4) that m-tables are a strong representation system for positive relational algebra $\mathcal{RA}^+$ (which includes selection, projection, join, union and renaming) under both set and bag semantics. This means that we can efficiently compute a new m-table that represents the result of a $\mathcal{RA}^+$ query over underlying m-tables. We should

emphasize here that an important feature of m-tables is that it can capture bag semantics, in contrast to other representation systems and current approaches, where operations under bag semantics are not clearly defined [15].

3. We prove that m-tables are strictly more expressive than c-tables, under both the closed and open world assumptions (Section 5). We also perform a detailed study of the expressive power of m-tables (also Section 5).
4. Given a query result as an m-table, interpreting the semantics to determine whether a tuple is a certain or a possible answer is not an easy task. We thus propose a labeling scheme (Section 6) that interprets the annotations and labels a type of "generalized" tuples with either certain or possible labels, along with other possible information. We show that, as a consequence of the labeling process, we can study the complexity of computing the certain and possible tuples of an m-table representation.

2 Background

In this section, we present the necessary background on representation systems of incomplete data. We will focus on *c-tables*, a representation system that will be of interest to us. We further provide an overview of semiring algebras and provenance semirings [13].

2.1 Representation Systems

We assume that a relational instance is defined over a countably infinite domain $\mathbb{D}$. For the sake of simplicity, we will present the definitions over a relational schema with a single relation with attribute set U; the definitions extend in a straightforward way to any database schema. We use the convention that a *tuple* is a function $t : U \to \mathbb{D}$, and we let $\mathbb{D}^U$ denote the set of all possible tuples with schema U.

An *incomplete database* $\mathcal{I}$ is any set of finite instances $I \subseteq \mathbb{D}^U$ (an instance of an incomplete database refers to a possible completion of the incomplete database). The standard definition of a complete database corresponds to a singleton set $\{I\}$. Representation systems can concisely describe an incomplete database: a *representation system* consists of a set of elements $\mathcal{M}$, and a function Mod that maps each $M \in \mathcal{M}$ to an incomplete database $\mathcal{I}$. For a query $q \in \mathcal{L}$, where $\mathcal{L}$ is a query language, we define the answer of q on the incomplete database $\mathcal{I}$ as $q(\mathcal{I}) = \{q(I) \mid I \in \mathcal{I}\}$. In this work, we will mainly focus on *positive Relational Algebra*, or $\mathcal{RA}^+$, which contains queries that are formed using the selection, projection, join, union and renaming, and the full *Relational Algebra*, $\mathcal{RA}$, which additionally uses difference.

▶ **Definition 1** (Closure)**.** A representation system is *closed* under a query language $\mathcal{L}$ if for any query $q \in \mathcal{L}$ and any $M \in \mathcal{M}$, there exists some $M' \in \mathcal{M}$ such that $q(\mathsf{Mod}(M)) = \mathsf{Mod}(M')$. We further say that it is *strong* for $\mathcal{L}$ if M' is computable.

In addition to the closure property, we are interested in representation systems where we can efficiently perform the following tasks [14, 24]:

Instance Membership. Given an instance I and $M \in \mathcal{M}$, is $I \in \mathsf{Mod}(M)$?

Tuple Membership. Given a tuple $t \in \mathbb{D}^U$ and $M \in \mathcal{M}$, does there exist some instance $I \in \mathsf{Mod}(M)$ such that $t \in I$?

Tuple Certainty. Given a tuple $t \in \mathbb{D}^U$ and $M \in \mathcal{M}$, does $t \in I$ for every $I \in \mathsf{Mod}(M)$?

We should mention here that tuple and instance membership, as well as tuple certainty, can be extended to be defined with respect to a given query q. For example, for tuple membership we can ask whether for a given tuple t, $M \in \mathcal{M}$ and a query q, there exists $I \in q(\mathsf{Mod}(M))$ such that $t \in I$.

A	B	C	
10	2	y	$x = 100 \vee y = 210$
40	x	x	$x \neq 100$

Figure 1 Example of a c-table.

Conditional Tables. A *conditional table, or c-table,* is expressed as (S, ϕ), where S is a table that contains variables along with constant values from $\mathbb{D}$, and ϕ is a function that associates a local condition ϕ_s (a boolean combination of equalities involving variables and constants) with each tuple $s \in S$.

As an example c-table, consider $C(A, B, C)$ with two tuples, t_1 and t_2. $t_1 = (10, 2, y)$ with conditions $(x = 100 \vee y = 210)$ and $t_2 = (40, x, x)$ with condition $(x \neq 100)$.

Let v denote a valuation that maps the variables in a c-table to values in $\mathbb{D}$. Under the *closed world assumption (CWA)*, a c-table $C = (S, \phi)$ represents: $\mathsf{Mod}_C(C) = \{I \mid \exists$ valuation v s.t. $I = \{v(t) \mid t \in S, v$ satisfies $\phi_t\}\}$. The CWA for missing data assume that all information about an incomplete database is modeled by its representation. Alternatively, a c-table can be defined under the *open world assumption (OWA)*, where an instance of an incomplete database can contain any number of tuples, not necessarily justified by the presence of a tuple in its representation. Under OWA: $\mathsf{Mod}_O(C) = \{I \mid \exists$ valuation v s.t. $I \supseteq \{v(t) \mid t \in S, v$ satisfies $\phi_t\}\}$. c-tables form a closed and strong representation system for $\mathcal{RA}$ [16] under CWA. However, c-tables are not closed even for $\mathcal{RA}^+$ under OWA. We will present a detailed comparison of c-tables with m-tables in Section 3.

2.2 Semiring Algebras and Provenance

A *commutative monoid* is a structure $(M, +_M, 0_M)$ where $+_M$ is an associative and commutative binary operation and 0_M is the identity for $+_M$. A *commutative semiring* is a structure $(K, +_K, \cdot_K, 0_K, 1_K)$, where $(K, +_K, 0_K)$ and $(K, \cdot_K, 1_K)$ are commutative monoids, $\cdot_K$ is distributive over $+_K$, and $a \cdot_K 0_K = 0_K \cdot_K a = 0_K$. Examples of commutative semirings are the natural number semiring $(\mathbb{N}, +, \cdot, 0, 1)$ and the boolean semiring $(\mathbb{B}, \vee, \wedge, \bot, \top)$. A *semiring homomorphism* is a mapping $h : K \to K'$ where K, K' are semirings and $h(0_K) = 0_{K'}$, $h(1_K) = 1_{K'}$, $h(a +_K b) = h(a) +_{K'} h(b)$, $h(a \cdot_K b) = h(a) \cdot_{K'} h(b)$.

The work on provenance semirings [13, 4, 12, 18] established the theoretical foundations and implementations for representing, computing and querying *annotated relations*. Many applications need to manipulate some "property" of tuples. These properties, viewed as annotations, and operations on these tuple annotations together form the *semiring* algebraic structure. These semiring structures adequately capture enough information for a variety of applications including obtaining *what* and *how* [5] provenance information. In particular, the *how* provenance of result tuples is captured by annotations from the provenance semiring, $(\mathbb{N}[X], +, \cdot, 0, 1)$, where $\mathbb{N}[X]$ represents the multivariate polynomials with indeterminates from X (a set of *provenance tokens* that we use to annotate input tuples).

Let U be a finite set of attributes and $(K, +, \cdot, 0, 1)$ be a commutative semiring. A *K-relation* is a function $R : \mathbb{D}^U \to K$, whose support, $\mathsf{supp}(R) = \{t \mid R(t) \neq 0\}$ is finite. The $+$ operation in the semiring represents alternate derivations for the same tuple, and the $\cdot$ operation represents the joint use of data to obtain the tuple. 1 represents a tuple that is present in the result and 0 represents the absence of that tuple. We refer the reader to [13] for details on how to execute relational algebra queries over K-relations.

3 How to Represent Missing Data

In this section, we develop a representation system, called *m-tables*, which allows us to represent relational data with missing tuples. Although there exists extensive literature on representation systems for incomplete and uncertain data (from *c*-tables [16], to more recent research [14, 24]), existing representation systems cannot represent and operate on missing data that may consist of zero to any number of tuples. Consider the following example.

▶ **Example 2.** Let $R(A, B, C)$ be a ternary relation, and suppose that the domain $\mathbb{D}$ is the natural numbers $\mathbb{N}$. Our goal is to represent the incomplete database $\mathcal{I}$ that contains all instances that satisfy the following conditions:

- The tuple $(1, 2, 3)$ must be included in any instance.
- The tuple $(2, 3, 4)$ may be present in an instance; if present, its multiplicity should be 2.
- Any other tuple, if present, must be of the form $(x, y, 3)$, where $x, y \in \mathbb{D}$ and $3 \leq x \leq 10$.

$\{R(1, 2, 3), R(2, 3, 4), R(2, 3, 4), R(4, 5, 3)\}$ and $\{R(1, 2, 3), R(4, 5, 3), R(5, 5, 3)\}$ are possible instances of $\mathcal{I}$, but $\{R(2, 3, 4), R(4, 5, 3)\}$ and $\{R(1, 2, 3), R(2, 3, 4), R(4, 5, 3)\}$ are not. Observe that an instance in $\mathcal{I}$ is a *bag*, and not a set. Each possible instance of this incomplete database $\mathcal{I}$ can contain zero to any number of tuples of the form $(x, y, 3)$, and this cannot be represented by a *c*-table (under either open or closed world semantics), *v*-table or other representation system.

An incomplete database like $\mathcal{I}$ is not only of theoretical interest, since it can be the result of answering a query over a partitioned database (can also be a column store), where some partition has failed during evaluation (additionally, some columns may be unavailable for processing). Example 2 will be the running example throughout this section.

3.1 Basic Definitions

In this section, we define the components of an *m*-table in a bottom-up manner and then proceed to discuss *m*-table semantics. Informally, the construction involves two requirements. First, we need to represent 'missing' values along with the associated domain and cardinality constraints; we also need to encode correlation constraints between the missing values. Second, we require that the representation allows for systematic propagation of the constraints of missing values through the algebraic operators. To achieve this, we define a schema over the missing values (encoding correlations) with cardinality constraints defined over the schema elements. Next, we observe that propagating the schema information along with domain constraints is akin to propagating the provenance information for tuples. So, we introduce suitable annotations for tuples and propagate these annotations using machinery from [13].

We now begin the construction of *m*-tables. Let $\mathbf{U}$ denote the set of possible attribute names and $\mathbb{D}$ denote the set of all constants. $\mathbb{D}$ represents the domain of all the attributes.

Missing values. The first component is a distinguished symbol m, which represents any *missing value*. Define $\hat{\mathbb{D}} = \mathbb{D} \cup \{m\}$. This notation is similar in spirit to other types of representation systems; the difference in our setting is that m can potentially represent multiple possible values, or even no values at all, instead of exactly one. One should think of m as representing a set or a bag (depending on the semantics) of possible values.

We introduce the notion of an *extended tuple* $\hat{t} \in \hat{\mathbb{D}}^U$, where $U \subseteq \mathbf{U}$, to represent tuples in the representation $M \in \mathcal{M}$. This notation is meant to distinguish from a tuple t in an instance of an incomplete database. Notice that the distinguished symbol m can only be part of a tuple $\hat{t}$ and not t. We define $\mathsf{m}(\hat{t}) = \{A \in U \mid \hat{t}[A] = m\}$ and $\bar{\mathsf{m}}(\hat{t}) = \{A \in U \mid \hat{t}[A] \neq m\}$.

In other words, $\mathsf{m}(\hat{t})$ includes the attributes in the tuple that have a missing value instead of a constant value. For example, to represent the incomplete database in Example 2, we introduce three extended tuples: $\hat{t}_1 = (1, 2, 3)$, $\hat{t}_2 = (2, 3, 4)$ and $\hat{t}_3 = (m, m, 3)$.

The set of extended tuples is not enough by itself to represent the incomplete database $\mathcal{I}$ of Example 2. Indeed, there is no way to distinguish that $(1, 2, 3)$ is a *certain* tuple, in the sense that it appears in all instances, and $(2, 3, 4)$ is a *possible* tuple with multiplicity 2, in the sense that it appears in some instances. Furthermore, we want to ensure that for the tuple $(m, m, 3)$, the first m can only take values between 3 and 10, which is not yet captured. We thus augment the table with annotations.

However, simply annotating tuples with domain conditions will not be sufficient. A structure is necessary to encode correlations between different missing values that appear in different tuples or attributes of the same tuple. For instance, consider the tuple $\hat{t}_3 = (m, m, 3)$; consider another relation $S(C, D, E)$ with tuples $\hat{s}_1 = (3, 4, 5)$ and $\hat{s}_2 = (3, 9, 10)$. If $\hat{t}_3$ were to be joined with the relation S on attribute C, then we will have two tuples in the result: $r_1 = (m, m, 3, 4, 5)$ and $r_2 = (m, m, 3, 9, 10)$. Observe that $\mathsf{m}(\hat{r}_1) = \mathsf{m}(\hat{r}_2) = \{A, B\}$; the pairs of m values across tuples are not independent and are bound by the values taken by $\mathsf{m}(\hat{t}_3)$. To capture this, we need to relate annotations to the m values in the tuples; we do so by introducing a second component, a database schema $\Sigma = \{T_1(U_1), T_2(U_2), \ldots, T_N(U_N)\}$, where $U_i \subseteq \mathbf{U}$ for each $i = 1, \ldots, N$.

For the running example, we need a schema that allows the presence of the tuple $(2, 3, 4)$ (with multiplicity 2) to be toggled and allows the tuple $(m, m, 3)$ to take on multiple values for m. We construct the schema $\Sigma = \{T_1(), T_2(A, B)\}$. Observe that relation T_1 has no attributes: this means that T_1 will behave like a boolean variable depending on whether T_1 is empty or contains the empty tuple $()$ (this is also because of the additional cardinality constraints that we introduce next).

An instantiation of Σ determines an instance of the incomplete database. The size of the possible instantiations of Σ are constrained by two vectors, $\mathbf{min} = (min_1, \ldots, min_N)$ and $\mathbf{max} = (max_1, \ldots, max_N)$, where $min_i, \max_i \in \mathbb{N} \cup \{\infty\}$. The number of tuples in every instantiation of $T_i \in \Sigma$ are lower bounded by min_i and upper bounded by max_i. For our running example, we add the cardinality constraints $min_1 = 0, max_1 = 1$ and $min_2 = 0, max_2 = \infty$. The constraints enforce that T_1 behaves like a boolean variable and T_2 can be instantiated to anything.

We next introduce annotations that capture all necessary properties of an extended tuple; as we will argue in the next section, annotations are also necessary to make the representation system complete for SPJU queries.

Annotations. To construct a suitable set of annotations for m-tables, we first need to define two new kinds of expressions. The first kind has expressions of the form $\alpha_i(U)$, where $i = 1, \ldots, N$ corresponds to the relation $T_i(U_i)$ of the schema Σ, with $U \subseteq \mathbf{U}$ and $|U| = |U_i|$. The condition $|U| = |U_i|$ is sufficient, since as we will see in the next section, applying a renaming operator can change the attributes in α_i. We define $\mathcal{K} = \{\alpha_i(U) \mid T_i(U_i) \in \Sigma, U \subseteq \mathbf{U}, |U| = |U_i|\}$. In the preceding definition, not requiring $U_i = U$ allows the reuse of α expressions across multiple attributes and tables.

