

NORTHEASTERN UNIVERSITY

PHD DISSERTATION

REVERSE QUERY OPTIMIZATION

Author: Neha Makhija

Doctoral Thesis Committee: Wolfgang Gatterbauer (advisor)
Mirek Riedewald
Renée Miller
Antoine Amarilli

*A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the*

Khoury College of Computer Sciences

**Northeastern University
Khoury College of
Computer Sciences**


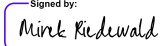


PhD Thesis Approval

Thesis Title: Reverse Query Optimization

Author: Neha Makhija

PhD Program: ☒ Computer Science ☐ Cybersecurity ☐ Personal Health Informatics

PhD Thesis Approval to complete all degree requirements for the above PhD program.

<small>DocuSigned by:</small>  <small>C30B8M4263CA48E</small>	<u>6/30/2025</u>
Thesis Advisor	<i>Date</i>
<small>Signed by:</small>  <small>CA675994105E4E0</small>	<u>7/2/2025</u>
Thesis Reader	<i>Date</i>
<small>Signed by:</small>  <small>B287529D3E07A57</small>	<u>7/11/2025</u>
Thesis Reader	<i>Date</i>
<small>Signed by:</small>  <small>EB9FC788E400432</small>	<u>6/30/2025</u>
Thesis Reader	<i>Date</i>
Thesis Reader	<i>Date</i>

KHOURY COLLEGE APPROVAL:

 Associate Dean for Graduate Programs	<u>14 July 2025</u>
	<i>Date</i>

COPY RECEIVED BY GRADUATE STUDENT SERVICES:

 Recipient's Signature	<u>14 July 2025</u>
	<i>Date</i>

Distribution: Once completed, this form should be attached as page 2, immediately following the title page of the dissertation document. An electronic version of the document can then be uploaded to the Northeastern University-UMI Website.

Abstract

While traditional data management typically answers questions like “What were the expenses of my company this quarter?”, the underlying data can be used to answer more powerful questions like “What expenses should my company have this quarter (given the business requirements)?”. These questions can generally be modeled as “Reverse” Data Management (RDM) problems that ask: “What optimal interventions on data produce a desired property in the output?” Several problems from this family have been studied and used to solve challenges in domains as diverse as query explainability, data debugging, knowledge representation, and effective SQL pedagogy. However, these variants, their complexity, and practical algorithms had always been studied in isolation- hence prior solutions do not generalize to slightly different problems and settings, and many questions in this space remain unsolved.

This dissertation proposes a novel paradigm for databases to compute the answers to such *Reverse Questions*: instead of creating specialized algorithms for easy (PTIME) and hard cases (NP-Complete), we can devise unified algorithms that can solve all problem instances and are guaranteed to terminate in PTIME for easy cases. This approach allows us to separate the task of designing efficient algorithms from the task of proving tractability. An ultimate goal would be to design instance optimal algorithms to execute Reverse Queries, however developing such algorithms is an open, challenging question for many problems across computer science with few known positive answers. Instead, in an effort to bridge this gap, we develop algorithms that are *coarse-grained instance optimal i.e.* they are guaranteed to run in polynomial time for all known tractable classes of instances, however they are coarse-grained instead of fine-grained *i.e.* they do not make guarantees on the precise running time, but provide guarantees based on if the optimal running time is known to be PTIME or not.

The proposed approach leads to both practical benefits in terms of instance-based time guarantees and ease of implementation, as well as new theoretical complexity results. We show that the proposed approach can be used to solve a wide range of Reverse Data Management problems, including problems well-known in databases such as *Resilience*, *Causal Responsibility*, and many variants of *Deletion Propagation*. We also show that ideas of unified algorithms are applicable and useful for problems that are not traditionally considered as Reverse Data Management problems (but are closely related in the spirit of modifying query output representations), such as the problem of finding the *Minimum Factorization of Provenance Expressions*.

A complementary contribution of this work is an “Automatic Hardness Gadget”-finder. Showing that a problem is NP-Hard is a fundamental task in theoretical computer science (and database theory), and many problems are known to be NP-Hard, but the proofs are often tedious and not easy to find or generalize. We build a tool that leverages a semantic specification of NP-Hardness to computationally build hardness proofs and hence resolve open theoretical complexity questions.

Acknowledgements

I am grateful to my advisor Dr. Wolfgang Gatterbauer, whose guidance has been valuable not just in developing my technical ability, but also in learning how to do good research even when things don't work out the way you expect - introducing me to the adage that *problems worthy of attack often fight back*. I am also immensely grateful for the support of my committee members, Dr. Mirek Riedewald, Dr. Renée Miller, and Dr. Antoine Amarilli; all of whom I have learned a lot from, and relied on many times for their sound and honest feedback and advice.

I am fortunate to have the opportunity to share a lab with so many talented and incredibly nice colleagues at the DataLab at Northeastern University, and I am grateful to all of them for their friendships and insights. My research has benefited from the discussions and collaborations with many, many fantastic members of the database community, especially through people I met at the "Logic and Algorithms in Database Theory and AI" program at Simons Institute, and at my internships at RelationalAI and IBM. Each of these experiences has been a source of inspiration and learning, and has shaped my research for the better.

I am grateful to my parents, who taught me the virtues of hard-work and dedication by example; and encouraged me to pursue my interests, even when they took me far from home. My parents, grandparents, and all of my family have always believed in me, and I am thankful for their blessings and love. I am indebted to my friends, who have over many a game-night, potluck, art-night, hike, phone call, grocery run or shared meal, made the journey of graduate school fulfilling and special. I am incredibly lucky to have Samar by my side in this journey, and this dissertation (and my life) is better for it.

Contents

Abstract	iii
Acknowledgements	iv
Contents	v
1 Introduction	1
1.1 Thesis Statement	1
1.2 Unified Algorithms for Reverse Query Optimization	2
1.3 Outline and Contributions	4
1.3.1 Resilience (Chapter 4)	4
1.3.2 Causal Responsibility (Chapter 5)	5
1.3.3 Generalized Deletion Propagation (Chapter 6)	5
1.3.4 Minimum Factorization of Provenance Formulas (Chapter 7)	5
1.3.5 Automatic Hardness Gadget-finder for Resilience problems (Chapter 8)	6
2 Background and Related Work	7
2.1 Problems in Focus in this Dissertation	7
2.2 Related Work and Themes	8
3 Preliminaries	10
3.1 Standard Notations and Definitions	10
3.2 Provenance Framework and Notation	11
3.3 Background on Linear Programming	12
UNIFIED ALGORITHMS FOR REVERSE DATA MANAGEMENT PROBLEMS	14
4 Resilience	15
4.1 Problem Statement	15
4.2 Chapter Overview and Contributions	15
4.3 ILP for Resilience	16
4.4 PTIME Relaxation of ILP[RES*]	19
4.5 Finding hardness certificates	19
4.5.1 Independent Join Paths (IJPs)	19
4.5.2 More Example IJPs	24
4.6 Complexity results for SJ-free CQs	24
4.6.1 Necessary notations	24
4.6.2 Dichotomies for RES under Sets and Bags	25
4.7 Three Approximation Algorithms	28
4.7.1 LP-based m-factor Approximation	28
4.7.2 Flow-based Approximations	28
4.8 Experiments	29
4.8.1 Experimental Settings	30
4.8.2 Key Takeaways from Experiments	31
4.9 Chapter Summary	33
5 Causal Responsibility	34
5.1 Problem Statement	34
5.2 Chapter Overview and Contributions	34
5.3 ILP for Causal Responsibility	36
5.4 PTIME Relaxation of ILP[RSP*]	37
5.4.1 MILP Relaxation for RSP	37

5.5	Complexity results for SJ-Free CQs	38
5.6	Additional Instance Based Complexity Results for Resilience and Causal Responsibility	42
5.7	A note on Approximation Algorithms	43
5.8	Experiments	44
5.9	Chapter Summary	45
6	Generalized Deletion Propagation	46
6.1	Chapter Overview and Contributions	48
6.2	Problem Statement	49
6.2.1	Defining Generalized Deletion Propagation	50
6.2.2	Capturing Prior Variants of Deletion Propagation with GDP	50
6.2.3	Capturing Natural New Variants of Deletion Propagation with GDP	52
6.3	ILP Framework for GDP	53
6.3.1	A basic ILP Formulation for GDP	54
6.3.2	Wildcard Semantics for $X[w]$ and $X[v]$	56
6.3.3	ILP with Smoothing Constraints	59
6.4	Recovering Existing Tractability Results	62
6.5	New Tractability Results	69
6.6	Experiments	70
6.7	A Note on System Implementation	76
6.8	Chapter Summary	77
7	Minimal Factorization of Provenance Formulas	78
7.1	Chapter Overview and Contributions	79
7.2	Problem Definition	80
7.3	Related Work	80
7.3.1	Boolean Factorization	81
7.3.2	Probabilistic Inference, Read-Once Formulas, and Dissociation	83
7.4	Search Space for minFACT	83
7.4.1	Factorizations and Variable Elimination Orders	83
7.4.2	Bijection Between VEOs and Query Plans	87
7.4.3	Details and Examples: VEO Instances, Table Prefixes and Factorization Forests	90
7.4.4	Connections of VEOs to Related Work	91
7.4.5	End to End Example: Query Plans, VEOs and Factorizations	92
7.4.6	Details about Minimal VEOs	93
7.5	ILP Formulation for minFACT	98
7.5.1	Example minFACT ILPs	99
7.5.2	Value of structural knowledge	102
7.6	PTIME Algorithms	102
7.6.1	MFMC-based Algorithm for minFACT	102
7.6.2	LP relaxation for minFACT and an LP relaxation-based approximation	108
7.7	Recovering Read-Once instances	110
7.8	Tractable Queries for minFACT	112
7.8.1	All queries with ≤ 2 minimal query plans	112
7.8.2	Two queries with ≥ 3 minimal query plans	114
7.8.3	Conjecture for Linear Queries	119
7.9	Hard Queries for minFACT	119
7.10	Application: A complete approach for Approximate Probabilistic Inference	122
7.10.1	A Proof-of-Concept for Improving Probabilistic Inference	125
7.11	Optimizations for computing minFACT	126
7.12	Experiments	128
7.12.1	Experimental Setup	128
7.12.2	Experimental Results	129
7.13	Chapter Summary	132

UNIFYING CRITERIA TO PROVE NP-HARDNESS	132
8 Automatic Hardness Gadgets	133
8.1 Related Work	133
8.2 Automatic Hardness Gadgets for Resilience	133
8.3 Scalability Experiment of newly found hard query	135
8.4 Chapter Summary	136
9 Conclusion	137
9.1 Future Directions	137
 APPENDIX	 139
A Nomenclature	140
B Disjunctive Logic Program for Automatic Hardness Gadget-finder: An Example	143
Bibliography	146

List of Figures

1.1	Dissertation Overview: Dissertation Structure with Highlighted Contributions	4
3.1	Example explaining the Provenance of a Query Output	12
4.1	An example hardness gadget for the triangle query	20
4.2	Composition of hardness gadgets	22
4.3	Simple hardness gadgets for prior known hard queries.	24
4.4	Flow approximation linearizations for Example 4.7.1.	29
4.5	Resilience Experiment: Hard 3-star query Q_3^\star	30
4.6	Resilience Experiment: Queries with self-joins under Bag Semantics	31
4.7	Resilience Experiment: Contrasting a query between Set and Bag Semantics	32
5.1	Overview of complexity results in Chapter 4 (Resilience) and Chapter 5 (Causal Responsibility)	35
5.2	Causal Responsibility Hardness Gadget	41
5.3	Causal Responsibility Experiment on TPC-H Data	44
6.1	A motivating example for Generalized Deletion Propagation that extends previously studied DP variants.	47
6.2	Generalized Deletion Propagation (GDP) Framework, and its relation to prior work	49
6.3	Propagation constraints in the ILP formulation of GDP	55
6.4	Wildcard semantics used to model an ILP for GDP	57
6.5	One-side guarantees enforced under wildcard semantics	58
6.6	Direction of lower bound constraints in ILP[GDP]	59
6.7	Intuition behind smoothing constraints in ILP[GDP]	62
6.8	GDP Experiments: Performance of ILP[GDP] as compared with previous specialized algorithms	71
6.9	GDP Experiments: The Power of Smoothing Constraints	72
6.10	GDP Experiments: A case study on real-world flight data	73
6.11	GDP experiment: Scalability of ILP[GDP] for different domain sizes	74
6.12	GDP experiment: Scalability of ILP[GDP] for different query sizes	75
6.13	GDP experiment: Memory consumption of ILP[GDP]	76
7.1	Complexity Landscape for the Minimum Factorization of Provenance problem	79
7.2	An overview of related work on the Exact, Minimal Equivalent Expression (MEE) problem applied to formulas.	82
7.3	Representation of a factorization as a mapping of witnesses to VE0s	84
7.4	An example query plan with its corresponding VE0	89
7.5	VE0 split operation	90
7.6	Examples of VE0 instances, VE0 table prefix instances and VE0 factorization forests	91
7.7	The incidence graphs, minimal dissociations, minimal query plans and factorization of Q_A^Δ	97
7.8	Example 7.4.10: Two legal query plans for Q_2^∞ . P_1 is a hierarchical minimal plan while P_2 is not.	97
7.9	ILP Formulation for minFACT	98
7.10	Example mveo for 3-chain query	99
7.11	Example 7.5.2: mveo for 3-star query Q_3^\star	102
7.12	A flow graph F for minFACT.	103
7.13	Example showing a set of mveo that cannot form an RP-ordering	104
7.14	Example of a factorization flow graph with a non-RP ordering	105
7.15	Example of a Nested RP-ordering	105
7.16	6-cycle query with endpoints	105
7.17	Example 7.6.3: A flow graph with leakage for the query Q^Δ	107
7.18	Example 7.6.4: From three mveos to a flow graph for the triangle query	107

7.19	Section 7.7: P_4 co-occurrence pattern ($w_1 - r - w_2 - s - w_3$).	110
7.20	Proof Theorem 7.7.1: $P_4(w_1 - p_r - w_2 - p_s - w_3)$ in factorization flow graph.	111
7.21	Proof Theorem 7.7.1: Impossibility of leakage in read-once Instances (Case 2).	111
7.22	Q_A^Δ graph for a single witness	115
7.23	Simplified Q_A^Δ graph for a single witness	115
7.24	Q_A^Δ instance, with shared x (Case 3)	115
7.25	Q_A^Δ instance, with shared xy (Case 4)	116
7.26	Q_A^Δ instance, with shared xz (Case 5)	116
7.27	Q_A^Δ instance, with shared yz (Case 6)	116
7.28	Flow graph with a single witness of Q_4^∞ under ordering Ω .	117
7.29	Lemma 7.8.8: Showing that if there is leakage in the graph, there must be two witnesses that can together form a leakage path.	117
7.30	Comparing Q^Δ and Q_A^Δ , two structurally similar queries with different complexities	119
7.31	Theorem 7.9.1: Hardness gadget for a query with triad $\{R, S, T\}$.	120
7.32	Theorem 7.9.1: Example graph with its independent set	121
7.33	Theorem 7.9.4: Hardness gadget for Q_{cod}^Δ	122
7.34	Connections of minimal factorization to prior work in probabilistic inference	123
7.35	Subsection 7.10.1: Parameterized database instance for evaluating probabilistic inference with $2k$ probabilistic variables.	125
7.36	Subsection 7.10.1: Expected value of exact and approximate probabilistic inference comparison for an increasing number of tuples (x-axis is k).	125
7.37	Experimental study of time and factorization length for 5 queries over all algorithms.	131
8.1	Automatically generated and visualized Hardness Gadgets for 5 previously open queries	135
8.2	Resilience experiment: For a newly proven hard SJ query	135
B.1	Automatically generated IJP for Q_{2-SJ}^∞	145

List of Tables

4.1	Complexity Landscape for Resilience	24
5.1	Complexity Landscape for Resilience and Responsibility	39
A.1	Notation Table	140
A.2	Example Queries	141
A.3	Common Abbreviations	141

List of Definitions

2.1.1	Definition: Reverse Data Management (RDM)	7
4.1.1	Definition: Resilience [59]	15
4.5.1	Definition: Join Path (JP)	20
4.5.2	Definition: Independent Join Path	21

4.6.1 Definition: Exogenous / Endogenous tuples	24
4.6.2 Definition: Domination [59]	25
4.6.3 Definition: Triad (different from [59])	25
4.6.4 Definition: Solitary variable [59]	25
4.6.5 Definition: Full domination [59]	25
4.6.6 Definition: Active or (fully) deactivated triads	25
5.1.1 Definition: Causal Responsibility [59]	34
6.2.1 Definition: Generalized Deletion Propagation (GDP)	50
6.2.2 Definition: DP-SS	50
6.2.3 Definition: DP-VS	51
6.2.4 Definition: ADP	51
6.2.5 Definition: SWP	52
6.4.1 Definition: Existential Connectivity Graph G_Q^\exists	66
6.4.2 Definition: Head Cluster Property	67
7.2.1 Definition: FACT	80
7.4.1 Definition: Variable Elimination Order (VE0)	84
7.4.2 Definition: VE0 instance	84
7.4.3 Definition: VE0 table prefix	85
7.4.4 Definition: VE0 factorization forest (VE0FF)	85
7.4.5 Definition: VE0 Merge	90
7.4.6 Definition: VE0 Split	90
7.4.7 Definition: Hierarchical query [39]	94
7.4.8 Definition: Hierarchical plan	94
7.4.9 Definition: Query dissociation	95
7.4.10 Definition: Hierarchical dissociation	95
7.4.11 Definition: Partial dissociation order	95
7.6.1 Definition: Running-Prefixes (RP) ordering	104
7.6.2 Definition: Leakage	106
7.8.1 Definition: VEO roots	112
7.10.1 Definition: Probabilistic query evaluation	122

List of Theorems, Lemmas, Propositions, and Corollaries

4.3.1 Lemma: Simplification for RES under Bag Semantics	18
4.3.2 Theorem: RES ILP correctness	18
4.5.1 Proposition: Triangle composition	21
4.5.2 Theorem: IJPs \Rightarrow NPC	22
4.5.3 Theorem: IJPs \Leftrightarrow NPC for SJ-free CQs	23
4.6.1 Theorem: Integrality of LP[RES*] for Linear Queries	25
4.6.2 Theorem: Integrality of LP[RES*] for queries with deactivated triads under set semantics	26
4.6.3 Theorem: Hardness of RES for non-linear queries under bag semantics	27
4.6.4 Corollary: Resilience Dichotomy under set semantics	27
4.6.5 Corollary: Resilience Dichotomy under bag semantics	27
4.7.1 Theorem: m -factor approximation for RES	28

5.3.1 Theorem: ILP[RSP*] Correctness	36
5.4.1 Theorem: PTIME solvability of MILP[RSP*]	38
5.5.1 Theorem: Integrality of MILP[RSP*] for Linear Queries	39
5.5.2 Theorem: Integrality of MILP[RSP*] with fully deactivated triads under set semantics	40
5.5.3 Theorem: Integrality of MILP[RSP*] when responsibility tuple is in dominating relation	40
5.5.4 Theorem: Hardness of RSP for tuples in not fully deactivated triads	41
5.5.5 Theorem: RSP is harder than RES	41
5.5.6 Corollary: Causal Responsibility Dichotomy under set semantics	42
5.5.7 Corollary: Causal Responsibility Dichotomy under bag semantics	42
5.5.8 Corollary: Tractable cases of RES and RSP coincide under bag semantics	42
5.6.1 Theorem: Integrality of MILP[RSP*] for read-once instances	42
5.6.2 Theorem: Integrality of MILP[RSP*] with certain functional dependencies	43
5.7.1 Theorem: m-factor approximation for RSP	44
6.3.1 Proposition: Correctness of $ILP_N[GDP]$	56
6.3.2 Proposition: Correctness of $ILP_W[GDP]$	58
6.3.3 Theorem: Correctness of $ILP_S[GDP]$ (also known as $ILP[GDP]$)	60
6.4.1 Theorem: Integrality of $LP[GDP]$ for tractable instances of DP-SS	63
6.4.2 Theorem: Integrality of $LP[GDP]$ for tractable instances of DP-VS	63
6.4.3 Theorem: Integrality of $LP[GDP]$ for tractable instances of SWP	64
6.4.4 Theorem: Integrality of $LP[GDP]$ for tractable instances of ADP-SS	68
6.5.1 Proposition: New tractable deletion propagation case via $LP[GDP]$	69
7.4.1 Theorem: Factorizations and VEOs	86
7.4.2 Theorem: minFACT with VEOs	91
7.4.3 Theorem: minFACT with mveos	92
7.5.1 Theorem: ILP[minFACT] correctness	99
7.5.2 Corollary: FACT is in NP	99
7.6.1 Theorem: Running Prefixes (RP) Property	106
7.6.2 Theorem: Constant-factor approximation algorithm for minFACT	109
7.6.3 Theorem: Relationship between MFMC and LP based approaches for minFACT	109
7.7.1 Theorem: minFACT of Read-Once Instances is tractable	110
7.8.1 Theorem: minFACT of 2-MQP Queries is tractable	112
7.8.2 Proposition: Total Unimodularity of 2-MQP Queries	113
7.8.3 Lemma: Sufficient conditions for Total Unimodularity [89]	113
7.8.4 Corollary: minFACT of Hierarchical Queries is tractable	114
7.8.5 Corollary: minFACT more tractable than PQE	114
7.8.6 Theorem: minFACT is tractable for Q_A^Δ	114
7.8.7 Theorem: minFACT is tractable for Q_4^∞	117
7.8.8 Lemma: Necessary condition for leakage	117
7.9.1 Theorem: minFACT is hard for queries with Active Triads	120
7.9.2 Proposition: Connection of factorization and independent set: Direction 1	120
7.9.3 Proposition: Connection of factorization and independent set: Direction 2	121
7.9.4 Theorem: minFACT is hard for queries with Co-Deactivated Triads	122
8.2.1 Corollary: Sufficient hardness condition via Hardness Gadgets	134
8.2.2 Corollary: Complexity bound of finding Hardness Gadgets	135

Introduction

1

A core research area in databases is query optimization, which focuses on finding efficient ways to execute queries over a database. Traditional queries run over an input database (or a *source* database) and return a result table (or an *output* table). However, in a modern analytical setting, we may want to ask questions that go beyond traditional queries, such as “What changes (or interventions) to the source database would lead to a certain change in the output table?”. These questions are widely known as *Reverse Data Management* (RDM) problems [118]. They are useful in many applications, such as intervention-based approaches for explanations [72, 119, 142], fairness [28, 62, 143], causal inference [63], and data repair [158].

The focus of this dissertation is to develop a unified and generalizable Reverse Query Optimization paradigm, and to provide theoretical guarantees for the performance of the proposed methods.

1.1 Thesis Statement

Thesis

My thesis is that it is possible to solve Reverse Data Management problems with a novel paradigm:

► instead of creating dedicated algorithms for easy (PTIME) and hard cases (NP-Complete), we can devise unified algorithms that can solve all problem instances and terminate in PTIME for all currently-known easy cases.

This approach allows us to separate the task of designing efficient algorithms from the task of proving tractability, allowing us to solve problems in a “coarse-grained instance optimal” manner *i.e.*, it is possible to solve all known tractable cases in polynomial time, without needing to know in advance which cases are tractable.

Dissertation Goals. The methods developed in this dissertation focus on theoretical guarantees as well as practical considerations. In particular, to develop efficient methods that support solving Reverse Query Optimization problems in practice, we aim to develop:

- **Generalized Algorithms:** We would like the algorithms developed for RDM problems to be general to the type of input: queries with and without self-joins, unions; data with and without functional dependencies; and different semantics (set, bag). In addition, we would like the algorithms to be applicable to a class of problems, rather than a single problem- thus leading to the formulation of *Generalized Deletion Propagation*, which encapsulates many variants of deletion propagation (a family of RDM problems that has been studied since the 1980s) as special cases.
- **Theoretical Guarantees:** We would also like the algorithms to be adaptive in performance for different inputs, and to be guaranteed to terminate in polynomial time for all currently known tractable cases.

1.1 Thesis Statement	1
1.2 Unified Algorithms (Coarse-Grained Instance Optimal Algorithms) for Reverse Query Optimiza- tion	2
1.3 Outline and Contributions	4

[118]: Meliou, Gatterbauer, and Suciu (PVLDB, 2011), ‘Reverse Data Management’. doi:10.14778/3402755.3402803

[72]: Glavic, Meliou, and Roy (Foundations and Trends in Databases, 2021), ‘Trends in explanations: Understanding and debugging data-driven systems’. doi:10.1561/9781680838817

[119]: Meliou and Suciu (SIGMOD, 2012), ‘Tiresias: the database oracle for how-to queries’. doi:10.1145/2213836.2213875

[142]: Roy and Suciu (SIGMOD, 2014), ‘A Formal Approach to Finding Explanations for Database Queries’. doi:10.1145/2588555.2588578

[28]: Chen, Manolios, and Riedewald (PVLDB, 2023), ‘Why Not Yet: Fixing a Top-k Ranking that is Not Fair to Individuals’. doi:10.14778/3598581.3598606

[62]: Galhotra, Brun, and Meliou (FSE, 2017), ‘Fairness testing: testing software for discrimination’. doi:10.1145/3106237.3106277

[143]: Salimi, Rodriguez, Howe, and Suciu (SIGMOD, 2019), ‘Interventional fairness: Causal database repair for algorithmic fairness’. doi:10.1145/3299869.3319901

[63]: Galhotra, Gilad, Roy, and Salimi (SIGMOD, 2022), ‘HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach’. doi:10.1145/3514221.3526149

[158]: Wang, Meliou, and Wu (SIGMOD, 2017), ‘QFix: Diagnosing errors through query histories’. doi:10.1145/3035918.3035925

- **Efficient in Practice:** We run experiments to show that our algorithms are efficient in practice, and compare them to existing specialized algorithms. For variants of the problem that have been studied only theoretically so far, we show the first experimental results.

The goals can be achieved by developing a new philosophy of algorithm design, which we call *unified algorithms* or *coarse-grained instance optimal algorithms* (for different data management problems) *i.e.* algorithms that are guaranteed to terminate in polynomial time for all currently known tractable cases. We introduce this notion abstractly in the next section [Section 1.2](#), and then describe specific constructions for the problems we study in this dissertation in [Chapters 4 to 7](#).

Problems in Focus in this Dissertation. We prove the above thesis by developing a unified algorithmic framework for some key Reverse Data Management problems. The first problem we focus on is *Resilience* [59], a well-known database problem that can be seen as the “simplest” RDM problem. We then extend this framework to *Causal Responsibility* [117], which is a more complex RDM problem that captures the notion of counterfactual causality in databases. Finally, instead of looking at a single RDM problem at a time, we define a new family of RDM problems called *Generalized Deletion Propagation* (GDP), which captures many well-known RDM problems such as resilience [59], *deletion propagation with source side effects* [23] and *view side effects* [102], *aggregated deletion propagation* [94], and the *smallest witness problem* [121].

Interestingly, the methods we develop are not restricted to traditional RDM problems, but can also be adapted to other questions in databases, such as *provenance factorization*, which that looks for a minimum-sized formula representation of the provenance of a query result. Although this is not a traditional RDM problem, it can be seen as RDM of a different kind, where the intervention is not on the source database, but rather on the query plan used to compute the query result provenance.

1.2 Unified Algorithms (Coarse-Grained Instance Optimal Algorithms) for Reverse Query Optimization

This dissertation and accompanying research [110–112] introduces a new philosophy of algorithm design in the form of *coarse-grained instance optimal algorithms*. The term *coarse-grained* refers to the fact this complexity measure only distinguishes between tractable and NPC cases (which we can refer to as intractable cases, assuming $P! = NP$), and does not care about the precise fine-grained time complexity of the algorithm (for different data management problems) *i.e.* algorithms that are guaranteed to terminate in polynomial time for all currently known tractable cases. The motivation behind such algorithms is to automatically leverage regularities in the data without explicitly receiving them as input. The use of such unified or coarse-grained instance optimal algorithms allows a major simplification of algorithm design and implementation, while maintaining strong theoretical guarantees. We have shown unified algorithms that can solve all known polynomial time solvable cases in polynomial time for different data management problems that prior to my work had independent, case-specific specialized algorithms. These unified algorithms have been tested empirically as well, and we observed

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

[117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176

[23]: Buneman, Khanna, and Tan (PODS, 2002), ‘On Propagation of Deletions and Annotations Through Views’. doi:10.1145/543613.543633

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[121]: Miao, Roy, and Yang (SIGMOD, 2019), ‘Explaining Wrong Queries Using Small Examples’. doi:10.1145/3299869.3319866

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[111]: Makhija and Gatterbauer (PODS, 2024), ‘Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries’. doi:10.1145/3651605

[112]: Makhija and Gatterbauer (PVLDB, 2025), ‘Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach for Generalized Deletion Propagation’. doi:10.14778/3742728.3742756

that, surprisingly, they often outperform specialized algorithms that have been designed for specific problems and query classes.

We have shown unified algorithms that can recover all known tractable cases for well-known data management problems like resilience [110], causal responsibility [110], deletion propagation (with its many variants) [112], and provenance factorization [111]. The challenges posed by each of these problems are described in subsequent chapters, but this section focuses on the problem-agnostic contribution of *unified algorithms*. These algorithms are guaranteed to terminate in polynomial time for all known tractable cases. By “tractable cases” we mean classes of instances for which we know that the problem can be solved in polynomial time in data complexity, usually with specific specialized algorithms that have been discovered with prior work. These “tractable cases” are typically defined by the query i.e. a query forms a tractable case if a problem can be solved in polynomial time over any database over the given query. But our tractability criteria are not restricted to queries, and can be defined by properties of the database as well – such as the presence of functional dependencies. The unified algorithms we have developed are guaranteed to terminate in polynomial time in the presence of these tractability properties, even without specifying them in advance. For example, functional dependencies are automatically leveraged, without needing to be explicitly discovered in advance. Our current unified algorithms use Integer Linear Programming (ILP) as a common framework to express the problem, and then use the combined properties of ILP and the problem characteristics to prove that for all tractable cases, an optimal solution of the Linear Programming (LP) relaxation¹ is integral as well. We have observed that for ILPs that enjoy this property, modern ILP solvers are able to solve them efficiently without knowing in advance that these are tractable problems.