The second kind of expressions are symbolic equations, which will be used to capture selection and join conditions in query evaluation. We define $\mathcal{E} = \{[x \ \mathsf{op} \ y] \mid x, y \in \mathbb{D} \cup \mathbf{U}, \mathsf{op} \in \{=, <, >, \leq, \geq, \neq\}\}$.

For example, if $A, B \in \mathbf{U}$ and $\mathbb{D} = \mathbb{N}$, both $[A = B]$ and $[A > 3]$ are valid expressions in $\mathcal{E}$. This definition is similar to the technique used in [4] to capture provenance for

queries with aggregates. We will use the above expressions to annotate each extended tuple. Formally, let $\hat{K}$ be the *polynomial semiring* with variables from $\mathcal{K} \cup \mathcal{E}$ and coefficients from $\mathbb{N}$: $(\mathbb{N}[\mathcal{K} \cup \mathcal{E}], +, \cdot, 0, 1)$. We can then define a $\hat{K}$-relation R that maps each extended tuple in $\hat{\mathbb{D}}^U$ to an element in the semiring $\hat{K}$. The coefficients from $\mathbb{N}$ in the polynomial enable the annotations to handle both set and bag semantics.

▶ **Example 3.** Continuing the running example, the annotation for the tuple $(1, 2, 3)$ is 1 (the tuple is certain). The annotation for the tuple $(2, 3, 4)$ is $2 \cdot \alpha_1()$, with $\alpha_1()$ operating like a boolean variable. Alternately, we could have had two $(2, 3, 4)$ tuples, each with annotation $\alpha_1()$. Finally, the annotation for the tuple $(m, m, 3)$ is $\alpha_2(A, B) \cdot [A \leq 10] \cdot [A \geq 3]$. Intuitively, for each instance of the relation $T_2(A, B)$, we first filter the tuples using the conditions of the expressions $[A \leq 10]$ and $[A \geq 3]$. The result will define a set of valuations from (A, B) to $(\mathbb{D}, \mathbb{D})$; each such valuation will correspond to a tuple in the possible world.

An annotation $R(\hat{t}) \in \mathbb{N}[\mathcal{K} \cup \mathcal{E}]$ has a natural interpretation as a query in $\mathcal{RA}^+$. We first write the polynomial $R(\hat{t})$ in the following canonical form[1]: $R(\hat{t}) = \sum_{k=1}^{n} (\prod_{k_i} \alpha_{k_i}(U_{k_i}) \cdot \prod_{k_j} \theta_{k_j})$ where, $\theta_{k_j} \in \mathcal{E}$. We can now interpret each monomial in $R(\hat{t})$ as a query, which involves a renaming operation (to match attributes in α_i and T_i), followed by a natural join, followed by a selection condition specified by the equations θ_{k_j} and followed by a projection on $\mathsf{m}(\hat{t})$. Formally, for the k-th monomial in $R(\hat{t})$, we associate the query: $q_k(R(\hat{t})) = \pi_{\mathsf{m}(\hat{t})} \left(\sigma_{\bigwedge_{k_j} \theta_{k_j}} \left(\bowtie_{k_i} (\rho_{A_{k_i}/U_i} T_{k_i}) \right) \right)$. The result of this query will be a relation defined over the attributes in $\mathsf{m}(\hat{t})$. To obtain the final query associated with $R(\hat{t})$, we first extend this definition over all attributes in U, and then take the union over all monomials. Formally, we map each annotation $R(\hat{t})$ to:

$$q(R(\hat{t})) \stackrel{\text{def}}{=} \bigcup_{k=1}^{n} \left(\pi_{\bar{\mathsf{m}}(\hat{t})} \{\hat{t}(A)\} \times q_k(R(\hat{t})) \right) \tag{1}$$

The query $q(R(\hat{t}))$ returns a relation (set or bag) defined over the attribute set U. In the case where $R(\hat{t}) = 1$, $q(R(\hat{t}))$ is the constant query that returns a relation with an empty tuple $()$.

▶ **Example 4.** For the tuples $\hat{t}_2$, $\hat{t}_3$ we have $q_2 = q(R(\hat{t}_2)) = (\pi_{A,B,C}\{\hat{t}_2\} \times \pi_{()}(T_1)) \cup (\pi_{A,B,C}\{\hat{t}_2\} \times \pi_{()}(T_1))$ and $q_3 = q(R(\hat{t}_3)) = \pi_C\{\hat{t}_3\} \times \pi_{A,B}(\sigma_{A \geq 3 \wedge A \leq 10}(T_2))$. Since $\hat{t}_2$ has two monomials, $\alpha_1() + \alpha_1()$, in its annotation, its final annotation is a union over the queries of each of its monomials.

We could equivalently define an annotation directly as a query in $\mathcal{RA}^+$. The choice to use semirings instead is because they form a more compact annotation and work seamlessly for both set and bag semantics. For instance, the simple annotation $100 \cdot \alpha_1()$ would have to be written as a union of 100 expressions T_1.

The query $q(\gamma)$ may not be well-defined for a given polynomial γ, since a selection condition θ_{k_j} or a projection operator may include an attribute that does not appear in any of the α_{k_i} terms. For example, the annotation $\alpha_2(A, B) \cdot [C = 1]$ for the tuple $(m, m, 3)$ would correspond to the query $\pi_C\{\hat{t}_3\} \times \pi_{A,B}(\sigma_{C=1}(T_2))$, which is not a valid expression.

It is easy to see that an annotation $R(\hat{t})$ is valid if and only if for every monomial $\prod_{k_i} \alpha_{k_i}(A_{k_i}) \cdot \prod_{k_j} \theta_{k_j}$ in the annotation the following hold: (1) $\mathsf{m}(\hat{t}) \subseteq \bigcup_{k_i} A_{k_i}$, and (2) for any $\theta_{k_j} = [x_1 \ \mathsf{op} \ x_2]$ such that $x_i \in \mathbf{U}$, $x_i \in \bigcup_{k_i} A_{k_i}$.

[1] A monomial with coefficient > 1 can be easily split into multiple monomials, thus conforming to the canonical form. For example, the annotation $2 \cdot \alpha_1()$ can be written as $\alpha_1() + \alpha_1()$.

Table 2 The m-table for the running example.

A	B	C	
1	2	3	1
2	3	4	$2 \cdot \alpha_1()$
m	m	3	$\alpha_2(A,B) \cdot [A \leq 10] \cdot [A \geq 3]$

▶ **Definition 5** (m-proper)**.** A $\hat{K}$-relation R over a set of attributes U is *m-proper* if for every extended tuple $\hat{t} \in \hat{\mathbb{D}}^U$, $R(\hat{t})$ is a valid annotation.

Using the conditions for a valid annotation, given a $\hat{K}$-relation, we can efficiently determine whether it is m-proper. We now formally define m-tables.

▶ **Definition 6.** A set of *m-tables*, or an *m-multitable*, is a tuple $(\mathbf{R}, \Sigma, \mathbf{min}, \mathbf{max})$ such that:

1. each $R_j \in \mathbf{R}$ is an m-proper $\hat{K}$-relation, where $\hat{K}$ is the polynomial semiring $(\mathbb{N}[\mathcal{K} \cup \mathcal{E}], +, \cdot, 0, 1)$ (recall that the elements of $\mathcal{K}$ are constructed from the elements of Σ),
2. $\Sigma = \{T_1(U_1), \ldots, T_N(U_N)\}$ is a database schema,
3. $\mathbf{min}, \mathbf{max} \in (\mathbb{N} \cup \{\infty\})^N$ are vectors of cardinality constraints.

To define a single m-table for a relation R, we can simply write it as $M = (R, \Sigma, \mathbf{min}, \mathbf{max})$.

As discussed before, the cardinality constraints min_i, max_i provide a lower and upper bound on the cardinality of an instance of the relation $T_i \in \Sigma$. In the case where $min_i = 0$ and $max_i = \infty$ for every i, we say that the m-table is *free* and for simplicity we omit $\mathbf{min}, \mathbf{max}$ from the m-table definition. We denote by $\mathcal{M}^f$ the set of all free m-tables. When $min_i = max_i = 1$ for every i, we can equivalently view each relation T_i as a function from attributes to values in $\mathbb{D}$; as we will see later, this will allow us to capture the semantics of c-tables. We denote by $\mathcal{M}^c$ the set of all such m-tables. Finally, we define an *$m^{\not=}$-table* as an m-table where the expressions in the annotation are restricted to use only $=, \neq$.

For our running example, the final m-table $M = (R, \Sigma, \mathbf{min}, \mathbf{max})$ will have $\Sigma = \{T_1(), T_2(A, B)\}$ and $min_1 = min_2 = 0$, $max_1 = 1, \max_2 = \infty$. The annotated relation R can be seen in Table 2.

3.2 Semantics

We present here the semantics of m-tables. Given $M = (\mathbf{R}, \Sigma, \mathbf{min}, \mathbf{max})$, we formally define the incomplete database $\mathsf{Mod}(M)$ that it represents under both set and bag semantics.

To explain the semantics behind m-tables, we draw a parallel with c-tables. For a c-table C, each possible instance of the incomplete database is produced by computing $v(C)$ for a valuation v over the variables in the c-table. In m-tables, instead of a valuation, we will use an instance $\mathbf{T}$ on the schema Σ, which satisfies the cardinality constraints; each such instance will produce a possible instance I of the incomplete database: $M[\mathbf{T}]$ in $\mathsf{Mod}(M)$. Under set semantics, the instance I will be a set, and under bag semantics it will be a bag.

We start by looking at a single extended tuple $\hat{t}$ with annotation $R(\hat{t})$, for some $R \in \mathbf{R}$ with attribute set U. Let $J = q(R(\hat{t}))(\mathbf{T})$. As we discussed in the previous section, each tuple $v \in J$ can be equivalently viewed as a total function $v : U \to \mathbb{D}$. We say that $v(\hat{t})$ is an *instantiation* of the extended tuple $\hat{t}$.

▶ **Definition 7** (Derivation Set/Bag)**.** Let $\hat{t}$ be an extended tuple with a valid annotation $R(\hat{t})$. The *derivation set (bag)* of $\hat{t}$ for a set (bag) instance $\mathbf{T}$ of Σ is defined as $q(R(\hat{t}))(\mathbf{T})$.

▶ **Example 8.** Consider the schema Σ and the cardinality constraints of our running example. Consider the following set instance $\mathbf{T}$ of Σ: $\mathbf{T} = \{T_2(2,4), T_2(3,4), T_1()\}$. Then, the derivation set of the tuple $(1,2,3)$ with annotation 1 is $\{(1,2,3)\}$. For the tuple $\hat{t}_2 = (2,3,4)$ with annotation $2 \cdot \alpha_1()$, the derivation set w.r.t. $\mathbf{T}$ will be $\{(2,3,4)\}$. Notice that because of the set semantics, the coefficient 2 in the annotation is effectively ignored. Finally, for $\hat{t}_3 = (m,m,3)$ with annotation $\alpha_2(A,B) \cdot [A \leq 10] \cdot [A \geq 3]$, we compute $q_3(\mathbf{T}) = \{(3,4,3)\}$.

If we switch to bag semantics, we can start with a bag instance of the schema Σ: $\mathbf{T} = \{T_2(2,4), T_2(3,4), T_2(3,4), T_1()\}$. The derivation bag of $(1,2,3)$ will be as before $\{(1,2,3)\}$. For the tuple $\hat{t}_2 = (2,3,4)$ with annotation $2 \cdot \alpha_1()$, the derivation bag w.r.t. $\mathbf{T}$ will now be $\{(2,3,4),(2,3,4)\}$. Observe that the coefficient 2 is now critical for the correct interpretation. Finally for $\hat{t}_3 = (m,m,3)$ with annotation $\alpha_2(A,B) \cdot [A \leq 10] \cdot [A \geq 3]$, we compute $q_3(\mathbf{T}) = \{(3,4,3),(3,4,3)\}$ as its derivation bag.

▶ **Definition 9** (m-table Semantics)**.** Let $M = (\mathbf{R}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-multitable. For $R \in \mathbf{R}$, define the query $\mathcal{Q}_R \stackrel{\text{def}}{=} \bigcup_{\hat{t}:R(\hat{t})\neq 0} q(R(\hat{t}))$. The *instantiation* of M under a (set or bag) instance $\mathbf{T}$ of Σ is $M[\mathbf{T}] = \{\mathcal{Q}_R(\mathbf{T}) \mid R \in \mathbf{R}\}$. Under set (bag) semantics, the incomplete database $\mathsf{Mod}_S(M)$ ($\mathsf{Mod}_B(M)$) that is represented by M is

$$\mathsf{Mod}_{S/B}(M) = \{M[\mathbf{T}] \mid \forall i = 1,\ldots,N : \mathbf{T}^{T_i} \text{ is a set/bag, } min_i \leq |\mathbf{T}^{T_i}| \leq max_i\},$$

In other words, for each instance $\mathbf{T}$, we construct a possible world of the incomplete database by taking the union of all the derivation sets (or bags) for each extended tuple (w.r.t. $\mathbf{T}$) in the annotated relation. This of course is equivalent to computing the query $\mathcal{Q}_R(\mathbf{T})$ for each $R \in \mathbf{R}$. To construct the incomplete database, we compute all possible worlds that correspond to every instance of Σ that satisfies the cardinality constraints $\mathbf{min}, \mathbf{max}$ of the m-table. We present next an example that sheds more light on the semantics of m-tables.

▶ **Example 10.** Consider the binary relation $R(A,B)$ along with two different schemas: $\Sigma_1 = \{T_1(A,B)\}$ and $\Sigma_2 = \{T_2(A), T_3(B)\}$.

Consider first the free m-table $M_1 = (\{R_1\}, \Sigma_1)$, where R_1 contains the tuple (m,m) with annotation $\alpha_1(A,B)$. It is easy to see that $\mathsf{Mod}_S(M_1)$ is the set of all possible instances of R over the domain $\mathbb{D}$, otherwise known as the no-information instance.

Second, consider the free m-table $M_2 = (\{R_2\}, \Sigma_2)$, where R_2 contains, again, a single tuple (m,m) with annotation $\alpha_2(A) \cdot \alpha_3(B)$. Observe now that the incomplete database $\mathsf{Mod}_S(M_2)$ does not include all possible instances, since for example, the instance $\{(1,1),(1,2),(2,1)\}$ can not be produced from M_2.

3.3 Examples and Applications

In this section, we show how to use m-tables to represent different types of missing data that occur in a practical setting. Recall our original motivation: we have a cluster of nodes executing relational queries. Suppose that one of the tables in this cluster is $R(A_1,\ldots,A_k)$, and that during the execution of a query, several nodes become unresponsive. We look at three different cases:

Missing Arbitrary Data. We are certain about several tuples in the table R, but we are missing an arbitrary part, for which we have no information. To represent this instance, our underlying m-table schema needs a single relation: $\Sigma = \{T_1(A_1,\ldots,A_k)\}$. For the annotated relation, we first include in R all the certain tuples with annotation 1. Then, we introduce one more tuple $(m,m,\ldots,m)$ with annotation $\alpha_1(A_1,\ldots,A_k)$.

Missing Data in Range-Partitioned Databases. In this scenario, table R is initially range-partitioned across different nodes in the cluster using an attribute, say A_1. We construct an m-table for this case as follows. For each responsive node, we add the tuples in the nodes as certain tuples with annotation 1. Our underlying schema is the same as before: $\Sigma = \{T_1(A_1, \ldots, A_k)\}$. Let $[x_i, y_i]$ be the range of each missing node i. We then add a tuple $(m, m, \ldots, m)$ with annotation $\alpha_1(A_1, \ldots, A_k) \cdot \sum_i([A_1 \geq x_i] \cdot [A_1 \leq y_i])$.