But creating a unified algorithm is not as simple as expressing the problem as an ILP. There are two main challenges: (1) finding a *right ILP* that enjoys the tractability properties we desire, and (2) proving that the ILP is indeed tractable for all known tractable cases. Picking a *right ILP* is important, as there can be many ILPs that express the same problem and hence have the same solution, but have completely different solutions to their Linear Programming relaxations, making a big difference in the performance of the algorithm [112]. We show some insight via example of a principled way to come up with the right ILP for a problem for a particular problem, by using techniques reminiscent of the cutting-plane algorithm [98] to “cut away fractional solutions”, which we believe can be generalized to other problems as well. This leaves us with the challenge of proving that the ILP is tractable for all known tractable cases – a question that is also of great interest to linear optimization researchers, and practitioners across domains. We found that for the problems we studied, existing tractability criteria for ILPs (like Total Unimodularity and Balanced Matrices) were not sufficient to prove our results; we had to come up with new criteria that were specific to the problem at hand, creating case-specific reductions to min cut problems in flow graphs. Thus while the proofs underlying our unified algorithms are complex, the algorithms themselves are simple and easy to implement, and simply work without prior knowledge of the tractability properties of the problem.

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[112]: Makhija and Gatterbauer (PVLDB, 2025), ‘Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach for Generalized Deletion Propagation’. doi:10.14778/3742728.3742756

[111]: Makhija and Gatterbauer (PODS, 2024), ‘Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries’. doi:10.1145/3651605

1: LP Relaxations arise by removing integrality constraints from variables in an ILP (thus variables are allowed to take a range of fractional values)

[112]: Makhija and Gatterbauer (PVLDB, 2025), ‘Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach for Generalized Deletion Propagation’. doi:10.14778/3742728.3742756

[98]: Kelley (JSIAM, 1960), ‘The Cutting-Plane Method for Solving Convex Programs’. doi:10.1137/0108053

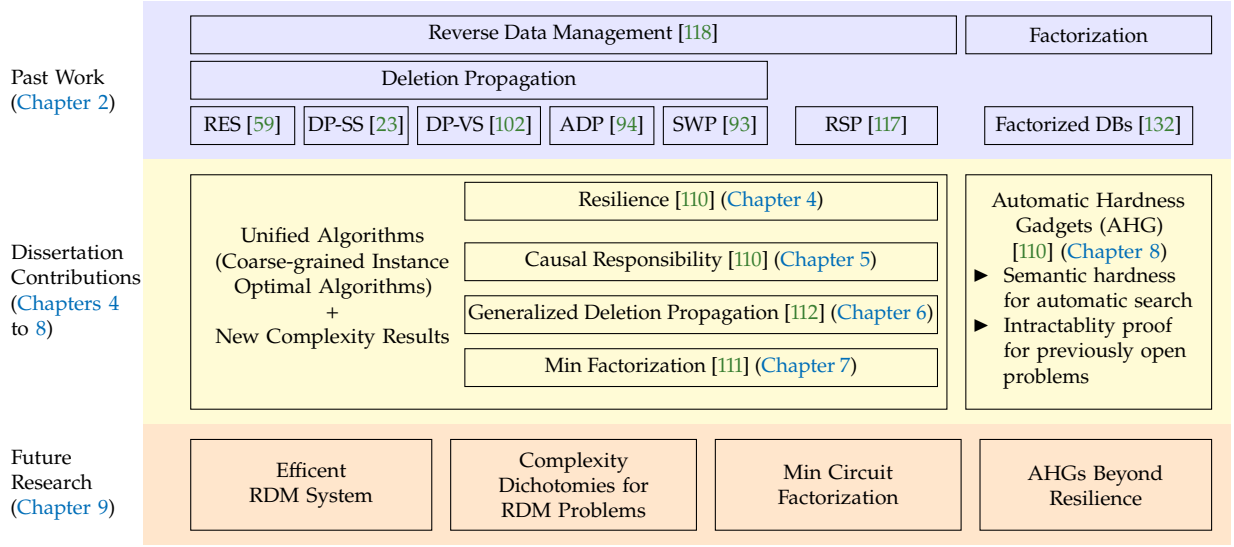


Figure 1.1: Overview of the results of the contributions of this dissertation in the context of past work and future research.

1.3 Outline and Contributions

Figure 1.1 shows the outline of this dissertation, along with a brief overview of the contributions in context of past work and future research. Chapter 2 provides the necessary background on Reverse Data Management problems, focusing on the specific problems we study in this dissertation. It also discusses additional related work and themes, such as instance optimal algorithms, intervention-based explanations, and the use of Integer Linear Programming (ILP) in databases. Chapter 3 introduces the necessary notation and definitions used in this dissertation, including standard database notation. Chapters 4 to 8 form the core of this dissertation, and the key contributions and outline is summarized below. Finally, Chapter 9 summarizes the contributions of this dissertation and discusses open conjectures and future research directions.

1.3.1 Resilience (Chapter 4)

Arguably, the simplest formulation of such RDM problems is “resilience”: What is the minimal number of tuples to delete from a database in order to eliminate all query answers? It was introduced by Freire et al. in 2015 [59], and the same work also showed a dichotomy of the complexity for resilience for self-join-free queries under set semantics. While a dichotomy for the general self-join case remains open, Freire et al. [60] gave partial complexity results in this space, and conjectured a sufficient and necessary condition for hardness of resilience. We proved this conjecture for self-join-free queries (with a slight deviation of the original specification) [110], and proved that the condition in question is a sufficient condition for hardness in queries with self-joins as well. We also show that resilience is easy under bag semantics if and only if the query is *linear* [110] - thus obtaining a dichotomy for the self-join-free case under bag semantics. Interestingly, we show that switching from set semantics to bag semantics only changes the objective function of the ILP, and the constraint matrix remains the same, however this leads to a different tractability characterization. This tractability difference

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

[60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

is *automatically* captured by our unified algorithm, without needing to know in advance that the query is tractable or the semantics used.

1.3.2 Causal Responsibility (Chapter 5)

The notion of causal responsibility is due to foundational work by Halpern, Pearl, et al. [86] which defined causal responsibility based on *minimal interventions* in the causal graph. Meliou et al. [117] adapted this concept to define causal responsibility for database queries and showed a dichotomy under set semantics for self-join-free queries. Our work proves a dichotomy under bag semantics, as well as more fine-grained result based on the relation of the tuple we find responsibility of [110]. The unified algorithm we developed for causal responsibility differs from the others in a key aspect - the LP relaxation of the unified ILP is not tight for all tractable cases. However, a certain Mixed ILP (MILP) relaxation of the ILP is tight for all tractable cases, and we show that this MILP can be solved in PTIME as well. Thus, we still do obtain a unified algorithm that is guaranteed to recover all known PTIME cases by terminating in PTIME. We show that popular ILP solvers like Gurobi do indeed show polynomial time behavior for these ILPs (with a tight tractable MILP), and thus our unified algorithm is practical as well.

[86]: Halpern and Pearl (BJPS, 2001), ‘Causes and Explanations: A Structural-Model Approach: Part 1: Causes’. doi:10.1093/bjps/axi147

[117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

1.3.3 Generalized Deletion Propagation (Chapter 6)

A contribution of this dissertation is to define *Generalized Deletion Propagation* (GDP), a new RDM problem that encapsulates many variants of *deletion propagation* (including *resilience* studied in Chapter 4) and introduces new, natural variants. Deletion propagation are well-studied problem in databases, where the goal is to find the minimal set of tuples to delete from a source database such that a query result is changed in a desired way. We can think of deletion propagation as a special case of Reverse Data Management, where the only interventions that are allowed are deletions. The GDP problem was born out of a desire to unify the many variants of deletion propagation that have been studied in isolation [23, 60, 93, 94, 102], and answer some very natural data analysis questions that have not been studied before. We show a principled way to create a unified algorithm for GDP as well, that is guaranteed to terminate in polynomial time for all known tractable cases and show experimental results that demonstrate the effectiveness of our approach. Although we use a similar ILP formulation setup for GDP as for resilience, we show an example of a problem for which “naive” ILP constructions do not have the desired tractability properties, and we had to come up with a new *smoothened* ILP formulation that did. This smoothened ILP uses insights like the use of a “wildcard” semantics and cutting plane like techniques, and leads to 2-3 orders of magnitude speedup in practice over the naive ILP formulation.

[23]: Buneman, Khanna, and Tan (PODS, 2002), ‘On Propagation of Deletions and Annotations Through Views’. doi:10.1145/543613.543633

[60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

[93]: Hu and Sintos (ICDT, 2024), ‘Finding Smallest Witnesses for Conjunctive Queries’. doi:10.4230/LIPIcs.ICDT.2024.24

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

1.3.4 Minimum Factorization of Provenance Formulas (Chapter 7)

At the heart of many RDM problems is the notion of *provenance*, which captures the origin or derivation of a query result. Provenance is a useful notion in explainable data management, and can be thought of as Boolean formula that captures the origin or derivation of a query

result. A fundamental question is: given a provenance formula, how can we succinctly represent it? At first glance, this question is very different from other RDM problems covered by this dissertation, however we see that a lot of the techniques used to develop RDM problems can be adapted to this problem as well. In fact, the problem can be seen as an extension of the RDM framework, where the query output is a provenance representation (instead of an output relation) and we perform interventions on the query plans used to create the output in order to create the smallest sized provenance representation.

If we think of provenance as an arbitrary Boolean formula given in DNF, then the problem of finding the smallest equivalent formula is equivalent to the intractable Σ_2^P -complete problem of Minimal Equivalent Expression [155]. However, when we take into account the structure that is inherent in provenance formulas, we show [111] that for all provenance formulas of self-join-free queries, not only is the problem in NP, but also there is a large class of queries for which we can provide a polynomial time algorithm to find the smallest equivalent formula. Moreover, such an algorithm is a *unified* algorithm as discussed in previous subsections, thus it can automatically detect and recover the tractable cases. Our work lies on the bedrock of formalizing a connection between factorizations, variable elimination orders and minimal query plans as used in the context of probabilistic databases. This work is not only a step in understanding a very fundamental question about tractability of Boolean formula factorization, but we show experimentally that it leads to better probabilistic inference approximations as well.

[155]: Umans (JCSS, 2001), ‘The minimum equivalent DNF problem and shortest implicants’. doi:10.1109/sfcs.1998.743506

[111]: Makhija and Gatterbauer (PODS, 2024), ‘Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries’. doi:10.1145/3651605

1.3.5 Automatic Hardness Gadget-finder for Resilience problems (Chapter 8)

A symmetric problem to developing unified algorithms for RDM problems is to prove the complexity class for these problems. For problems that are known to be NP-Hard, it is believed that no polynomial time algorithm exists to solve them. Thus, when we encounter a problem for which we cannot show that our unified algorithm terminates in polynomial time, we would like to show that the problem is NP-Hard (and thus the problem cannot be solved in polynomial time unless $P = NP$). A typical approach to proving hardness results for a problem is to reduce from a known NP-Hard problem. These reductions are often done by hand, and require a deep understanding of the problem at hand. For the earlier discussed problem of *resilience*, hardness results for different queries are often different from one another, and complex in their own ways. Since this process is complex, the general tractability is open and hardness results for many queries are unknown, and it is not clear how to obtain them. Instead of doing these reductions by hand, we show [110] that we can automatically generate hardness reductions for a wide range of queries, by leveraging the structure of the query and the database. In our method, we identified and declaratively specified a set of sufficient properties that a hardness reduction must satisfy, and then used a Disjunctive Logic Programming (DLP) solver to automatically generate the reductions that satisfied these declaratively specified semantic hardness properties. We were able to generate hardness reductions for 5 out of 8 queries for which hardness results were previously unknown (2 of these queries were independently shown to be tractable later).

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

Background and Related Work

2

We begin by a brief review of work in Reverse Data Management (RDM) and discuss problems that are the focus of this dissertation - resilience, causal responsibility, and deletion propagation variants, and minimal factorization of provenance formulas. This section only gives a brief overview of the problems in focus, with the formal definitions and details are given in the respective chapters. In [Section 2.2](#) we discuss related work on some broader themes that are closely tied to this dissertation, such as explanations, fairness, and linear optimization. Related work that is more specifically tied to one chapter is discussed in the respective chapters (such as related work on factorization in [Chapter 7](#) or automatic hardness proofs in [Chapter 8](#)).

2.1 Problems in Focus in this Dissertation

Reverse Data Management (RDM). The term *Reverse Data Management* (RDM) [118] was proposed in 2011, but examples of reverse data management problems have been studied in databases since the 1980s [44]. In these problems one wants to achieve a certain effect in the output data, and needs to act on the input (source) data in order to achieve that effect.

Definition 2.1.1 (Reverse Data Management (RDM)) *Given a source database \mathcal{D} and a query Q , a desired output property P , and an optional objective function $f(Q, \mathcal{D}, \Gamma)$, an RDM problem is to find a set of interventions Γ on \mathcal{D} that produce a new database \mathcal{D}' such that $Q(\mathcal{D}')$ satisfies P and $f(Q, \mathcal{D}, \Gamma)$ is optimized.*

RDM problems are useful in many applications, such as intervention-based approaches for explanations [72, 91, 142, 161], fairness [62, 143], causal inference [63], and data repair [158]. The Tiresias system [119] solves how-to problems, a type of reverse data management problem, using Mixed Integer Linear Programming (MILP). However, its focus is on building the semantics of a query language for how-to problems that can be translated to an MILP, unlike this dissertation, which focuses on building a unified method that can recover tractable cases.

Deletion Propagation (DP) and View-Update. A particular class of RDM problems that we focus on this dissertation are Deletion Propagation problems. Intuitively, one can think of deletion propagation as RDM problems where the only interventions permitted are deletion. The problem of *deletion propagation* seeks to delete a set of tuples in the input tables in order to delete a particular tuple from the view. Intuitively, this deletion should be achieved with minimal side effects, where side effects were initially commonly defined with either of two objectives: (a) deletion propagation with *source side effects* seeks a minimum set of input tuples Γ in order to delete a given output tuple; whereas (b) deletion propagation with *view side effects* seeks a set of input tuples that results in a minimum number of output tuple deletions in the view, other than the tuple of interest [23]. The resilience problem [59, 60] is a variant of the deletion propagation problem with source-side effects focusing on

2.1 Problems in Focus in this Dissertation	7
2.2 Related Work and Themes	8

- [118]: Meliou, Gatterbauer, and Suciu (PVLDB, 2011), ‘Reverse Data Management’. doi:10.14778/3402755.3402803
- [44]: Dayal and Bernstein (TODS, 1982), ‘On the Correct Translation of Update Operations on Relational Views’. doi:10.1145/319732.319740
- [72]: Glavic, Meliou, and Roy (Foundations and Trends in Databases, 2021), ‘Trends in explanations: Understanding and debugging data-driven systems’. doi:10.1561/9781680838817
- [91]: Herschel, Hernández, and Tan (PVLDB, 2009), ‘Artemis: A System for Analyzing Missing Answers’. doi:10.14778/1687553.1687588
- [142]: Roy and Suciu (SIGMOD, 2014), ‘A Formal Approach to Finding Explanations for Database Queries’. doi:10.1145/2588555.2588578
- [161]: Wu and Madden (PVLDB, 2013), ‘Scorpion: Explaining Away Outliers in Aggregate Queries’. doi:10.14778/2536354.2536356
- [62]: Galhotra, Brun, and Meliou (FSE, 2017), ‘Fairness testing: testing software for discrimination’. doi:10.1145/3106237.3106277
- [143]: Salimi, Rodriguez, Howe, and Suciu (SIGMOD, 2019), ‘Interventional fairness: Causal database repair for algorithmic fairness’. doi:10.1145/3299869.3319901
- [63]: Galhotra, Gilad, Roy, and Salimi (SIGMOD, 2022), ‘HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach’. doi:10.1145/3514221.3526149
- [158]: Wang, Meliou, and Wu (SIGMOD, 2017), ‘QFix: Diagnosing errors through query histories’. doi:10.1145/3035918.3035925
- [119]: Meliou and Suciu (SIGMOD, 2012), ‘Tiresias: the database oracle for how-to queries’. doi:10.1145/2213836.2213875
- [23]: Buneman, Khanna, and Tan (PODS, 2002), ‘On Propagation of Deletions and Annotations Through Views’. doi:10.1145/543613.543633
- [59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592
- [60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

Boolean queries. It can be considered the simplest RDM problem, and we discuss it in detail in [Chapter 4](#). There are several variants to resilience such as destroying a pre-specified fraction of witnesses from the database instead of all witnesses [94]. We introduce a more general problem formulation in [Chapter 6](#) that captures all prior deletion propagation variants as a special case.

Causal Responsibility. Foundational work by Halpern, Pearl, et al. [31, 86, 87] defined the concept of *causal responsibility* based *minimal interventions* in the input. Meliou et al. [117] adapted this concept to define causal responsibility for database queries and proposed a flow algorithm to solve the tractable cases. The problem is closely related to resilience, and can be thought of as finding resilience for a specific tuple, with some additional constraints (that relate to making the tuple counterfactual). Their tractable cases are closely related as well - In fact in [Chapter 5](#), we show that under bag semantics, resilience is intractable for a self-join-free query iff causal responsibility is intractable for that query.

Factorization of boolean Provenance formulas. Implicit in solving RDM problems is a notion of provenance, that captures how the output is produced from the input. Provenance is represented as a boolean formula under the semiring framework [78]. A natural question is about how this provenance can be succinctly represented *i.e.* what is the size of the smallest Boolean formula that captures the provenance of a given database under a query? Surprisingly, much of ideas we used to solve RDM problems can apply in this setting (although resulting in far more complicated proofs). Interestingly, even the complexity characterization of this problem and that of resilience have some connections, and we discuss these in [Chapter 7](#).

2.2 Related Work and Themes

Explanations and fairness in Data Management. Data management research has recognized the need to derive *explanations* for query results and surprising observations [72]. Existing work on explanations uses many approaches [108]. This includes the approach of modifying the input (*i.e.* performing *interventions*) [91, 95, 116, 117, 142, 161], which is our focus as well. Recent approaches show that explanations may benefit a variety of applications, such as ensuring or testing fairness [62, 134, 143] or finding bias [163]. We believe our unified algorithms that solve both easy and hard cases with one algorithm will also find applications for these applications.

Intervention-Based Explanations in AI. Formal Explainability in AI (FXAI) [113] distinguishes between two types of explanations: *Abductive* explanations (or locally sufficient reasons [12]) identify a minimal subset of features that, when fixed to their original values, are sufficient to *guarantee the original prediction*. They are also known as ‘Why?’ explanations as they explain why a prediction is the way it is. *Contrastive* explanations identify a minimal subset of features that, when altered from their original values, are sufficient to *change the original prediction*. They are also known as ‘Why not?’ explanations as they explain why the prediction is not different from what it is. These notions also extend to relational query explanations, and we can interpret the Smallest Witness Problem (SWP) [93, 121] as an instance of abductive explanation, and the Resilience Problem (RES) [110] as a contrastive explanation. Generalized Deletion Propagation (GDP) subsumes both SWP and RES and can give

- [94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892
- [31]: Chockler and Halpern (JAIR, 2004), ‘Responsibility and Blame: A Structural-Model Approach’. doi:10.1613/jair.1391
- [86]: Halpern and Pearl (BJPS, 2001), ‘Causes and Explanations: A Structural-Model Approach: Part I: Causes’. doi:10.1093/bjps/axi147
- [87]: Halpern and Pearl (BJPS, 2005), ‘Causes and Explanations: A structural-model Approach. Part II: Explanations’. doi:10.1093/bjps/axi148
- [117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176
- [72]: Glavic, Meliou, and Roy (Foundations and Trends in Databases, 2021), ‘Trends in explanations: Understanding and debugging data-driven systems’. doi:10.1561/9781680838817
- [108]: Lim, Dey, and Avrahami (CHI, 2009), ‘Why and why not explanations improve the intelligibility of context-aware intelligent systems’.
- [91]: Herschel, Hernández, and Tan (PVLDB, 2009), ‘Artemis: A System for Analyzing Missing Answers’. doi:10.14778/1687553.1687588
- [95]: Huang, Chen, Doan, and Naughton (PVLDB, 2008), ‘On the provenance of non-answers to queries over extracted data’. doi:10.14778/1453856.1453936
- [116]: Meliou, Gatterbauer, Moore, and Suciu (MUD, 2009), ‘Why so? or Why no? Functional Causality for Explaining Query Answers’. doi:10.48550/arxiv.0912.5340
- [117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176
- [142]: Roy and Suciu (SIGMOD, 2014), ‘A Formal Approach to Finding Explanations for Database Queries’. doi:10.1145/2588555.2588578
- [161]: Wu and Madden (PVLDB, 2013), ‘Scorpion: Explaining Away Outliers in Aggregate Queries’. doi:10.14778/2536354.2536356
- [62]: Galhotra, Brun, and Meliou (FSE, 2017), ‘Fairness testing: testing software for discrimination’. doi:10.1145/3106237.3106277
- [134]: Pradhan, Zhu, Glavic, and Salimi (SIGMOD, 2022), ‘Interpretable data-based explanations for fairness debugging’. doi:10.1145/3514221.3517886
- [143]: Salimi, Rodriguez, Howe, and Suciu (SIGMOD, 2019), ‘Interventional fairness: Causal database repair for algorithmic fairness’. doi:10.1145/3299869.3319901
- [163]: Youngmann, Cafarella, Moskovitch, and Salimi (ICDE, 2023), ‘On Explaining Confounding Bias’. doi:10.1109/icde55515.2023.00144
- [113]: Marques-Silva (Reasoning Web Summer School, 2023), ‘Logic-Based Explainability in Machine Learning’. doi:10.1007/978-3-031-31414-8_2

both *abductive* and *contrastive* explanations in the same framework. Notice that ‘Why’ and ‘Why not’ explanations have been understood differently in the context of database provenance [116, 117]: ‘Why’ has been used to understand why a given tuple is in the output (a ‘prediction’ is true) whereas ‘why not’ to understand why a tuple is not in the output (a ‘prediction’ is false).

Bag semantics. Real-world databases consist of bags instead of sets *i.e.* tuples may be duplicated in the input and output relations. This gap between database theory and database practice has been pointed out years ago [27]. However, studying properties of CQs under bag semantics is often considerably harder. For example, the connection between local and global consistency has only been recently solved for bags [11, 162], and the fundamental problems of query containment of CQs under bag semantics remain open despite recent progress [99, 105]. This dissertation and accompanying work gives the first dichotomy result for any reverse data management problems under bag semantics (showing dichotomies for self-join-free queries for the problems of resilience and causal responsibility).

Instance Optimal Algorithms. Our notion of “*coarse-grained instance optimality*” is inspired by the notion of instance optimality in complexity theory [140]. Instance optimal algorithms are defined as those that for every input perform better than every other correct algorithm (up to a constant factor) [140]. Problem-specific requirements of a “any correct algorithm” are used to define lower bounds for instance optimality. It is a very strong notion of optimality, and is often not achievable in practice. However, the need for instance optimality or beyond worst case complexity analysis has been increasingly recognized since worst-case complexity analysis can be overly pessimistic and fails to capture the efficient real world performance of many algorithms such as in ILP optimization and machine learning. Instance optimal algorithms have also been sought for some problems in databases such as top- k score aggregation [54], and join computation [3, 7, 100, 125].

Holistic Join Algorithms or Worst-Case Optimal Algorithms. Our approach has an interesting conceptual connection to “holistic” join algorithms [4] that rely on not just a single tree decomposition (thus one query plan) but rather multiple tree decompositions (thus multiple plans) for different output tuples. We see a similarity in our approach, where we choose between a different intervention for each input tuple. This parallel is most closely seen in the case of *minimal factorization of provenance formulas* (Chapter 7), where we assign different query plans to different witnesses.

- [12]: Bassan, Amir, and Katz (PMLR, 2024), ‘Local vs. Global Interpretability: A Computational Complexity Perspective’. doi:10.5555/3692070.3692196
- [93]: Hu and Sintos (ICDT, 2024), ‘Finding Smallest Witnesses for Conjunctive Queries’. doi:10.4230/LIPICs.ICDT.2024.24
- [121]: Miao, Roy, and Yang (SIGMOD, 2019), ‘Explaining Wrong Queries Using Small Examples’. doi:10.1145/3299869.3319866
- [110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715
- [116]: Meliou, Gatterbauer, Moore, and Suciu (MUD, 2009), ‘Why so? or Why no? Functional Causality for Explaining Query Answers’. doi:10.48550/arxiv.0912.5340
- [117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176
- [27]: Chaudhuri and Vardi (PODS, 1993), ‘Optimization of real conjunctive queries’. doi:10.1145/153850.153856
- [11]: Atserias and Kolaitis (SIGMOD Rec. 2022), ‘Structure and complexity of bag consistency’. doi:10.1145/3542700.3542719
- [162]: Yannakakis (SIGMOD Rec. 2022), ‘Technical Perspective: Structure and Complexity of Bag Consistency’. doi:10.1145/3542700.3542718
- [99]: Khamis, Kolaitis, Ngo, and Suciu (TODS, 2021), ‘Bag Query Containment and Information Theory’. doi:10.1145/3472391
- [105]: Konstantinidis and Mogavero (PODS, 2019), ‘Attacking Diophantus: Solving a Special Case of Bag Containment’. doi:10.1145/3294052.3319689
- [140]: Roughgarden (2020), *Beyond the Worst-Case Analysis of Algorithms*. doi:10.1017/9781108637435
- [54]: Fagin, Lotem, and Naor (PODS, 2001), ‘Optimal aggregation algorithms for middle-ware’. doi:10.1145/375551.375567
- [3]: Abo-Khamis, Im, Moseley, Pruhs, and Samadian (Procedia Computer Science, 2021), ‘Instance Optimal Join Size Estimation’. doi:10.1016/j.procs.2021.11.019
- [7]: Alway, Blais, and Salihoglu (ICDT, 2021), ‘Box Covers and Domain Orderings for Beyond Worst-Case Join Processing’. doi:10.4230/LIPICs.ICDT.2021.3
- [100]: Khamis, Ngo, Ré, and Rudra (TODS, 2016), ‘Joins via Geometric Resolutions: Worst Case and Beyond’. doi:10.1145/2967101
- [125]: Ngo, Nguyen, Re, and Rudra (PODS, 2014), ‘Beyond worst-case analysis for joins with minesweeper’. doi:10.1145/2594538.2594547
- [4]: Abo Khamis, Ngo, and Suciu (PODS, 2017), ‘What Do Shannon-Type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?’. doi:10.1145/3034786.3056105

This chapter introduces notation and background that is used throughout this dissertation. A summary of the notation is also given in the appendix (Chapter A). Section 3.2 gives a brief overview of the provenance framework as used in this dissertation. Section 3.3 gives important background on linear programming, which is a key tool we use to model and solve the problems in this dissertation. We discuss notions from polyhedral theory that are instrumental for some results in this dissertation.

3.1 Standard Notations and Definitions	10
3.2 Provenance Framework and Notation	11
3.3 Background on Linear Programming	12

3.1 Standard Notations and Definitions

We write \mathcal{D} for a database, i.e. the set of tuples in the relations.

A *conjunctive query* (CQ) is a first-order formula $Q(\mathbf{y}) = \exists \mathbf{x} (g_1 \wedge \dots \wedge g_m)$ where the variables $\mathbf{x} = (x_1, \dots, x_\ell)$ are called existential variables, \mathbf{y} are called the head or free variables, and each atom g_i represents a relation $g_i = R_{j_i}(\mathbf{x}_i)$ where $\mathbf{x}_i \subseteq \mathbf{x} \cup \mathbf{y}^1$. W.l.o.g., we discuss only connected queries (results for disconnected queries typically follow immediately by factorizing each of the query components *independently*.)

A *self-join-free* (sj-free) CQ is one where no relation symbol occurs more than once and thus every atom represents a different relation. Thus, for sj-free CQs, one may refer to atoms and relations interchangeably.

A Boolean CQ is a CQ with no free variables, i.e. $\mathbf{y} = \emptyset$. Notice that a query has at least one output tuple iff the Boolean variant of the query (obtained by making all the free variables existential) is true. We write $D \models Q$ to denote that that query Q evaluates to true over database instance \mathcal{D} , and $D \not\models Q$ to denote it evaluates to false.

We write $\text{var}(X)$ for the set of variables occurring in atom/ relation/ query/ formula X and $\text{at}(x)$ for the set of atoms that contain variable x . We write $[\mathbf{w}/\mathbf{x}]$ as a valuation (or substitution) of query variables \mathbf{x} by constants \mathbf{w} . These substitutions may be written explicitly by “domain-annotating” variables with domain constants as subscripts. *Domain-annotated tuples* use such domain-annotated variables as subscripts, e.g. r_{x_1, y_2} represents a tuple of relation $R(x, y)$ with $x = 1$ and $y = 2$. We sometimes informally omit the variables and use the notation $r_{v_1 v_2 \dots v_a}$ where $v_1 v_2 \dots v_a$ are the domain values of $\text{var}(R)$ in the order that they appear in atom R . Thus, r_{12} also represents $R(1, 2)$.

Queries are interpreted as hypergraphs with edges formed by atoms and nodes by variables. Two hyperedges are connected if they share at least one node. We use concepts like paths and reachable nodes on the hypergraph of a query in the usual sense [18].

A *witness* \mathbf{w} is a valuation of \mathbf{x} that is permitted by \mathcal{D} and that makes Q true (i.e. $D \models Q[\mathbf{w}/\mathbf{x}]$).^{*} with \mathbf{w}/\mathbf{x} representing substitution of \mathbf{x} by \mathbf{w} . The set of witnesses is then

$$\text{witnesses}(Q, D) = \{\mathbf{w} \mid D \models Q[\mathbf{w}/\mathbf{x}]\}.$$

1: W.l.o.g., we assume that \mathbf{x}_i is a tuple of only variables and don’t write the constants. Selections can always be directly pushed into the database before executing the query. In other words, for any constant in the query, we can first apply a selection on each relation and then consider the modified query with a column removed. In other words, for any constant in the query, we can first apply a selection on each relation and then consider the modified query with a column removed.

[18]: Bollobás (1998), *Modern graph theory*.
doi:10.1007/978-1-4612-0619-4

^{*} Note that our notion of witness slightly differs from the one used in provenance literature where a “witness” refers to a subset of the input database records that is sufficient to ensure that a given output tuple appears in the result of a query [30].

Since every witness implies exactly one set of up to m tuples from \mathcal{D} that make the query true, we will slightly abuse the notation and also refer to this set of tuples as “witnesses.” For example, consider the 2-chain query $Q_2^\infty := R(x, y), S(y, z)$ over the database $D = \{r_{12}: R(1, 2), s_{23}: S(2, 3), s_{24}: S(2, 4)\}$. Then the witnesses(Q_2^∞, D) = $\{(1, 2, 3), (1, 2, 4)\}$ and their respective tuples (also henceforth referred to as witnesses) are $\{r_{12}, s_{23}\}$, and $\{r_{12}, s_{24}\}$. A set of witnesses may be represented as a connected hypergraph (called the witness hypergraph), where tuples are the nodes of the graph and each witness is a hyperedge around a set of tuples.

Computational Complexity Measure. Unless otherwise specified, all complexity results in this dissertation are focused on the *data complexity* [156] of RDM problems in question *i.e.* the complexity of the problem as the size of the data \mathcal{D} increases but the size of the query Q remains fixed. We refer to $\text{RES}(Q)$, $\text{RSP}(Q)$, $\text{GDP}(Q)$, $\text{FACT}(Q)$ to discuss the complexity of the problems of query Q over an arbitrary input instance.

[156]: Vardi (STOC, 1982), ‘The Complexity of Relational Query Languages (Extended Abstract)’. doi:10.1145/800070.802186

3.2 Provenance Framework and Notation

The terms provenance and lineage are used in the literature with slightly different meanings. While lineage was originally formalized in [37], we follow the modern treatment of *data provenance* as denoting a proposition formula that corresponds to the *Boolean provenance semiring* of Green et al. [78, 79], which is the commutative semiring of positive Boolean expressions ($\mathbb{B}[X]$, $\vee, \wedge, 0, 1$). We sometimes write \vee as semiring-plus (\oplus) and \wedge as times (\otimes).

We assign to every tuple $t \in D$ a *provenance token*, *i.e.* we interpret each tuple as a Boolean variable. Then the provenance formula (equivalently, provenance expression) φ_p of a query $Q := R_1(x_1), \dots, R_m(x_m)$ on \mathcal{D} is the positive Boolean DNF formula

$$\text{Prov}(Q, D) = \bigvee_{\theta: D \models Q[\theta(x)/x]} R_1(\theta(x_1)) \wedge \dots \wedge R_m(\theta(x_m))$$

where $D \models Q[\theta(x)/x]$ denotes that $\theta(x)$ is a *valuation* or assignment of x to constants in the active domain that make the query true over database \mathcal{D} . Notice that for sj-free queries, this DNF is always m -partite as each disjunct contains one tuple from each of the m tables and that the notions of provenance polynomial and provenance formula are interchangeable.

Read-once. For a formula φ , we denote by $\text{var}(\varphi)$ the set of variables that occur in φ , and by $\text{len}(\varphi)$ its length, *i.e.*, the number of its literals.[†] A provenance is called *read-once* if it can be represented in *read-once* form, *i.e.* there is an equivalent formula in which each literal appears exactly once [74, 85, 127]. This is possible iff that equivalent formula can be built up recursively from the provenance tokens by disjunction (and conjunction), *s.t.* whenever $\varphi = \varphi_1 \vee \varphi_2$ (or $\varphi = \varphi_1 \wedge \varphi_2$), then $\text{var}(\varphi_1) \cap \text{var}(\varphi_2) = \emptyset$.

Witnesses (Additional Provenance-based Formalism). We saw before that a *witness* \mathbf{w} a valuation of all variables \mathbf{x} that is permitted by \mathcal{D} and

[37]: Cui, Widom, and Wiener (TODS, 2000), ‘Tracing the lineage of view data in a warehousing environment’. doi:10.1145/357775.357777

[78]: Green, Karvounarakis, and Tannen (PODS, 2007), ‘Provenance semirings’. doi:10.1145/1265530.1265535

[79]: Green and Tannen (PODS, 2017), ‘The Semiring Framework for Database Provenance’. doi:10.1145/3034786.3056125

[74]: Golumbic and Gurvich (2011), ‘Read-once functions’. doi:10.1017/cbo9780511852008.011

[85]: Gurvich (Uspekhi Mat. Nauk (in Russian), 1977), ‘Repetition-free Boolean functions’.

[127]: Olteanu and Huang (SUM, 2008), ‘Using OBDDs for efficient query evaluation on probabilistic databases’. doi:10.1007/978-3-540-87993-0_26

[†] Notice that the length of a Boolean expression φ is also at times defined as the total number of symbols (including operators and parentheses, *e.g.* in [35]). In our formulation, we only care about the number of variable occurrences.

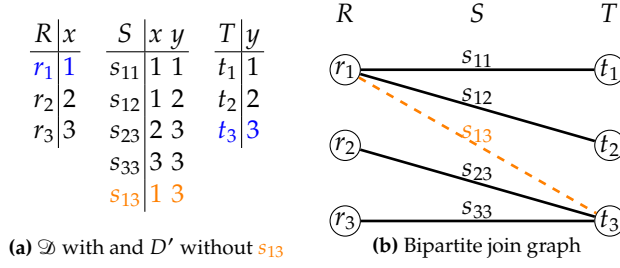


Figure 3.1: Example 3.2.1: (a): Database instance with provenance tokens to the left of each tuple, e.g. s_{12} for $S(1, 2)$. (b): $\text{Prov}(Q_2^*, D)$ for $Q_2^* := R(x), S(x, y), T(y)$ represented as bipartite graph. \mathcal{D} denotes the database with the orange tuple s_{13} and D' denotes the database without it.

that makes Q true (i.e. $D \models Q[\mathbf{w}/\mathbf{x}]$). Since every witness implies exactly one set of $\leq m$ tuples from \mathcal{D} that make the query true, we also refer to this set of tuples as a “witness.” We will also slightly abuse notation and use “witness” to refer to a product term in a DNF of the provenance polynomial.

Example 3.2.1 (Provenance) Consider the Boolean 2-star query $Q_2^* := R(x), S(x, y), T(y)$ over the database D' in Figure 3.1 (ignore the tuple s_{13} for now). Each tuple is annotated with a Boolean variable (or provenance token) r_1, r_2, \dots . The *provenance* φ_p is the Boolean expression about which tuples need to be present for Q_2^* to be true:

$$\varphi_p = r_1 s_{11} t_1 \vee r_1 s_{12} t_2 \vee r_2 s_{23} t_3 \vee r_3 s_{33} t_3 \quad (3.1)$$

This expression contains $|\text{var}(\varphi_p)| = 10$ variables, however has a length of $\text{len}(\varphi_p) = 12$ because variables r_1 and t_3 are repeated 2 times each. The witnesses are $\text{witnesses}(Q_2^*, D') = \{(1, 1), (1, 2), (2, 3), (3, 3)\}$ and their respective tuples are $\{r_1, s_{11}, t_1\}$, $\{r_1, s_{12}, t_2\}$, $\{r_2, s_{23}, t_3\}$, and $\{r_3, s_{33}, t_3\}$.

The provenance can be re-factored into a read-once factorization φ' which is a factorized representation of the provenance polynomial in which every variable occurs once, and thus $\text{len}(\varphi') = |\text{var}(\varphi')| = 10$. It can be found in PTIME in the size of the database [75]:

$$\varphi' = r_1 (s_{11} t_1 \vee s_{12} t_2) \vee (r_2 s_{23} \vee r_3 s_{33}) t_3$$

[75]: Golumbic, Mintz, and Rotics (JDAM, 2006), ‘Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees’. doi:10.1016/j.dam.2005.09.016

3.3 Background on Linear Programming

We use *Integer Linear Programs* and their *relaxations* to model and solve the problems in this dissertation. This section introduces the background knowledge on linear programs, their complexity, and how they are used in practice.

Linear Programs (LP). Linear Programs are standard optimization problems [1, 144] in which the objective function and the constraints are linear. A standard form of an LP is $\min \mathbf{c}^\top \mathbf{x}$ s.t. $\mathbf{W}\mathbf{x} \geq \mathbf{b}$, where \mathbf{x} denotes the variables, the vector \mathbf{c}^\top denotes weights of the variables in the objective, the matrix \mathbf{W} denotes the weights of \mathbf{x} for each constraint, and \mathbf{b} denotes the right-hand side of each constraint. If the variables are constrained to be integers, the resulting program is called an Integer Linear Program (ILP), while a program with some integral variables is referred to as a Mixed Integer Linear Program (MILP). The *LP relaxation* of an ILP program is obtained by removing the integrality constraint for all variables.

[1]: Aardal, Nemhauser, and Weismantel (2005), *Handbooks in Operations Research and Management Science: Discrete Optimization*. doi:10.1016/s0927-0507(05)x1200-2
[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

Polyhedral Results in Linear Optimization. A linear program $\min\{\mathbf{c}^\top \mathbf{x} \mid \mathbf{W}\mathbf{x} \geq \mathbf{b}\}$ can also be viewed as the problem of minimizing a linear function $\mathbf{c}^\top \mathbf{x}$ over a polyhedron P defined by the inequalities $\mathbf{W}\mathbf{x} \geq \mathbf{b}$ [144]. The polyhedron P is called the *feasible region*, and any vector in P a *feasible solution*. The function $\mathbf{c}^\top \mathbf{x}$ is called the *objective function* and can be interpreted as a direction vector in the polyhedron. Any feasible solution attaining the optimum objective value is called an *optimum solution*. A *face* F of a polyhedron P is a set of optimum solutions of $\min\{\mathbf{c}^\top \mathbf{x} \mid \mathbf{W}\mathbf{x} \geq \mathbf{b}\}$ for some $\mathbf{c} \in \mathbb{R}^n$. For a given objective function $\mathbf{c}^\top \mathbf{x}$, we refer to the face of P that contains all optimum solutions as the *optimal face* of P . An *extreme point* or a *vertex* or *corner point* of a polyhedron P is a point in P that cannot be expressed as a convex combination of other points in P . A polytope is a bounded polyhedron. A polytope is an *integer polytope* if all its vertices (or extreme points) have integer coordinates.

Complexity of solving LPs. The question of when an Integer Linear Program (ILP) is tractable has many theoretical and practical consequences [34]. Since we model our problem as an ILP, we can leverage some known results for ILPs to evaluate the complexity of our problem. Solving ILPs is NPC and part of Karp’s 21 problems [97], while LPs can be solved in PTIME with Interior Point methods [32, 80]. The complexity of MILPs is exponential in the number of integer variables. However, there are conditions under which ILPs become tractable. In particular, if the LP relaxation of an ILP creates an integer polytope, then the ILP can be solved in PTIME. In particular, if there is an optimal integral assignment to the LP relaxation, then the original ILP can be solved in PTIME as well. This can be done by using *crossover methods* that can find an extreme point solution from an interior point solution in PTIME [67, 115]. A lot of work studies conditions under which Linear Programs have integer polytopes [2, 34, 58, 106, 144]. A famous example is the max-flow min-cut problem which can be solved with LP relaxations despite integrality constraints. The *max-flow Integrality Theorem* [144, Theorem 5.22] states that a flow graph always has an integer max flow (or integral max-cut) [58], for any integral capacities on the edges. This is a sufficient condition for a set of inequalities to have *total dual integrality (TDI)* and Edmonds-Giles showed that TDI systems always have integral polytopes [48]. There are many other structural characteristics that define when the LP is guaranteed to have an integer polytope, and thus where ILPs are in PTIME. For example, if the constraint matrix of an ILP is *Totally Unimodular* [144] then the LP always has the same optima. Similarly, if the constraint matrix is *Balanced* [33], several classes of ILPs are PTIME. We use the results of Balanced Matrices to show that the resilience and responsibility of any read-once data instances can be found in PTIME (Section 5.6). However, for other PTIME cases, we have ILP constraint matrices that do not fit into any previous *structural* tractability characterization. In such situations, we are able to use existing results indirectly (by showing an intermediate flow representation that accurately represents the ILP) to show that the polytope defined by the ILP is an integer polytope, and thus the ILP can be solved in PTIME.

We also notice that one need not enforce that the entire polytope is integral to solve the ILP in PTIME. If we can show that for a given objective function, the optimal face of the polytope has integer vertices, then we can still apply the crossover methods to find an integral solution in PTIME. We use this idea to show the tractability of additional problems in this dissertation, particularly in settings in which the complexity of a problem is different under set and bag semantics (for example in Theorem 4.6.2). Throughout this dissertation, we will say that an *LP*

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

[34]: Cornuéjols and Guenin (JDM, 2002), ‘Ideal clutters’. doi:10.1016/S0166-218X(01)00344-4

[97]: Karp (Complexity of Computer Computations, 1972), ‘Reducibility among combinatorial problems’. doi:10.1007/978-1-4684-2001-2_9

[32]: Cohen, Lee, and Song (JACM, 2021), ‘Solving linear programs in the current matrix multiplication time’. doi:10.1145/3424305

[80]: Grötschel, Lovász, and Schrijver (1993), ‘The Ellipsoid Method’. doi:10.1007/978-3-642-78240-4_4

[67]: Ge, Wang, Xiong, and Ye (INFORMS, 2025), ‘From an Interior Point to a Corner Point: Smart Crossover’. doi:10.1287/ijoc.2022.0291

[115]: Megiddo (ORSA JOC, 1991), ‘On Finding Primal- and Dual-Optimal Bases’. doi:10.1287/ijoc.3.1.63

[34]: Cornuéjols and Guenin (JDM, 2002), ‘Ideal clutters’. doi:10.1016/S0166-218X(01)00344-4

[58]: Ford and Fulkerson (CJM, 1956), ‘Maximal flow through a network’. doi:10.4153/cjm-1956-045-5

[106]: Lau, Ravi, and Singh (2011), *Iterative methods in combinatorial optimization*. doi:10.1017/cbo9780511977152

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

[58]: Ford and Fulkerson (CJM, 1956), ‘Maximal flow through a network’. doi:10.4153/cjm-1956-045-5

[48]: Edmonds and Giles (1977), ‘A Min-Max Relation for Submodular Functions on Graphs’. doi:10.1016/S0167-5060(08)70734-9

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

[33]: Conforti, Cornuéjols, and Vušković (JDM, 2006), ‘Balanced matrices’. doi:10.1016/j.disc.2005.12.033

solves a problem exactly, or that *an LP is integral*, if we can prove that the optimal face of the LP polytope has integer vertices (sometimes after a “presolve” phase - some preprocessing heuristics applied by ILP solvers [84]), and hence the ILP can be solved in PTIME.

Linear Optimization Solvers. A key advantage of modeling problems as ILPs is of a practical nature. There are many highly-optimized ILP solvers, both commercial [83] and free [123] which can obtain exact results fast, in practice. ILP formulations are standardized, and thus programs can easily be swapped between solvers. Any advances made over time by these solvers (improvements in the presolve phase, heuristics, and even novel techniques) can automatically make implementations of these problems better over time. As such, ILPs are a powerful tool that have been used in numerous fields, including databases.

For our experimental evaluation we use Gurobi.[‡] Gurobi uses an LP based branch-and-bound method to solve ILPs and MILPs [81]. Gurobi uses many heuristics to speed up the search for an optimal solution, including a presolve phase that simplifies the problem before solving it [84]. This means that it first computes an LP relaxation bound and then explores the search space to find integral solutions that move closer to this bound. If an integral solution is encountered (either through the simplex method that finds extreme points, or via crossover methods) that has the same objective function value as an LP relaxation optimum solution, then the solver has found a guaranteed optimal solution, and it does not need to complete its exhaustive search of the space. Through many examples throughout this dissertation, we see that whenever we can prove that the optimal face of the LP polytope has integer vertices, then we observe that our original ILP formulation solved via Gurobi terminates in PTIME even without changing the formulation or letting the solver know anything about the theoretical complexity.

ILPs and Constraint Optimization in Databases. Integer Linear Programming has been used in databases for problems such as in solving package queries [20], query optimization [153], and general optimization applications [149]. However, other than our recent work on the resilience problem [110], we are unaware of any work in databases that uses ILPs to automatically recover tractable cases by proving that the LP relaxation polytope has an optimal face with integer vertices and thus the original ILP problem can be solved in guaranteed PTIME. We show that a straightforward application of that earlier idea to our generalized problem formulation does not work as the LP relaxation of the naive formulation can give fractional optimal solutions (see [Example 6.3.2](#) and [Figure 6.7](#)). In [Subsections 6.3.2](#) and [6.3.3](#) we develop *new techniques that allowed us to prove that the natural LP relaxation of the resulting non-obvious ILP formulation has the ILP = LP property*. We also show the effect in our experiments [Figure 6.9](#) with a reduction from over 3 hours to under 20 seconds.

[84]: Gurobi Optimization (2025), *How does presolve work?*

[83]: Gurobi Optimization (2022), *Gurobi Optimizer Reference Manual*.

[123]: Mitchell, OSullivan, and Dunning (2011), *PuLP: a linear programming toolkit for python*.

[81]: Gurobi Optimization (2021), *Mixed-Integer Programming (MIP) – A Primer on the Basics*.

[84]: Gurobi Optimization (2025), *How does presolve work?*

[20]: Brucato, Abouzied, and Meliou (CACM, 2019), ‘Scalable computation of high-order optimization queries’. doi:10.1145/3299881

[153]: Trummer and Koch (SIGMOD, 2017), ‘Solving the Join Ordering Problem via Mixed Integer Linear Programming’. doi:10.1145/3035918.3064039

[149]: Siksnyš and Pedersen (SSDBM, 2016), ‘SolveDB: Integrating Optimization Problem Solvers Into SQL Databases’. doi:10.1145/2949689.2949693

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[‡] Gurobi offers a free academic license <https://www.gurobi.com/academia/academic-program-and-licenses/>.

Resilience is one of the key algorithmic problems underlying various forms of reverse data management: *What is the minimal number of tuples to delete from a database in order to eliminate all query answers?* A long-open question is determining the conjunctive queries (CQs) for which resilience can be solved in PTIME. In this chapter, we propose a unified Integer Linear Programming (ILP) formulation that can solve all known easy cases of resilience, and also new cases, including self-joins and bag semantics.

4.1 Problem Statement

Definition 4.1.1 (Resilience [59]) *Given a boolean query Q and database \mathcal{D} , we say that $k \in \text{RES}(Q, D)$ if and only if $D \models Q$ and there exists some contingency set $\Gamma \subseteq D$ with $|\Gamma| \leq k$ such that $D - \Gamma \not\models Q$.*

In other words, $k \in \text{RES}(Q, D)$ means that there is a set of k or fewer tuples in \mathcal{D} , the removal of which makes the query false. We are interested in the optimization version $\text{RES}^*(Q, D)$ of this decision problem: given Q and \mathcal{D} , find the *minimum* k so that $k \in \text{RES}(Q, D)$. A larger k implies that the query is more “resilient” and requires the deletion of more tuples to change the query output. A contingency size of minimum size is called a *resilience set*.

Our goal is to understand the complexity of solving these problems. The first result by Buneman et al. [22] showed that the problem is (NPC) for conjunctive queries (CQs) with projections. Later work that introduced the term resilience [59] and focused on boolean CQs showed that a large fraction of self-join-free CQs (“triad-free queries”) can be solved in PTIME, solving the complexity of self-join-free (SJ-free) queries. However, few results are known for the cases of CQs *with self-joins* [60]. This state is similar to other database problems where establishing complexity results for self-joins is often considerably more involved than for self-join-free queries (e.g., compare the dichotomy results on probabilistic databases for either self-join-free queries [39] with those for self-joins [40]). Moreover, all these problems have been studied only for set semantics, whereas relational databases actually use bag semantics i.e., they allow duplicate tuples [27]. Like self-joins, bags usually make problems harder to analyze [11, 99, 162], and few complexity results for bag semantics exist.

4.2 Chapter Overview and Contributions

This chapter gives the first dichotomy results under bag semantics for problems in reverse data management. Our attack on the problem is unconventional: Rather than deriving a dedicated PTIME algorithm for certain queries (and proving hardness for the rest), we instead propose a unified Integer Linear Program (ILP) formulation for all problem variants (self-joins or not, sets or bags, Functional Dependencies or not). We then show that, for all PTIME queries, *the Linear Program (LP) relaxation of our*

4.1	Problem Statement	15
4.2	Chapter Overview and Contributions	15
4.3	ILP for Resilience	16
4.4	PTIME Relaxation of ILP[RES*]	19
4.5	Finding hardness certificates	19
4.6	Complexity results for SJ-free CQs	24
4.7	Three Approximation Algorithms	28
4.8	Experiments	29
4.9	Chapter Summary	33

This chapter is based on: Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *Proc. ACM Manag. Data* 1, 4, Article 228 (December 2023), 27 pages. <https://doi.org/10.1145/3626715> [110]. Code is available online: <https://github.com/northeastern-datalab/resilience-responsibility-ilp/>

[22]: Buneman, Khanna, and Tan (ICDT, 2001), ‘Why and Where: A Characterization of Data Provenance’. [doi:10.1007/3-540-44503-x_20](https://doi.org/10.1007/3-540-44503-x_20)

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. [doi:10.14778/2850583.2850592](https://doi.org/10.14778/2850583.2850592)

[60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. [doi:10.1145/3375395.3387647](https://doi.org/10.1145/3375395.3387647)

[39]: Dalvi and Suciu (VLDBJ, 2007), ‘Efficient query evaluation on probabilistic databases’. [doi:10.1007/s00778-006-0004-3](https://doi.org/10.1007/s00778-006-0004-3)

[40]: Dalvi and Suciu (JACM, 2012), ‘The dichotomy of probabilistic inference for unions of conjunctive queries’. [doi:10.1145/2395116.2395119](https://doi.org/10.1145/2395116.2395119)

[27]: Chaudhuri and Vardi (PODS, 1993), ‘Optimization of real conjunctive queries’. [doi:10.1145/153850.153856](https://doi.org/10.1145/153850.153856)

[11]: Atserias and Kolaitis (SIGMOD Rec. 2022), ‘Structure and complexity of bag consistency’. [doi:10.1145/3542700.3542719](https://doi.org/10.1145/3542700.3542719)

[99]: Khamis, Kolaitis, Ngo, and Suciu (TODS, 2021), ‘Bag Query Containment and Information Theory’. [doi:10.1145/3472391](https://doi.org/10.1145/3472391)

[162]: Yannakakis (SIGMOD Rec. 2022), ‘Technical Perspective: Structure and Complexity of Bag Consistency’. [doi:10.1145/3542700.3542718](https://doi.org/10.1145/3542700.3542718)

ILP has the same optimal value, thereby proving that existing ILP solvers are *guaranteed* to solve problems for those queries in PTIME.

Contributions and Outline. We propose a unified framework for solving resilience, give new theoretical results, approximation guarantees, and experimental results:

- ① *Unified ILP framework:* We propose an ILP formulation for the problems of resilience that can not only encode all previously studied variants of the problem, but can also encode *all* formulations of the problem, including self-joins and bag semantics (Section 4.3). This unified encoding allows us to model and solve problems for which currently no algorithm (whether easy or hard) has been proposed. It also allows us to study LP relaxation (Section 4.4) of our formulation, which form the basis of several of our theoretical results.
- ② *Unified hardness criterion:* We prove a variant of an open conjecture from PODS 2020 [60] by defining a structural certificate called Independent Join Path (IJP) and proving that it implies hardness (Section 4.5). We use this certificate to both all prove hardness for all hard queries in our dichotomies, and later in Chapter 8 will use it to obtain computationally derived hardness certificates for previously open queries with self-joins.
- ③ *First results for resilience under bag semantics:* We give full dichotomy results for resilience under bag semantics for the special case of SJ-free CQs (Section 4.6).
- ④ *Recovering PTIME cases:* We prove that for all prior known PTIME cases of SJ-free queries (as well those shown to be PTIME in this chapter) (under both set and bag semantics), our ILP is solved in guaranteed PTIME by standard solvers (Section 4.6). This means that our formulation is unified not only in being able to model all cases but also in that it is guaranteed to *recover all known PTIME cases by terminating in PTIME*. This new way of modeling the problem opens up a new route for solving various open problems in reverse data management: by proposing a universal algorithm for solving all variants, future development does not depend on finding new dedicated PTIME algorithms, but rather on proving that the universal method terminates in PTIME (in similar spirit to proofs in this chapter).
- ⑤ *Novel approximations:* We show 3 different approximation algorithms for resilience. The first approach based on LP-rounding provides a guaranteed m -factor approximation (where m is the number of atoms in the query) for all queries (*including self-joins and bag semantics*). The other two are new flow-based approximation techniques designed for hard queries without self-joins (Section 4.7).
- ⑥ *Experimental Study:* We compare various approaches proposed in this chapter on different problem instances: easy or hard, for set or bag semantics, queries with self-joins, and Functional Dependencies. Our results establish the accuracy of our asymptotic predictions, uncover novel practical trade-offs, and show that our approach and approximations create an end-to-end solution (Section 7.12).

[60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’.
doi:10.1145/3375395.3387647

4.3 ILP for Resilience

We construct an Integer Linear Program $\text{ILP}[\text{RES}^*(Q, D)]$ from a CQ Q and a database \mathcal{D} which returns the solution to the optimization problem $\text{RES}^*(Q, D)$ for any Boolean CQ (even with self-joins) under either set

or bag semantics.¹ This section focuses on the correctness of the ILP. Section 4.4 later investigates how easy cases can be solved in PTIME, despite the problem being NPC in general.

To construct the ILP, we need to specify the decision variables, constraints and objective. As input to the ILP, we first run the query on the database instance to compute all the witnesses. This can be achieved with a modified witness query, a query that returns keys for each table, and thus each returned row is a *set* of tuples from each of the tables.²

1. Decision Variables. We create an indicator variable $X[t] \in \{0, 1\}$ for each tuple t in the database instance \mathcal{D} . A value of 1 for $X[t]$ means that t is included in a contingency set, and 0 otherwise. For bag semantics, Lemma 4.3.1 shows that it suffices to define a single variable for a set of duplicate tuples (intuitively, an optimal solution chooses either all or none).

2. Constraints. Each witness must be destroyed in order to make the output false for a Boolean query (or equivalently, to eliminate all output tuples from a non-Boolean query). A witness is destroyed, when at least one of its tuples is removed from the input. Thus, for each witness, we add one constraint enforcing that at least one of its tuples must be removed. For example, for a witness $\mathbf{w} = \{r_i, r_j, r_k\}$ we add the constraint that $X[r_i] + X[r_j] + X[r_k] \geq 1$.³

3. Objective. Under set semantics, we simply want to minimize the number of tuples deleted. Since for bag semantics we have made a simplification that we use only one variable per unique tuple, marking that tuple as deleted has cost equal to deleting all copies of the tuple. Thus, we weigh each tuple by the number of times it occurs to create the minimization objective.

1: Notice that we also write ILP[problem] for the optimal value of the program

2: Whether in set or bag semantics, duplicate tuples have the same key.

3: Notice that for SJ-free queries, the number of tuples in each constraint is exactly equal to the number of atoms in the query. But for queries with self-joins, the number of tuples in each constraint is not fixed (is lower when a tuple joins with itself).

Example 4.3.1 (RES ILP) Consider the Boolean two-chain query with self-join $Q_{2-SJ}^\infty := R(x, y), R(y, z)$ and a database \mathcal{D} with a single table $R \{(1, 1), (2, 3)(3, 4)\}$ The query over \mathcal{D} has 2 witnesses:

x	y	z	
1	1	1	$\mathbf{w}_1 = \{r_{11}\}$
2	3	4	$\mathbf{w}_2 = \{r_{23}, r_{34}\}$

Each distinct tuple has a decision variable. Thus, our ILP has 3 variables $X[r_{11}]$, $X[r_{23}]$, and $X[r_{34}]$. We create a constraint for each unique witness in the output, resulting in two constraints:

$$\begin{aligned} X[r_{11}] &\geq 1 \\ X[r_{23}] + X[r_{34}] &\geq 1 \end{aligned}$$

Finally, the objective is to minimize the tuples deleted, thus, to minimize: $X[r_{11}] + X[r_{23}] + X[r_{34}]$. Solving this results in an objective of 2 at $X[r_{11}] = 1$, $X[r_{23}] = 1$, $X[r_{34}] = 0$. Intuitively, one can see that $\text{RES}(Q, D) = 2$ as removing r_{11} and r_{23} from R is the smallest change required to make the query false.