Missing Data in Column Stores. Suppose that the table R is stored in columnar format. In this case, the columns may not be sharded, but observe that all columns may not be accessed at the same time. A particular node may be accessed several times during query processing while stitching the columns together. One of these accesses might fail and result in a missing column. Let's say we are missing the column corresponding to attribute A_1. We can use m-tables to represent this type of missing data as follows. Every tuple will be of the form $(m, a_2, \ldots, a_k)$, where $a_i \in \mathbb{D}$ and will have annotation $\alpha_j(A_1)$, where we introduce a distinguished unary relation T_j for every tuple. Moreover, we add cardinality constraints such that $min_j = max_j = 1$.

4 $\mathcal{RA}^+$ Algebra for m-tables

In this section we present the specifics of executing operators in the positive relational algebra ($\mathcal{RA}^+$) over m-tables, thus proving that m-tables form a strong representation system for $\mathcal{RA}^+$ under both set and bag semantics. Recall that an m-multitable M is a tuple $(\{R_1, \ldots, R_\ell\}, \Sigma, \mathbf{min}, \mathbf{max})$, where each R_i is an m-proper $\hat{K}$-relation. Thus, in order to define relational operators over m-tables we will need to modify the standard algebra operators over K-relations. We next present how each operator works.

Selection. Let $R : \hat{\mathbb{D}}^U \to \hat{K}$ and let θ be a selection predicate of the form $(A \ \mathsf{op} \ x)$, where $A \in U$, $x \in \mathbb{D}$ and $\mathsf{op} \in \{=, <, >, \leq, \geq, \neq\}$. Then, the selection $\sigma_\theta R : \hat{\mathbb{D}}^U \to \hat{K}$ is defined as

$$(\sigma_\theta R)(\hat{t}) = \begin{cases} R(\hat{t}) \cdot [A \ \mathsf{op} \ x] & \text{if } \hat{t}(A) = m, \\ R(\hat{t}) \cdot [\hat{t}(A) \ \mathsf{op} \ x] & \text{otherwise.} \end{cases}$$

Observe that if $\hat{t}(A) \neq m$, we can immediately evaluate the condition by checking whether the expression $(\hat{t}(A) \ \mathsf{op} \ x)$ is true or not. If it is true, then $(\sigma_\theta R)(\hat{t}) = R(\hat{t})$; otherwise $(\sigma_\theta R)(\hat{t}) = 0$. These semantics coincide with the algebra on K-relations. When $\hat{t}(A) = m$, the attribute value is unknown and the extended tuple $\hat{t}$ may potentially satisfy the selection predicate. Thus, we need to keep the expression uninterpreted as part of the annotation. The case where the condition is of the form $(A \ \mathsf{op} \ B)$ is similar and thus omitted.

Projection. Let $R : \hat{\mathbb{D}}^U \to \hat{K}$ and $U' \subseteq U$. The projection $\pi_{U'} R : \hat{\mathbb{D}}^{U'} \to \hat{K}$ is defined as $(\pi_{U'} R)(\hat{t}) = \sum_{t' : R(t') \neq 0 \wedge \hat{t} = t' \text{ on } U'} R(t')$.

Union. Let $R_1, R_2 : \hat{\mathbb{D}}^U \to \hat{K}$. Then the union $R_1 \cup R_2 : \hat{\mathbb{D}}^U \to \hat{K}$ is defined as $(R_1 \cup R_2)(\hat{t}) = R_1(\hat{t}) + R_2(\hat{t})$.

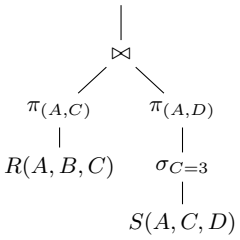
Renaming. Let $R : \hat{\mathbb{D}}^U \to \hat{K}$ and let $\beta : U \to U'$ be a bijection. To define the semantics for the renaming operator ρ_β, we need to rename the attributes in the annotation as well. For this, we define $\beta(\alpha_i(A)) = \alpha_i(\beta(A))$ and also $\beta([x \ \mathsf{op} \ y]) = [\beta(x) \ \mathsf{op} \ \beta(y)]$. (The function β

A	B	C	
a_1	b_1	50	1
a_2	b_2	400	1
a_3	b_3	30	1

(a) Table R.

A	C	D	
a_1	3	450	1
a_2	3	m	$\alpha_1(D)$
a_3	3	100	$\alpha_2()$
m	m	m	$\alpha_3(A, C, D) \cdot [D > 100]$

(b) Table S.



(c) Query Plan.

Figure 2 The initial m-tables R, S and the query plan for the running example.

behaves as the identity function for constants and attributes not in U.) For an annotation $R(\hat{t})$, we now define $\beta(R(\hat{t}))$ as the result of applying β to each element of the polynomial. We can now define $\rho_\beta R$ to be a $\hat{K}$-relation over U' such that: $(\rho_\beta R)(\hat{t}) = \beta(R(\hat{t} \circ \beta))$.

Cartesian Product. Let $R_i : \hat{\mathbb{D}}^{U_i} \to \hat{K}$ for $i = 1, 2$ and let $\hat{t}|U_i$ represent the restriction of the tuple $\hat{t}$ to the attributes of U_i. Then $R_1 \times R_2 : \hat{\mathbb{D}}^{U_1 \cup U_2} \to \hat{K}$ is defined as $(R_1 \times R_2)(\hat{t}) = R_1(\hat{t}_1) \cdot R_2(\hat{t}_2)$ where, $\hat{t}_i = \hat{t}|U_i, i = 1, 2$. An important point is that we assume w.l.o.g. that R_1, R_2 do not share any attributes in the annotations that are not in $U_1 \cup U_2$. If this happens, it is easy to rename these attributes such that there is no conflict.

We should note here that we defined the cartesian product instead of a natural join operator for simplicity of presentation: the natural join can be easily defined as a sequence of renaming, cartesian product, selection and projection.

Given a query $Q \in \mathcal{RA}^+$, and an m-multitable $M = (\{R_1, \ldots, R_\ell\}, \Sigma, \mathbf{min}, \mathbf{max})$, let us define by $\bar{Q}(M)$ the tuple $(\{Q(R_1, \ldots, R_\ell)\}, \Sigma, \mathbf{min}, \mathbf{max})$. Here we should note that we have not yet shown that $\bar{Q}(M)$ is a valid m-table; for this, we need to prove that $Q(R_1, \ldots, R_\ell)$ is an m-proper $\hat{K}$-relation. Before we do this, we first give a detailed example of applying the algebraic operations we defined to m-tables.

▶ **Example 11.** We illustrate querying over m-tables through an example. Consider the m-multitable $M = (\{R, S\}, \Sigma, \mathbf{min}, \mathbf{max})$. R is a complete relation (i.e. all annotations are 1), S has missing data, $\Sigma = \{T_1(D), T_2(), T_3(A, C, D)\}$, $\mathbf{min} = (1, 0, 0)$ and $\mathbf{max} = (1, 1, \infty)$. We present tables R and S in Figures 2a and 2b, respectively, with the initial annotations for the extended tuples. Observe that we have appended multiple extended tuples to relation S to represent its missing data. The relational query to be executed on the database is given in Figure 2c and the results obtained after applying the $\hat{K}$-relational algebra operators are presented in Figure 3.

▶ **Lemma 12.** *Let $M = (\{R_1, \ldots, R_\ell\}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-multitable, and Q be a query in $\mathcal{RA}^+$. Denote $R' = Q(R_1, \ldots, R_\ell)$. Then*

1. *R' is an m-proper $\hat{K}$-relation*
2. *$\mathcal{Q}_{R'} = Q(\mathcal{Q}_{R_1}, \ldots, \mathcal{Q}_{R_\ell})$ under both set and bag semantics, i.e., the incomplete database represented by R' and the resulting incomplete database after applying query Q are the same.*

▶ **Corollary 13.** *The $\mathcal{RA}^+$ operations defined for m-multitables map m-multitables to m-multitables, and thus form a well-defined algebra over m-multitables.*

A	C	D	
a_1	3	450	1
a_2	3	m	$\alpha_1(D)$
a_3	3	100	$\alpha_2()$
m	m	m	$\alpha_3(A,C,D)\cdot[D>100]\cdot[C=3]$

(a) Step 1: $S' = \sigma_{C=3}(S)$.

A	D	
a_1	450	1
a_2	m	$\alpha_1(D)$
a_3	100	$\alpha_2()$
m	m	$\alpha_3(A,C,D)\cdot[D>100]\cdot[C=3]$

(b) Step 2: $S'' = \pi_{A,D}(S')$.

A	C	
a_1	50	1
a_2	400	1
a_3	30	1

(c) Step 3: $R' = \pi_{A,C}(R)$.

A'	D	
a_1	450	1
a_2	m	$\alpha_1(D)$
a_3	100	$\alpha_2()$
m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]$

(d) Step 4.1: $S'' = \rho_{\{A\to A',C\to C'\}}(S'')$.

A	C	A'	D	
a_1	50	a_1	450	1
a_2	400	a_2	m	$\alpha_1(D)$
a_3	30	a_3	100	$\alpha_2()$
a_1	50	m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_1]$
a_2	400	m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_2]$
a_3	30	m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_3]$

(e) Steps 4.2, 4.3: Result $= \sigma_{A=A'}(R'\times S'')$.

A	C	D		label	ϕ
a_1	50	450	1	certain	T
a_2	400	m	$\alpha_1(D)$	certain	T
a_3	30	100	$\alpha_2()$	possible	T
a_1	50	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_1]$	possible	$(D>100)\wedge(C'=3)\wedge(A'=a_1)$
a_2	400	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_2]$	possible	$(D>100)\wedge(C'=3)\wedge(A'=a_2)$
a_3	30	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_3]$	possible	$(D>100)\wedge(C'=3)\wedge(A'=a_3)$

(f) Step 4.4: Projection along with application of SimpleLabel algorithm.

Figure 3 Execution of the query plan over m-tables and obtaining m-labeled tuples.

▶ **Theorem 14.** *The m-multitables form a strong representation system for positive relational algebra for both set and bag semantics. Moreover, evaluating a positive relational algebra query on m-multitables has polynomial data complexity.*

We conclude this section by observing that one can apply all the known optimizations on relational algebra plans when querying m-tables, since the standard algebraic identities under bag semantics are preserved (see also Appendix, for more details).

5 The Expressive Power of m-tables

In this section, we discuss the expressive power of m-tables. We first compare the expressiveness of m-tables to c-tables under both the closed and open world assumption. Then, we characterize the set of incomplete databases that can be expressed through m-tables. Our results in this section hold only for the case of set semantics.

m-tables versus c-tables. Our first result shows that the class of $m^{\not=}$-tables in $\mathcal{M}^c$ can capture precisely the expressiveness of c-tables under CWA.

▶ **Theorem 15.** *The c-tables under CWA and the $m^{\not=}$-tables in $\mathcal{M}^c$ have the same expressive power, that is:*
1. *For every c-table C, there exists an $m^{\not=}$-table $M \in \mathcal{M}^c$ such that $\mathsf{Mod}_S(M) = \mathsf{Mod}_C(C)$;*
2. *For every $m^{\not=}$-table $M \in \mathcal{M}^c$, there exists a c-table C such that $\mathsf{Mod}_C(C) = \mathsf{Mod}_S(M)$.*

The detailed proof of this statement is given in the Appendix. As we discussed earlier, m-tables are *strictly* more expressive than c-tables under closed world semantics, since c-tables under CWA cannot express incomplete databases with arbitrarily large instances.

Under the open world assumption, c-tables can express incomplete databases with arbitrarily large instances.

▶ **Proposition 16.** *For every c-table C, there exists an $m^{\not=}$-table M s.t. $\mathsf{Mod}_S(M) = \mathsf{Mod}_O(M)$.*

It turns out that general m-tables are strictly more expressive than c-tables under OWA, in the sense that there exists an m-table M such that $\mathsf{Mod}_S(M)$ is not expressible through a c-table under OWA. Indeed, consider the following example.

▶ **Example 17.** Let $M = (R, \{T_1(A)\}, (0), (\infty))$, where $R(A, B)$ is a binary $\hat{K}$-relation that consists of a single extended tuple $(m, 1)$ with annotation $\alpha_1(A)$. Suppose there exists a c-table C such that $\mathsf{Mod}_S(M) = \mathsf{Mod}_O(C)$. Since the instance $\{(1, 1)\}$ belongs in $\mathsf{Mod}_S(M)$, it must also belong in $\mathsf{Mod}_O(C)$. But then $\{(1, 1), (1, 2)\}$ must also belong in $\mathsf{Mod}_O(C)$; however, this contradicts that C expresses $\mathsf{Mod}_S(M)$, since $(1, 2)$ cannot belong in any instance of $\mathsf{Mod}_S(M)$.

To summarize, m-tables can express a strictly larger class of incomplete databases in comparison to c-tables under both the closed and open world assumption.

Characterizing the Expressiveness. Following [14], we define $\mathcal{N} = \{I \mid I \subseteq \mathbb{D}^U, I \text{ finite}\}$ as the *zero-information* incomplete database. Each subset of $\mathcal{N}$ forms an incomplete database; our goal is to characterize the subsets of $\mathcal{N}$ which are representable by m-tables. We also define $\mathcal{Z}_U = \{\{t\} \mid t \in \mathbb{D}^U\}$ as the incomplete database that represents the set of all relations with exactly one tuple.

▶ **Definition 18** ([14])**.** Let $\mathcal{L}$ be a query language. An incomplete database $\mathcal{I}$ is *$\mathcal{L}$-definable* if there exists a query $Q \in \mathcal{L}$ such that $\mathcal{I} = Q(\mathcal{Z}_U)$. We further say that a representation system is *$\mathcal{L}$-complete* if it can represent any $\mathcal{L}$-definable incomplete database.

We are primarily interested in $\mathcal{RA}$-definable and $\mathcal{RA}^+$-definable incomplete databases. We start by applying a result of [14], which shows that an incomplete database $\mathcal{I}$ is $\mathcal{RA}$-definable if and only if $\mathcal{I}$ is representable by a c-table under CWA. Combining this with Theorem 15, we obtain that $m^{\not=}$-tables in $\mathcal{M}^c$ (as c-tables) capture exactly the incomplete databases that are expressed through an $\mathcal{RA}$ query over $\mathcal{Z}_U$:

▶ **Corollary 19.** *$m^{\not=}$-tables are $\mathcal{RA}$-complete; every $m^{\not=}$-table in $\mathcal{M}^c$ is $\mathcal{RA}$-definable.*

This result characterizes the expressivity of m-tables using $\mathcal{Z}_U$ as the starting point; it turns out that a small fragment of m-tables is enough to capture all of $\mathcal{RA}$ over $\mathcal{Z}_U$. To understand the true expressive power of m-tables, we need to use $\mathcal{N}$ as the starting point.

▶ **Proposition 20.** *For every m-table M, there exists a query $Q \in \mathcal{RA}$ s.t.* $\mathsf{Mod}_S(M) = Q(\mathcal{N})$.

The above proposition tells us that every m-table can be expressed as a relational query over $\mathcal{N}$. For the construction in the proof, which is presented in the Appendix, we need the difference operator to define the appropriate query Q.