Example 4.3.2 (RES ILP: Bag Semantics) Assume the same problem as Example 4.3.1, but we allow duplicates in the input. Concretely assume r_{23} appears twice: $R' = \{(1, 1) : 1, (2, 3) : 2, (3, 4) : 1\}$. The variables

and constraints stay the same, only the objective function changes now to

$$\min \{X[r_{11}] + 2X[r_{23}] + X[r_{34}]\}$$

Removing r_{11} and r_{23} is no longer optimal since it incurs a cost of 3. The optimal solution is now at $X[r_{11}] = 1$, $X[r_{23}] = 0$, $X[r_{34}] = 1$, with the objective value 2.

Before we prove the correctness of $\text{ILP}[\text{RES}^*(Q, D)]$ in [Theorem 4.3.2](#), we will justify our decision to use a single decision variable per unique tuple with the help of [Lemma 4.3.1](#).

Lemma 4.3.1 (Simplification for RES under Bag Semantics) *There exists a resilience set where for each distinct tuple in D , either all occurrences of the tuple are in the resilience set, or none are.*

Proof Intuition ([Lemma 4.3.1](#)). We show that if a tuple t is in a contingency set Γ but a duplicate tuple t' is not, then removing t leads to a now smaller contingency set Γ' .

Proof [Lemma 4.3.1](#). Assume there exists an optimally minimal resilience set R such that it contains a tuple t , but it does not contain an identical tuple t' . Since t and t' are identical, they join with the same tuples and must participate in same number of witnesses. Since t' is not in the resilience set, for every witness \mathbf{w}_i that contains t' , there must be at least one tuple x_i that is in the resilience set. All the witnesses that t participates in, must also contain a tuple from the set of x_i . If none of the x_i tuples is t itself, then we can safely remove t from R . Thus, R is not minimal, and we have a contradiction.

However, in the case that there exists an $x_i = t$, this implies that \mathbf{w}_i contains t and t' (along with 0 or more other tuples $T[\mathbf{w}_i]$). Since t and t' are identical, it follows that there is an identical witness created due to joining t' with itself. This witness too must be destroyed - hence one of $T[\mathbf{w}_i]$ is in the resilience set, and we safely remove t , leading to a contradiction. \square

Theorem 4.3.2 (RES ILP correctness) $\text{ILP}[\text{RES}^*(Q, D)] = \text{RES}^*(Q, D)$ for any CQ Q and database \mathcal{D} under set or bag semantics.

Proof Intuition. We prove validity by showing that any satisfying solution would necessarily destroy all witnesses i.e. make the query false. Optimality is proved by showing that any valid resilience set would be a valid solution for the ILP.

Proof [Theorem 4.3.2](#). The proof is divided into parts to separately show the validity and optimality of $\text{ILP}[\text{RES}^*(Q, D)]$. An invalid solution would not destroy all the witnesses in the output, while a suboptimal solution would have size bigger than the minimum resilience set.

- **Proof of Validity:** Assume a solution is invalid i.e. after deleting the tuples in the resilience Set, the number of output witnesses is not 0. Since Q is monotone, this witness existed in the original database as well. A witness can only survive if all the tuples in the witness are not a part of the resilience Set. Such a solution would hence violate the constraint for the surviving witness and hence would not be generated by the ILP.

- **Proof of Optimality:** Assume a solution is not optimal i.e. there exists a strictly smaller, valid resilience Set R' . We could translate this set into a variable assignment \tilde{X} to $X[t]$ where $\tilde{X}[t] = 1$ if $t \in R'$. Since R' is a valid resilience Set, it would satisfy all the constraints to destroy all witnesses in \mathcal{D} and also be a valid solution for $\text{ILP}[\text{RES}^*(Q, D)]$. Thus, it cannot be smaller than the optimal solution for $\text{ILP}[\text{RES}^*(Q, D)]$. \square

We would like to stress to the reader that changing from sets to bags affects only the objective function, not the constraint matrix. Later in [Section 4.6](#), we will prove that for queries such as Q_A^Δ , the problem of finding resilience becomes NPC under bag semantics, while it is solvable in PTIME under set semantics. This observation is significant because most of the literature on tractable cases in ILP focuses exclusively on analyzing the constraint matrix. For example, if an ILP has a constraint matrix that is Totally Unimodular it is PTIME no matter the objective function [[145](#), Section 19].

4.4 PTIME Relaxation of $\text{ILP}[\text{RES}^*]$

The previous sections introduced a unified ILPs to solve for RES. However, ILPs are NPC in general, and we would like stronger runtime guarantees for cases where RES can be solved in PTIME. We do this with the introduction of LP relaxations, which generally act as lower bounds for minimization problems. However, in [Section 4.6](#) we prove that these relaxations $\text{LP}[\text{RES}^*]$ is actually always equal to the corresponding ILPs for all easy cases. Thus, whether easy or hard, exact or approximate, problems can be solved within the same framework, with the same solver, with minimal modification, and with the best-achievable time guarantees.

LP Relaxation for RES. LP Relaxations are constructed by relaxing (removing) integrality constraints on variables. In $\text{ILP}[\text{RES}^*]$, a tuple indicator variable $X[t]$ only takes values 0 or 1. $\text{LP}[\text{RES}^*]$ removes that constraint and allows the variables any (“fractional”) value in $[0, 1]$.

4.5 Finding hardness certificates

Freire et al. [[60](#)] conjectured that the ability to construct a particular certificate called “Independent Join Path” is a sufficient criterion to prove hardness of resilience for a query. We prove here that not the original but a slight variation of that idea is indeed correct. We also prove that this construction is a *necessary criterion for hardness of self-join-free queries* and conjecture it to be also necessary for any query. In addition, in [Chapter 8](#), we also give a Disjunctive Logic Program ($\text{DLP}[\text{RESIJP}]$) that can create hardness certificates and use it to prove hardness for 5 previously open queries with self-joins.

4.5.1 Independent Join Paths (IJPs)

We slowly build up intuition to define IJPs ([Definitions 4.5.1 and 4.5.2](#)). Recall the concept of a *canonical database* for a minimized CQ resulting from replacing each variable with a *different constant* [[25](#), [154](#)]. For example

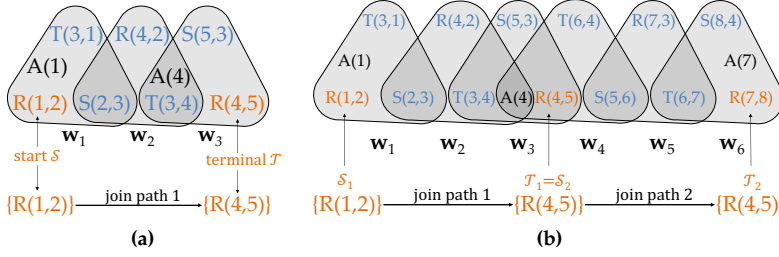


Figure 4.1: (a) IJP for triangle query Q_A^Δ . (b) IJPs are composed by sharing their endpoints (start or terminal tuples).

$A(1), R(1, 2), S(2, 3), T(3, 1)$ is a canonical database for the triangle query $Q_A^\Delta := A(x), R(x, y), S(y, z), T(z, x)$. Intuitively, one can think of a *witness* as more general than a canonical database in that several variables may map to the *same constant*. A join path is then a set of witnesses that share enough constants to be connected (this sharing of constants can be best formalized as a *partition of the constants* among a fixed number of witnesses). In addition, join paths are defined with two “*isomorphic*” sets of tuples, the start \mathcal{S} and terminal \mathcal{T} (both together called the “*endpoints*”). We call two sets of tuples *isomorphic* iff there a bijective mapping between the constants of the sets that preserves the sets of shared constants across table attributes. For example, $\mathcal{S}_1 = \{R(1, 2), A(2), R(2, 2)\}$ is isomorphic to $\mathcal{S}_2 = \{R(3, 4), A(4), R(4, 4)\}$ but not to $\mathcal{S}_2 = \{R(3, 4), A(4), R(4, 5)\}$.

Definition 4.5.1 (Join Path (JP)) *A database \mathcal{D} forms a Join Path from a set of tuples \mathcal{S} (start) to \mathcal{T} (terminal), for query Q if*

1. *Each tuple in \mathcal{D} participates in some witness (i.e. \mathcal{D} is reduced).*
2. *The witness hypergraph is connected.*
3. *\mathcal{S} and \mathcal{T} form a valid endpoint pair, i.e.:*
 - (i) *\mathcal{S} and \mathcal{T} are isomorphic and non-identical.*
 - (ii) *There is no endogenous tuple $t \in \mathcal{D}, t \notin \mathcal{S} \cup \mathcal{T}$ whose constants are a subset of the constants of tuples in $\mathcal{S} \cup \mathcal{T}$.*

Example 4.5.1 (Join paths) Consider again the query Q_A^Δ . The following database of 9 tuples (Figure 4.1a) $\mathcal{D} = \{A(1), A(4), R(1, 2), R(4, 2), R(4, 5), S(2, 3), S(5, 3), T(3, 1), T(3, 4)\}$ where $A(1)$ and $A(4)$ are exogenous, forms a join path from $\mathcal{S} = \{R(1, 2)\}$ to $\mathcal{T} = \{R(4, 5)\}$. It has 3 witnesses $w_1 = \{A(1), R(1, 2), S(2, 3), T(3, 1)\}$, $w_2 = \{A(4), R(4, 2), S(5, 3), T(3, 4)\}$, and $w_3 = \{A(4), R(4, 5), S(5, 3), T(3, 4)\}$. This join path can also be interpreted as a partition $\{\{x^1\}, \{x^2, x^3\}, \{y^1, y^2\}, \{y^3\}, \{z^1, z^2, z^3\}\}$ on the canonical databases for three witnesses $w_i = \{A(x^i), R(x^i, y^i), S(y^i, z^i), T(z^i, x^i)\}, i = 1, 2, 3$, expressing the shared constants in each subset. Then above database instance results from the following valuation v of the quotient set $\{[x^1], [x^2], [y^1], [y^3], [z^1]\}$ to constants: $v : (x^1, y^1, z^1, x^2, y^3) \rightarrow (1, 2, 3, 4, 5)$. Notice that \mathcal{S} and \mathcal{T} form a valid endpoint pair because (i) \mathcal{S} and \mathcal{T} are isomorphic with the mapping $f = \{1 : 3, 2 : 4\}$ and (ii) there is no endogenous tuple with constants only from $\{1, 2, 3, 4\}$. $A(1)$ and $A(4)$ violate the subset requirement, however they are exogenous, so the definition is fulfilled.

We also call two join paths *isomorphic* if there is a bijective mapping between the shared constants across the witnesses. Given a fixed query, we usually leave away the implied qualifier “*isomorphic*” when discussing join paths. We talk about the “*composition*” of two join paths if one endpoint of the first is identical to an endpoint of the second, and all other constants are different. We call a composition of join paths “*non-leaking*” if the composition adds no additional witnesses that were not

already present in any of the non-composed join paths.

Example 4.5.2 (Join path composition) Consider the composition of two JPs shown in Figure 4.1b. They are isomorphic because there is a reversible mapping $(1, 2, 3, 4, 5) \rightarrow (4, 5, 6, 7, 8)$ from one to the other. They are composed because they share no constants except for their endpoints: The terminal $\mathcal{T}_1 = \{R(4, 5)\}$ of the first is identical to the start of the second (\mathcal{S}_2). The composition is non-leaking since no additional witnesses results from their composition.

Proposition 4.5.1 (Triangle composition) *Assume a join path (JP) with endpoints \mathcal{S} and \mathcal{T} . If 3 isomorphic JPs composed in a triangle with directions as shown in Figure 4.2 are non-leaking, then any composition of JPs is non-leaking.*

Proof Intuition (Proposition 4.5.1). Since JPs can be asymmetric, the composability due to sharing the \mathcal{S} tuples in two isomorphic JPs differs from sharing \mathcal{S} and \mathcal{T} . We show that the three JP interactions in Figure 4.2 act as sufficient base cases to model all types of interactions. We show via induction that sharing the same end tuples across multiple JPs cannot leak if it does not leak in the base case.

Proof Proposition 4.5.1. Condition (3ii) of Definition 4.5.1 implies that given two canonical join paths (they are isomorphic, and all constants are distinct), sharing the constants in one end point of each guarantees that the only endogenous tuples that the join paths shares are the endpoint tuples. What can happen is that this sharing of endpoints creates *additional* witnesses which will affect the resilience of the resulting database instance. What we like to prove is that if the composition from Figure 4.2 is not leaking, then *any* composition is non-leaking (and thus creates no new witnesses).

From the condition that the endpoints of a join path have disjoint constants, and the fact that any two join paths can share maximally one endpoint, it follows that the tuples from two join paths that are not sharing any endpoint cannot create additional witnesses; additional witnesses can only be created by two join paths sharing an endpoint.

Since join paths can be asymmetric, there are three ways that two join paths can create additional witnesses: they are either sharing the start tuples, or the terminal tuples, or one start tuple is identical to the other end tuple. All three cases are covered by Figure 4.2.

It remains to be shown that sharing the same end tuples across multiple join paths can't add additional witnesses. This follows now from induction with the three base cases covered above. To illustrate, assume adding a third join path by their terminal to a start tuples shared by two join paths leads to additional witnesses. Then from the isomorphism between the two prior join paths it follows that a new witness would have to be created from having only one join path with the end tuples as start tuples. This is a contradiction. The same argument can be used for adding join paths to the other three bases cases. \square

Definition 4.5.2 (Independent Join Path) *A Join Path \mathcal{D} forms an Independent Join Path (IJP) if it fulfills two additional conditions:*

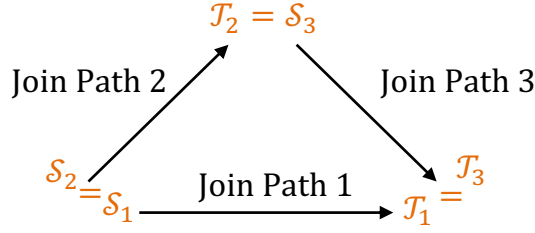


Figure 4.2: 3 JPs composed in a triangle with shown edge directions.

4. “OR-property”: Let c be the resilience of Q on \mathcal{D} . Then resilience is $c - 1$ in all 3 cases of removing either \mathcal{S} or \mathcal{T} or both.
5. Any composition of two or more isomorphic JPs is non-leaking.

Example 4.5.3 (IJPs) Consider again the JP from Figure 4.1a. The resilience is $c = 2$ as removing $\Gamma = \{S(2, 3), T(3, 4)\}$ destroys all 3 witnesses. Removing $\mathcal{S} = \{(1, 2)\}$ destroys w_1 , and it suffices to just remove one tuple $\Gamma' = \{T(3, 4)\}$ to destroy the remaining 2 witnesses. Similarly, for removing either \mathcal{T} , or both \mathcal{S} and \mathcal{T} . This proves the OR-property of this JP. Further composing 3 JPs in a triangle as shown in Figure 4.2 is non-leaking (the resulting database has 9 witnesses), and thus this JP is an IJP.

We now prove that the ability to create an IJP for a query proves its resilience to be hard. This was left as an open conjecture in [60].

Theorem 4.5.2 (IJPs \Rightarrow NPC) *If there is a database \mathcal{D} that forms an IJP for a query Q , then $\text{RES}(Q)$ is NPC.*

Proof Intuition. This proof uses the IJP to build a reduction from Vertex Cover to $\text{RES}(Q)$. The IJP acts as a hardness gadget for the edges. We show that the decision version of resilience of an appropriately chosen database can be used to solve the decision version of vertex cover of a given graph.

Proof Theorem 4.5.2. The proof follows from a simple reduction from vertex cover. Assume Q can form IJPs of resilience c . Take any directed simple graph $G(V, E)$ with n nodes and m edges. Encode each node $v \in V$ with a unique tuple $v = (\langle v_1 \rangle, \langle i_v \rangle, \dots, \langle v_d \rangle) \in R$ where d is the arity of R . Encode each edge $(v, u) \in E$ as separate IJP from $R(v)$ to $R(u)$ with fresh constants except their endpoints. Then G has a Vertex Cover of size k iff resilience $\text{RES}^*(Q, D)$ is $k + m(c - 1)$.

Notice that the semantic condition 5 is needed. It guarantees that there is no tuple (other than the endpoints) that are shared between two different join paths (corresponding to the edges), and no additional joins are created. Without that condition, the join paths are not *independent* and *leakage* across join paths could otherwise change the resilience of the composition. \square

We next prove that the ability to create an IJP for a self-join-free CQ is not only a sufficient but also a *necessary criterion* for hardness. This suggests that IJPs are a strictly more powerful criterion for resilience than the previous notion of triads [59]: they capture the same hardness for SJ-free queries, but can also prove hardness for queries with self-joins that do not contain a triad.

Theorem 4.5.3 (IJPs \Leftrightarrow NPC for SJ-free CQs) *The resilience of a SJ-free CQ under set semantics is NPC iff it has an IJP.*

Proof Intuition (Theorem 4.5.3). We generalize all past hardness results [59] for SJ-free queries by showing that the same hardness criteria (*triads*) that was necessary and sufficient for hardness, can always be used to construct an IJP and show this construction.

Proof Theorem 4.5.3. We already know from Theorem 4.5.2 that IJPs \Rightarrow NPC. We also know from [59] that all hard queries must have an active triad. Recall that an *active triad* is a set of three endogenous (and therefore, non-dominated) atoms, $\mathcal{T} = \{R_1, R_2, R_3\}$ such that for every pair $i \neq j$, there is a path from R_i to R_j that uses no variable occurring in the other atom of \mathcal{T} . It remains to be shown that queries with triads also have an IJP.

Let q be a query with triad $\mathcal{T} = \{R_1, R_2, R_3\}$. We will choose appropriate constants to build an IJP from $\{R_1(\mathbf{a})\}$ to $\{R_1(\mathbf{b})\}$ consisting of three witnesses $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ s.t. \mathbf{w}_1 and \mathbf{w}_2 share R_2 and \mathbf{w}_2 and \mathbf{w}_3 share R_3 . In other words, we compose the 3 paths from the triad as $R_1(\mathbf{a}) \rightarrow R_2(\mathbf{c}) \rightarrow R_3(\mathbf{d}) \rightarrow R_1(\mathbf{b})$.

We will assume that no variable is shared by all three elements of \mathcal{T} (we can ignore any such variable by setting it to a constant). Our proof splits into two cases:

Case 1: $\text{var}(R_1), \text{var}(R_2), \text{var}(R_3)$ are pairwise disjoint: We use unique constants $a, b, c, d, w_1, w_2, w_3$ and add tuples $R_1(a, a, \dots, a)$, $R_2(c, c, \dots, c)$, $R_3(d, d, \dots, d)$, and $R_1(b, b, \dots, b)$ to \mathcal{D} .

To define the relations corresponding to the other atoms in \mathbf{w}_1 , we first partition the variables of q into 3 disjoint sets: $\text{var}(q) = \text{var}(R_1) \cup \text{var}(R_2) \cup W_1$. Now for each atom $A_i \in q \setminus \{R_1, R_2\}$, arrange its variables in these three groups. Then define a tuple $(a; d; w_1)$ to relation R_i of \mathcal{D} corresponding to atom A_i . For example, all the variables $v \in \text{var}(R_1)$ are assigned the value a and all the variables $v \in W_1$ are assigned w_1 . Repeat the same process analogously for witnesses \mathbf{w}_2 and \mathbf{w}_3 .

From our construction \mathbf{w}_1 and \mathbf{w}_2 share only one single endogenous tuple: $R_2(\mathbf{c})$. This follows from the fact that there is no other tuple that dominates (has a subset of variables) of endogenous tuples. It follows that every endogenous tuple in \mathbf{w}_1 needs to contain at either at least constant a or w_1 (and optionally c) Similarly every endogenous tuple in \mathbf{w}_2 needs to contain at either at least constant d or w_2 (and optionally c).

It follows that the resilience of the resulting database is identical to vertex cover of the graph $R_1(\mathbf{a}) \rightarrow R_2(\mathbf{c}) \rightarrow R_3(\mathbf{d}) \rightarrow R_1(\mathbf{b})$, which fulfills condition (4) of Definition 4.5.2. Condition (5) follows from the same fact that every tuple in one join path needs to contain at least one constant not contained a tuple from another join path, other than the maximally one shared endpoint.

Case 2: $\text{var}(R_i) \cap \text{var}(R_j) \neq \emptyset$ for some $i \neq j$: The previous construction can now be generalized from Case 1 by partitioning $\text{var}(R_i)$ into those unshared, those shared with R_{i-1} , and those shared with R_{i+1} (addition here is mod 3) and verifying that the resulting database still fulfills the same conditions (4) and (5). \square

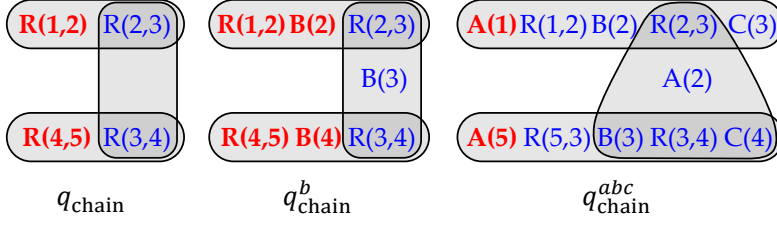


Figure 4.3: Simple hardness gadgets for prior known hard queries.

4.5.2 More Example IJPs

We also give simpler automatically derived IJPs for $k = 3$ for the following 3 previously known hard queries. The original hardness proofs for those queries [59] are pretty involved and cover several pages. Our new hardness proofs are just Figure 4.3 given Theorem 4.5.2.

4.6 Complexity results for SJ-free CQs

Table 4.1: Complexity Landscape for Resilience. All results follow from our unified framework. Results highlighted with a yellow background are new. Some theorems apply to multiple settings (thus shown repeatedly) and include earlier results as special cases.

Type of query	Example	Set Semantics	Bag Semantics
Linear Queries	Q_2^∞	PTIME (thm. 4.6.1)	PTIME (thm. 4.6.1)
With Deactivated Triads	$Q_{AB}^\Delta, Q_A^\Delta$	PTIME (thm. 4.6.2)	NPC (thm. 4.6.3)
With Active Triads	Q^Δ, Q_3^\star	NPC (thm. 4.5.3)	

This section gives complexity results for both RES or SJ-free queries, under set and bag semantics (see Table 4.1). Our results include both prior known results and new results. Importantly, all our hard cases are derived with our unified hardness criterion (IJPs) from Section 4.5, and all tractable cases follow from our unified algorithms in Sections 4.3 and 4.4.

4.6.1 Necessary notations

Before diving into the proofs, we define a few key concepts stemming from domination (Definition 4.6.2) that lead up to the three structural criteria (Definition 4.6.6) which completely describe our dichotomy results. Notice that the notion of *triads* has been previously defined [59]. However, we extend this notion and make it more-fine grained. The previous definition of *triad* now corresponds exactly to the special case of “*active triads*.”

Definition 4.6.1 (Exogenous / Endogenous tuples) *A tuple is exogenous if it must not or need not participate in a contingency set, and endogenous otherwise.*

Prior work [117] has defined relations (or atoms) to be exogenous or endogenous, i.e. when all tuples in any relation (or relation of the atom) are either exogenous or endogenous. We use but also generalize this notation to allow *individual tuples* to be declared exogenous (but keep them endogenous by default). We will see later in Section 4.5 that this generalization allows us to formulate resilience and responsibility with a

simple universal hardness criterion.* The set of exogenous tuples $E \subset D$ can be provided as an additional input parameter as in $\text{RES}(Q, D, E)$ and $\text{RSP}(Q, D, t, E)$. We assume a database instance has no exogenous tuples unless explicitly specified, and we omit the parameter for simplicity.

Definition 4.6.2 (Domination [59]) *In a query Q with endogenous atoms A and B , we say A dominates B iff $\text{var}(A) \subset \text{var}(B)$.*

Definition 4.6.3 (Triad (different from [59])) *A triad is a set of three atoms, $\mathcal{T} = \{R_1, R_2, R_3\}$ s.t. for every pair $i \neq j$, there is a path from R_i to R_j that uses no variable occurring in the third atom of \mathcal{T} .*

Definition 4.6.4 (Solitary variable [59]) *In a query Q a variable v in relation A is solitary if, in the query hypergraph it cannot reach any endogenous atom $B \neq A$ without passing through one of the nodes in $\text{var}(A) - v$.*

Definition 4.6.5 (Full domination [59]) *An atom A of CQ Q is fully dominated iff for all non-solitary variables $y \in \text{var}(A)$ there is another atom B such that $y \in \text{var}(B) \subset \text{var}(A)$.*

Definition 4.6.6 (Active or (fully) deactivated triads) *A triad is deactivated iff at least one of its three atoms is dominated by another atom of the query. A triad is fully deactivated iff at least one of its three atoms is fully dominated by another atom of the query. A triad is active iff none of its atoms are dominated.*

We call queries *linear* if they do not contain triads. Here we depart from prior work that referred to linear queries as queries without what we now call active triads [59]. We instead say that queries without active triads are *linearizable*.[†]

Example 4.6.1 Consider the triad $\{R, S, T\}$ in all 3 queries Q^Δ , Q_A^Δ , and Q_{AB}^Δ . The triad is deactivated in Q_A^Δ and Q_{AB}^Δ because A dominates both R and T . The triad is fully deactivated in Q_{AB}^Δ because T is fully dominated by A and B . The triad is active in Q^Δ since none of the three tables in the triad are dominated. The chain with ends query Q_{2WE}^∞ has no triad and is thus linear.

$Q^\Delta := R(x, y), S(y, z), T(z, x)$
 $Q_A^\Delta := A(x), R(x, y), S(y, z), T(z, x)$, and
 $Q_{AB}^\Delta := A(x), R(x, y), S(y, z), T(z, x), B(z)$,
 respectively

4.6.2 Dichotomies for RES under Sets and Bags

This section proves that for all SJ-free CQs, either $\text{LP}[\text{RES}^*]$ solves RES exactly (and the problem is hence easy for any instance), or we can form an IJP (and thus the problem is hard). Our results cover both set and bag semantics (see Table 5.1).

Theorem 4.6.1 (Integrality of $\text{LP}[\text{RES}^*]$ for Linear Queries) $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}^*(Q, D)$ for all database instances \mathcal{D} under set or bag semantics if Q is linear.

* In more detail, we will formulate hardness of responsibility via an Independent Join Path which is only possible because one specified tuple is exogenous, e.g. Theorem 5.5.4.

[†] The intuition of “linearity” is that the vertices of the dual hypergraph H_d of Q can be mapped onto a line s.t. H_d has the running intersection property [13].

Proof Intuition (Theorem 4.6.1). We prove that all optimal LP solutions correspond to a valid integral solution by using the flow graph for resilience introduced by prior work [117] as an intermediary. We show that every LP solution corresponds to a valid cut of the graph. It follows from the max-flow Integrality Theorem [58] that there must be an optimal integral solution with the same objective value.

Proof Theorem 4.6.1. Prior approaches show that the witnesses generated by a linear query Q over database instance \mathcal{D} can be encoded in a flow graph [117] such that each path of the flow graph represents a witness and each edge with non-infinite weight represents a tuple. The flow graph is such that an edge participates in a path iff the corresponding tuple is part of the corresponding witness. The min-cut of this graph (or the minimum edges to remove to disconnect the source from the target), is equal to $\text{RES}(Q, D)$.

We use this prior result to prove that $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}^*(Q, D)$ by showing that the Linear Program solution represents all possible cuts for the flow graph, and vice versa. For each constraint of the LP (that corresponds to a witness), we have a source to target path in the flow graph. Thus, whenever we have an edge cut in a flow graph path, a corresponding constraint of the LP is fulfilled by deleting the corresponding tuple (and vice versa). As discussed in Section 3.3, it is well established (through the flow integrality theorem [144, Theorem 5.22] and Edmonds-Giles theorem on Total Dual Integrality [48, Theorem 7.1]) that the LP polytope of the cuts of a flow graph is integral, i.e. all extreme points of the polytope are integral, and hence the optimal solution of the ILP can be found in PTIME. \square

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

[48]: Edmonds and Giles (1977), 'A Min-Max Relation for Submodular Functions on Graphs'. doi:10.1016/s0167-5060(08)70734-9

Theorem 4.6.2 (Integrality of $\text{LP}[\text{RES}^*]$ for queries with deactivated triads under set semantics) $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}^*(Q, D)$ for all database instances \mathcal{D} under set semantics if all triads in Q are deactivated.

Proof Intuition (Theorem 4.6.2). Prior work [59] has shown that queries that contain only deactivated triads (previously called dominated triads) can be linearized due to domination (Definition 4.6.2). We show that this linearization does not change the optimal solution to the LP formulation under set semantics. Notice that domination does not work under bag semantics, which leads to a different tractability frontier.

Proof Theorem 4.6.2. Assume R, S, T are the tables in a deactivated triad in Q and W.l.o.g. assume R is dominated by a relation A . We show that R can be made exogenous because there exists an optimal resilience set that does not contain any tuple from R . We can see that for each tuple r_i in R , there exists a tuple a_i in A such that a_i participates in the same or more witnesses than r_i . Thus, whenever we have a contingency set that contains r_i , we can replace it with a_i and still fulfill the same constraints, without increasing the cost of the contingency set. We can do this because under set semantics, we only care about the number of tuples in a contingency set, since the cost assigned to removing a tuple is always 1. We can thus make all tuples in R exogenous (allowing them to take on value 0 in the ILP solution or essentially removing the variable from the constraints) without changing the optimal solution of the ILP. Such a process is typically part of the presolve step of ILP solvers [84], and we can remove variables whose constraints are subsumed by the constraints of other variables. This process can be performed in PTIME

[84]: Gurobi Optimization (2025), *How does presolve work?*

(we can see this is naively true by comparing the constraints of all pairs of variables in the ILP).