▶ **Theorem 21.** *The set of incomplete databases expressed by free m-tables and by $Q(\mathcal{N})$, for $Q \in \mathcal{RA}^+$, are the same, that is:*
1. *For every free m-table M, there exists a query $Q \in \mathcal{RA}^+$ such that* $\mathsf{Mod}_S(M) = Q(\mathcal{N})$.
2. *For every $Q \in \mathcal{RA}^+$, there exists a free m-table M such that* $\mathsf{Mod}_S(M) = Q(\mathcal{N})$.

In other words, free m-tables capture exactly the incomplete databases that can be constructed by computing a positive relational algebra query over $\mathcal{N}$. As for general m-tables, we have shown that they describe a subset of the incomplete databases that can be computed through a relational algebra query over $\mathcal{N}$. It is not clear whether the converse holds, that is, if every incomplete database $\mathcal{I} = Q(\mathcal{N})$, where $Q \in \mathcal{RA}$, can be represented by m-tables. We leave this as part of future work.

6 Labeling Schemes

Interpreting the semantics of an m-table, and in particular the annotation $R(\hat{t})$ for each extended tuple $\hat{t}$ can be non-trivial. Additionally, given an m-table it is not immediately clear whether a tuple is certain or possible in the corresponding incomplete database. To address this issue, we describe a way to interpret an m-table, under set semantics, such that it tells the user which tuples are certain, and which tuples are possible (and under which conditions). We would like to emphasize that the labeling in this section is done only under set semantics.

6.1 Semantics of Labels

We first propose a labeling scheme for missing data. Each tuple $\hat{t}$, following the structure of m-tables, will be an extended tuple that takes values from $\hat{\mathbb{D}} = \mathbb{D} \cup \{m\}$ and every extended tuple will be associated with one of certain and possible labels. Formally, we define:

▶ **Definition 22** (Labeling)**.** An *m-labeled tuple* is a triple of the form $(\hat{t}, \lambda, \phi)$ such that:
- $\hat{t} : U \to \hat{\mathbb{D}}$ is an extended tuple,
- $\lambda \in \{\mathsf{certain}, \ \mathsf{possible}\}$,
- ϕ is a conjunction of expressions $(x \ \mathsf{op} \ y)$, where $x, y \in \mathbf{U} \cup \mathbb{D}$ and $\mathsf{op} \in \{=, <, >, \neq, \geq, \leq\}$.

An *m-labeling scheme* is a finite set of m-labeled tuples.

By viewing the attributes in ϕ as variables, we can view ϕ as a logical formula over $\mathbf{U}$. Given an assignment $v : \mathbf{U} \to \mathbb{D}$ that satisfies ϕ, we obtain an *instantiation* $v \circ \hat{t}$ of the tuple $\hat{t}$, where $v(\hat{t}[A]) = v(A)$ if $A \in \mathsf{m}(\hat{t})$, otherwise $v(\hat{t}[A]) = \hat{t}[A]$. We say that the set of all such instantiations is the *expansion* of $(\hat{t}, \phi)$ and we denote it as $\mathcal{D}(\hat{t}, \phi)$. Note that if ϕ is not satisfiable, then $\mathcal{D}(\hat{t}, \phi) = \emptyset$. Also, if $\mathsf{m}(\hat{t}) = \emptyset$ (so $\hat{t}$ has only constants) and $\phi = \mathsf{T}$ (i.e. the boolean formula ϕ is always true), we simply have $\mathcal{D}(\hat{t}, \phi) = \{\hat{t}\}$.

Let $\mathcal{I}$ be an incomplete database, and let $\mathsf{pos}(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} I$ denote the set of all possible tuples in $\mathcal{I}$. Recall that for $t \in \mathbb{D}^U$, we say that t is certain in $\mathcal{I}$ if for every $I \in \mathcal{I}$ we have $t \in I$; and that t is possible in $\mathcal{I}$ if there exists $I \in \mathcal{I}$ such that $t \in I$. We next generalize the definitions of certainty and possibility for extended tuples. We say that $(\hat{t}, \phi)$ is *certain*

in $\mathcal{I}$ if for every $I \in \mathcal{I}$ we have $\mathcal{D}(\hat{t}, \phi) \cap I \neq \emptyset$. We also say that $(\hat{t}, \phi)$ is *possible* in $\mathcal{I}$ if $\mathcal{D}(\hat{t}, \phi) \subseteq \mathsf{pos}(\mathcal{I})$. Observe that if $\hat{t}$ takes only values from $\mathbb{D}$ and $\phi = \mathsf{T}$, then the above definitions collapse to the standard definitions of possible and certain tuples.

We will focus on m-labeling schemes with the following property: for any $(\hat{t}, \mathsf{certain}, \phi)$, we have that $\phi = \mathsf{T}$. In other words, every extended tuple with label certain has no constraints on its possible values. In this case we simplify the notation of an m-labeled tuple to $(\hat{t}, \mathsf{certain})$ and its expansion to $\mathcal{D}(\hat{t})$: we call this a *simple m-labeling scheme*.

▶ **Definition 23** (Soundness)**.** Let $\mathcal{I}$ be an incomplete database, and $\mathcal{S}$ a simple m-labeling scheme. $\mathcal{S}$ is *c-sound* w.r.t. $\mathcal{I}$ if for every $(\hat{t}, \mathsf{certain}, \phi) \in \mathcal{S}$, $(\hat{t}, \phi)$ is certain in $\mathcal{I}$. $\mathcal{S}$ is *p-sound* w.r.t. $\mathcal{I}$ if for every $(\hat{t}, \mathsf{possible}, \phi) \in \mathcal{S}$, $(\hat{t}, \phi)$ is possible in $\mathcal{I}$.

If $\mathcal{S}$ is both c-sound and p-sound, we simply say that $\mathcal{S}$ is sound. A sound labeling scheme is a conservative under-approximation of an incomplete database.

▶ **Definition 24** (Completeness)**.** Let $\mathcal{I}$ be an incomplete database, and $\mathcal{S}$ a simple m-labeling scheme. $\mathcal{S}$ is *c-complete* w.r.t. $\mathcal{I}$ if for every $(\hat{t}, \mathsf{T})$ that is certain in $\mathcal{I}$, there exists $(\hat{t}', \mathsf{certain}) \in \mathcal{S}$ such that $\mathcal{D}(\hat{t}') \subseteq \mathcal{D}(\hat{t})$. $\mathcal{S}$ is *p-complete* w.r.t. $\mathcal{I}$ if for every $t \in \mathsf{pos}(\mathcal{I})$, there exists $(\hat{t}, \lambda, \phi) \in \mathcal{S}$ such that $t \in \mathcal{D}(\hat{t}, \phi)$.

Analogous to the definition of soundness, a complete labeling scheme is a conservative over-approximation of an incomplete database. A sound and complete labeling scheme captures exactly both the generalized possible and certain tuples.

▶ **Example 25.** Suppose we are given any incomplete database $\mathcal{I}$ for the relation $R(A, B, C)$. Consider the m-labeling scheme $\mathcal{S}$ that consists only of a single tuple: $((m, m, m), \mathsf{possible}, \mathsf{T})$. We first claim that this is a c-sound labeling. This trivially holds, since $\mathcal{S}$ contains no extended tuples with a certain tuple. We also claim that $\mathcal{S}$ is p-complete. Indeed, the expansion of $((m, m, m), \mathsf{T})$ is $\mathbb{D}^{(A,B,C)}$, and thus any tuple in $\mathsf{pos}(\mathcal{I})$ will also belong in $\mathcal{D}((m, m, m), \mathsf{T})$. This construction implies that we can always construct a trivial c-sound and p-complete labeling scheme for any incomplete database.

As we will see shortly, it is computationally hard to construct a sound and complete m-labeling for every incomplete database and $\mathcal{RA}^+$ query. However, we will show that a sound and complete simple m-labeling is possible for a particular case of incomplete databases that are defined through m-tables. We should emphasize here that an m-labeling scheme is not a representation system of $\mathcal{I}$, since we cannot reconstruct $\mathcal{I}$ from $\mathcal{S}$.

6.2 A Simple Label Inference Algorithm

We describe a simple procedure that, given an m-table M, constructs a c-sound and p-complete simple m-labeling scheme for $\mathsf{Mod}_S(M)$. We can use this procedure, together with the completeness of m-tables for $\mathcal{RA}^+$, to construct a c-sound and p-complete simple m-labeling for $Q(\mathcal{I})$ over an incomplete database $\mathcal{I}$ that is represented by an m-table.

The algorithm SimpleLabel is given in Algorithm 1. The intuition for the inference procedure is that, an extended tuple $\hat{t}$ is certain only in the case where there exists a monomial that has no constraints in $\mathcal{E}$, and further, the lower bounds on the cardinalities of α_i's (min_i's) are at least 1 (line 5).

If the tuple is labeled as possible, we compute the formula ϕ in lines 10-13 by taking the conjunction of the expressions in $\mathcal{E}$. Figure 3 shows the resulting labels after applying SimpleLabel on the final result of the running example of Section 4.

Algorithm 1 SimpleLabel.

$\mathcal{S} \leftarrow \emptyset$
for each $\hat{t} \in R$ s.t. $R(\hat{t}) \neq 0$ **do**
  Let $R(\hat{t}) = \sum_{k=1}^{n} \prod_{i_k} \alpha_{i_k}(A_{i_k}) \cdot \prod_{j=1}^{m_k} \theta_j$
  for $k = 1, \ldots, n$ **do**
    if $\exists i_k$ s.t. $min_{i_k} = 0$ or $m_k > 0$ **then**
      $\lambda \leftarrow$ possible
    else
      $\lambda \leftarrow$ certain
    end if
    $\phi \leftarrow \mathsf{T}$
    for all $\theta_j = [x \text{ op } y]$ **do**
      $\phi \leftarrow \phi \wedge (x \text{ op } y)$
    end for
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\hat{t}, \lambda, \phi)\}$
  end for
end for
return $\mathcal{S}$

▶ **Proposition 26.** *Given an m-table M,* SimpleLabel *computes in polynomial time (data complexity) a c-sound and p-complete simple m-labeling scheme $\mathcal{S}$ w.r.t. to* $\mathsf{Mod}_S(M)$.

From the previous proposition, we see that SimpleLabel provides conservative labeling, i.e., if a tuple is marked as certain, then it is definitely so, but all certain tuples may not be identified. Even though SimpleLabel does not produce sound and complete labelings, we can prove several interesting properties if we restrict the expressive power of m-tables.

▶ **Lemma 27.** *Let $M = (\{R\}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-table such that, for every $i = 1, \ldots, N$ we have $max_i = \infty$. Then,* SimpleLabel *produces a p-sound m-labeling scheme.*

This lemma tells us that whenever there is no upper bound on the size of the relations in Σ, we can efficiently construct a p-sound and p-complete labeling scheme, and thus capture exactly the possible tuples.

▶ **Lemma 28.** *Let $M = (\{R\}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-table such that, for every $i = 1, \ldots, N$ we have $min_i = 0$. Then,* SimpleLabel *produces a c-certain m-labeling scheme.*

Lemma 28 is the analogue of Lemma 27: if the size of each relation in Σ is lower bounded by 0, then we obtain a c-certain and c-sound m-labeling and thus compute exactly the certain answers. Combining the two lemmas:

▶ **Theorem 29.** *If $M = (\{R\}, \Sigma)$ is a free m-table, then* SimpleLabel *produces a sound and complete m-labeling scheme w.r.t.* $\mathsf{Mod}_S(M)$.

6.3 Certainty and Possibility in m-tables

▶ **Proposition 30.** *Let M be a free $m^{\not=}$-table. Then, the following tasks can be completed in polynomial time data complexity: (1) tuple certainty, (2) tuple possibility, and (3) tuple q-possibility and q-certainty for $q \in \mathcal{RA}^+$.*

Regarding arbitrary $m^{/=}$-tables, in the Appendix, we show why SimpleLabel fails to produce a p-sound labeling scheme when the upper cardinality constraints are different than ∞, and why it fails to obtain a c-certain labeling when the lower cardinality constraints are different than 0. We conclude with two results on the complexity of tuple certainty and possibility for general $m^{/=}$ tables.

▶ **Proposition 31.** *Tuple possibility in $m^{/=}$-tables is NP-complete (data complexity).*

▶ **Proposition 32.** *Tuple certainty in $m^{/=}$-tables is coNP-complete (data complexity).*

7 Related Work

Incomplete databases have been extensively studied in various contexts; we refer the reader to [1, 25] for a survey and to [22] for a broad perspective on this area.

Numerous models for uncertain information are discussed and compared in [14]. *Conditional tables* (c-tables) [16, 17] are considered one of the most expressive representation system for representing incomplete databases. The $\mathcal{R}^A_{prop}$ model [24] has also been shown to be *closed*, *complete* and as expressive as c-tables. We have provided a detailed comparison of m-tables with c-tables in Section 3 and we have shown that m-tables are strictly more expressive than c-tables (and thus than the $\mathcal{R}^A_{prop}$ as well). In [2, 3], the focus is on providing complexity and decidability results for querying over incomplete databases and we have utilized results from [2] to show complexity results for obtaining certain answers with m-tables.

The use of many-valued logic to handle missing information has been proposed in [6, 9, 26]; this is complementary to our work, and adding multi-valued logic into m-tables should be interesting future work. Our definitions of certain and possible answers are similar to the certain answers defined in [6, 23]; however, the notion of certainty and possibility in our work is defined for extended tuples that represent a set of tuples and not just for single tuples. In this context, our work is also related to the partial results work of [19], where the idea is to execute a given query on an incomplete database in the usual way, and then provide insights into the possible anomalies of the result tuples by labeling the tuples and attribute values with potential errors. They do not focus on developing a formal framework to provide all possible results and they lack a systematic approach to obtain labels for 'partial' results.

Querying over incomplete databases under the open world assumption has been explored in [21] and [8]. In both these pieces of work, the focus is on decidability results on whether *complete* queries can be obtained over possibly incomplete data, with constraints on the missing data. However, our focus is on obtaining a representation system that provides all possible results, while trying to label certain answers.

An extended tuple with all 'm' values is similar to the idea proposed in [10], where they introduce a tuple with all attribute values as 'open' to represent an unknown number of missing tuples. However, their work does not extend beyond the 'open' value; their focus is not on obtaining a representation system or to identify certain answers.

Queries over data integrated (DI) sources have a similar scenario where, frequently, some subset of the information will be uncertain or not available [7, 11, 20, 21]. Solution approaches proposed in this context use schema information of the sources to improve the answers. Our work can be seen as complementary to the work done in this area and our work is applicable in DI scenarios as well.

8 Conclusion

In this paper, we proposed a new representation system called m-tables, which can represent various forms of missing data and generalizes many existing representation systems. We showed that m-tables form a *closed* and *strong* representation system for both set and bag semantics, and are strictly more expressive than c-tables. Further, we propose a simple labeling algorithm that labels tuples as certain or possible by interpreting the annotations of the m-table. One immediate line for future work is to extend m-table semantics to aggregate and group by operations. Another interesting direction is to use the annotations and labeling scheme to "repair" a result when missing data becomes available in the future.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

2 Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, SIGMOD'87, pages 34–48, New York, NY, USA, 1987. ACM. doi:10.1145/38713.38724.

3 Antoine Amarilli and Michael Benedikt. Finite open-world query answering with number restrictions (extended version). *CoRR*, abs/1505.04216, 2015. URL: http://arxiv.org/abs/1505.04216.

4 Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'11, pages 153–164, New York, NY, USA, 2011. ACM. doi:10.1145/1989284.1989302.

5 James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1(4):379–474, April 2009. doi:10.1561/1900000006.

6 Marco Console, Paolo Guagliardo, and Leonid Libkin. Approximations and refinements of certain answers via many-valued logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 349–358, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12813.

7 AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

8 Wenfei Fan and Floris Geerts. Capturing missing tuples and missing values. 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 169–178, 2010. doi:10.1145/1807085.1807109.

9 G. H. Gessert. Four valued logic for relational database systems. *SIGMOD Rec.*, 19(1):29–35, March 1990. doi:10.1145/382274.382401.

10 Georg Gottlob and Roberto Zicari. Closed world databases opened through null values. In *Proceedings of the 14th International Conference on Very Large Data Bases*, VLDB'88, pages 50–61, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc. URL: http://dl.acm.org/citation.cfm?id=645915.671794.

11 G. Grahne. Information integration and incomplete information. In *IEEE Data Eng. Bull.*, pages 46–52, 2002.

12 Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB'07, pages 675–686. VLDB Endowment, 2007. URL: http://dl.acm.org/citation.cfm?id=1325851.1325929.

13 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'07, pages 31–40, New York, NY, USA, 2007. ACM. doi:10.1145/1265530.1265535.

14 Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *Proceedings of the 2006 International Conference on Current Trends in Database Technology*, ICDT'06, pages 278–296, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11896548_24.

15 Paolo Guagliardo and Leonid Libkin. Making sql queries correct on incomplete databases: A feasibility study. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'16, pages 211–223, New York, NY, USA, 2016. ACM. doi:10.1145/2902251.2902297.

16 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984. doi:10.1145/1634.1886.

17 Tomasz Imielinski and Witold Lipski, Jr. On representing incomplete information in a relational data base. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB'81, pages 388–397. VLDB Endowment, 1981. URL: http://dl.acm.org/citation.cfm?id=1286831.1286869.

18 Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD'10, pages 951–962, New York, NY, USA, 2010. ACM. doi:10.1145/1807167.1807269.

19 Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. Partial results in database systems. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD'14, pages 1275–1286, New York, NY, USA, 2014. ACM. doi:10.1145/2588555.2612176.

20 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'02, pages 233–246, New York, NY, USA, 2002. ACM. doi:10.1145/543613.543644.

21 Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB'96, pages 402–412, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. URL: http://dl.acm.org/citation.cfm?id=645922.673332.

22 Leonid Libkin. Incomplete data: What went wrong, and how to fix it. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'14, pages 1–13, New York, NY, USA, 2014. ACM. doi:10.1145/2594538.2594561.

23 Leonid Libkin. Sql's three-valued logic and certain answers. *ACM Trans. Database Syst.*, 41(1):1:1–1:28, March 2016. doi:10.1145/2877206.

24 Anish Das Sarma, Omar Benjelloun, Alon Halevy, and Jennifer Widom. Working models for uncertain data. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE'06, pages 7–, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/ICDE.2006.174.

25 Ron van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publishers, Norwell, MA, USA, 1998. doi:10.1007/978-1-4615-5643-5_10.

26 Kwok-bun Yue. A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Rec.*, 20(3):43–49, September 1991. doi:10.1145/126482.126487.

Better Streaming Algorithms for the Maximum Coverage Problem*

Andrew McGregor[1] and Hoa T. Vu[2]

1 **University of Massachusetts Amherst, Amherst, MA, USA**
mcgregor@cs.umass.edu
2 **University of Massachusetts Amherst, Amherst, MA, USA**
hvu@cs.umass.edu

Abstract

We study the classic NP-Hard problem of finding the maximum k-set coverage in the data stream model: given a set system of m sets that are subsets of a universe $\{1, \ldots, n\}$, find the k sets that cover the most number of distinct elements. The problem can be approximated up to a factor $1 - 1/e$ in polynomial time. In the streaming-set model, the sets and their elements are revealed online. The main goal of our work is to design algorithms, with approximation guarantees as close as possible to $1 - 1/e$, that use sublinear space $o(mn)$. Our main results are:

- Two $(1 - 1/e - \epsilon)$ approximation algorithms: One uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space[1] whereas the other uses only a single pass but $\tilde{O}(\epsilon^{-2}m)$ space.
- We show that any approximation factor better than $(1-(1-1/k)^k)$ in constant passes requires $\Omega(m)$ space for constant k even if the algorithm is allowed unbounded processing time[2]. We also demonstrate a simple *single-pass*, $(1-\epsilon)$ approximation algorithm using $\tilde{O}(\epsilon^{-2}mk)$ space.

We also study the maximum k-vertex coverage problem in the dynamic graph stream model. In this model, the stream consists of edge insertions and deletions of a graph on N vertices. The goal is to find k vertices that cover the most number of distinct edges.

- We show that any constant approximation in constant passes requires $\Omega(N)$ space for constant k whereas $\tilde{O}(\epsilon^{-2}N)$ space is sufficient for a $(1-\epsilon)$ approximation and arbitrary k in a single pass.
- For regular graphs, we show that $\tilde{O}(\epsilon^{-3}k)$ space is sufficient for a $(1-\epsilon)$ approximation in a single pass. We generalize this to a $(\kappa - \epsilon)$ approximation when the ratio between the minimum and maximum degree is bounded below by κ.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases algorithms, data streams, approximation, maximum coverage

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.22

1 Introduction

The *maximum set coverage problem* is a classic NP-Hard problem that has a wide range of applications including facility and sensor allocation [33], information retrieval [6], influence maximization in marketing strategy design [29], and the blog monitoring problem where we want to choose a small number of blogs that cover a wide range of topics [41]. In this

* This work was supported by NSF Awards CCF-0953754, IIS-1251110, CCF-1320719, and a Google Research Award.
[1] $\tilde{O}(\cdot)$ suppresses polylog factors.
[2] Note that $\lim_{k\to\infty}(1-(1-1/k)^k) = 1 - 1/e$.

problem, we are given a set system of m sets that are subsets of a universe $[n] := \{1, \ldots, n\}$. The goal is to find the k sets whose union covers the largest number of distinct elements. For example, in the application considered by Saha and Getoor [41], the universe corresponds to n topics of interest to a reader, each subset corresponds to a blog that covers some of these topics, and the goal is to maximize the number of topics that the reader learns about if she can only choose k blogs.

It is well-known that the greedy algorithm, which greedily picks the set that covers the most number of uncovered elements, is a $1 - 1/e$ approximation. Furthermore, unless $P = NP$, this approximation factor is the best possible [24].

The *maximum vertex coverage problem* is a special case of this problem in which the universe corresponds to the edges of a given graph and there is a set corresponding to each node of the graph that contains the subset of edges that are incident to that node. For this problem, algorithms based on linear programming are known to achieve a 3/4 approximation for general graphs [1] and a 8/9 approximation for bipartite graphs [15]. Assuming $P \neq NP$, there does not exist a polynomial-time approximation scheme. Recent work has focused on finding purely combinatorial algorithms for this problem [14].

Streaming Algorithms. Unfortunately, for both problems, the aforementioned greedy and linear programming algorithms do not scale well to massive data sets. This has motivated a significant research effort in designing algorithms that could handle large data in modern computation models such as the data stream model and the MapReduce model [34, 10]. In the data stream model, the k-set coverage problem and the related set cover problem have received a lot of attention in recent research [26, 18, 9, 45, 23, 7].

Two variants of the data stream model are relevant to our work. In the *streaming-set model* [41, 25, 23, 40, 44, 31], the stream consists of m sets $S_1, \ldots, S_m$ and each S_i is encoded as the list of elements in that set along with a unique ID for the set. For simplicity, we assume that $\mathrm{ID}(S_i) = i$. In the dynamic graph stream model [2, 3, 4, 5, 27, 28, 25, 13, 20, 8, 32, 37, 36], relevant to the maximum vertex coverage problem, the stream consists of insertions and deletions of edges of the underlying graph. For a recent survey of research in graph streaming, see [35]. Note that any algorithm for the dynamic graph stream model can also be used in the streaming-set model; the streaming-set model is simply a special case in which there are no deletions and edges are grouped by endpoint.

1.1 Related Work

Maximum Set Coverage. Saha and Getoor [41] gave a swap based 1/4 approximation algorithm that uses a single pass and $\tilde{O}(kn)$ space. At any point, their algorithm stores k sets explicitly in the memory as the current solution. When a new set arrives, based on a specific rule, their algorithm either swaps it with the set with the least contribution in the current solution or does nothing and moves on to the next set in the stream. Subsequently, Ausiello et al. [9] gave a slightly different swap based algorithm that also finds a 1/4 approximation using one pass and the same space. Yu and Yuan [45] claimed an $\tilde{O}(n)$ space, single-pass algorithm with an approximation factor around 0.3 based on the aid of computer simulation.

Recently, Badanidiyuru et al. [10] gave a generic single-pass algorithm for maximizing a monotone submodular function on the stream's objects subject to the cardinality constraint that at most k objects are selected. Their algorithm guarantees a $1/2 - \epsilon$ approximation. At a high level, based on a rule that is different from [41, 9] and a guess of the optimal value, their algorithm decides if the next object (which is a set in our case) is added to the current solution. The algorithm stops when it reaches the end of the stream or when k

objects have been added to the solution. In the k-set coverage problem, the rule requires knowing the coverage of the current solution. As a result, a careful adaptation to the k-set coverage problem uses $\tilde{O}(\epsilon^{-1}n)$ space. For constant ϵ, this result directly improves upon [41, 9]. Subsequently, Chekuri et al. [19] extended this work to non-monotone submodular function maximization under constraints beyond cardinality.

The set cover problem, which is closely related to the k-set coverage problem, has been studied in [41, 26, 18, 23, 7]. See [7] for a comprehensive summary of results and discussion.

Maximum Vertex Coverage. The streaming k-vertex coverage problem was studied by Ausiello et al. [9]. They first observed that simply outputting the k vertices with highest degrees is a $1/2$ approximation; this can easily be done in the streaming-set model. The main results of their work were $\tilde{O}(kN)$-space algorithms that have better approximation for special types of graph. Their results include a 0.55 approximation for regular graphs and a 0.6075 approximation for regular bipartite graphs. Note that their paper only considered the streaming-set model whereas our results for maximum vertex coverage will consider the more challenging dynamic graph stream model.

1.2 Our Contributions

Maximum k-set coverage. Our main goal is to achieve the $1 - 1/e$ approximation that is possible in the non-streaming or offline setting.

- We present polynomial time data stream algorithms that achieve a $1 - 1/e - \epsilon$ approximation for arbitrarily small ϵ. The first algorithm uses one pass and $\tilde{O}(\epsilon^{-2}m)$ space whereas the second algorithm uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space. We consider both algorithms to be pass efficient but the second algorithm uses much less space at the cost of using more than one pass. We note that storing the solution itself requires $\Omega(k)$ space. Thus, we consider $\tilde{O}(\epsilon^{-2}k)$ space to be surprisingly space efficient.
- For constant k, we show that $\Omega(m)$ space is required by any constant pass (randomized) algorithm to achieve an approximation factor better than $(1-(1-1/k)^k)$ with probability at least 0.99; this holds even if the algorithm is permitted exponential time. To the best of our knowledge, this is the first non-trivial space lower bound for this problem. However, with exponential time and $\tilde{O}(\epsilon^{-2}mk)$ space we observe that a $1 - \epsilon$ approximation is possible in a single pass.

For a slightly worse approximation, a $1/2 - \epsilon$ approximation in one pass can be achieved using $\tilde{O}(\epsilon^{-3}k)$ space. This follows by building on the result of Badanidiyuru et al. [10]. However, we provide a simpler algorithm and analysis. Finally, we design a $1/3 - \epsilon$ approximation algorithm for the budgeted maximum set coverage problem using one pass and $\tilde{O}(n)$ space. In this version, each set S has a cost w_S in the interval $[0, L]$. The goal is to find a collection of sets whose total cost does not exceed L that cover the most number of distinct elements. Khuller et al. [30] presented a polynomial time and $1-1/e$ approximation algorithm based on the greedy algorithm and an enumeration technique. Our results are summarized in Figure 1.

Shortly after our submission, in an independent work, Bateni et al. [12] presented a single-pass, $\tilde{O}(\epsilon^{-3}m)$ space algorithm that finds a $1-1/e-\epsilon$ approximation for the maximum k-set coverage problem. We note that our approach also works in their *edge arrival* model in which the stream reveals the set-element relationships one at a time.

Maximum k-vertex coverage. Compared to the most relevant previous work [9], we study this problem in a more general model, i.e., the dynamic graph stream model. We manage

Upper/Lower bound	Number of passes	Space	Approximation	Constraint
U	$O\left(\epsilon^{-1}\right)$	$\tilde{O}(\epsilon^{-2}k)$	$1-1/e-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-3}k)$	$1/2-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-2}m)$	$1-1/e-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-2}mk)$	$1-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-1}n)$	$1/3-\epsilon$	B
L	constant	$\Omega(mk^{-2})$	$(1-(1-1/k)^k)+\epsilon$	C

Figure 1 Summary of results for MaxSetCoverage, C: cardinality, B: budgeted.

Upper/Lower bound	Number of passes	Space	Approximation
U	1	$\tilde{O}(\epsilon^{-2}N)$	$1-\epsilon$
U	1	$\tilde{O}(\epsilon^{-3}k)$	$\kappa-\epsilon$
L	1	$\Omega(N\kappa^3/k)$	$\kappa+\epsilon$

Figure 2 Summary of results for MaxVertexCoverage. κ is ratio of lowest degree to highest degree.

to achieve a better approximation and space complexity for general graphs even when comparing to their results for special types of graph. Our results are summarized in Figure 2. In particular, we show that

- $\tilde{O}(\epsilon^{-2}N)$ space is sufficient for a $1-\epsilon$ approximation (or a $3/4-\epsilon$ approximation if restricted to polynomial time) and arbitrary k in a single pass. The algorithms in [9] use $\tilde{O}(kN)$ space and achieve an approximation worse than 0.61 even for special graphs.
- Any constant approximation in constant passes requires $\Omega(N)$ space for constant k.
- For regular graphs, we show that $\tilde{O}(\epsilon^{-3}k)$ space is sufficient for $1-\epsilon$ approximation in a single pass. We generalize this to an $\kappa-\epsilon$ approximation when the ratio between the minimum and maximum degree is bounded below by κ. We also extend this result to hypergraphs.

Our techniques. On the algorithmic side, our basic approach is a "guess, subsample, and verify" framework. At a high level, suppose we design a streaming algorithm for approximate k-coverage that assumes a priori knowledge of a good guess of the optimal coverage. We show that it is a) possible to run same algorithm on a subsampled universe defined by a carefully chosen hash function and b) remove the assumption that a good guess was already known.

If the guess is at least nearly correct, running the algorithm on the subsampled universe results in a small space complexity. However, there are two main challenges. First, an algorithm instance with a wrong guess could use too much space. We simply terminate those instances. The second issue is more subtle. Because the hash function is not fully independent, we appeal to a special version Chernoff bound. The bound needs not guarantee a good approximation unless the guess is near-correct. To this end, we use the F_0 estimation algorithm to verify the coverage of the solutions. Finally, we return the solution with maximum estimate coverage. This framework allows us to restrict the analysis solely to the near-correct guess. The analysis is, therefore, significantly simpler.

Some of our other algorithmic ideas are inspired by previous works. The "thresholding greedy" technique was inspired by [18, 22, 11]. However, the analysis is different for our problem. Furthermore, to optimize the number of passes, we rely on new observations.

Another algorithmic idea in designing one-pass space-efficient algorithm is to treat the sets differently based on their contributions. During the stream, we immediately add the sets

with large contributions to the solution. We store the contribution of each remaining sets explicitly and solve the remaining problem offline. Har-Peled et al. [26] devised a somewhat similar strategy but the details are different.