The resulting ILP now has a reduced dimension, since all tuples in R are now exogenous. We next can show that this new constraint matrix that creates a reduced-dimension polytope represents the cuts of a flow graph, and thus all extreme points of the polytope are integral. Let Q' be the query where for each deactivated triad, all dominated tables have been made exogenous. We can then construct a flow graph for Q' by using the same method as prior approaches [59] to create flow graphs for queries with exogenous tables via dissociation. All dissociated, exogenous tuples in the flow graph can simply be represented with infinite capacity edges (since they never need to be part of an optimal solution). This flow graph is now equivalent to the flow graph for Q that represents the reduced-dimension polytope (and it preserves an optimal solution of the original ILP). Thus, we can use the flow integrality theorem and the integrality of TDI systems as discussed in Section 3.3 and used in Theorem 4.6.1 to show that the polytope is integral, and thus $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}^*(Q, D)$. \square

Theorem 4.6.3 (Hardness of RES for non-linear queries under bag semantics) *RES(Q) is NPC under bag semantics if Q is not linear.*

Proof Intuition (Theorem 4.6.3). For queries with active triads, the IJPs (Theorem 4.5.3) imply hardness for bag semantics as well. We prove that *all triads* are hard by showing that including a fixed number of copies of a dominating table is equivalent to making it exogenous. Thus, domination does not work under bag semantics, and any triad (even a fully deactivated one) implies hardness.

Proof Theorem 4.6.3. A non-linear query by definition must contain triads. If the query contain active triads, then Theorem 4.5.3 can be applied in the bag semantics setting as well to show that RES is NPC. However, the same IJP does not directly work for (fully) deactivated triads - since the endpoints are part of the triad tables, they can be dominated by another tuple in the IJP. Then the optimal resilience would be to choose the dominating tuple, thus no longer fulfilling the first criteria of independence. Hence, we must have a slightly different IJP with the property that the dominating table is exogenous. To make the dominating table exogenous, it suffices that we have c_w copies of each tuple from the table in the IJP (where c_w is the number of witnesses in the IJP under set semantics), and 1 copy of all other tuples. Using Lemma 4.3.1 where we showed that it is never beneficial to remove some copies of a tuples, and the fact that the resilience of the IJP is at most k , we can see that it is never necessary to remove tuples from the dominating table. \square

The results in this section, along with Theorem 4.5.3 imply the following dichotomies under both set and bag semantics:

Corollary 4.6.4 (Resilience Dichotomy under set semantics) *Under set semantics, $\text{RES}^*(Q, D)$ is in PTIME for queries that do not contain active triads, otherwise it is NPC.*

Corollary 4.6.5 (Resilience Dichotomy under bag semantics) *Under bag*

semantics, $\text{RES}^*(Q, D)$ is in PTIME for queries that do not contain triads, otherwise it is NPC.

4.7 Three Approximation Algorithms

We describe one LP-based approximation algorithm and two flow-based approximation algorithms for RES, all three of which apply to both set and bag semantics.

4.7.1 LP-based m -factor Approximation

For a given query with m atoms, we use a standard LP rounding technique [157] with the threshold of $1/m$ i.e., we round up variables whose value is $\geq 1/m$ or set them to 0 otherwise.

[157]: Vazirani (2001), *Approximation algorithms*.
doi:10.1007/978-3-662-04565-7

Theorem 4.7.1 (m -factor approximation for RES) *The LP Rounding Algorithm is a PTIME m -factor approximation for RES.*

Proof Intuition (Theorem 4.7.1). Verification of PTIME solvability and the m -factor bound is trivial, and correctness follows by showing validity of each constraint for a rounded solution.

Proof Theorem 4.7.1. The LP-Rounding algorithm is PTIME since it requires the solution of a linear program, which can be found in PTIME, and a single iteration over the tuple variables. We also see that it is bounded by $m * \text{LP}[\text{RES}^*(Q, D)]$ since each variable is multiplied by at most m , and since $\text{LP}[\text{RES}^*(Q, D)] \leq \text{ILP}[\text{RES}^*(Q, D)]$, the algorithm is at most m -factor the optimal value. Thus, it remains to prove that X_I returned by the rounding, satisfies all constraints of $\text{ILP}[\text{RES}^*(Q, D)]$. We know that for every constraint, we involve at most m tuple variables[‡]. Since the sum of these variables in X_f must be at least 1 (due to the constraints of $\text{LP}[\text{RES}^*]$), there must exist at least one tuple variable in each constraint with value $\geq 1/m$. Thus, in X_I , for each constraint, there is a tuple variable t such that $X_I[t] = 1$ and all constraints are satisfied.

□

4.7.2 Flow-based Approximations

Non-linear queries cannot be encoded as a flow graph since they do not have the running-intersection property. The idea behind flow-based approximations is to add either witnesses or tuples (while keeping the other constant) to linearize a non-linear query. This works since adding more tuples or witnesses can only increase RES for monotone queries. Since there are multiple arrangements to linearize a query, we take the minimum over all non-symmetric arrangements, explained next for the two variants:

Constant Tuple Linearization Approximation (Flow-CT). We keep the same tuples as the original database in each arrangement. However, since the query is non-linear, these flow graphs may have spurious paths that do not correspond to any original witnesses, thus inadvertently adding

[‡] For SJ-free cases, exactly m tuples are involved, but for queries with self-join a witness can have less than m tuples

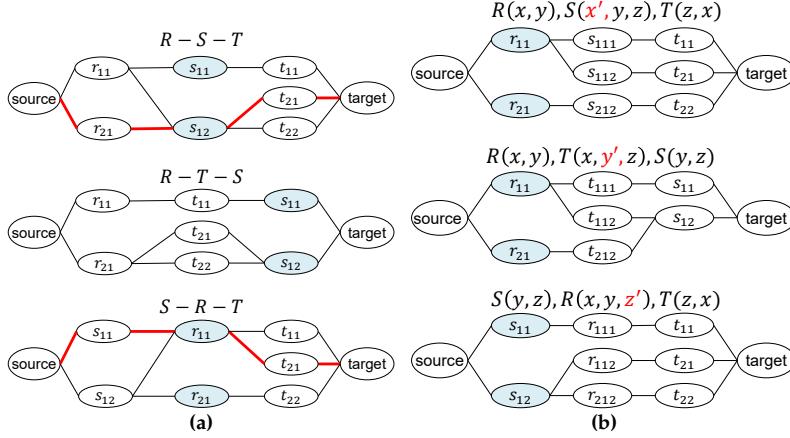


Figure 4.4: Flow approximation linearizations for Example 4.7.1.

witnesses. For a query with m atoms, there are up to $m! / 2$ linearizations due to the number of asymmetric ways to order them.

Constant Witness Linearization Approximation (Flow-CW). We keep the same witnesses as the original database instance in each linearization, however the query is changed by adding variables to tables (which is equivalent to dissociating tuples) to make it linear. The number of such linearizations is equal to the number of minimal dissociations [66].

Example 4.7.1 Consider the Q^Δ query with the following witnesses:

x	y	z	
1	1	1	$w_1 = \{r_{11}, s_{11}, t_{11}\}$
1	1	2	$w_2 = \{r_{11}, s_{12}, t_{21}\}$
2	1	2	$w_3 = \{r_{21}, s_{12}, t_{22}\}$

Then there are 3 Flow-CT linearizations (Figure 4.4a) and 3 Flow-CW linearizations (Figure 4.4b). The approximated resilience corresponds to the minimum of the min-cut over all linearized flow graphs. In this example, we see that both Flow-CT and Flow-CW happen to return the optimal value of 2 as approximation.

[66]: Gatterbauer and Suciu (VLDB, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

4.8 Experiments

Our experimental objective is to answer the following questions: (1) How does our ILP scale for PTIME queries, and how does it compare to previously proposed algorithms that use flow-based encodings [117]? (2) Are our LP relaxations (proved to be correct for PTIME queries in Section 4.6) indeed correct in practice? (3) What is the scalability of ILPs and LPs for settings that are proved NPC? (4) What is the quality of our approximations from Section 4.7?

Algorithms. ILP denotes our ILP formulations for RES. ILP(10) denotes the solution obtained by stopping the solver after 10 seconds.[§] LP denotes

[§] Solvers often already have the optimal solution by this cutoff, despite the ILP taking longer to terminate. This is because although the solver has stumbled upon an optimal solution, it may not yet have a proof of optimality (in cases where LP!=ILP).

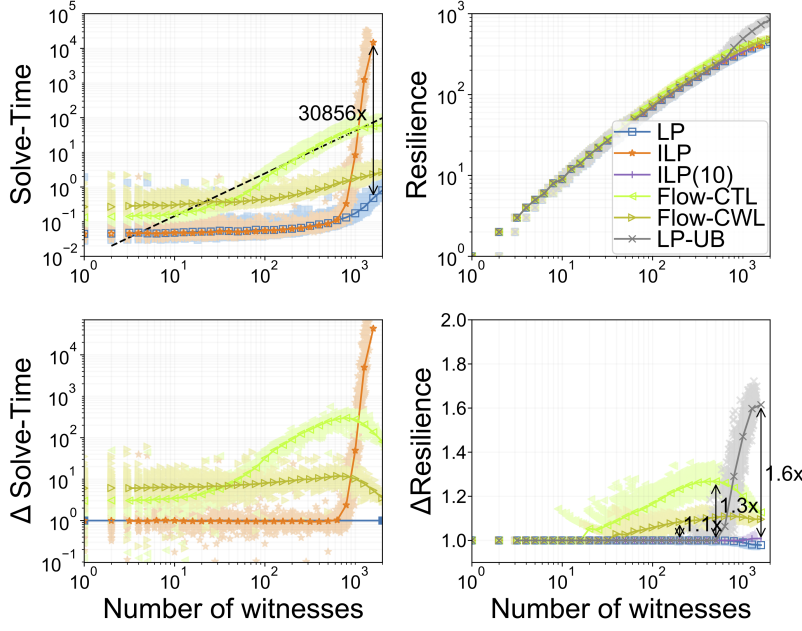


Figure 4.5: Setting 1: Hard 3-star query Q_3^* .

LP relaxations for RES. Flow denotes an implementation of the prior max-flow min-cut algorithm for RES for queries that are in PTIME [59, 117].[¶] LP-UB denotes our m -factor upper bound obtain by the LP rounding algorithm. Flow-CW and Flow-CT represent our approximations via Constant Witness Linearization and Constant Tuple Linearizations, respectively.

Data. We use synthetic data for our experiments. For any synthetic data experiment, we fix the maximum domain size, and sample randomly from all possible tuples. For testing our methods under bag semantics, each tuple is replicated by a random number that is smaller than a pre-specified max bag size.

Software and Hardware. We implement the algorithms using Python 3.8.5 and solve the respective optimization problems with Gurobi Optimizer 8.1.0 [83]. Experiments are run on an Intel Xeon E5-2680v4 @2.40GHz machine available via a cluster.

Experimental Protocol. For each plot we run 30 runs of logarithmically and monotonically increasing database instances. We plot all obtained points with a low saturation, and then draw a trend line between the median points from logarithmically increasing sized buckets. All plots are log-log, with the x-axis representing the number of witnesses. The y-axis for plots on the left shows the solve-time (in seconds) taken by the solver to solve a RES or mincut problem.⁴ We include a dashed line to show linear scalability as reference in the log-log plot.

4.8.1 Experimental Settings

Setting 1: Resilience Under Set Semantics. We consider the 3-star query $Q_3^* := R(x), S(y), T(z), W(x, y, z)$ which contains an active triad and is hard (Figure 4.5). The top plots show the growth of solve-time and resilience for increasing instances, while the bottom plots show the

4: The build-times to create the ILP or flow graphs are not plotted since they were negligible in comparison to the solve-time.

[¶] For the min-cut algorithm, we also experimented with both LP and Augmented Path-based algorithms via the NetworkX library [146]. Since the time difference in the methods was not significant, we leave it out and all running times reported in the figures use the same LP library Gurobi [83].

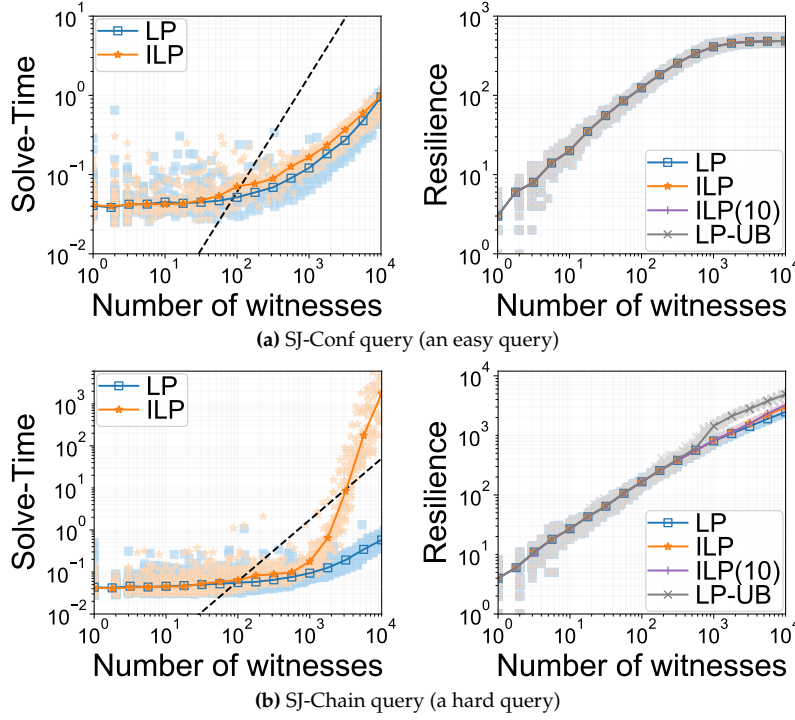


Figure 4.6: Setting 2: Queries with self-joins under Bag Semantics.

growths as a fraction of the optimal.^{||} We see that the solve-time of ILP[RES*] quickly shoots up, while LP[RES*] and the approximations remain PTIME. The bottom plots show a more zoomed-in look, and we see even in the worst case instances, the approximations are only between 1.1x to 1.6x off.

Setting 2: Queries with Self-Joins under Bag Semantics. Figure 4.6 compares two queries with self-joins: SJ-conf: $-R(x, y), R(x, z), A(x), C(z)$ is easy and SJ-chain: $-R(x, y), R(y, z)$ is hard. The stark difference in the solve-time growth clearly indicates their theoretical complexity. While LP-UB increases as the instance grows, it is still far from the 4-factor bound theorized in the worst case. We see that ILP-10 remains a good indicator for the objective value, even when the ILP takes far longer.

Setting 3: Resilience Under Set vs. Bag Semantics. Figures 4.7a and 4.7b show Q_A^Δ , a query that contains a deactivated triad. It is easy under set semantics and hard for bag semantics. However, surprisingly, even with a high max bag size of $1e4$, we always observed $LP[RES^*] = ILP[RES^*]$, and the growth of ILP solve-time remained polynomial. The approximation algorithms are slower, and almost always optimal, differing by less than 1.1x to the optimal in the worst case.

4.8.2 Key Takeaways from Experiments

We summarize the key takeaways from our experiments:

Result 1. (Scalability of ILP for PTIME Cases) For easy cases, solving our ILP encoding is in PTIME.

^{||} The optimal solve-time is LP[RES*] and the optimal resilience is from ILP[RES*].

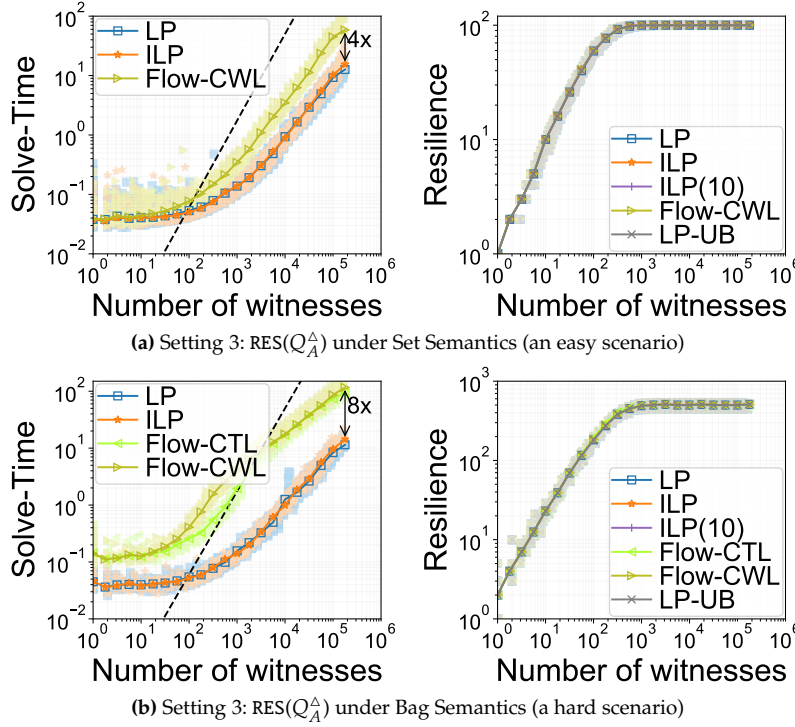


Figure 4.7: Setting 4: $\text{RES}(Q_A^\Delta)$ is easy for sets and hard for bags.

We see the scalability of ILP for PTIME cases in Figure 4.6a. As expected, solving the ILP formulation takes similar time as LP.** We will see in the next chapters, that our unified approach is at times even faster than a previously proposed dedicated flow algorithm.

Result 2. (Correctness of LP for PTIME Cases) Over all experiments, $\text{LP}[\text{RES}^*] = \text{ILP}[\text{RES}^*]$.

Figures 4.6a and 4.7a demonstrate the correctness of the LP relaxation for tractable cases: all algorithms output the same RES values.

Result 3. (Scalability of ILP and its Relaxations for Hard Cases) For hard queries, the LP relaxations always take polynomial time, whereas the ILP solution may take exponential time. However, in practice (and in the absence of “hardness-creating interactions” in data) the ILP can often be solved efficiently.

Figures 4.5 and 4.6b show hard cases. The difference in solve-time is best seen in Figures 4.5 and 4.6, where the ILP overtakes linear scalability. However, interestingly some hard queries don’t show exponential time complexity, and for more complicated queries it actually quite difficult to even synthetically create random data for which solving the ILP shows exponential growth.

Result 4. (Approximation quality) LP-UB is better in practice than the worst-case m -factor bound. The flow based approximations give better approximations, but are slower than the LP relaxation.

** We observed some very surprising cases where the ILP was consistently faster than the LP. We learned from Gurobi Support that this may be due to optimizations applied to the ILP that are not applied to the LP [5], and if such optimizations eliminate numerical issues in the LP [82] such as any issues caused due to the limitations of floating-point arithmetic.

Figures 4.5 and 4.6b show that the results from approximation algorithms are well within theorized bounds and run in PTIME. All approximations are very close to the exact answer, and we need the Δ plots in Figure 4.5 to see any difference between exact and approximate results. We observe that in this case Flow-CW performs better than Flow-CT and is faster as well. LP-UB is faster than the flow-based approximations but can be worse. We also see that the LP approximation is worst when the ILP takes much longer than the LP.

These settings help us answer another interesting question: (5) Do experimental scalabilities give hints about the hardness of queries? We see a rather surprising result.

Result 5. (Practical ILP scalability) *Hard queries may or may not show exponential time requirement in practice.*

Figure 4.6b is a hard query that shows exponential growth. However, while exponential growth of solve-time is a hint for the hardness of a query, the converse is not necessarily true (Figure 4.7b). This explains why our approach of using ILP to solve the problem is *practically motivated*: For realistic instances, or even dense instance but more complicated queries, scenarios where the hardness of the problem actually renders the problem infeasible may be rare.

4.9 Chapter Summary

This chapter presented a novel way of determining the complexity of resilience. We give a unified encoding as ILP and then investigate when an LP approximation is guaranteed to give an integral solution, thereby proving that modern solvers can return the answer in guaranteed PTIME. While this approach is known in the optimization literature [145], it has so far not been applied as *proof method* to establish dichotomy results in reverse data management. Since the resulting theory is somewhat simpler and naturally captures all prior known PTIME cases, we believe that this approach will also help in related open problems for reverse data management, in particular a so far elusive complete dichotomy for resilience of queries with self-joins [60].

Causal Responsibility

5

How responsible is a certain cause for a particular effect? This is a fundamental question that is studied beyond the domain of databases and computer science. Foundational work by Halpern, Pearl, et al. [31, 86, 87] defined the concept of *causal responsibility* that is based *minimal interventions* in the world (in our setting, the input). Meliou et al. [117] adapted this concept to define causal responsibility for database queries. The definition of causal responsibility uses the same idea of minimal interventions that we explored with resilience to provide explanations at a more fine-grained tuple level. For any desired input tuple, users can calculate the “responsibility” of that tuple based on the mathematical notions of counterfactual causality. Then one can derive explanations by ranking input tuples using their responsibilities: tuples with a high degree of responsibility are better explanations for a particular query result. This makes causal responsibility an invaluable tool for query explanations and debugging.

We will see that the problem of causal responsibility is closely related to resilience, and is in fact a strictly more complex problem. In this chapter, we study the complexity of causal responsibility and develop a unified algorithm to solve it.

5.1 Problem Statement

Definition 5.1.1 (Causal Responsibility [59]) *Given query Q and an input tuple t , we say that $k \in \text{RSP}(Q, D, t)$ if and only if $D \models Q$ and there is a contingency set $\Gamma \subseteq D$ with $|\Gamma| \leq k$ such that $D - \Gamma \models Q$ but $D - (\Gamma \cup \{t\}) \not\models Q$.*

In other words, causal responsibility aims to determine whether a *particular input tuple t* (the *responsibility tuple*) can be made “counterfactual” by deleting a set of other input tuples Γ of size k or less. Counterfactual here means that the query is true with that input tuple present, but false if it is also deleted. In contrast to resilience, the problem of responsibility is defined for a *particular tuple t* in \mathcal{D} , and instead of finding a Γ that will leave no witnesses for $D - \Gamma \models q$, we want to preserve only witnesses that involve t , so that there is no witness left for $D - (\Gamma \cup \{t\}) \models Q$. Responsibility measures the *degree* of causal contribution of a particular tuple t to the output of a query as a function of the size of a minimum contingency set (the *responsibility set*). We are again interested in the optimization version of this problem: $\text{RSP}^*(Q, D, t)$.

5.2 Chapter Overview and Contributions

The structure and contributions of this chapter mirror that of the previous Chapter 4. Almost all ideas and techniques from Chapter 4 carry over to this chapter, however, they need to be expanded to account for the extra complexity of the causal responsibility problem. Thus, most proofs, constructions and results in this chapter build on the results from Chapter 4, and it is recommended to read that chapter first.

5.1	Problem Statement	34
5.2	Chapter Overview and Contributions	34
5.3	ILP for Causal Responsibility	36
5.4	PTIME Relaxation of ILP[RSP*]	37
5.5	Complexity results for SJ-Free CQs	38
5.6	Additional Instance Based Complexity Results for Resilience and Causal Responsibility	42
5.7	A note on Approximation Algorithms	43
5.8	Experiments	44
5.9	Chapter Summary	45

This chapter is based on: Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *Proc. ACM Manag. Data* 1, 4, Article 228 (December 2023), 27 pages. <https://doi.org/10.1145/3626715> [110]. Code is available online: <https://github.com/northeastern-datalab/resilience-responsibility-ilp/>

[31]: Chockler and Halpern (JAIR, 2004), ‘Responsibility and Blame: A Structural-Model Approach’. doi:10.1613/jair.1391

[86]: Halpern and Pearl (BJPS, 2001), ‘Causes and Explanations: A Structural-Model Approach: Part 1: Causes’. doi:10.1093/bjps/axi147

[87]: Halpern and Pearl (BJPS, 2005), ‘Causes and Explanations: A structural-model Approach. Part II: Explanations’. doi:10.1093/bjps/axi148

[117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176

Note that it is possible that a given tuple cannot be made counterfactual. For example, given witnesses $\{\{r_{11}\}, \{r_{11}, r_{12}\}\}$, tuple r_{12} cannot be made counterfactual without deleting r_{11} , which in turn would delete both witnesses.

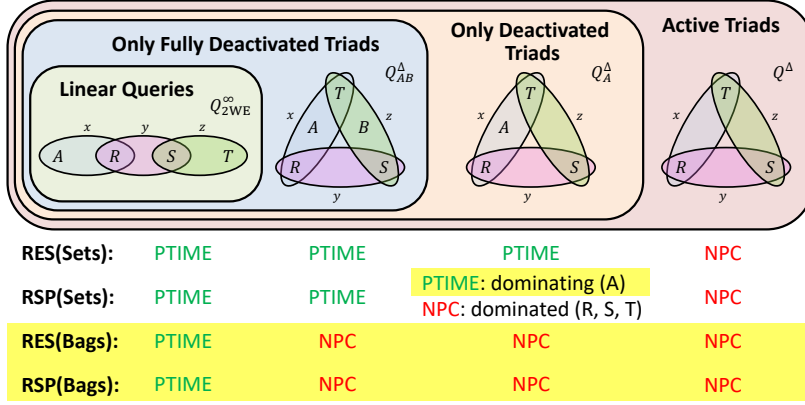


Figure 5.1: Overview of complexity results for self-join-free conjunctive queries (SJ-free CQs) that follow from our unified framework in Chapters 4 and 5. Results highlighted with a yellow background are new. RES stands for resilience and RSP for causal responsibility.

Contributions and Outline. Similar to our contributions for resilience in the previous chapter, we give new theoretical results, approximation guarantees, and experimental results for causal responsibility.

- ① *Unified ILP framework:* We propose an ILP formulation for the problems of causal responsibility that builds on the ILP formulation for resilience from Chapter 4. It still holds the same properties that can not only encode all previously studied variants of the problem, but can also encode *all* formulations of the problem, including self-joins and bag semantics (Section 5.3). An interesting twist that arises when extending the ILP formulation to causal responsibility is that the LP relaxation is not sufficient to prove these unified properties but rather we need an MILP relaxation (Section 5.4).
- ② *First results for causal responsibility under bag semantics:* We give dichotomy results for causal responsibility under bag semantics for the special case of SJ-free CQs (Section 5.5). We show that under bag semantics, the PTIME cases for resilience and responsibility are surprisingly exactly the same (Figure 5.1).
- ③ *Recovering PTIME cases:* We prove that for all prior known PTIME cases of SJ-free queries (as well those shown to be PTIME in this chapter) (under both set and bag semantics), our ILP is solved in guaranteed PTIME by standard solvers (Section 5.5). This means that our formulation is unified not only in being able to model all cases but also in that it is guaranteed to *recover all known PTIME cases by terminating in PTIME*.
- ④ *More tractable cases:* Our approach can also solve causal responsibility and resilience for otherwise hard queries in PTIME for certain types of database instances, such as *read-once* instances, or instances that obey certain *Functional Dependencies* (not necessarily known at the query level) (Section 5.6). In addition, we uncover more tractable cases for causal responsibility, due to obtaining more fine-grained complexity results (Section 5.5).
- ⑤ *Novel approximations:* We observe that the three approximation algorithms for resilience from Section 4.7 can be adapted to causal responsibility, and that the same approximation guarantees hold (Section 5.7).
- ⑥ *Experimental Study:* We run similar experiments for causal responsibility as for resilience in Section 5.8. Our results establish the accuracy of our asymptotic predictions, and show that our ILP formulation is able to solve causal responsibility problems in practice, at times even faster than specialized solutions.

5.3 ILP for Causal Responsibility

The ILP for RSP builds upon ILP[RES*] with an important additional consideration. While the goal of ILP[RES*] was to destroy *all* output witnesses, in ILP[RSP*(Q, D, t)] we must also ensure that not all the output is destroyed. To enforce this, we need additional constraints and additional decision variables to track the witnesses that are destroyed.

1. Decision Variables. ILP[RSP*(Q, D, t)] has two types of decision variables:

- (a) $X[t]$: Tuple indicator variables are defined for all tuples in the set of witnesses we wish to destroy.
- (b) $X[\mathbf{w}]$: Witness indicator variables help preserve at least 1 witness that contains t . We track all witnesses that contain t and set $X[\mathbf{w}] = 1$ if the witness is destroyed and $X[\mathbf{w}] = 0$ otherwise.

2. Constraints. We deal with three types of constraints.

- (a) Resilience Constraints: Every witness that does not contain t must be destroyed. As before, for such witnesses $\mathbf{w}_i = (r_i, r_j \dots r_k)$ we enforce $X[r_i] + X[r_j] + \dots + X[r_k] \geq 1$
- (b) Witness Tracking Constraints: For those witnesses that contain t , we need to track if the witness is destroyed. If any tuple that participates in a witness is deleted, then the witness is deleted as well. Thus, we can enforce that $X[\mathbf{w}] \geq X[t]$ where $t \in \mathbf{w}$. Notice that we just care about tuples that need to be potentially deleted, i.e. only tuples that occur in witnesses without t .
- (c) Counterfactual Constraint: A single constraint ensures that at least one of the witnesses that contains the responsibility tuple is preserved. As example, if only the witnesses $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ contain t , then this constraint is $X[\mathbf{w}_1] + X[\mathbf{w}_2] + X[\mathbf{w}_3] \leq 2$.

3. Objective. The objective is the same as for ILP[RES*(Q, D)]: we minimize the number of tuples deleted (weighted by the number of occurrences).

Theorem 5.3.1 (ILP[RSP*] Correctness) $\text{ILP}[\text{RSP}^*(Q, D, t)] = \text{RSP}^*(Q, D, t)$ of a tuple t in database instance \mathcal{D} under CQ Q under set or bag semantics.