For the k-vertex coverage problem, we show that simply running the streaming cut-sparsifier algorithm is sufficient and optimal up to a polylog factor. The novelty is to treat it as an interesting corner case of a more space-efficient algorithm for near regular graphs, i.e., κ is bounded below.

One of the novelties is proving the lower bound via a randomized reduction from the k-party set disjointness problem.

2 Algorithms for maximum k-set coverage

In this section, we design various algorithms for approximating MaxSetCoverage in the data stream model. Our main algorithmic results in this section are two $1 - 1/e - \epsilon$ approximation algorithms. The first algorithm uses one pass and $\tilde{O}(\epsilon^{-2}m)$ space whereas the second algorithm uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space. We also briefly explore some other trade-offs in a subsequent subsection.

Notation. If $\mathcal{A}$ is a collection of sets, then $\mathcal{C}(\mathcal{A})$ denotes the union of these sets.

2.1 $(1 - 1/e - \epsilon)$ approximation in one pass and $\tilde{O}(\epsilon^{-2}m)$ space

Approach. The algorithm adds sets to the current solution if the number of new elements they cover exceeds some threshold. The basic algorithm relies on an estimate z of the optimum coverage OPT. The threshold for including a new set in the solution is that it covers at least z/k new elements. Unfortunately, this threshold is too high to ensure that we selected sets that achieve the required $1 - 1/e - \epsilon$ approximation and we may want to revisit adding a set, say S, that was not added when it first arrived. To facilitate this, we will explicitly store the subset of S that were uncovered when S arrived in a collection of sets $\mathcal{W}$. By the fact that S was not added immediately, we know that this subset is not too large. At the end of the pass, we continue augmenting out current solutions using the collection $\mathcal{W}$.

Technical Details. For the time being, we suppose that the algorithm is provided with an estimate z such that $\text{OPT} \leq z \leq 4\,\text{OPT}$. The algorithm uses C to keep track of the elements that have been covered so far. Upon seeing a new set S, the algorithm stores $S \setminus C$ explicitly in $\mathcal{W}$ if S covers few new elements. Otherwise, the algorithm adds S to the solution and updates C immediately. At the end of the stream, if there are fewer than k sets in the solution, we use the greedy approach to find the remaining sets from $\mathcal{W}$.

The basic algorithm maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. We also maintain a collection of sets $\mathcal{W}$ described above. The algorithm proceeds as follows:

1. Initialize $C = \emptyset$, $I = \emptyset$, $\mathcal{W} = \emptyset$.
2. For each set S in the stream:
   a. If $|S \setminus C| < z/k$ then $\mathcal{W} \leftarrow \mathcal{W} \cup \{S \setminus C\}$.
   b. If $|S \setminus C| \geq z/k$ then $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.
3. Post-processing: Greedily add $k - |I|$ sets from $\mathcal{W}$ and update I and C appropriately.

▶ **Lemma 1.** *There exists a single-pass, $O\left(k \log m + mz/k \cdot \log n\right)$-space algorithm that finds a $1 - 1/e$ approximation of* MaxSetCoverage.

Proof. We observe that storing the set of covered elements C requires at most $\mathrm{OPT} \log n = O(z \log n)$ bits of space. For each set S such that $S \setminus C$ is stored explicitly in $\mathcal{W}$, we need $O\left(z/k \cdot \log n\right)$ bits of space. Storing I requires $O(k \log m)$ space. Thus, the algorithm uses the space claimed since $k < m$.

After the algorithm added the ith set S to the solution, let a_i be the number of new elements that S covers and b_i be the total number of covered elements so far. Furthermore, for $i > 0$, let $c_i = \mathrm{OPT} - b_i$. Define $a_0 := b_0 := 0$ and $c_0 := \mathrm{OPT}$. At the end of the stream, suppose $|I| = j$. Then, $c_j \leq \mathrm{OPT} - zj/k \leq \mathrm{OPT}(1 - 1/k)^j$. Now, we consider the sets that were added in post-processing. We then proceed with the usual inductive argument to show that $c_i \leq (1 - 1/k)^i \,\mathrm{OPT}$ for $i > j$. Before the algorithm added the $(i+1)$th set for $i \geq j$, there must be a set that covers at least c_i/k new elements. Therefore, $c_{i+1} = c_i - a_{i+1} \leq c_i(1 - 1/k) \leq \mathrm{OPT}(1 - 1/k)^{i+1}$. The approximation follows since $c_k \leq \mathrm{OPT}(1 - 1/k)^k \leq 1/e \cdot \mathrm{OPT}$. ◀

Following the approach outlined in Section 2.3 we may assume $z = O(\epsilon^{-2} k \log m)$ and that $\mathrm{OPT} \leq z \leq 4\,\mathrm{OPT}$.

▶ **Theorem 2.** *There exists a single-pass, $\tilde{O}(\epsilon^{-2}m)$ space algorithm that finds a $1 - 1/e - \epsilon$ approximation of* MaxSetCoverage *with high probability.*

Remark. After the initial submission of this paper, we observed that a slight modification of the above algorithm can be used to attain a $1 - 1/(4b)$ approximation for any $b > 1$ if we are permitted unlimited post-processing time and increase the space by a factor of b. Specifically, we increase the threshold for when to add a set immediately to the solution from z/k to bz/k and then find the optimal set of $k - |I|$ sets from $\mathcal{W}$ to add in post-processing. For example, setting $b = 4\epsilon^{-1}$ yields a $1 - \epsilon$ approximation using $\tilde{O}(\epsilon^{-3}m)$ space. See the full version for further details [38].

2.2 $(1 - 1/e - \epsilon)$ approximation in $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space

Approach. Our second algorithm is based on the standard greedy approach but instead of adding the set that increases the coverage of the current solution the most at each set, we add a set if the number of new elements covered by this set exceeds a certain threshold. This threshold decreases with each pass in such a way that after only $O(\epsilon^{-1})$ passes, we have a good approximate solution but the resulting algorithm may use too much space. We will fix this by first randomly subsampling each set at different rates and running multiple instantiations of the basic algorithm corresponding to different rates of subsampling.

The basic "decreasing threshold" approach has been used before in different contexts [11, 18, 22]. The novelty of our approach is in implementing this approach such that the resulting algorithm uses small space and a small number of passes. For example, a direct implementation of the approach by Badanidiyuru and Vondrák [11] in the streaming model may require $O(\epsilon^{-1} \log(m/\epsilon))$ passes and $O(n)$ space[3].

Technical Details. We will assume that we are given an estimate z of OPT such that $\mathrm{OPT} \leq z \leq 4\,\mathrm{OPT}$. We will later remove this assumption. We start by designing a $(1 - 1/e - \epsilon)$ approximation algorithm that uses $\tilde{O}(k + z)$ space and $O(\epsilon^{-1})$ passes. We will subsequently use a sampling approach to reduce the space to $\tilde{O}(\epsilon^{-2}k)$.

[3] Note that their work addressed the more general problem of maximizing sub-modular functions.

As with the previous algorithm, the basic algorithm in this section also maintains maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. The algorithm proceeds as follows:

1. Initialize $C = \emptyset$ and $I = \emptyset$
2. For $j = 1$ to $1 + \lceil \log_\alpha(4e) \rceil$ where $\alpha = 1 + \epsilon$:
   a. Make a pass over the stream. For each set S in the stream:
      i. If $|I| < k$ and $|S \setminus C| \geq \frac{z/k}{(1+\epsilon)^{j-1}}$ then $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

▶ **Lemma 3.** *There exists an $O(\epsilon^{-1})$-pass, $O(k \log m + z \log n)$-space algorithm that finds a $1 - 1/e - \epsilon$ approximation of* MaxSetCoverage.

To analyze the algorithm, we introduce some notation. After the ith set was picked, let a_i be the number of new elements covered by this set and let b_i be the total number of covered elements so far. Furthermore, let $c_i = \text{OPT} - b_i$. We define $a_0 := 0$ and $b_0 := 0$.

▶ **Lemma 4.** *Suppose the algorithm picks k' sets. For $0 \leq i \leq k' - 1$, $a_{i+1} \geq c_i/(\alpha k)$.*

Proof. Suppose the algorithm added the $(i+1)$th set S during the jth pass. Consider the set of covered elements C just before the algorithm added the set S.

We first consider the case where $j = 1$. Then, the algorithm only adds S if

$$|S \setminus C| \geq z/k \geq \text{OPT}/k \geq c_i/k \geq c_i/(\alpha k) \ .$$

Now, we consider the case where $j > 1$. Note that just before the algorithm added S, there must exist a set S' (which could be S) that had not been already added where $|S' \setminus C| \geq c_i/k$. This follows because the optimum collection of k sets covers at least c_i elements that are currently uncovered and hence one of these sets must cover at least c_i/k new elements. But since S' had not already been added, we know that S' was not added during the first $j - 1$ passes and thus, $|S' \setminus C| < z/(k\alpha^{j-2})$. Therefore,

$$z/(k\alpha^{j-2}) > |S' \setminus C| \geq c_i/k$$

and in particular, $z/(k\alpha^{j-1}) > c_i/(k\alpha)$. Since the algorithm picked S, we have $a_{i+1} = |S \setminus C| \geq z/(k\alpha^{j-1}) \geq c_i/(k\alpha)$ as required. ◀

Proof of Lemma 3. It is immediate that the number of passes is $O(\epsilon^{-1})$. The algorithm needs to store the sets I and C. Since $|C| \leq z$, the total space is $O(k \log m + z \log n)$.

To argue about the approximation factor, we first prove by induction that we always have $c_i \leq \left(1 - \frac{1}{\alpha k}\right)^i \text{OPT}$ for $i \leq k'$. Trivially, $c_0 \leq (1 - \frac{1}{\alpha k})^0 \,\text{OPT}$. Suppose $c_i \leq (1 - \frac{1}{\alpha k})^i \,\text{OPT}$. Then, according to Lemma 4, $a_{i+1} \geq c_i/(\alpha k)$. Thus,

$$c_{i+1} = c_i - a_{i+1} \leq c_i - \frac{c_i}{\alpha k} = c_i \left(1 - \frac{1}{\alpha k}\right) \leq \text{OPT} \left(1 - \frac{1}{\alpha k}\right)^{i+1} .$$

Suppose the final solution contains k sets. Then

$$c_k \leq \left(1 - \frac{1}{\alpha k}\right)^k \text{OPT} \leq e^{-1/\alpha} \,\text{OPT} \leq (1/e + \epsilon) \,\text{OPT} \,.$$

As a result, the final solution covers $b_k = \text{OPT} - c_k \geq (1 - 1/e - \epsilon) \,\text{OPT}$.

Suppose the collection of sets $\mathcal{S}$ chosen by the algorithm contains fewer than k sets. We define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$ to be the set of elements in S that are not covered by the final solution. For each set S in the optimum solution $\mathcal{O}$, if S is unpicked, then $|\tilde{S}| \leq z/(4ek)$. Therefore,

$$\mathrm{OPT} = \left|\bigcup_{S\in\mathcal{O}\cap\mathcal{S}} S\right| + \left|\bigcup_{S\in\mathcal{O}\setminus\mathcal{S}} \tilde{S}\right| \leq |\mathcal{C}(\mathcal{S})| + \sum_{S\in\mathcal{O}\setminus\mathcal{S}} \left|\tilde{S}\right| \leq |\mathcal{C}(\mathcal{S})| + \frac{z}{4e} \leq |\mathcal{C}(\mathcal{S})| + \frac{\mathrm{OPT}}{e} .$$

Hence, $|\mathcal{C}(\mathcal{S})| \geq (1 - 1/e)\,\mathrm{OPT}$. ◀

Following the approach outlined in Section 2.3 we may assume $z = O(\epsilon^{-2}k \log m)$ and that $\mathrm{OPT} \leq z \leq 4\,\mathrm{OPT}$.

▶ **Theorem 5.** *There exists an $O(1/\epsilon)$-pass, $\tilde{O}(\epsilon^{-2}k)$ space algorithm that finds a $1 - 1/e - \epsilon$ approximation of* MaxSetCoverage *with high probability.*

2.3 Removing Assumptions via Guessing, Sampling, and Sketching

In this section, we address the fact that in the previous two sections we assumed a priori knowledge of a constant approximation of the maximum number of elements that could be covered and that this optimum was of size $O(\epsilon^{-2}k \log m)$.

Addressing both issues are interrelated and are based on a subsampling approach. The basic idea is to run the above algorithms on a new instance formed by removing occurrences of certain elements in $[n]$ from all the input sets. The goal is to reduce the maximum coverage to $\min(n, O(\epsilon^{-2}k \log m))$ while ensuring that a good approximation in the subsampled instance corresponds to a good approximation in the original instance. In the rest of this section we will assume that $k = o(\epsilon^2 n/\log m)$ since otherwise this bound is trivial.

Subsampling. Assume we know a value v that satisfies $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$. Let c be some sufficiently large constant and set $\lambda = c\epsilon^{-2}k \log m$. Let $h : [n] \to \{0, 1\}$ be drawn from a family of 2λ-wise independent hash functions where

$$p := \Pr[h(e) = 1] = \lambda/v.$$

The space to store h is $\tilde{O}(\epsilon^{-2}k)$. For any set S that is a subset of $[n]$, we define

$$S' := \{e \in S : h(e) = 1\}.$$

We use the following Chernoff bound for limited independent random variables.

▶ **Theorem 6** (Schmidt et al. [42])**.** *Let $X_1, \ldots, X_n$ be boolean random variables. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Suppose $\mu \leq n/2$. If X_i are $\lceil \gamma\mu \rceil$-wise independent, then*

$$\Pr[|X - \mu| \geq \gamma\mu] \leq \exp\left(-\lfloor \min(\gamma, \gamma^2) \cdot \mu/3 \rfloor\right) .$$

The next lemma and its corollary will allow us to argue that approximating the maximum coverage amongst the elements $\{e \in [n] : h(e) = 1\}$ gives only a slightly weaker approximation of the maximum coverage amongst the original set of elements.

▶ **Lemma 7.** *With high probability*[4]*, for all collections of k sets $S_1, \ldots, S_k$ in the stream, $|S'_1 \cup \ldots \cup S'_k| = |S_1 \cup \ldots \cup S_k| p \pm \epsilon v p$.*

[4] We consider $1 - 1/\operatorname{poly}(m)$ or $1 - 1/\operatorname{poly}(n)$ as high probability.

Proof. Fix any collection of k sets $S_1, \ldots, S_k$. Let $D = |S_1 \cup \ldots \cup S_k|$ and $D' = |S'_1 \cup \ldots \cup S'_k|$. We first observe that since $k = o(\epsilon^2 n/\log m)$, we may assume that $\lambda = o(n)$.

$$\mu := \mathrm{E}[D'] = pD \leq p\,\mathrm{OPT} < 2pv = 2\lambda \leq n/2.$$

Appealing to the Chernoff bound with limited independence Theorem 6, we have

$$\Pr[|D' - \mu| \geq \epsilon vp] = \Pr[|D' - \mu| \geq \gamma Dp] \leq \exp\left(-\lfloor \min(\gamma, \gamma^2) \cdot \mu/3 \rfloor\right)$$

where $\gamma = \epsilon v/D$ since the hash function was $\lceil \gamma\mu \rceil = \lceil \epsilon vp \rceil$-wise independent. But note that

$$\exp\left(-\lfloor \min(\gamma, \gamma^2) \cdot \frac{\mu}{3} \rfloor\right) = \exp\left(-\lfloor \min(1, \gamma) \cdot \frac{\epsilon vp}{3} \rfloor\right) \leq \exp\left(-\lfloor \frac{1}{2} \cdot \frac{ck \log m}{3} \rfloor\right) \leq \frac{1}{m^{10k}}$$

where we use the fact that $\gamma = \epsilon v/D \geq \epsilon/2$ because $D \leq \mathrm{OPT} \leq 2v$. The lemma follows by taking the union bound over all $\binom{m}{k}$ collections of k sets. ◀

In particular, the following corollary establishes that a $1/t$ approximation when restricted to elements in $\{e \in [n] : h(e) = 1\}$ yields a $(1/t - 2\epsilon)$ approximation and at most $p\,\mathrm{OPT}(1 + \epsilon) = O(\epsilon^{-2} k \log m)$ of these elements can be covered by k sets.

▶ **Corollary 8.** *Let* OPT' *be optimum number of elements that can be covered from* $\{e \in [n] : h(e) = 1\}$. *Then,*

$$p\,\mathrm{OPT}(1 + \epsilon) \geq \mathrm{OPT}' \geq p\,\mathrm{OPT}(1 - \epsilon)$$

Furthermore if $U_1, \ldots, U_k$ *satisfies* $|U'_1 \cup \ldots \cup U'_k| \geq p\,\mathrm{OPT}(1 - \epsilon)/t$ *for* $t \geq 1$ *then*

$$|U_1 \cup \ldots \cup U_k| \geq \mathrm{OPT}(1/t - 2\epsilon)\ .$$

Proof. The fact that $\mathrm{OPT}' \geq p\,\mathrm{OPT}(1 - \epsilon)$ follows by applying Lemma 7 to the optimum solution. According to Lemma 7, for all collections of k sets $U_1, \ldots, U_k$, we have

$$|U'_1 \cup \ldots \cup U'_k| = |U_1 \cup \ldots \cup U_k|p \pm \epsilon vp \leq p\,\mathrm{OPT}(1 + \epsilon)$$

which implies the first inequality.

Now, suppose $|U'_1 \cup \ldots \cup U'_k| \geq p\,\mathrm{OPT}(1-\epsilon)/t$. Since $|U'_1 \cup \ldots \cup U'_k| - \epsilon vp \leq |U_1 \cup \ldots \cup U_k|p$, we deduce that $|U_1 \cup \ldots \cup U_k| \geq \mathrm{OPT}(1 - \epsilon)/t - \epsilon v \geq \mathrm{OPT}(1/t - 2\epsilon)$. ◀

Hence, since we know v such that $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$, then we know that

$$(1 - \epsilon)\lambda \leq \mathrm{OPT}' \leq 2(1 + \epsilon)\lambda \tag{1}$$

with high probability according to Corollary 8. Then, by setting $z = 2(1 + \epsilon)\lambda$, we ensure that $\mathrm{OPT}' \leq z \leq 4\,\mathrm{OPT}'$.

Guessing v and F_0 Sketching. We still need to address how to compute v such that $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$. The natural approach is to make $\lceil \log_2 n \rceil$ guesses for v corresponding to $1, 2, 4, 8 \ldots$ since one of these will be correct.[5] We then perform multiple parallel instantiations of the algorithm corresponding to each guess. This increases the space by a factor of $O(\log n)$.

[5] The number of guesses can be reduced to $\lceil \log_2 k \rceil$ if the size of the largest set is known since this gives a k approximation of OPT. The size of the large set can be computed in one additional pass if necessary.

But how do we determine which instantiation corresponds to the correct guess? The most expedient way to deal with this question is to sidestep the issue as follows. Instantiations corresponding to guesses that are too small may find it is possible to cover $\omega(\epsilon^{-2}k\log m)$ elements so we will terminate any instantiation as soon as it covers more than $O(\epsilon^{-2}k\log m)$ elements. Note that by Corollary 8 and Equation 1, we will not terminate the instantiation corresponding to the correct guess.

Among the instantiations that are not terminated we simply return the best solution. To find the best solution we want to estimate $|\cup_{i\in I} S_i|$, i.e., the coverage of the corresponding sets *before* the subsampling. To compute this estimate in small space we can use the *F_0-sketching* technique. For the purposes of our application, we can summarize the required result as follows:

▶ **Theorem 9** (Cormode et al. [21]). *There exists an $\tilde{O}(\epsilon^{-2}\log\delta^{-1})$-space algorithm that, given a set $S\subseteq[n]$, can construct a data structure $\mathcal{M}(S)$, called an F_0* sketch *of S, that has the property that the number of distinct elements in a collection of sets $S_1, S_2, \ldots, S_r$ can be approximated up to a $1+\epsilon$ factor with probability at least $1-\delta$ given the collection of F_0 sketches $\mathcal{M}(S_1), \mathcal{M}(S_2), \ldots, \mathcal{M}(S_r)$.*

For the algorithms in the previous section we can maintain a sketch $\mathcal{M}(C)$ of the set of covered elements in $\tilde{O}(\epsilon^{-2}\log\delta^{-1})$ space and from this can estimate the desired coverage. We set $\delta \leftarrow \Theta(1/n\cdot\log n)$ so that coverages of all non-terminated instances are estimated up to a factor $(1+\epsilon)$ with high probability.

2.4 Other Algorithmic Results

In this final subsection, we briefly review some other algorithmic results for MaxSetCoverage, either with different trade-offs or for a "budgeted" version of the problem.

2.4.1 $(1-\epsilon)$ approximation in one pass and $\tilde{O}(\epsilon^{-2}m)$ space

In the previous subsection, we gave a single-pass $1-1/e-\epsilon$ approximation using $\tilde{O}(\epsilon^{-2}m)$ space. Here we observe that if we are permitted $\tilde{O}(\epsilon^{-2}mk)$ space and unlimited post-processing time then a $1-\epsilon$ approximation can be achieved directly from the F_0 sketches.

Specifically, in one pass we construct the F_0 sketches of all m sets, $\mathcal{M}(S_1), \ldots, \mathcal{M}(S_m)$ where the failure probability of the sketches is set to $\delta = 1/(nm^k)$. Thus, at the end of the stream, one can $1+\epsilon$ approximate the coverage $|S_{i_1}\cup\ldots\cup S_{i_k}|$ for each collection of k sets $S_{i_1}, \ldots, S_{i_k}$ with probability at least $1-1/(nm^k)$. Since there are at most $\binom{m}{k} \le m^k$ collections of k sets, appealing to the union bound, we could guarantee that the coverages of all of the collections of k sets are preserved up to a $1+\epsilon$ factor with probability at least $1-1/n$. The space to store the sketches is $\tilde{O}(\epsilon^{-2}mk)$.

▶ **Theorem 10.** *There exists a single-pass, $\tilde{O}(\epsilon^{-2}mk)$-space algorithm that finds a $1-\epsilon$ approximation of* MaxSetCoverage *with high probability .*

2.4.2 $(1/2-\epsilon)$ approximation in one pass and $\tilde{O}(\epsilon^{-3}k)$ space

We next observe that it is possible to achieve a $1/2-\epsilon$ approximation using a single pass and $\tilde{O}(\epsilon^{-3}k)$ space. Consider the following simple single-pass algorithm that uses an estimate z of OPT such that $\text{OPT}\le z\le(1+\epsilon)\,\text{OPT}$. As with previous algorithms, the basic algorithm in this section also maintains $I\subseteq[m]$, $C\subseteq[n]$ where I corresponds to the ID's of the (at

most k) sets in the current solution and C is the the union of the corresponding sets. The algorithm proceeds as follows:

1. Initialize $C = \emptyset$ and $I = \emptyset$.
2. For each set S in the stream:
   a. If $|S \setminus C| \geq z/(2k)$ and $|I| < k$ then $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

The described algorithm is a $1/2 - \epsilon$ approximation. To see this, if the solution consists of k sets, then the final solution obviously covers at least $z/2 \geq \text{OPT}/2$ elements. Now we consider the case in which the collection of sets $\mathcal{S}$ chosen by the algorithm contains fewer than k sets. We define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$ to be the set of elements in S that are not covered by the final solution. For each set S in the optimum solution $\mathcal{O}$, if S is unpicked, then $|\tilde{S}| \leq z/(2k)$. Therefore,

$$\text{OPT} = \left|\bigcup_{S\in\mathcal{O}\cap\mathcal{S}} S\right| + \left|\bigcup_{S\in\mathcal{O}\setminus\mathcal{S}} \tilde{S}\right| \leq |\mathcal{C}(\mathcal{S})| + \sum_{S\in\mathcal{O}\setminus\mathcal{S}} |\tilde{S}| \leq |\mathcal{C}(\mathcal{S})| + \frac{z}{2}$$
$$\leq |\mathcal{C}(\mathcal{S})| + \frac{\text{OPT}(1+\epsilon)}{2} .$$

and thus $|\mathcal{C}(\mathcal{S})| \geq \frac{1-\epsilon}{2}\,\text{OPT}$.

We note that the above algorithm uses $O(k \log m + z \log n)$ space but we can use an argument similar to that used in Section 2.3 to reduce this to $\tilde{O}(\epsilon^{-3}k)$. The only difference is since we need z such that $\text{OPT}' \leq z \leq (1+\epsilon)\,\text{OPT}'$ we will guess v in powers of $1 + \epsilon/4$ and set $\lambda = 16c\epsilon^{-2}k \log m$. Then Equation 1, becomes $(1-\epsilon/4)\lambda \leq \text{OPT}' \leq (1+\epsilon/4)^2\lambda$ and hence $z = (1+\epsilon/4)^2\lambda$ is a sufficiently good estimate.

▶ **Theorem 11.** *There exists a single-pass, $\tilde{O}(\epsilon^{-3}k)$ space algorithm that finds a $1/2 - \epsilon$ approximation of* MaxSetCoverage *with high probability.*

2.4.3 Budgeted Maximum Coverage

In this variation, each set S has a cost $w_S \in [0, L]$. The problem asks to find the collection of sets whose total cost is at most L that covers the most number of distinct elements. For $I \subseteq [n]$, we use $w(I)$ to denote $\sum_{i\in I} w_{S_i}$.

We present the algorithm assuming knowledge of an estimate z such that $\text{OPT} \leq z \leq (1+\epsilon)\,\text{OPT}$; this assumption can be removed by running the algorithm for guesses $1, (1+\epsilon), (1+\epsilon)^2, \ldots$ for z and returning the best solution found. The basic algorithm maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. The algorithm proceeds as follows:

1. Initialize $C = \emptyset$ and $I = \emptyset$
2. For each set S in the stream:
   a. If $|S \setminus C| \geq \frac{2z}{3} \cdot \frac{w_S}{L}$ then:
      i. If $w(I) + w_S > L$: Terminate and return:

$$I \leftarrow \begin{cases} I & \text{if } |C| \geq |S| \\ \{ID(S)\} & \text{if } |C| < |S| \end{cases}$$

      ii. $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

▶ **Lemma 12.** *If the clause in line 2ai is never satisfied, then thealgorithm returns a $1/3 - \epsilon$ approximation.*

Proof. Suppose the collection of sets chosen by the algorithm is $\mathcal{S}$. We define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$ to be the set of elements in S that are not covered by the final solution. For each set S in the optimum solution $\mathcal{O}$, if S is unpicked, then $|\tilde{S}| \leq 2z/3 \cdot w_S/L$. Therefore,

$$\begin{aligned}\mathrm{OPT} = \left|\bigcup_{S\in\mathcal{O}\cap\mathcal{S}} S\right| + \left|\bigcup_{S\in\mathcal{O}\setminus\mathcal{S}} \tilde{S}\right| &\leq |\mathcal{C}(\mathcal{S})| + \sum_{S\in\mathcal{O}\setminus\mathcal{S}} |\tilde{S}| \leq |\mathcal{C}(\mathcal{S})| + \frac{2z}{3} \\ &\leq |\mathcal{C}(\mathcal{S})| + \frac{2\,\mathrm{OPT}(1+\epsilon)}{3},\end{aligned}$$

and thus $|\mathcal{C}(\mathcal{S})| \geq \frac{1-2\epsilon}{3}\,\mathrm{OPT}$. ◀

▶ **Lemma 13.** *If the clause in line 2ai is satisfied at some point, then the algorithm returns a* $1/3$ *approximation.*

Proof. Suppose the clause is satisfied when the set S is being considered. Then

$$|S \setminus C| + |C| \geq \frac{2z}{3} \cdot \frac{w_S + w(I)}{L} \geq \frac{2z}{3}$$

where we used the fact that $w_S + w(I) > L$. The claim then follows immediately. ◀

▶ **Theorem 14.** *There exists a single-pass,* $\tilde{O}(\epsilon^{-1}n)$*-space algorithm that finds a* $1/3 - \epsilon$ *approximation of budgeted* MaxSetCoverage.

3 Algorithms for Maximum k-Vertex Coverage

In this section, we present algorithms for the maximum k-vertex coverage problem. We present our results in terms of hypergraphs for full generality. The generalization to hypergraphs can also be thought of as a natural "hitting set" variant of maximum coverage, i.e., the stream consists of a sequence of sets and we want to pick k elements in such a way to maximize the number of sets that include a picked element.

Notation. Given a hypergraph G and a subset of nodes S, we define $\mathcal{C}_G(S)$ to be the number of edges that contain at least one node in S. Recall that the maximum k-vertex coverage problem is to approximate the maximum value of $\mathcal{C}_G(S)$ over all sets S containing k nodes. We use E_G and V_G to denote the set of edges and nodes of the hypergraph G respectively.

The size of a cut $(S, V \setminus S)$ in a hypergraph G, denoted as $\delta_G(S)$, is defined as the number of hyperedges that contain at least one node in both S and $V \setminus S$. In the case that G is weighted, $\delta_G(S)$ denotes the total weight of the cut. A core idea to our approach is to use *hypergraph sparsification*:

▶ **Definition 15** (ϵ-sparsifier)**.** Given a hypergraph $G = (V, E)$, we say that a weighted subgraph $H = (V, E')$ is an ϵ-sparsifier for G if for all $S \subseteq V$, $\delta_G(S) \approx_\epsilon \delta_H(S)$.

Any graph on N nodes has an ϵ-sparsifier with only $\tilde{O}(\epsilon^{-2}N)$ edges [43]. Similarly, any hypergraph in which the maximum size of the hyperedges is bounded by d (rank d hypergraphs) has an ϵ-sparsifier with only $\tilde{O}(\epsilon^{-2}dN)$ edges. Furthermore, an ϵ-sparsifier can be constructed in the dynamic graph stream model using one pass and $\tilde{O}(\epsilon^{-2}dN)$ space [25, 27].

First, we show that it is possible to approximate all the coverages by constructing a sparsifier of a slightly modified graph. In particular, we construct the sparsifier H of the

graph G' with an extra node v, i.e., $V_{G'} = V_G \cup \{v\}$, and for every hyperedge $e \in E_G$, we put the hyperedge $e \cup \{v\}$ in $E_{G'}$. It is easy to see that for all S that is a subset of V_G, $\mathcal{C}_G(S) = \delta_{G'}(S)$. Therefore, it is immediate that we could $1+\epsilon$ approximate all the coverages in G by constructing the sparsifer of G'.