Proof Intuition (Theorem 5.3.1). Like Theorem 4.3.2, we prove validity and then optimality. We show that for any responsibility set we can assign values to the ILP variables such that they can form a satisfying solution (this follows from that fact that the responsibility set must preserve at least one witness containing t).

Proof Theorem 5.3.1. Similar to Theorem 4.3.2, we show validity and optimality.

- **Proof of Validity:** An invalid solution is not counterfactual, i.e., either it does not destroy all witnesses without t or it destroys all witnesses. The former violates the resilience constraints, while the latter violates the Counterfactual constraint.
- **Proof of Optimality:** Any strictly smaller, valid responsibility set R' can also be translated into variable assignment \tilde{X} to $X[t]$ such that it satisfies all constraints (where $\tilde{X}[t] = 1$ if $t \in R'$). Since R' is valid, at least one tuple for each witness that does contain t is destroyed - thus resilience constraints are fulfilled. There must be at least one witness \mathbf{w}_p containing t that is preserved. For this witness, we know

that $X[w_p] = 0$ is valid (since there is no $t' \in w_p$ s.t. $X[t'] = 1$). Thus, the counterfactual constraint is also fulfilled since $\sum X[w] < |W_p|$. Thus, ILP[RSP*] Q, D calculates the optimal responsibility. \square

Example 5.3.1 Consider $Q_2^\infty := R(x, y), S(x, y)$ and database instance \mathcal{D} with $R = (1, 1), S = \{(1, 1), (1, 2), (1, 3)\}$.

x	y	z	
1	1	1	$\mathbf{w}_1 = \{r_{11}, s_{11}\}$
1	1	2	$\mathbf{w}_2 = \{r_{11}, s_{12}\}$
1	1	3	$\mathbf{w}_3 = \{r_{11}, s_{13}\}$

How do we calculate the responsibility of s_{11} ? First, we must destroy the two witnesses that do not contain s_{11} i.e. \mathbf{w}_2 and \mathbf{w}_3 . The tuple indicator variables we need are - $X[r_{11}]$, $X[s_{12}]$, $X[s_{13}]$. (Notice that s_{11} is not tracked itself.) Since we need to track \mathbf{w}_1 to ensure it isn't destroyed, we need the witness indicator variable $X[\mathbf{w}_1]$. The resilience constraints are:

$$X[r_{11}] + X[s_{12}] \geq 1$$

$$X[r_{11}] + X[s_{13}] \geq 1$$

The witness tracking constraints apply only to $X[\mathbf{w}_1]$:

$$X[\mathbf{w}_1] \geq X[r_{11}]$$

Finally, we use the counterfactual constraint to enforce that at least one witness is preserved. In this example, this implies directly that \mathbf{w}_1 may not be destroyed.

$$X[\mathbf{w}_1] \leq 0$$

Solving this ILP gives us an objective of 2 when $X[s_{12}] = 1$ and $X[s_{13}] = 1$ and all other variables are set to 0. Notice that setting $X[r_{11}]$ to 1 will force $X[\mathbf{w}_1]$ to take value 1 and hence violate the counterfactual constraint. Intuitively, r_{11} cannot be in the responsibility set because deleting it will delete all output witnesses, and not allow s_{11} to be counterfactual.

5.4 PTIME Relaxation of ILP[RSP*]

The previous section introduced unified ILPs to solve for RSP. Just like we did for resilience, we would like to use a relaxation of the ILP to solve the tractable cases of RSP in PTIME. For responsibility, the relaxation is more intricate, and we need to introduce a Mixed Integer Linear Program (MILP) MILP[RSP*].

5.4.1 MILP Relaxation for RSP

It turns out that an LP relaxation is not optimal for PTIME cases (Example 5.4.1). We introduce a Mixed Integer Linear Program MILP[RSP*], where tuple indicator variables are relaxed and take values in $[0, 1]$

whereas witness indicator variables are restricted to values $\{0, 1\}$. Typically, MILPs are exponential in the number of integer variables i.e. if there are n integer binary variables, a solver explores 2^n possible branches of assignments. However, despite having an integer variable for every witness that contains t (thus up to linear in the size of the database), we show that $\text{MILP}[\text{RSP}^*]$ is in PTIME.

Theorem 5.4.1 (PTIME solvability of $\text{MILP}[\text{RSP}^*]$) *For any CQ Q and tuple t , $\text{MILP}[\text{RSP}^*(Q, D, t)]$ can be solved in PTIME in the size of database \mathcal{D} .*

Proof Intuition. All witness indicator variables are combined into one counterfactual constraint. This constraint is always satisfied when any one of the variable takes value 0, irrespective of other variable values. Thus, we only need to explore the assignments where exactly 1 variable takes on value 0, thus a linear number of assignments in the size of the database.

Proof Theorem 5.4.1. Assume that there are c_t witnesses that contain t . The counterfactual constraint enforces that at least one of these witnesses is preserved. Notice that the witness indicator variables have no effect on the objective, and can be set to any value so long as all constraints are fulfilled. Any assignment where 1 witness is preserved, and the rest are destroyed fulfills all constraint (even witness tracking constraints, which only enforce that a witness is destroyed if one of its tuples is destroyed, but does not enforce that the witness cannot be destroyed otherwise). Thus, we can restrict ourselves to c_t potential assignments of witness indicator variables instead of 2^{c_t} . Trivially, we can now solve the problem by running c_t Linear Programs (where the only variables are tuple indicator variables and the witness indicator variables are fixed to one out of c_t assignments). Since c_t is polynomial in the database size, we see that $\text{MILP}[\text{RSP}^*]$ can be solved in PTIME. In practice, ILP solvers solve the problem faster than the algorithm in the proof, since they leverage common insights across the c_t Linear Programs. \square

We show experimentally in [Section 5.8](#) that a typical ILP solver indeed solves $\text{MILP}[\text{RSP}^*]$ in PTIME.

Example 5.4.1 Consider again the problem in [Example 5.3.1](#). The solution of $\text{ILP}[\text{RSP}^*]$ was 2 at $X[s_{12}] = 1$, $X[s_{13}] = 1$, $X[r_{11}] = 0$ and $X[w_1] = 0$. What happens if we relax the integrality constraints and allow $0 \leq X[v] \leq 1$ for all variables? We can get a smaller satisfying solution 1.5 at the point $X[s_{12}] = 0.5$, $X[s_{13}] = 0.5$, $X[r_{11}] = 0.5$ and $X[w_1] = 0.5$. This value is $\text{LP}[\text{RSP}^*]$ and is *not* guaranteed to be equal to $\text{ILP}[\text{RSP}^*]$. If we instead create $\text{MILP}[\text{RSP}^*]$ and apply *integrality constraints only for the witness indicator variables*, then $X[w_1]$ is forced to be in $\{0, 1\}$ while all other variables can be fractional. We see that the $\text{LP}[\text{RSP}^*]$ solution is no longer permitted, and solving $\text{MILP}[\text{RSP}^*]$ results in the true RSP value of 2. We show in [Section 5.5](#) that $\text{MILP}[\text{RSP}^*] = \text{ILP}[\text{RSP}^*]$ for all easy cases.

5.5 Complexity results for SJ-Free CQs

This section follows a similar pattern as the previous one to prove that for every SJ-free CQ, either $\text{MILP}[\text{RSP}^*]$ solves RSP exactly (and the problem

Table 5.1: Complexity Landscape for Resilience and Responsibility. All results follow from our unified framework. Results highlighted with a yellow background are new. Some theorems apply to multiple settings (thus shown repeatedly) and include earlier results as special cases.

Type of query	Example	Set Semantics		Bag Semantics	
		RES	RSP	RES	RSP
Linear Queries	Q_2^∞	PTIME (thm. 4.6.1)	PTIME (thm. 5.5.1)	PTIME (thm. 4.6.1)	PTIME (thm. 5.5.1)
With Only Fully Deactivated Triads	Q_{AB}^Δ	PTIME (thm. 4.6.2)	PTIME (thm. 5.5.2)	NPC (thm. 4.6.3)	NPC (thm. 5.5.5)
With Only Deactivated Triads 1. Dominating Relation 2. Dominated Relation	Q_A^Δ		PTIME (thm. 5.5.3)		
			NPC (thm. 5.5.4)		
With Active Triads	Q^Δ, Q_3^\star	NPC (thm. 4.5.3)	NPC (thm. 5.5.5)		

is hence easy), or we can form an IJP for RSP. We also compare the complexity of RSP with the complexity of RES in Table 5.1.

Theorem 5.5.1 (Integrality of MILP[RSP*] for Linear Queries) *For all database instances \mathcal{D} under set or bag semantics, $\text{MILP}[\text{RSP}^*(Q, D, t)] = \text{RSP}^*(Q, D, t)$ if Q is linear.*

Proof Intuition (Theorem 5.5.1). Prior work [117] solves responsibility by solving the min-cut problem for multiple flow graphs, by creating one flow graph for each witness containing t . We prove a correspondence between all valid MILP[RSP*] solutions and valid cuts of the flow graphs, thereby showing that both approaches lead to identical solutions.

Proof Theorem 5.5.1. Let X_m be an optimal variable assignment generated by solving $\text{MILP}[\text{RSP}^*(Q, D, t)]$. There must be at least one witness $w_p \in D$ such that $t \in w_p$ and $X_m[w_p] = 0$ i.e. the witness is not destroyed (this follows from the fact that all witness variables take values in $\{0, 1\}$ and the counterfactual clause enforces that they all cannot take value 1). For such a witness, any tuple $t \in w_p$, must have $X[t] = 0$ since it satisfies the witness tracking constraints. We also know that since Q is a linear query, the witnesses can be encoded in a flow graph to find the responsibility [59, 117]. We can map the values of X_m to the flow graph, where $X_m[t]$ now denotes if an edge in the flow graph is cut or not. Consider $X_m[t] = 0$, since it is not modeled in MILP[RSP*]. We see that this disconnects all paths in the graph (since paths that do not contain t are disconnected by virtue of the resilience constraints of MILP[RSP*]). If we set the weight of all tuples in w_p to ∞ , the cut value does not change since these tuples were not part of the cut. Prior work [117] has shown that $\text{RSP}(Q, D)$ for linear queries can be calculated by taking the minimum of minimum cuts of all flow graphs such that have 1 of witnesses that contains t , has weight of all other tuples edges set to ∞ . Thus, $\text{MILP}[\text{RSP}^*(Q, D, t)]$ is at least as much as the responsibility computed by a flow graph. In addition to this, the flow graph with the smallest cut also fulfills all the solutions for MILP[RSP*] (since at least one witness containing t is preserved, and all witnesses not containing t are cut). Thus, the optimal value of $\text{RSP}(Q, D, t)$ can be mapped back to a MILP[RSP*] assignment. \square

[117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

[117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176

[117]: Meliou, Gatterbauer, Moore, and Suciu (PVLDB, 2010), ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. doi:10.14778/1880172.1880176

Theorem 5.5.2 (Integrality of MILP[RSP*] with fully deactivated triads under set semantics) $\text{MILP}[\text{RSP}^*(Q, D, t)] = \text{RSP}^*(Q, D, t)$ for any database \mathcal{D} under set semantics if all triads in Q are fully deactivated.

Proof Intuition (Theorem 5.5.2). This follows from the fact that fully deactivated triads can be linearized without changing the optimal solution [59] and Theorem 5.5.1.

Proof Theorem 5.5.2. Assume Q contains a triad with tables R, S, T . However, since the triad is fully deactivated, at least one of these tables must be dominated by set of tables $A_1, A_2 \dots$. W.l.o.g., assume R is fully dominated. We show that R can be made exogenous because there exists an optimal responsibility set that does not contain any tuple from R . If r_i is part of the responsibility set, then it can be replaced with a_i where $a_i \subset r_i$ while still destroying the same or more witnesses. However, it is still possible that including a_i in the responsibility set may destroy all witnesses. This is possible only if a_i dominates t as well. If all a_i such that $a_i \subset r_i$ dominate t , then it must be that r_i dominates t (since r_i is fully dominated and uniquely determined by the tuples that dominate it). It is not possible for such an r_i to be in the responsibility set as it would destroy all witnesses containing t . Thus, no tuple from R can be used in the responsibility set, the size of the responsibility set will not change if we make R exogenous i.e. add all the variables of the query to R . Let Q' be the query where for each deactivated triad, all fully dominated tables have been made exogenous. Then $\text{RSP}(Q', D, t) = \text{RSP}(Q, D, t)$. Q' is linear, and we can then use Theorem 5.5.1 to show that $\text{MILP}[\text{RSP}^*(Q, D)]$ is optimal. \square

Theorem 5.5.3 (Integrality of MILP[RSP*] when responsibility tuple is in dominating relation) $\text{LP}[\text{RSP}^*(Q, D, t)] = \text{RSP}^*(Q, D, t)$ for all database instances \mathcal{D} under set semantics if Q does not contain any active triad and t belongs to an atom that dominates some atom in all deactivated triads in Q .

Proof Intuition (Theorem 5.5.3). We prove that in every deactivated triad dominated by A , it is always safe to make the dominated table R exogenous since any tuple from R in the responsibility set is either replaceable, or invalid. This linearizes the query, and the rest follows from Theorem 5.5.1. Notice that prior work [59] identified as tractable cases those without any *active* triad, which a special case of our more general tractable cases.

Proof Theorem 5.5.3. Let R be the table in a deactivated triad that A dominates. We show that no tuple of R is required in the responsibility set, and we can make it exogenous. If some r_i is in the responsibility database, it can be replaced with some a_i if the variables and valuation of a_i are a strict subset of r_i and then a_i deletes all the witnesses as before, and potentially some more. This is permitted unless removal of a_i deletes all witnesses containing t as well. However, since a_i and t belong to the same table, this is not possible. Thus, at least one table from each triad can be made exogenous, and the query can be replaced with a linear query. \square

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), 'The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries'.
doi:10.14778/2850583.2850592

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), 'The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries'.
doi:10.14778/2850583.2850592

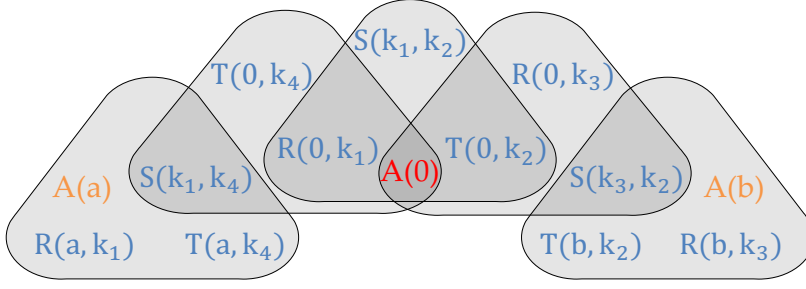


Figure 5.2: IJP for $RSP(Q_A^\Delta)$ for tables R , S and T

Theorem 5.5.4 (Hardness of RSP for tuples in not fully deactivated triads) $RSP(Q, D, t)$ is NPC if t belongs to an atom that is part of a triad that is not fully deactivated.

Proof Intuition (Theorem 5.5.4). The key principle behind this proof is our more fine-grained notion of exogenous tuples. A tuple a such that a has all the same values for the same variables as t and $\text{var}(a) \subseteq \text{var}(t)$ is necessarily exogenous since it is not possible for t to become counterfactual if a is removed.

Proof Theorem 5.5.4. If T is part of an active triad, the same IJP as RES is proof for this theorem. However, if T is part of a deactivated triad, then we need to slightly modify the hardness proof. Let A be the table that dominates one of the tables in the deactivated triad. In our IJP we ensure that an atom from A is an *exogenous tuple*- one that cannot be deleted. This is possible by constructing an a_i that dominates t . Since this is always possible, we can now construct the rest of the IJP. We connect a witness containing a_0 to two others by using two tables of the deactivated triad. Then we finally add two more witnesses to the triad with the common tuple being the third table of the deactivated triad. We treat the A table as the endpoints of the IJP. Since a_i is exogenous, the gadget must choose between the first or the second table to destroy all witnesses in the IJP. Such a gadget does not form new witnesses when composed as well as any two isomorphs share only tuples from A . \square

In Figure 5.2, we show an example for the IJP that greatly simplifies the previous hardness gadget (the earlier gadget was a reduction from 3SAT whose variable gadget had 80 witnesses)

Theorem 5.5.5 (RSP is harder than RES) If $RES(Q)$ is NPC for a query Q under set or bag semantics then so is $RSP(Q)$.

Proof Intuition (Theorem 5.5.5). We give a reduction from $RES(Q)$ to $RSP(Q)$ in both set and bag semantics by adding a witness to the given database instance and selecting a tuple whose responsibility is equal the resilience of the original instance. Our approach extends a prior result [59] that applied only to set semantics.

Proof Theorem 5.5.5. Consider an arbitrary database instance \mathcal{D} and add all tuples from a witness w_r that is disjoint from all tuples in \mathcal{D} . The responsibility of the resulting database instance is simply the resilience of \mathcal{D} (since all witnesses in \mathcal{D} must be destroyed, and the other singleton witness must be preserved). Thus, we can reduce $RSP(Q, D, t)$ to $RES(Q, D)$ and $RSP(Q)$ must be hard whenever $RES(Q)$ is. \square

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

These results imply the following dichotomies under both set and bag semantics:

Corollary 5.5.6 (Causal Responsibility Dichotomy under set semantics) *Under set semantics, $\text{RSP}(Q, D, t)$ is in PTIME for queries that contain only fully deactivated triads or deactivated triads that are dominated by the relation of t , otherwise it is NPC.*

Corollary 5.5.7 (Causal Responsibility Dichotomy under bag semantics) *Under bag semantics, $\text{RSP}(Q, D, t)$ is in PTIME for queries that do not contain any triads, otherwise it is NPC.*

Notice that the tractability frontier for bag semantics notably differs from set semantics, where the tractable cases for $\text{RSP}(Q)$ are a *strict subset* of those for $\text{RES}(Q)$. For bags, they coincide:

Corollary 5.5.8 (Tractable cases of RES and RSP coincide under bag semantics) *Under bag semantics, the tractable cases for $\text{RSP}(Q)$ are exactly the same as for $\text{RES}(Q)$.*

5.6 Additional Instance Based Complexity Results for Resilience and Causal Responsibility

We give here two cases for when our unified algorithm is *guaranteed to terminate* in PTIME for generally hard queries - for both resilience and causal responsibility. The interesting aspect is that our unified algorithm terminates in PTIME if the database instance fulfills those conditions, but the algorithm *does not need to know about these conditions as input*, it just automatically leverages those during query time. We believe that this really shows the power of our unconventional approach of proposing one unified approach for all problems and then proving termination in PTIME for increasing number of cases (instead of starting from a dedicated PTIME solution for special cases).

Read-Once Instances. We show that database instances which allow a read-once factorization of the provenance for a given query are always tractable. A Boolean function is called *read-once* if it can be expressed as a Boolean expression in which every variable appears exactly once [35, 74, 75]. We call a database \mathcal{D} *read-once instance* for query Q if the provenance of the query over \mathcal{D} can be represented by a read-once expression.

Theorem 5.6.1 (Integrality of $\text{MILP}[\text{RSP}^*]$ for read-once instances) *$\text{LP}[\text{RES}^*]$ and $\text{MILP}[\text{RSP}^*]$ always have optimal, integral solutions under set or bag semantics for all database instances \mathcal{D} that are read-once for query Q .*

Proof Theorem 5.6.1. We use a structural property of the constraint matrix of the LP to show that $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}(Q, D)$. A $\{0, 1\}$ -matrix M is balanced iff M has no square submatrix of odd order, such that each row and each column of the submatrix has exactly two 1s. If a matrix M is balanced, then the polytope $Mx \geq 1$ is Total Dual Integral (TDI),

[35]: Crama and Hammer (2011), *Boolean Functions: Theory, Algorithms, and Applications*. doi:10.1017/cbo9780511852008.003

[74]: Golumbic and Gurvich (2011), 'Read-once functions'. doi:10.1017/cbo9780511852008.011

[75]: Golumbic, Mintz, and Rotics (JDAM, 2006), 'Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees'. doi:10.1016/j.dam.2005.09.016

which means all vertices of the polytope are integral [144]. For such a system, the optimal Linear Program solution will always have an Integral solution. We first show that the constraint matrix of $\text{LP}[\text{RES}^*(Q, D)]$ is 0, 1-balanced when \mathcal{D} is read-once. A 0, 1 balanced matrix is one that does not contain any odd square submatrix having all row sums and all column sums equal to 2.

Assume the constraint matrix is unbalanced. Then there must be a set of witnesses $(w_1, w_2, w_3 \dots)$ such that w_1 and w_2 share tuple t_1 but not t_2 , and w_2 and w_3 share t_2 . This defines a P_4 , which is not permitted in a read-once instance. Thus, the constraint matrix is balanced and $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}(Q, D)$.

Now for $\text{ILP}[\text{RSP}^*(Q, D, t)]$, if there is a tuple x that exists in a witness with t (w_i) as well as in a witness without t (w_j), then x must exist in all witnesses ($w_k \dots$) containing t to prevent the formation of a P_4 . (There would be a P_4 as w_k and w_i share t , w_i and w_j share x but w_k and w_j do not share t or x .) If x participates in all witnesses containing t it cannot be part of the responsibility set as it would violate the counterfactual constraint by preserving no witnesses. Hence, the responsibility set consists wholly of tuples that do not interact with t and the problem reduces to resilience, which we know is PTIME for read-once instances. \square

Functional Dependencies (FDs). A Functional Dependency (FD) is a constraint between two sets of attributes X and Y in a relation of a database instance \mathcal{D} . We say that X functionally determines Y ($X \rightarrow Y$) if whenever two tuples $r_1, r_2 \in R$ contain the same values for attributes in X , they also have the same values for attributes in Y [104]. Prior work introduced an *induced rewrites* procedure [59] which, given a set of FDs, rewrites a query to a simpler query without changing the resilience or responsibility. If the query after an induced rewrite is in PTIME, then the original could be solved after performing a transformation. We prove that any instance that is PTIME after an induced rewrite is *automatically* easy for our ILPs. Thus, if there are *undetected* FDs in the data that would allow a PTIME rewrite, our framework guarantees PTIME performance, while prior approaches would classify it as hard.

Theorem 5.6.2 (Integrality of $\text{MILP}[\text{RSP}^*]$ with certain functional dependencies) *Let Q' be the induced rewrite of Q under a set of FDs. If $\text{RES}(Q')$ or $\text{RSP}(Q')$ are in PTIME under set or bag semantics then $\text{LP}[\text{RES}^*]$ and $\text{MILP}[\text{RSP}^*]$ always have optimal integral solutions under the same semantics.*

Proof Theorem 5.6.2. Prior work [59] showed that FDs can make things easy and be used to transform non-linear queries to linear queries. We can make the same argument as Theorem 5.5.1 to show that $\text{LP}[\text{RES}^*(Q)]$ or $\text{MILP}[\text{RSP}^*(Q)]$ cannot be smaller than the resilience or responsibility respectively found by the min-cut algorithm of the flow graph produced by the query after linearization. \square

5.7 A note on Approximation Algorithms

All three approximation algorithms for resilience from Section 4.7 can be trivially applied to causal responsibility as well. It is interesting however, to note that the approximation guarantees for resilience (shown for the LP based approximation algorithm) also hold for causal responsibility.

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

[104]: Kolahi (2009), 'Functional Dependency'. doi:10.1007/978-0-387-39940-9_1247

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), 'The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries'. doi:10.14778/2850583.2850592

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), 'The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries'. doi:10.14778/2850583.2850592

Theorem 5.7.1 (m-factor approximation for RSP) *The LP Rounding Algorithm is a PTIME m-factor approximation for RSP.*

Proof Theorem 4.7.1. We can assume the correctness of the approximation for RES as shown in Theorem 4.7.1. Now to prove the correctness of the approximation for MILP[RSP*] as well, we need to verify the extra constraints. We must ensure that the resultant variable assignment fulfills the Counterfactual Constraints to ensure that not all witnesses are deleted. However, since in the Mixed ILP, the witness variables already took on integral values, there was at least one witness w_p containing t such that $X[w] = 0$. This implies that in the MILP, all tuples t in w_p have $X_f[t] = 0$. They will stay 0 after rounding as well, and thus the Witness Tracking Constraints and Counterfactual Constraint are still satisfied. \square

5.8 Experiments

We evaluate our algorithms for causal responsibility with the exact same experimental setup as in Section 7.12, but now with TPC-H data [152]. We use the TPC-H data generator at logarithmically increasing scale factors, creating 18 databases ranging from scale factor 0.01 to 1.

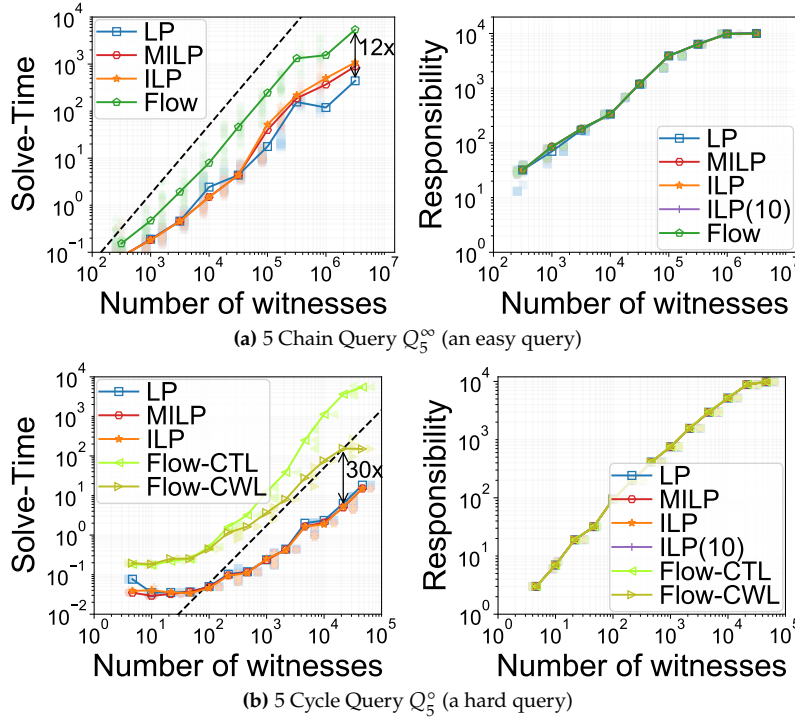


Figure 5.3: TPC-H data with FDs.

Causal Responsibility Scalability With TPC-H Data. Figure 5.3 shows results for the 5-chain query Q_5^∞ :- Customer(custname, custkey), Orders(custkey, orderkey), Lineitem(orderkey, psid), Partsupplier(id, suppkey) and 5-cycle query Q_5^o :- Customer(custname, custkey), Orders(custkey, orderkey), Lineitem(orderkey, psid), Partsupplier(id, suppkey), Supplier(suppkey, suppname) over TPC-H data. While in general Q_5^∞ is NPC, a careful reader may notice that all the joins have primary-foreign key dependencies. We do not inform our algorithms about these dependencies nor make any changes to accommodate them. Yet the solver is able to leverage the dependencies from the data instance and ILP[RES*]

scales in PTIME, and we see that the ILP is faster than the both dedicated flow algorithm and flow approximation, respectively. In both cases, all algorithms (exact and approximate) return the correct responsibility.

We observe similar trends as in [Section 7.12](#).

Result 6. (Scalability of ILP for PTIME Cases) *For easy cases, solving our ILP encoding is in PTIME and at times even faster than a previously proposed dedicated flow algorithm.*

We see the scalability of ILP for PTIME cases in [Figure 5.3a](#). As expected, solving the ILP formulation takes similar time as LP. We see that Gurobi can solve responsibility around 12 times faster for a PTIME query ([Figure 5.3a](#)) than the previously proposed flow encoding.

Result 7. (Correctness of LP for PTIME Cases) *Over all tractable instances, $\text{MILP}[\text{RSP}^*] = \text{ILP}[\text{RSP}^*]$.*

[Figure 5.3a](#) corroborate the correctness of the MILP relaxation for PTIME queries, as expected due to the theorems proved in [Section 5.5](#).

Result 8. (Scalability of ILP and its Relaxations for Hard Cases) *For hard queries, we observe that the time taken by the MILP relaxations grows polynomially, while the time taken by the ILP solution grows exponentially. However, in practice (ain the presence of functional dependencies), the ILP can often be solved efficiently.*

[Figure 5.3b](#) shows a hard query. Interestingly, it does not show exponential time complexity. This can be explained by the fact that the ILP solver is able to leverage the dependencies in the data instance, and thus solve the problem in PTIME.

5.9 Chapter Summary

In this chapter, we saw that the ideas of unified ILP encodings can be extended beyond the simple problem of resilience to the more involved problem of causal responsibility. In particular, rather than seeing a problem instance solvable by an LP relaxation (with equivalence to a minimum cut in a flow graph), we need to see the problem instance as a PTIME solvable MILP problem (corresponding to minimum cut in multiple flow graphs). In the next chapter, we will see how the same ideas can be extended to an entire class of RDM problems, namely *deletion propagation* problems.

Generalized Deletion Propagation

6

Deletion Propagation (DP) was proposed as early as 1982 [44] and corresponds to the basic view-update problem: *Assume we want to delete a given tuple from a view, which tuples should we delete from the database to accomplish this goal?* Since different tuples in the view may be generated from the same tuple in the source database, deleting tuples from the source can have additional “side effects” beyond the user-requested deletion. Thus, the problem is not only to determine tuples to delete from the source such that the requested view deletion is achieved, but also to minimize other possible side effects, leading to a combinatorial optimization problem. Different choices about what constitutes a “side effect” and different optimization goals have led to several well-motivated variants of Deletion Propagation being proposed and studied over the last 40+ years. These variants have found a variety of applications, including query explainability and debugging.