▶ **Theorem 16.** *There exists a single-pass, $\tilde{O}(\epsilon^{-2}dN)$-space algorithm that finds a $1-\epsilon$ approximation of* MaxVertexCoverage *of rank d hypergraphs with high probability.*

The above theorem assumes unbounded post-processing time. If k is constant, the post-processing will be polynomial. For larger k, if we still require polynomial running time then, after constructing the ϵ-sparsifier H, we could either use the $(1-(1-1/d)^d)$ approximation algorithm via linear programming [1] or the folklore $(1-1/e)$ approximation greedy algorithm.

3.1 Algorithm for Near-Regular Hypergraphs

In this subsection, we show that is possible to reduce the space used to $\tilde{O}(\epsilon^{-3}dk)$ in the case of hypergraphs that are regular or nearly regular. Define $\kappa \leq 1$ to be the ratio between the smallest degree and the largest degree; for a regular hypergraph $\kappa = 1$. We show that a $(\kappa - \epsilon)$ approximation is possible using $\tilde{O}(\epsilon^{-3}dk)$ space for rank d hypergraphs. This also implies a $(1-\epsilon)$ approximation for regular hypergraphs.

▶ **Theorem 17.** *There exists a single-pass, $\tilde{O}(\epsilon^{-3}dk)$-space algorithm that finds a $(\kappa-\epsilon)$ approximation of* MaxVertexCoverage *of hypergraphs of rank d with high probability .*

Proof. Suppose we uniformly sample a set S of k nodes. Let $L_S(y) = \max(0, |y \cap S| - 1)$. Then the coverage of S satisfies

$$\mathcal{C}_G(S) = \sum_{y \in E_G} I[S \cap y \neq \emptyset] = \sum_{y \in E_G} (|S \cap y| - L_S(y)) \geq kt_1 - \sum_{y \in E_G} L_S(y) \ .$$

where the last inequality follows since every node in S covers at least t_1 hyperedges.

Let $\xi_y(j)$ denote the event that j nodes in the hyperedge y are in S and let $|y|$ denote the number of nodes in y. We have

$$\mathrm{E}\left[L_S(y)\right] = \sum_{j=1}^{|y|}(j-1)\Pr\left[\xi_y(j)\right] = \left(\sum_{j=0}^{|y|} j \Pr\left[\xi_y(j)\right]\right) - 1 + \Pr\left[\xi_y(0)\right].$$

The sum $\sum_{j=0}^{|y|} j \Pr\left[\xi_y(j)\right]$ is the expected value of the hypergeometric distribution and therefore it evaluates to $|y|k/N$. Furthermore,

$$\Pr\left[\xi_y(0)\right] = \prod_{i=0}^{k-1}\left(1 - \frac{|y|}{N-i}\right) \leq \left(1 - \frac{|y|}{N}\right)^k \leq \exp\left(-\frac{k|y|}{N}\right) \leq 1 - \frac{k|y|}{N} + \frac{1}{2}\left(\frac{k|y|}{N}\right)^2.$$

The last inequality follows from taking the first three terms of the Taylor's expansion. Hence,

$$\mathrm{E}\left[L_S(y)\right] \leq \frac{k|y|}{N} - 1 + 1 - \frac{k|y|}{N} + \frac{1}{2}\left(\frac{k|y|}{N}\right)^2 = \frac{1}{2}\left(\frac{k|y|}{N}\right)^2.$$

Hence, if $N \geq 4kd/\epsilon$, then

$$\sum_{y \in E_G} \mathrm{E}\left[L_S(y)\right] \leq \frac{1}{2}\sum_{y \in E_G}\left(\frac{k|y|}{N}\right)^2 \leq \frac{1}{2}d\left(\frac{k}{N}\right)^2 \sum_{y \in E_G}|y| \leq \frac{1}{2}d\left(\frac{k}{N}\right)^2 Nt_2 \leq \frac{1}{8}\epsilon k t_2.$$

By an application of Markov's inequality,

$$\Pr\left[\sum_{y\in E_G} L_S(y) \geq \epsilon k t_2\right] \leq 1/8 \ .$$

Thus, if we sample $O(\log N)$ sets of k nodes in parallel, with high probability, there is a sample set S of k nodes satisfying $\sum_{y\in E_G} L_S(y) \leq \epsilon k t_2$ which implies that $\mathcal{C}_G(S) \geq kt_1 - \epsilon k t_2 \geq (\kappa - \epsilon)\,\mathrm{OPT}$. If $N \leq 4kd/\epsilon$, we simply construct the sparsifier of G' as described above to achieve a $1-\epsilon$ approximation. ◀

4 Lower Bounds

In this section, we prove space lower bounds for data stream algorithms that approximate MaxSetCoverage or MaxVertexCoverage. In particular, these imply that improving over an $(1-1/e)$ approximation of MaxSetCoverage with constant passes and constant k requires $\Omega(m)$ space. Recall that, still assuming k is constant, we designed a constant-pass algorithm that returned a $(1-1/e-\epsilon)$ approximation using $\tilde{O}(\epsilon^{-2}k)$ space. For constant k, we also show that improving over a κ approximation (where κ is the ratio between the lowest degree and the highest degree) for MaxVertexCoverage requires $\Omega(N\kappa^3)$ space. Our algorithm returned a $\kappa-\epsilon$ approximation using $\tilde{O}(\epsilon^{-3}k)$ space.

Approach. We prove both bounds by a reduction from r-player set-disjointness in communication complexity. In this problem, there are r players where the ith player has a set $S_i \subseteq [u]$. It is promised that exactly one of the following two cases happens.

- Case 1 (NO instance): All the sets are pairwise disjoint.
- Case 2 (YES instance): There is a unique element $e \in [u]$ such that $e \in S_i$ for all $i \in [r]$.

The goal of the communication problem is the rth player answers whether the input is a YES instance or a NO instance correctly with probability at least 0.9. We shall denote this problem by $\mathsf{DISJ}_r(u)$.

The communication complexity of the above problem in p-round, one-way model (where each round consists of player 1 sending a message to player 2, then player 2 sending a message to player 3 and so on) is $\Omega(u/r)$ [17] even if the players may use public randomness. This implies that in any randomized communication protocol, the maximum message sent by a player contains $\Omega(u/(pr^2))$ bits. Without loss of generality, we could assume that $|S_1 \cup S_2 \cup \ldots \cup S_r| \geq u/4$ via a padding argument.

▶ **Theorem 18.** *Assuming $n = \Omega(\epsilon^{-2}k\log m)$, any constant-pass algorithm that finds a $(1+\epsilon)(1-(1-1/k)^k)$ approximation of* MaxSetCoverage *with probability at least 0.99 requires $\Omega(m/k^2)$ space even when all the sets have the same size.*

Proof. Recall that $(1+\epsilon)(1-(1-1/k)^k) \geq 1-1/e+O(\epsilon)$. Our proof is a reduction from $\mathsf{DISJ}_k(m)$. Consider a sufficiently large n where k divides n. For each $i \in [m]$, let $\mathcal{P}_i$ be a random partition of $[n]$ into k sets $V_1^i, \ldots, V_k^i$ of equal size. Each partition is chosen independently and the players agree on these partitions using public randomness before receiving the input.

For each player j, if $i \in S_j$, then she puts V_j^i in the stream. According to the aforementioned assumption, the stream consists of at least $m/4$ sets.

If the input is a NO instance, then for each $i \in [m]$, there is at most one set V_j^i in the stream. Hence, the stream consists of independent random sets of size n/k. Therefore, for each

$e \in [n]$ and any k sets $V_{j_1}^{i_1}, \ldots, V_{j_k}^{i_k}$ in the stream, $\Pr\left[e \in V_{j_1}^{i_1} \cup \ldots \cup V_{j_k}^{i_k}\right] = 1 - (1-1/k)^k$. By an application of Chernoff bound for negatively correlated boolean random variables [39],

$$\Pr\left[\left||V_{j_1}^{i_1} \cup \ldots \cup V_{j_k}^{i_k}| - \left(1 - \left(1 - \frac{1}{k}\right)^k\right) n\right| > \epsilon \left(1 - \left(1 - \frac{1}{k}\right)^k\right) n\right]$$
$$\leq 3\exp\left(-\epsilon^2 \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \frac{n}{3}\right) \leq 3\exp\left(-\epsilon^2(1-1/e)n/3\right) \leq \frac{1}{m^{10+k}}.$$

The last inequality holds when n is a sufficiently large multiple of $k\epsilon^{-2}\log m$. Therefore, the maximum coverage in this case is at most $(1+\epsilon)(1-(1-1/k)^k)n$ with probability at least $1 - 1/m^{10}$ by taking the union bound over all $\binom{m}{k} \leq m^k$ possible k sets.

If the input is a YES instance, then clearly, the maximum coverage is n. This is because there exists $i \in [m]$ such that $i \in S_1 \cap \ldots \cap S_k$ and therefore $V_1^i, \ldots, V_k^i$ are in the stream.

Therefore, any constant pass and $O(s)$-space algorithm that finds a $(1+2\epsilon)(1-(1-1/k)^k)$ approximation of the maximum coverage with probability at least 0.99 implies a protocol to solve the k-party disjointness problem using $O(s)$ bits of communication. Thus, $s = \Omega(m/k^2)$ as required. ◀

Consider the sets $S_1, \ldots, S_r \subseteq [u]$ that satisfy the unique intersection promise as in $\mathsf{DISJ}_r(u)$. Let X be the r by u matrix in which the row X_i is the characteristic vector of S_i. Suppose there are $r' = \Omega(r^2)$ players. Chakrabarti et al. [16] showed that if each entry of X is given to a unique player and the order in which the entries are given to the players is random, then the players need to use $\Omega(u/r)$ bits of communication to tell whether the sets is a YES instance or a NO instance with probability at least 0.9. Thus, in any randomized protocol, the maximum message sent by a player contains $\Omega(u/r^3)$ bits. Hence, using the same reduction and assuming constant k, we show that the same lower bound holds even for random order stream.

▶ **Theorem 19.** *Assuming $n = \Omega(\epsilon^{-2}k\log m)$, any constant-pass algorithm that finds a $(1+\epsilon)(1-(1-1/k)^k)$ approximation of* MaxSetCoverage *with probability at least 0.99 requires $\Omega(m/k^3)$ space even when all the sets have the same size and arrive in random order.*

Next, we prove a lower bound for the k-vertex coverage problem for graphs where the ratio between the minimum degree and the maximum degree is at least κ. We show that for constant k, beating κ approximation for constant κ requires $\Omega(N)$ space.

Since κ can be smaller than any constant, this also establishes that $\Omega(N)$ space is required for any constant approximation of MaxVertexCoverage.

▶ **Theorem 20.** *For $\epsilon > 0$, any constant-pass algorithm that finds a $(\kappa+\epsilon)$ approximation of* MaxVertexCoverage *with probability at least 0.99 requires $\Omega(N\kappa^3/k)$ space.*

Proof. Initially, assume $k = 1$. We consider the multi-party set disjointness problem $\mathsf{DISJ}_t(N')$ where $t = 1/\kappa$ and $N' = N/t$. Here, there are t players and the input sets are subsets of $[N']$. We consider a bipartite graph where the set of possible nodes are $L \cup R$ where $L = \{u_i\}_{i\in[N']}$ and $R = \{v_{i,j}\}_{i\in[N'],j\in[t]}$. Note that this graph has $(t+1)N' = \Theta(N)$ nodes. However we only consider a node to exist if the stream contains an edge incident to that node.

The j-th player defines a set of edges on this graph based on their set S_j as follows. If $i \in S_j$ she puts the edge between u_i and $v_{i,j}$. If $S_1, \ldots, S_t$ is a YES instance, then there must be a node u_i that has degree t. If A is a NO instance, then every node in the graph

has degree at most 1. Hence the ratio of minimum degree to maximum degree is at least $1/t = \kappa$ as required.

Thus, for $k = 1$, a $1/t$ approximation with probability at least 0.99 on a graph of N nodes implies a protocol to solve $\mathsf{DISJ}_t(N')$. Therefore, the algorithm requires $\Omega(N\kappa^3)$ space. For general k, we make k copies of the above construction to deduce the lower bound $\Omega(N\kappa^3/k)$. ◀

Acknowledgements. We thank Sagar Kale for discussions of related work.

References

1 Alexander A. Ageev and Maxim Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *IPCO*, volume 1610 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 1999.

2 Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2237–2246. JMLR.org, 2015.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467. SIAM, 2012.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14. ACM, 2012.

5 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *APPROX-RANDOM*, volume 8096 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2013.

6 Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering for fast approximate document retrieval. *ACM Trans. Inf. Syst.*, 33(3):11:1–11:35, 2015.

7 Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC*, pages 698–711. ACM, 2016.

8 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *SODA*, pages 1345–1364. SIAM, 2016.

9 Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and Vangelis Th. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012.

10 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *KDD*, pages 671–680. ACM, 2014.

11 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, pages 1497–1514. SIAM, 2014.

12 MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Almost optimal streaming algorithms for coverage problems. *CoRR*, abs/1610.08096, 2016.

13 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC*, pages 173–182. ACM, 2015.

14 Édouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Georgios Stamoulis. A 0.821-ratio purely combinatorial algorithm for maximum k-vertex cover in bipartite graphs. In *LATIN*, volume 9644 of *Lecture Notes in Computer Science*, pages 235–248. Springer, 2016.

15 Bugra Caskurlu, Vahan Mkrtchyan, Ojas Parekh, and K. Subramani. On partial vertex cover and budgeted maximum coverage problems in bipartite graphs. In *IFIP TCS*, volume 8705 of *Lecture Notes in Computer Science*, pages 13–26. Springer, 2014.

16 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:62, 2011.

17 Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117. IEEE Computer Society, 2003.

18 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *SODA*, pages 1365–1373. SIAM, 2016.

19 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015.

20 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *SODA*, pages 1326–1344. SIAM, 2016.

21 Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng.*, 15(3):529–540, 2003.

22 Graham Cormode, Howard J. Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. In *CIKM*, pages 479–488. ACM, 2010.

23 Yuval Emek and Adi Rosén. Semi-streaming set cover - (extended abstract). In *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 453–464. Springer, 2014.

24 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

25 Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *PODS*, pages 241–247. ACM, 2015.

26 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In *PODS*, pages 371–383. ACM, 2016.

27 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *FOCS*, pages 561–570. IEEE Computer Society, 2014.

28 Michael Kapralov and David P. Woodruff. Spanners and sparsifiers in dynamic streams. In *PODC*, pages 272–281. ACM, 2014.

29 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

30 Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

31 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *6th Innovations in Theoretical Computer Science*, 2015.

32 Christian Konrad. Maximum matching in turnstile streams. In *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer, 2015.

33 Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, pages 1650–1654. AAAI Press, 2007.

34 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. *TOPC*, 2(3):14, 2015.

35 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

36 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In *MFCS (2)*, volume 9235 of *Lecture Notes in Computer Science*, pages 472–482. Springer, 2015.

37 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *PODS*, pages 401–411. ACM, 2016.

38 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *CoRR*, abs/1610.06199, 2016. URL: `http://arxiv.org/abs/1610.06199`.

39 Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.

40 Jaikumar Radhakrishnan and Saswata Shannigrahi. Streaming algorithms for 2-coloring uniform hypergraphs. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, pages 667–678, 2011. `doi:10.1007/978-3-642-22300-6_57`.

41 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, pages 697–708. SIAM, 2009.

42 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

43 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

44 He Sun. Counting hypergraphs in data streams. *CoRR*, abs/1304.7456, 2013. URL: `http://arxiv.org/abs/1304.7456`.

45 Huiwen Yu and Dayu Yuan. Set coverage problems in a one-pass data stream. In *SDM*, pages 758–766. SIAM, 2013.