“Side effects” are usually measured in the number of tuples affected by a modification. Two important types of side effects that have been studied are source side effects [23, 44, 30pt] and view side effects [23, 101, 102]: *Source side effects* (DP-SS) measure the number of tuples deleted from the source database to fulfill the user request, while *view side effects* (DP-VS) measure the number of unintended tuples deleted from the same view. A recent variant on the DP-SS problem is the *aggregated deletion propagation* (ADP-SS) problem [94] in which a certain number of tuples should be deleted from the view, but it is not specified which tuples. A different, seemingly unrelated problem is the recently proposed *smallest witness problem* (SWP) [93, 121], where a user would like to preserve the view as is, but delete as many tuples from the source as possible. Although SWP has so far not been understood to be a variant of DP, we show that this problem shares the same structure as other DP problems, can be solved using the same techniques, and – when combined with other DP problems – opens up a new space of natural DP variants.

Despite the long history of Deletion Propagation, at least 3 challenges remain. This chapter shows that these 3 challenges can be largely addressed by casting the existing problems as special cases of a unified general “General Deletion Propagation” framework.

Challenge 1: Countless well-motivated variants. DP has been studied in many forms over the last 40+ years. However, one can imagine many more variants that are all well-motivated, and that have not yet been studied. These variants can arise from different definitions of side effects, different constraints on allowed side effects, and different optimization goals. [Example 6.0.1](#) gives just one such example of DP that has not been described by prior work (we explore the wider space of variants more thoroughly in [Section 6.2](#)).

Example 6.0.1 An airline company wants to cut costs by reducing the number of flights it offers, and reduce its total operational expenditure by at least 2%. There are various types of costs incurred by the flight company, such as the fuel cost of the flight and the airport fee at the locations they operate at. While cutting costs, the airline wants

6.1	Chapter Overview and Contributions	48
6.2	Problem Statement	49
6.3	ILP Framework for GDP	53
6.4	Recovering Existing Tractability Results	62
6.5	New Tractability Results	69
6.6	Experiments	70
6.7	A Note on System Implementation	76
6.8	Chapter Summary	77

This chapter is based on: *Neha Makhija and Wolfgang Gatterbauer. 2025. Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach for Generalized Deletion Propagation. PVLDB [112]. Code is available online: <https://github.com/north-eastern-datalab/generalized-deletion-propagation>*

[44]: Dayal and Bernstein (TODS, 1982), ‘On the Correct Translation of Update Operations on Relational Views’. doi:10.1145/319732.319740

[23]: Buneman, Khanna, and Tan (PODS, 2002), ‘On Propagation of Deletions and Annotations Through Views’. doi:10.1145/543613.543633

[101]: Kimelfeld (PODS, 2012), ‘A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies’. doi:10.1145/2213556.2213584

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[93]: Hu and Sintos (ICDT, 2024), ‘Finding Smallest Witnesses for Conjunctive Queries’. doi:10.4230/LIPIcs.ICDT.2024.24

[121]: Miao, Roy, and Yang (SIGMOD, 2019), ‘Explaining Wrong Queries Using Small Examples’. doi:10.1145/3299869.3319866

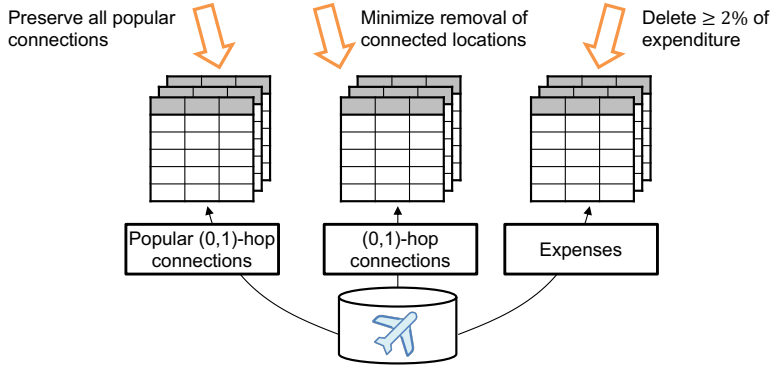


Figure 6.1: Example 6.0.1: A motivating example for Generalized Deletion Propagation (GDP) that extends previously studied DP variants.

to ensure that it minimizes the effect on its connectivity network i.e. pairs of locations that are connected directly or via 1 layover called “(0, 1)-hop connections.” Additionally, the airline would like to ensure that it maintains a profitable service and so it would like to preserve of all of its most popular connections.

This problem has all the ingredients of a Deletion Propagation problem: the source database is the set of all flights and airports; the view is the set of all location pairs that have direct or 1-hop connection between them. The airline would like to delete a certain amount of flight and airport costs (corresponding to cancelling flights, and not having service to an airport) - but it would like to minimize the side effect on the view (the connectivity network) and preserve output tuples of a different view (that shows the most popular connections). This problem is a mixture of Aggregated Deletion Propagation (which involves deleting an arbitrary fraction of a view), and the Smallest Witness Problem (which involves preserving a view), but is also an extension in many ways (discussed further in Section 6.3). For example, the side effects are not measured in the original source or view, but *in a different view (!)*.

Challenge 2: Dissimilar algorithms for similar problems. Since DP variants have been studied in isolation, the algorithms used to solve these problems are often dissimilar - ranging from dynamic programming-based approaches to flow-based algorithms. Even for one variant, different queries currently require different algorithms. Thus, new variants are often solved “from scratch” and algorithmic insights are not carried over. DP variants are NPC (NP-complete) in general, but are PTIME for certain queries. To solve DP for a query optimally, one needs to know the algorithm that can correctly solve the problem variant for the given query in PTIME (if such an algorithm exists), and know that the query and database fulfill the requirements that allow applying the specialized algorithm. Since algorithms that are specific to the variant and query, they are not generalizable, easily implementable, or extensible to new variants and query classes. As discussed in the introduction of this dissertation, we propose a unified “*coarse-grained instance optimal*” framework. Our framework includes *all* previously studied DP variants, including even problems that were not previously phrased as DP (SWP), and new variants as well.

Challenge 3: Algorithms and tractability criteria are unknown for many real-world queries and scenarios. DP problems are typically studied for self-join-free conjunctive queries under set semantics, because queries with self-joins are known to be notoriously difficult to analyze, and several complexity boundaries have been open for over a decade [102].

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

In practice, however, queries often contain unions, are not self-join-free, and are executed under bag semantics. Only very few algorithms and tractability results are known for these more complicated settings, such as for queries with self-joins [60, 101], unions of conjunctive queries [17], and queries under bags semantics [110]. The overall tractability criterion for queries for such “real-world” queries is overall ill-understood.

6.1 Chapter Overview and Contributions

We solve the challenges outlined above by introducing a unified framework for Deletion Propagation (DP) problems. We define Generalized Deletion Propagation (GDP), show that this definition encapsulates existing variants as well many natural new variants, and give a unified algorithm to solve GDP. In the process, we recover known tractability results, derive new theoretical results, and provide an experimental validation.

- ① We define Generalized Deletion Propagation (GDP) in [Section 6.2](#). This definition not only covers all known DP variants, but also includes the Smallest Witness Problem (SWP, which has so far been treated as completely different), and covers new well-motivated variants. Our definition allows us to reason about the many DP variants systematically, thus addressing **Challenge 1**.
- ② We present an Integer Linear Programming (ILP) formulation for the GDP problem in [Section 6.3](#). This formulation allows us to *use one solution for all variants* of DP, thus providing the first step in addressing **Challenge 2**. The ILP formulation can cover queries with unions and self-joins, and both the set and bag semantics settings, thus giving a valuable tool to address **Challenge 3**.
- ③ While providing ILP formulations is a typical approach for solving NPC optimization problems, our key technical contribution addressing **Challenge 3** is proposing an ILP *with the right algorithmic properties*: We show in [Section 6.4](#) that for *all known* PTIME cases, our ILP formulation is solvable in PTIME via an LP relaxation. Thus, we do not need dedicated PTIME algorithms for special cases; our theory shows that standard ILP solvers default to solving these cases in PTIME. This means that the ILP framework can be directly used to solve all tractable instances of DP, thus resolving **Challenge 2** for all known PTIME cases. Notice that it is *not trivial* to come up with the right ILP formulation. We show that a more obvious ILP formation does not have the desired PTIME guarantees, and can be over 2 orders of magnitude slower in practice.
- ④ We uncover a new tractable case for well-known variants of the DP problem, thus showing that our framework is a powerful tool to address **Challenge 3**, the long-standing open challenge of capturing the exact tractability boundary. Concretely, we prove in [Section 6.5](#) that the ILP formulation of a query with union and self-join that can be solved in PTIME under bag semantics.
- ⑤ We experimentally evaluate the efficiency of our contributions in [Section 6.6](#). Our approach performs comparably and sometimes even better than specialized algorithms for particular DP variants, and can solve new tractable cases that were not previously known.

[60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

[101]: Kimelfeld (PODS, 2012), ‘A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies’. doi:10.1145/2213556.2213584

[17]: Bodirsky, Semanisinová, and Lutz (LICS, 2024), ‘The Complexity of Resilience Problems via Valued Constraint Satisfaction Problems’. doi:10.1145/3661814.3662071

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

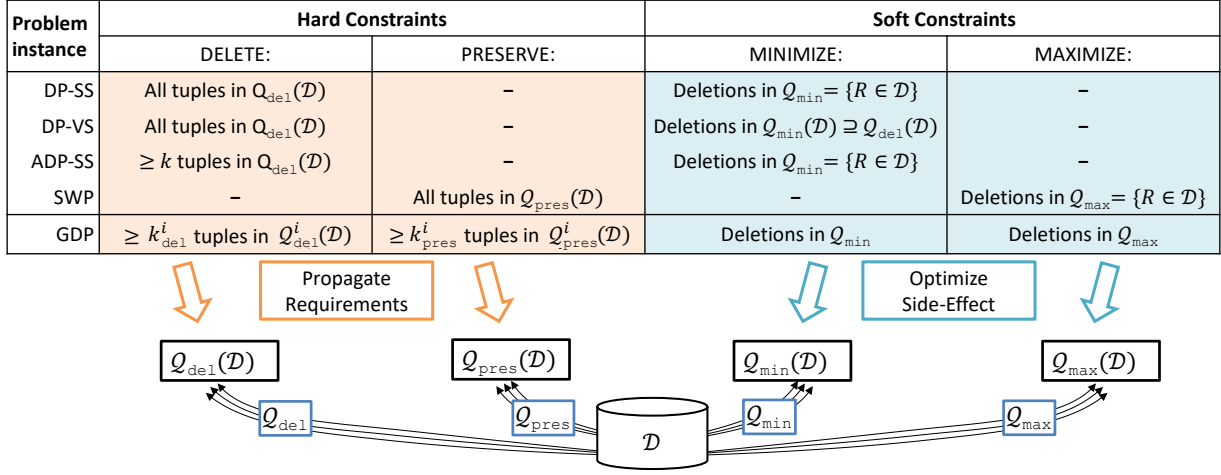


Figure 6.2: Generalized Deletion Propagation (GDP) is defined over 4 different sets of views, two of which model hard constraints, and the other two model soft constraints (optimization objectives). Our approach encapsulates previously studied NPC variants of the deletion propagation problem as special cases: Deletion Propagation with Source Side Effects (DP-SS) [44], Deletion Propagation with View Side Effects (DP-VS) [103], Aggregated Deletion Propagation with Source Side Effects (ADP-SS) [94], Smallest Witness Problem (SWP) [93]. Notice that GDP is a generalization of the prior variants in multiple senses: 1) It allows for side effects on a view different from the original. 2) It allows each type of constraint to be enforced over multiple views. 3) It allows for a combination of constraints and measured side effects.

6.2 Problem Statement

We introduce Generalized Deletion Propagation (GDP) which generalizes all prior variants of deletion propagation, but also allows for new variants to be defined. The new variants are motivated by the following observations:

1. The number of deletions in the source or view are not the only two possible side effects; one could also care about side effects on *another view* that is different from the one in which the deletion occurs (then the source tables can simply be considered just another set of views).
2. It is natural to enforce constraints or optimize side effects over multiple views.
3. Prior variants focus on a specific type of side effect (either source or view) and a specific type of constraint (either deletion or preservation). In practice, one might want to combine these constraints and side effects (e.g., minimizing deletions from one view while maximizing deletions from another). All three of these extensions are motivated with examples in [Subsection 6.2.3](#).

We observe that with 4 different sets of views, we can model the whole range of existing problems and can also combine individual constraints in arbitrary ways. [Definition 6.2.1](#) thus defines *generalized deletion propagation* as a constraint optimization problem over four set of views. These sets of views correspond to four primitive operations that typically occur in deletion propagation variants - a requirement to delete tuples from a view, a requirement to preserve tuples in a view, a requirement to minimize side effects on a view, and a requirement to maximize side effects on a view. [Subsection 6.2.2](#) discusses how the GDP definition encapsulates all past variants of DP as special cases (also depicted in [Figure 6.2](#)), while [Subsection 6.2.3](#) motivates new variants of DP that are captured by the GDP framework.

6.2.1 Defining Generalized Deletion Propagation

Before we define GDP, we introduce some notation. We use bold notation for vectors (as in \mathbf{x}) and superscript for entries (as in x^i). \mathbb{Q} represents an ordered set of queries, and Q^i represents the i^{th} query in \mathbb{Q} . $|Q(\mathcal{D})|$ is defined as the number of output tuples in $Q(\mathcal{D})$ and $|\mathbb{Q}(\mathcal{D})| = \sum_{Q \in \mathbb{Q}} |Q(\mathcal{D})|$ as the number of output tuples across all views in \mathbb{Q} . We define $\Delta Q(\mathcal{D}, \Gamma)$ as the set of output tuples in $Q(\mathcal{D})$ that are deleted as a consequence of deleting Γ from the database \mathcal{D} and hence are not present in $Q(\mathcal{D} \setminus \Gamma)$. Similarly, we define $\Delta \mathbb{Q}(\mathcal{D}, \Gamma)$ as the set of tuples deleted from all views in \mathbb{Q} :

$$|\Delta \mathbb{Q}(\mathcal{D}, \Gamma)| = \sum_{Q^i \in \mathbb{Q}} |Q^i(\mathcal{D})| - |Q^i(\mathcal{D} \setminus \Gamma)|$$

Definition 6.2.1 (Generalized Deletion Propagation (GDP)) *Given four ordered sets of monotone queries $\mathbb{Q}_{\text{del}}, \mathbb{Q}_{\text{pres}}, \mathbb{Q}_{\text{min}}$ and \mathbb{Q}_{max} over a database \mathcal{D} , and vectors of positive integers \mathbf{k}_{del} and \mathbf{k}_{pres} of size equal to the number of views in \mathbb{Q}_{del} and \mathbb{Q}_{pres} respectively, the GDP problem is the task of determining a set of input tuples $\Gamma \subseteq \mathcal{D}$ such that*

$$|\Delta \mathbb{Q}_{\text{min}}(\mathcal{D}, \Gamma)| - |\Delta \mathbb{Q}_{\text{max}}(\mathcal{D}, \Gamma)|$$

is minimized and the following hard constraints are satisfied:

1. *Deleting Γ from the database \mathcal{D} deletes at least k_{del}^i output tuples from the i^{th} view defined by \mathbb{Q}_{del} for all i i.e.,*

$$|Q_{\text{del}}^i(\mathcal{D} \setminus \Gamma)| \leq |Q_{\text{del}}^i(\mathcal{D})| - k_{\text{del}}^i$$

2. *Deleting Γ from the database \mathcal{D} preserves at least k_{pres}^i output tuples from the i^{th} view defined by \mathbb{Q}_{pres} for all i i.e.,*

$$|Q_{\text{pres}}^i(\mathcal{D} \setminus \Gamma)| \geq k_{\text{pres}}^i$$

6.2.2 Capturing Prior Variants of Deletion Propagation with GDP

We next show how each of the previously studied variants of the deletion propagation problem is a special case of GDP.

Deletion Propagation with Source Side Effects (DP-SS) [23, 44] and Resilience (RES) [17, 59, 60, 110, 120]

Deletion Propagation with source side effects (DP-SS) is one of the two originally formulated variants of the deletion propagation problem [23].

Definition 6.2.2 (DP-SS) *Given a view defined by a query Q over a database \mathcal{D} , and an output tuple $t \in Q(\mathcal{D})$, the deletion propagation with source side effects problem is the task of determining a set of input tuples $\Gamma \subseteq \mathcal{D}$ such that $|\Gamma|$ is minimized and t is not contained in $Q(\mathcal{D} \setminus \Gamma)$. In other words,*

$$\min |\Gamma| \text{ s.t. } t \notin Q(\mathcal{D} \setminus \Gamma)$$

[23]: Buneman, Khanna, and Tan (PODS, 2002), 'On Propagation of Deletions and Annotations Through Views'. doi:10.1145/543613.543633

DP-SS is a special case of GDP - we can solve a DP-SS problem by setting Q_{del}^1 to be a query with constants selecting for the values of t , $k_{\text{del}}^1 = 1$ and setting \mathbb{Q}_{min} to be the set of identity queries that select all tuples from any relation in \mathcal{D} . The key observation is that source side effects can also be represented by computing a set of queries \mathbb{Q}_{min} , and then the difference between source and view side effects results from the choice of query that defines the view.

Resilience (RES) is a variant of DP-SS that focuses on Boolean queries and asks for the minimum number of deletions needed to make a query false. It has been called the “simplest” of all deletion propagation problems [59], and a large amount of literature has been dedicated to understanding its complexity [17, 59, 60, 110, 120]. The complexity results for the RES problem also imply complexity results for the DP-SS problem [59]. Existing work has shown a complexity dichotomy for self-join-free conjunctive queries, both under set [59] and bag semantics [110], yet only few tractability results for queries with self-joins and unions are known [17, 60, 110]. The RES problem can be modelled as a special case of GDP similarly as DP-SS, with the added restriction that Q_{del}^1 is a boolean query.

Deletion Propagation: View Side Effect (DP-VS) [23, 101–103]

Deletion Propagation with View Side effects (DP-VS) has the same deletion propagation requirement (or “hard constraint”) as DP-SS, but does so with the goal of minimizing the side effects on the view in which the deletion occurs.

Definition 6.2.3 (DP-VS) *Given a view defined by a query Q over a database \mathcal{D} , and an output tuple $t \in Q(\mathcal{D})$, the deletion propagation with view side effects problem is the task of determining a set of input tuples $\Gamma \subseteq \mathcal{D}$ such that $|\Delta Q(\mathcal{D}, \Gamma)|$ is minimized and t is not contained in $Q(\mathcal{D} \setminus \Gamma)$. In other words,*

$$\min |Q(\mathcal{D})| - |Q(\mathcal{D} \setminus \Gamma)| \text{ s.t. } t \notin Q(\mathcal{D} \setminus \Gamma)$$

DP-VS is a special case of GDP where $\mathbb{Q}_{\text{del}}(\mathcal{D})$ contains a single query whose output is the output tuple t (just like in DP-SS), $k_{\text{del}}^1 = 1$, and \mathbb{Q}_{min} has as single query Q from the original DP-VS problem. A complexity dichotomy has been shown for the DP-VS problem for self-join-free CQs under set semantics [102].

Aggregated Deletion Propagation with Source Side effect (ADP-SS) [94]

The Aggregated Deletion Propagation (ADP-SS) formulation extends the previous DP-SS by requiring the deletion of any k output tuples from a view, rather than a specific output tuple.

Definition 6.2.4 (ADP) *Given a view defined by a query Q over a database \mathcal{D} , and a positive integer k , the Aggregated Deletion Propagation (ADP) problem is the task of determining a set of input tuples $\Gamma \subseteq \mathcal{D}$ such that $|\Gamma|$ is minimized and at least k tuples are removed from $Q(\mathcal{D})$ as a consequence of removing Γ from \mathcal{D} . In other words,*

$$\min |\Gamma| \text{ s.t. } |Q(\mathcal{D} \setminus \Gamma)| \leq |Q(\mathcal{D})| - k$$

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

[17]: Bodirsky, Semanisinová, and Lutz (LICS, 2024), ‘The Complexity of Resilience Problems via Valued Constraint Satisfaction Problems’. doi:10.1145/3661814.3662071

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

[60]: Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[120]: Miao, Li, and Cai (TCS, 2020), ‘The parameterized complexity and kernelization of resilience for database queries’. doi:10.1016/j.tcs.2020.08.018

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

Even though not explicit in the name of the problem, ADP-SS cares about minimizing source side effects (which can be captured by GDP in the same manner as for DP-SS). A complexity dichotomy has been shown for the ADP problem for self-join-free conjunctive queries under set semantics [94].

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

Smallest Witness Problem (SWP) [121]

The Smallest Witness Problem was not proposed as a DP problem, but was noted to bear a strong but unspecified resemblance to the deletion propagation variants [121]. We show that this resemblance is due to the fact that – when modelled as a constraint optimization problem – the correspondence of deletions of input and output tuples are based on exactly the same constraints. Concretely, SWP can be seen as a “preservation propagation” problem, where the goal is to find the largest set of tuples that can be removed from the database without affecting the results of a query. Using negation, we reformulate this as minimization problem (to maintain consistency with other definitions in this section):

[121]: Miao, Roy, and Yang (SIGMOD, 2019), ‘Explaining Wrong Queries Using Small Examples’. doi:10.1145/3299869.3319866

Definition 6.2.5 (SWP) *Given a view defined by a query Q over a database \mathcal{D} , the smallest witness problem is the task of determining a set of input tuples $\Gamma \subseteq \mathcal{D}$ such that $|\Gamma|$ is maximized and $\Delta V(\mathcal{D}, \Gamma)$ is exactly 0. In other words,*

$$\min -|\Gamma| \text{ s.t. } |Q(\mathcal{D} \setminus \Gamma)| = |Q(\mathcal{D})|$$

A complexity dichotomy has been shown for the SWP problem for self-join-free conjunctive queries under set semantics [93]. Interestingly, the tractable cases for SWP are a subset of the tractable cases for DP-VS, reaffirming that these variants have a structural connection and should be studied together.

[93]: Hu and Sintos (ICDT, 2024), ‘Finding Smallest Witnesses for Conjunctive Queries’. doi:10.4230/LIPICs.ICDT.2024.24

6.2.3 Capturing Natural New Variants of Deletion Propagation with GDP

Our GDP formulation allows for the definition of new variants of the deletion propagation problem based on at least three types of extensions to the existing variants. These extensions can also be combined in arbitrary ways, leading to a rich set of new problems that can be defined using the GDP framework.

Extension 1: New types of side effects. The existing variants of Deletion Propagation problem have focused on minimizing source side effects or view side effects. However, one can easily imagine the same underlying source database being used to generate multiple views, and the user wanting to delete tuples from one view while minimizing side effects on another view. In Example 6.0.1, we saw an airline that wanted to cut down expenses by reducing the number of flights, and terminating airport leases. The side effects of this intervention would need to be measured on a completely different view built on the same database, i.e., the view showing the connectivity network for the airline includes places that do not have a direct flight between them.

Extension 2: Constraints over multiple views. Existing variants of Deletion Propagation only focus on a single view from which deletions are propagated. However, one can imagine a scenario where tuples from multiple views are deleted. Note that performing deletions on

multiple views one at a time is not the same as performing deletions on all views simultaneously, and the side effects of performing DP on each view independently may be higher than performing DP on all views simultaneously. As we saw in [Example 6.0.1](#), the airline wanted to cut down on multiple costs such as fuel costs and airport lease costs. Cutting 2% of total costs is not necessarily the same as cutting 1% of fuel costs and 1% of airport lease costs. Depending on the current structure of the airline, a different percent of cost-cutting in each category may be required, and it is always better to jointly optimize over all the expense views.

Extension 3: Combination of Deletion and Preservation Constraints.

Current variants of the DP problem only specify a deletion constraint, and optimize over the tuples preserved (whether in the source or view). However, there may be some important tuples that should not be removed from the database, and the user may want to enforce a preservation constraint on these tuples. The user may also want an aggregated preservation constraint, where a certain number of tuples must be preserved in the view. In [Example 6.0.1](#), we saw that the airline wanted to cut down on costs but ensure that it keeps its most popular routes operational, thus combining a deletion and preservation constraint.

6.3 ILP Framework for GDP

This section specifies an Integer Linear Program (ILP) $\text{ILP}[\text{GDP}]$ which returns an optimal solution for GDP for any instance supported by [Definition 6.2.1](#). We proceed in three steps, first providing a basic ILP formulation and subsequently improving it in two steps. Our approach works even if some views are defined *with self-joins*, or if the underlying database uses *bag semantics*. We focus in this section on proving correctness, while [Section 6.4](#) later investigates how known tractable cases can be solved in PTIME, despite the problem being NPC in general. The input to the $\text{ILP}[\text{GDP}]$ are the four sets of view-defining queries \mathbb{Q}_{del} , \mathbb{Q}_{pres} , \mathbb{Q}_{min} , \mathbb{Q}_{max} over a database \mathcal{D} . Note that any of these sets can be empty as well.* As input to our computation, we also assume as given the *set of witnesses* for each output tuple in any of the computed views, which can be obtained in PTIME by running the *full version* Q^F of each query Q and computing the associated provenance polynomial. The full version Q^F of a query Q is the query that we get by making any existential variables into head variables (or equivalently, by removing all projections). For example, the full version of $Q(x) :- R(x, y), S(y, z)$ is $Q^F(x, y, z) :- R(x, y), S(y, z)$. The use of witnesses as an intermediary between input (database) and output (view) tuples is a key modeling step that allows us to formulate DP problems with linear constraints. We thus associate with each output tuple a set of witnesses and use these sets of witnesses to construct $\text{ILP}[\text{GDP}]$.

In a slight abuse of notation we write $v \in \mathbb{Q}(\mathcal{D})$ for $v \in \bigcup_{Q \in \mathbb{Q}} Q(\mathcal{D})$ and similarly, $w \in \mathbb{Q}^F(\mathcal{D})$ for $w \in \bigcup_{Q \in \mathbb{Q}} Q^F(\mathcal{D})$. We write that $v \in w$ if $v \in Q(\mathcal{D})$ is a projection of $w \in Q^F(\mathcal{D})$ onto the head variables of Q . For example, for the earlier example of $Q(x) :- R(x, y), S(y, z)$ and $Q^F(x, y, z) :- R(x, y), S(y, z)$, assume we have two witnesses $w_1 = Q^F(1, 2, 3)$, $w_2 = Q^F(1, 3, 2)$, $w_3 = Q^F(2, 1, 3)$, and two view tuples $v_1 = Q^1(1)$, $v_2 = Q^1(2)$. Then $v_1 \in w_1$, $v_1 \in w_2$, $v_1 \notin w_3$, $v_2 \notin w_1$, etc. It is

* Notice that the problem is still defined (though trivial) even if all sets are empty: Then any set of interventions satisfy the problem, and the objective value is always 0.

very important to note that we treat output tuples of different views as distinct, even if they correspond to the same set of tuples in the input database. Thus, we can have $v_1 \in Q_{\text{del}}^i(\mathcal{D})$ and $v_2 \in Q_{\text{pres}}^j(\mathcal{D})$ with $Q_{\text{del}}^i = Q_{\text{pres}}^j$, and the valuation of variables for v_1 is the same as for v_2 , but we will still treat them as distinct: $v_1 \neq v_2$ (similarly for views). Notice that this modeling decision appears at first to create inconsistencies, as our algorithm theoretically permits v_1 to be deleted from the view while v_2 is preserved. However, as we discuss later, this *does not create inconsistencies* and is actually *crucial* for the tractability proofs in [Section 6.4](#).

6.3.1 A basic ILP Formulation for GDP

We first define a naive ILP $\text{ILP}_N[\text{GDP}]$ with three components: the ILP variables, an ILP objective function, and ILP constraints.

ILP Variables

We introduce binary variables $X[t]$ for each input tuple t in the relations from \mathcal{D} which takes on value 1 if the corresponding tuple is deleted, and 0 otherwise. Similarly, we introduce binary variables $X[v]$ for each output tuple v in each of the view-defining queries in $\mathcal{Q}_{\text{del}}(\mathcal{D})$, $\mathcal{Q}_{\text{pres}}(\mathcal{D})$, $\mathcal{Q}_{\text{min}}(\mathcal{D})$, $\mathcal{Q}_{\text{max}}(\mathcal{D})$, and $X[w]$ for each witness w in the full version of those queries.

ILP Objective Function (“Soft constraints”)

The only possible side effects of deleting a set of input tuples on a view defined by a monotone query are deletions of tuples in the view. As defined in [Definition 6.2.1](#), we thus count the side effects as the number of output tuples deleted from $\mathcal{Q}_{\text{min}}(\mathcal{D})$ plus the number of tuples preserved in $\mathcal{Q}_{\text{max}}(\mathcal{D})$, respectively. Minimizing the number of tuples preserved in a view is equivalent to maximizing the number of tuples deleted in that view, which is equivalent to minimizing -1 times the number of tuples deleted in that view. Thus, our overall goal is to minimize the following objective function:

$$f(\mathbf{X}) = \sum_{v \in \mathcal{Q}_{\text{min}}(\mathcal{D})} X[v] - \sum_{v \in \mathcal{Q}_{\text{max}}(\mathcal{D})} X[v]$$

ILP Constraints (“Hard constraints”)

The basic ILP formulation has two types of constraints: (1) *User constraints (UCs)* are those that are application-specific and are specified by the user. (2) *Propagation constraints (PCs)* encode the various relationships between tuple variables, witness variables, and view variables needed for consistency. In other words, PCs capture the effect of the hard user constraints on the input database, and then the effect of the input database on various views.

(1) User constraints (UCs). These are the deletion and preservation constraints that are specified by the user on the view definitions \mathcal{Q}_{del} and $\mathcal{Q}_{\text{pres}}$, respectively. The deletion constraints specify that at least

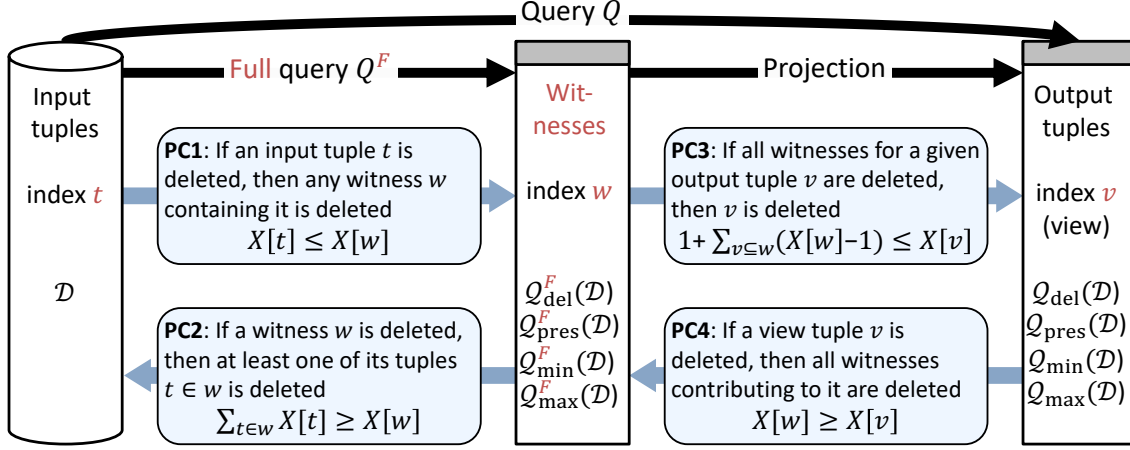


Figure 6.3: Propagation constraints in our ILP formulation $\text{ILP}_N[\text{GDP}]$, explained in the direction of propagating deletions and thus providing lower bounds on the variables. The witness variables are the bridge between the tuple variables and the view variables, and represent the output tuples of the corresponding full query.

k_{del}^i tuples must be deleted from each view $Q_{\text{del}}^i \in \mathbb{Q}_{\text{del}}$, while the preservation constraints specify that at least k_{pres}^i tuples must be preserved in each view $Q_{\text{pres}}^i \in \mathbb{Q}_{\text{pres}}$ (which is equivalent to deleting at most $|Q_{\text{pres}}^i(\mathcal{D})| - k_{\text{pres}}^i$ tuples):

$$\begin{aligned} \sum_{v \in Q_{\text{del}}^i(\mathcal{D})} X[v] &\geq k_{\text{del}}^i & \forall Q_{\text{del}}^i \in \mathbb{Q}_{\text{del}} \\ \sum_{v \in Q_{\text{pres}}^i(\mathcal{D})} X[v] &\leq |Q_{\text{pres}}^i(\mathcal{D})| - k_{\text{pres}}^i & \forall Q_{\text{pres}}^i \in \mathbb{Q}_{\text{pres}} \end{aligned}$$

(2) Propagation constraints (PCs). These constraints encode the relationships between input tuples, witnesses, and tuples in the views to obtain upper and lower bounds on each. Any deletion in a view needs to be reflected also in the input database, and as consequence also in the other views. It is this necessary “*propagation of deletions*” from views (output tuples) to the database (input tuples) that gave this family of problems its name [23].

Figure 6.3 shows a summary of the propagation constraints, split into two parts: the propagation constraints between input tuple variables and witness variables (PC1 and PC2), and between witness variables and view variables (PC3 and PC4). Notice that all PCs are bidirectional in that they compare two types of variables and give an upper bound for one and a lower bound for the other. Thus, each constraint can be explained in two ways (depending on the direction of the propagation), but not all constraints need to be applied to all views (recall our wildcard semantics). We first describe the constraints, and then discuss when they are enforced.

- **PC1:** (\rightarrow) If an input tuple t is deleted, then any witness w containing it is deleted. (\leftarrow) If a witness w is not deleted, then neither of its tuples $t \in w$ is deleted.

$$X[t] \leq X[w], t \in w$$

- **PC2:** (\leftarrow) If a witness w is deleted, then at least one of its tuples $t \in w$ is deleted. (\rightarrow) Alternatively, if all tuples $t \in w$ are not deleted, then the witness w is not deleted.

$$\sum_{t \in w} X[t] \geq X[w] \quad (6.1)$$

- **PC3:** (\rightarrow) If all witnesses for a given output tuple v are deleted, then v is deleted. (\leftarrow) If v is not deleted, then at least one witness w for v is not deleted.

$$1 + \sum_{v \subseteq w} (X[w] - 1) \leq X[v]$$

- **PC4:** (\leftarrow) If a view tuple v is deleted, then all witnesses contributing to it are deleted. (\rightarrow) If a witness w is not deleted, then any view tuple $v \subseteq w$ is not deleted.

$$X[w] \geq X[v], v \subseteq w$$

Naive ILP

We define $\text{ILP}_N[\text{GDP}]$ as the program resulting from our definitions of ILP variables, objective function, and constraints, and will sometimes refer to it as the “naive ILP”.

Proposition 6.3.1 (Correctness of $\text{ILP}_N[\text{GDP}]$) *The interventions given by an optimum solution of $\text{ILP}_N[\text{GDP}]$ for any \mathcal{D} , \mathbb{Q}_{del} , \mathbb{Q}_{pres} , \mathbb{Q}_{min} , \mathbb{Q}_{max} , \mathbf{k}_{del} , \mathbf{k}_{pres} are an optimum solution to GDP over the same input.*

The direct mapping from the variables, objective and constraints of GDP into our ILP formulation from this section forms the proof.

6.3.2 Wildcard Semantics for $X[w]$ and $X[v]$

The binary variables for each input tuple $X[t]$ are always faithful to deletions in the database \mathcal{D} (a tuple is either deleted or present). However, for witness variables $X[w]$ and output tuple variables $X[v]$ we use a semantics that we call “*wildcard semantics*.” The intuition is that user constraints on deletion views provide hard lower bounds on deletions in the database (we need to provide at least that many deletions), while minimization views provide upper bounds (more deletions than necessary get automatically penalized by the optimization objective). This results in a one-sided guarantee. For example, setting $X[v_1] = 1$ for $v_1 \in Q_{\text{del}}^i(\mathcal{D})$ means it is necessarily deleted, and setting $X[v_2] = 0$ for $v_2 \in Q_{\text{pres}}^i(\mathcal{D})$ means it is necessarily preserved. However, in this semantics we cannot infer the actual status from $X[v_1] = 0$ and $X[v_2] = 1$. This semantics allows us to simplify the ILP by having fewer constraints; and, it turns out to be crucial for the tractability proofs in [Section 6.4](#).

Example 6.3.1 (wildcard semantics) Consider a database \mathcal{D} with facts $\{R(1, 1), R(2, 2), S(1)\}$, and query $Q(x) := R(x, y), S(y)$. Consider a DP-SS problem where tuple $Q(1)$ should be deleted from the output. We introduce the tuple variables $X[R(1, 1)]$, $X[R(1, 2)]$, $X[S(1)]$, witness variables $X[Q^F(1, 1)]$, $X[Q^F(1, 2)]$, and view variables $X[Q(1)]$, $X[Q(2)]$. We show in [Figure 6.4](#) some possible variable assignments

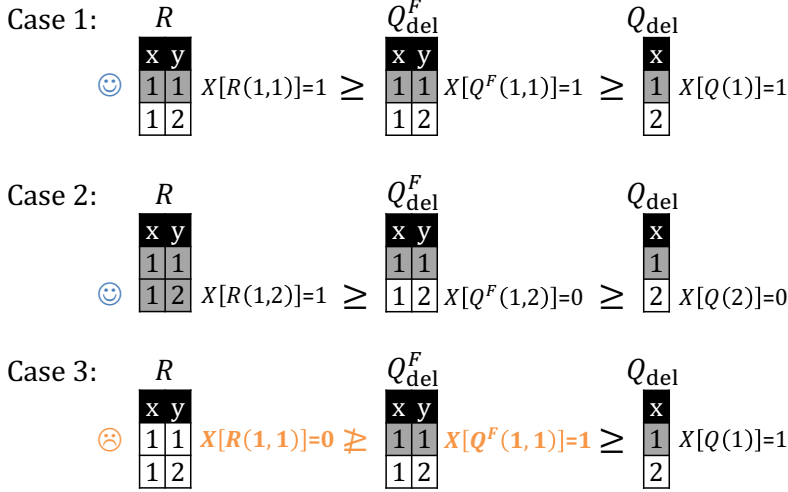


Figure 6.4: Example 6.3.1: The true state of deletions in the database \mathcal{D} is always faithfully represented by database variables (e.g., $R(1,2)$ is deleted and thus $X[R(1,2)] = 1$ and is grayed out). However, deletions in the views defined by a query in \mathcal{Q}_{del} need to provide only lower bounds for modeling DP-SS (e.g., setting $X[Q^F(1,2)] = 0$ in case 2 is ok even though the view tuple would be deleted).

and discuss if they satisfy the wildcard semantics.

Case 1: A feasible solution is setting $X[R(1,1)]$, $X[Q^F(1,1)]$, $X[Q(1)]$ to 1, and all other variables to 0, i.e. tuple $R(1,1)$ is deleted from the database, witness $Q^F(1,1)$ is deleted from the full query, and tuple $Q(1)$ is deleted from the view. In this case, all variables are faithful to a set of actual deletions in the database and views.

Case 2: Another solution modifies $X[R(1,2)]$ to 1, while the other variables remain the same (including $X[Q^F(1,2)] = 0$). Since the witness $Q^F(1,2)$ would be deleted once $R(1,2)$ is deleted, this solution is not faithful to any set of interventions (if 0 assignments are interpreted as required preservations). Notice, however, that this variable assignment causes no harm in the correct fulfillment of the user constraints. Marking a witness as not deleted when it is, is not a problem, since this can never mark the user constraint as satisfied if it isn't in reality.

Case 3: In contrast, a solution with $X[R(1,1)] = 0$, $X[Q^F(1,1)] = 1$, $X[Q(1)] = 1$ is incorrect. It falsely claims to satisfy the user constraint by deleting the tuple $Q(1)$ from the view, but it does not actually delete any input tuples that would lead to this deletion.

Example 6.3.1 showed that for a variable $X[w]$ for a witness w in $\mathcal{Q}_{del}^F(\mathcal{D})$, it is important that we do not claim it is deleted if it is not (as this would not truly satisfy the user requirement). Thus, $X[w] = 1$ must imply that $w \notin \mathcal{Q}_{del}^F(\mathcal{D})$. However, deleting w while having $X[w] = 0$ is not a problem, because this can never represent an unsatisfactory interventions (as is the case when a user required deletions that are not truly carried out). Thus, we use a semantics for witnesses in \mathcal{Q}_{del}^F where $X[w] = 1$ implies witness w is deleted, while $X[w] = 0$ acts as a “wildcard”, allowing the witness to be deleted or not. In other words, truth assignments to tuples in $\mathcal{Q}_{del}(\mathcal{D})$ provide a lower bound on the deletions of tuples in the database (Figure 6.4). The exact same reasoning applies to the $X[v]$ variables for view tuples in $\mathcal{Q}_{del}(\mathcal{D})$ as well.

Similarly, witness and view variables for \mathcal{Q}_{max}^F and \mathcal{Q}_{max} provide upper bounds on tuple deletions in the database. For these views, a solution stating that a witness / view tuples is not deleted when it is, is not a problem since the user constraints specify a lower bound. Thus, here too we allow the same semantics that $X[w] = 1$ and $X[v] = 1$ if w / v is deleted, and $X[w] = 0$ and $X[v] = 0$ are wildcard values where the

	$X[w] = 0$	$X[w] = 1$	$X[v] = 0$	$X[v] = 1$
$\mathbb{Q}_{\max}(\mathcal{D})$	*	$w \notin \mathbb{Q}_{\max}^F(\mathcal{D})$	*	$v \notin \mathbb{Q}_{\max}(\mathcal{D})$
$\mathbb{Q}_{\text{del}}(\mathcal{D})$	*	$w \notin \mathbb{Q}_{\text{del}}^F(\mathcal{D})$	*	$v \notin \mathbb{Q}_{\text{del}}(\mathcal{D})$
$\mathbb{Q}_{\text{pres}}(\mathcal{D})$	$w \in \mathbb{Q}_{\text{pres}}^F(\mathcal{D})$	*	$v \in \mathbb{Q}_{\text{pres}}(\mathcal{D})$	*
$\mathbb{Q}_{\min}(\mathcal{D})$	$w \in \mathbb{Q}_{\min}^F(\mathcal{D})$	*	$v \in \mathbb{Q}_{\min}(\mathcal{D})$	*

Figure 6.5: Table showing the one-sided guarantees that any variable assignment has on solution to a GDP problem. For cases with wildcards (“*”), the true value of the variable can be either 0 or 1.

witness/view tuple may or may not be deleted.

Symmetrically, for $\mathbb{Q}_{\text{pres}}(\mathcal{D})$ and $\mathbb{Q}_{\min}(\mathcal{D})$, we need to ensure that if a tuples and witnesses is said to be *preserved* (i.e. their variables are set to 0), then it is actually preserved. Thus, $X[w] = 0$ and $X[v] = 0$ for w, v in \mathbb{Q}_{pres} and \mathbb{Q}_{\min} imply that the corresponding witness or view tuple is not deleted, while $X[w] = 1$ and $X[v] = 1$ represent a wildcard value, allowing the corresponding witness or view tuple to be deleted or not. Figure 6.5 captures the semantics of the $X[w]$ and $X[v]$ variables for each type of query.

Selective application of PCs. We use the wildcard semantics for witnesses and view variables described in Subsection 6.3.1 to obtain a more efficient ILP. Concretely, we don’t apply the PCs in directions that are not required to enforce the wildcard semantics.

PC1 and PC3 encode lower bounds on the witness and view variables, respectively. They ensure that $X[w] = 0$ and $X[v] = 0$ only when w and v are not deleted. Thus, they need to be applied to \mathbb{Q}_{pres} and \mathbb{Q}_{\min} , but do not to \mathbb{Q}_{del} and \mathbb{Q}_{\max} . Similarly, PC2 and PC4 are upper bounds on the witness and view variables, respectively. They ensure that $X[w] = 1$ and $X[v] = 1$ only when w and v are deleted. Thus, they need to be applied to \mathbb{Q}_{del} and \mathbb{Q}_{\max} , but not to \mathbb{Q}_{pres} and \mathbb{Q}_{\min} . Figure 6.6 summarizes the selective application of PCs to the different views.

Wildcard ILP. We refer to the “wildcard ILP” or $\text{ILP}_w[\text{GDP}]$ solution to GDP as the basic ILP that applies the PCs only selectively, namely PC1 and PC3 to \mathbb{Q}_{pres} and \mathbb{Q}_{\min} (but not PC2 nor PC4), and PC2 and PC4 to \mathbb{Q}_{del} and \mathbb{Q}_{\max} (but not PC1 nor PC3).

Proposition 6.3.2 (Correctness of $\text{ILP}_w[\text{GDP}]$) *The interventions suggested by an optimum solution of $\text{ILP}_w[\text{GDP}]$ are an optimum solution to GDP over the same input.*

Proof Intuition. The proof is based on the fact that any optimal solution under traditional semantics is an optimal solution in the wildcard semantics, and to enforce the wildcard semantics it suffices to apply PCs selectively (which is possible as argued before).

Proof Proposition 6.3.2. We argue that every solution allowed by the wildcard semantics corresponds to a feasible solution of the ILP under naive semantics and vice versa. Additionally, we need to show that the optimal value of the ILP under wildcard semantics is the same as the optimal value of the ILP under naive semantics. That every solution of the ILP under naive semantics corresponds to a solution of the ILP under wildcard semantics is easy to see since the wildcard semantics is a generalization of the naive semantics. To argue the other direction, we need to show that every solution of the ILP under wildcard semantics corresponds to a solution of the ILP under naive semantics with the same optimal value. Such a corresponding solution can be obtained by keeping the values of all $X[t]$ variables the same, and replacing all $X[v]$ and $X[w]$ variables

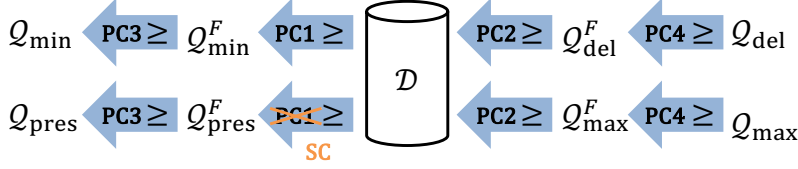


Figure 6.6: Arrows in this figure illustrate the constraints in the direction of lower bounds (but recall that constraints are bidirectional). Notice that our wildcard semantics applies constraints only selectively to different views (Subsection 6.3.2). Also shown is how our Smoothing Constraints (SC) replace PC1 for $\mathbb{Q}_{\text{pres}}^F(\mathcal{D})$ (Subsection 6.3.3). It is that replacement that gives us a powerful PTIME guarantee for PTIME problems (see later Figure 6.9 from the experiments).

to what is implied by the variables under naive semantics. We can see such a solution will still be feasible under the naive semantics since the wildcard semantics enforce one-sided guarantees on the values of the variables in the direction of the user constraints for witnesses and view tuples in \mathcal{V}_{del} and $\mathcal{V}_{\text{pres}}$. We now need to argue about the optimal solutions under naive and wildcard semantics. First, we observe that using wildcard semantics can never lead to a solution where the corresponding naive solution has a less optimal objective value. This is due to the fact that we have one-sided guarantees on the values of witnesses and view tuples in \mathcal{V}_{del} and $\mathcal{V}_{\text{pres}}$, that ensure that the objective value under wildcard semantics is never strictly better than the objective value under naive semantics. Combined with the fact that every solution under naive semantics is a solution under wildcard semantics, we can conclude that the optimal value of the ILP under wildcard semantics is the same as the optimal value of the ILP under naive semantics. \square

6.3.3 ILP with Smoothing Constraints

The wildcard semantics alone does not give noticeable performance improvements or PTIME guarantees for our ILP. However, it allows us to enable a surprising optimization: we will tighten one type of constraint in a way that the resulting solution space (a polyhedron) preserves an optimal solution, yet also affords desirable properties on the performance of the resulting ILP and also the optimal solution for its LP relaxation. It is those seemingly superfluous constraints that play a key role in the results of Section 6.4 where we show that an ILP with smoothing constraints $\text{ILP}_S[\text{GDP}]$ can be solved in PTIME for all known tractable cases. Hence, we also refer to $\text{ILP}_S[\text{GDP}]$ as simply $\text{ILP}[\text{GDP}]$.

The user constraints and propagation constraints suffice to correctly model GDP as an ILP. The purpose of the *Smoothing Constraints* (SC) is to make the objective of the LP relaxation closer to the objective of the ILP (in certain cases we see that the smoothing constraint makes the optimal objective value of LP relaxation equal to that of the ILP). In the language of linear optimization, adding these extra bounds is equivalent to adding cutting planes [98] to the polytope defined by the LP relaxation.

We identify a smoothing constraint that can be added to describe the relation between tuple variables, and the witness variables of \mathbb{Q}_{pres} . Recall that for \mathbb{Q}_{pres} , we would like to preserve a certain number of view variables. A view variable v is preserved if at least one of its witnesses w is preserved. Recall that due to our wildcard semantics of $X[w]$ in \mathbb{Q}_{pres} ,

setting $X[w] = 1$ means that we “do not care” whether the witness is deleted or not. In other words, we can say that for any view variable v , there is only one w with $X[w] = 0$ and the other witnesses can be set to 1. Now assume that a tuple t participates in multiple witnesses w_1, w_2, \dots, w_k corresponding to the same view tuple v (It may also participate in more witnesses, but we do not care about those here). We know through PC1 that $X[t] \leq X[w]$ for a given v and $t \in w, w \supseteq v$.

It is correct to now also enforce that $\sum_{i \in [1, k]} X[w_i] \geq k - 1$ i.e., only one $X[w]$ in this set is preserved (the rest may also be preserved, but due to the wildcard semantics they will still have $X[w] = 1$). Now we can also enforce that $X[t] \leq (\sum_{i \in [1, k]} X[w_i]) - (k - 1)$, since all but 1 values of $X[w]$ are set to 1, and only the final value decides the upper bound on $X[t]$. Thus, we get a smoothing constraint, applied to every $v \in \mathbb{Q}_{\text{pres}}(\mathcal{D})$:

$$X[t] \leq 1 + \sum_{\substack{w: t \in w \\ w \supseteq v}} (X[w] - 1)$$

Correctness of $\text{ILP}_S[\text{GDP}]$ with wildcard semantics and smoothing constraints. We refer to $\text{ILP}_S[\text{GDP}]$ as the ILP that has only the PCs that are required for wildcard semantics and has replaced the PC1 constraints on \mathbb{Q}_{pres} in the basic ILP with SC instead.

Theorem 6.3.3 (Correctness of $\text{ILP}_S[\text{GDP}]$ (also known as $\text{ILP}[\text{GDP}]$)) *The interventions suggested by an optimum solution of $\text{ILP}_S[\text{GDP}]$ with wildcard semantics and smoothed constraints form an optimum solution to GDP.*

Proof Intuition. Adding the smoothing constraint to the wildcard ILP always preserves at least one optimal solution - this follows also from the argument above the smoothing constraint can be derived by logically following the wildcard semantics.

Proof Theorem 6.3.3. Under wildcard semantics, each solution set of interventions in the database corresponds to multiple ILP solutions, since the value of $X[w]$ and $X[v]$ variables are not necessarily faithful to the variables. For \mathbb{Q}_{pres} , we can assume that a solution with wildcard semantics exists such that for every view tuple $v \in \mathbb{Q}_{\text{pres}}(\mathcal{D})$, there is only one witness such that $v \subseteq w$ and $X[w] = 0$ i.e. we can assume that for each $v \in \mathbb{Q}_{\text{pres}}(\mathcal{D})$, there is a unique w that is marked as preserved (note that more may be preserved, but unmarked due to the wildcard semantics). The optimal value of the ILP under this assumption is the same as the optimal value of the ILP under wildcard semantics (and hence naive semantics). Due to this assumption, we can say that a given tuple and view tuple, only one witness is preserved i.e. $\sum_{i \in [1, k]} X[w_i] \geq k - 1$ where $w_1 \dots w_k$ are the witnesses that correspond to v and contain t . Now we can also enforce that $X[t] \leq (\sum_{i \in [1, k]} X[w_i]) - (k - 1)$, since all but 1 values of $X[w]$ are set to 1, and only the final value decides the upper bound on $X[t]$. This is equivalent to adding the smoothing constraint, and hence addition of the smoothing constraint does not change the optimal solution of the ILP. \square

An interesting asymmetry. We notice an interesting asymmetry at play. We could apply a symmetric smoothing constraint in the case for PC4 on \mathbb{Q}_{del} . Interestingly, such an additional smoothing constraint would be identical to our original PC1, as every view tuple corresponds to exactly

one witness. Thus, we do not need to add any additional smoothing constraints for \mathbb{Q}_{del} .

Reducing the size of the ILP. The smoothing constraints may subsume some propagation constraints. These subsumed propagation constraints can be removed from the ILP without affecting any solution of the ILP or LP relaxation.

The Power of Smoothing Constraints. [Example 6.3.2](#) is an intuitive example of a SWP problem instance modelled as a GDP problem, where the smoothing constraints ensure that the optimal value of the ILP is equivalent to the optimal value of its LP relaxation in the GDP framework. Later in [Theorem 6.4.3](#), we show that this is the case for all prior known PTIME cases of SWP.

Example 6.3.2 (Power of smoothing) Consider again the \mathcal{D} from [Example 6.3.1](#): with $R(1, 1)$, $R(1, 2)$, and $S(1)$. We want to solve the smallest witness problem $\text{SWP}(Q_{\text{pres}}, \mathcal{D})$ for $Q_{\text{pres}}(x) :- R(x, y), S(x)$. To model it as GDP, we set \mathbb{Q}_{pres} to be $\langle Q_{\text{pres}} \rangle$ and $k_{\text{pres}} = \langle k_{\text{pres}} \rangle$ with $k_{\text{pres}} = 1$, which is the number of output tuples in $Q_{\text{pres}}(\mathcal{D})$. We also set $\mathbb{Q}_{\text{max}} = \langle Q_{\text{max}}^1(x, y) :- R(x, y), Q_{\text{max}}^2(x) :- S(x) \rangle$ and $\mathbb{Q}_{\text{del}} = \mathbb{Q}_{\text{min}} = \emptyset$. Our GDP formulation is as follows:

$$f(\mathbf{X}) = -(X[Q_{\text{max}}^1(1, 1)] + X[Q_{\text{max}}^1(1, 2)] + X[Q_{\text{max}}^2(1)])$$

s.t. following constraints (and integrality constraints):

$$\begin{aligned} X[Q_{\text{pres}}(1)] &\leq 0 && \text{(UC)} \\ X[Q_{\text{pres}}^F(1, 1)] + X[Q_{\text{pres}}^F(1, 2)] - 1 &\leq X[Q_{\text{pres}}(1)] && \text{(PC3)} \\ X[R(1, 1)] &\leq X[Q_{\text{pres}}^F(1, 1)] && \text{(PC1)} \\ X[S(1)] &\leq X[Q_{\text{pres}}^F(1, 1)] && \text{(PC1)} \\ X[R(1, 2)] &\leq X[Q_{\text{pres}}^F(1, 2)] && \text{(PC1)} \\ X[S(1)] &\leq X[Q_{\text{pres}}^F(1, 2)] && \text{(PC1)} \\ X[R(1, 1)] &\leq X[Q_{\text{max}}^{1F}(1, 1)] && \text{(PC2)} \\ X[R(1, 2)] &\leq X[Q_{\text{max}}^{1F}(1, 2)] && \text{(PC2)} \\ X[S(1)] &\leq X[Q_{\text{max}}^{2F}(1)] && \text{(PC2)} \\ X[Q_{\text{max}}^{1F}(1, 1)] &\leq X[Q_{\text{max}}^1(1, 1)] && \text{(PC4)} \\ X[Q_{\text{max}}^{1F}(1, 2)] &\leq X[Q_{\text{max}}^1(1, 2)] && \text{(PC4)} \\ X[Q_{\text{max}}^{2F}(1)] &\leq X[Q_{\text{max}}^2(1)] && \text{(PC4)} \end{aligned}$$

Observe that the optimal solution for the ILP is -1 which occurs when either one of the tuples in R is deleted, i.e. either of $X[R(1, 1)]$ or $X[R(1, 2)]$ is set to 1. However, the LP relaxation has a smaller non-integral optimum of -1.5 for $X[R(1, 1)] = X[R(1, 2)] = X[S(1)] = 0.5$. This is due to the fact that both $X[Q_{\text{pres}}^F(1, 1)]$ and $X[Q_{\text{pres}}^F(1, 2)]$ can take values 0.5, which is why $X[Q_{\text{pres}}^1(1)]$ can be set to 0 while fulfilling all constraints.

Our smoothing constraint for this example is the following

$$X[S(1)] \leq X[Q_{\text{pres}}^F(1, 1)] + X[Q_{\text{pres}}^F(1, 2)] - 1 \quad \text{(SC)}$$

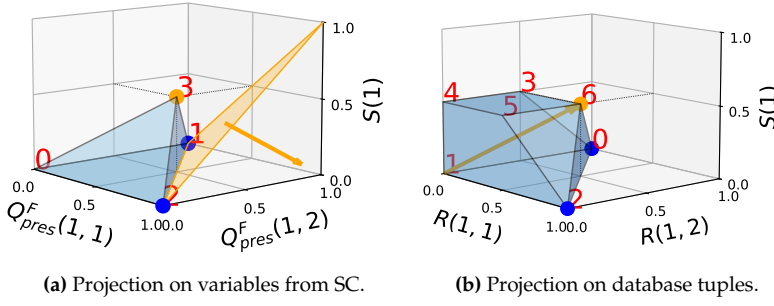


Figure 6.7: Example 6.3.2: Our Smoothing Constraint (SC) acts as a cutting plane, removing a non-integral optimal point from the LP relaxation of our ILP formulation.

Notice that it can replace the PC1 constraints $X[S(1)] \leq X[Q_{\text{pres}}^F(1, 1)]$ and $X[S(1)] \leq X[Q_{\text{pres}}^F(1, 2)]$, since it is a strictly tighter constraint.

The SC ensures that if $X[S(1)]$ is set to 0.5, then $X[Q_{\text{pres}}^F(1, 1)] + X[Q_{\text{pres}}^F(1, 2)] \geq 1.5$, thus violating PC4, and thereby effectively removing the non-integer solution. To gain more intuition, Figure 6.7 shows the polytope of the LP relaxation of our wildcard formulation projected on either the variables involved in SC (Figure 6.7a), or the three input tuples (Figure 6.7b). The optimal LP solution corresponds to the orange point (point 3 in Figure 6.7a, point 6 in Figure 6.7b), and the two optimal ILP solutions correspond to the two blue points (points 1 and 2 in Figure 6.7a, points 0 and 2 in Figure 6.7b). Notice how our SC (shown as yellow cutting plane in Figure 6.7a) cuts away the non-integer solution, leaving only points 1 and 2, and their convex extension. Similarly, this constraint cuts away all points with $X[S(1)] > 1$ (not shown in Figure 6.7b), leaving points 0 and 2 and their convex combination as the optimum solutions to the new LP.

From the workings of modern solvers we know that any ILP problem can be solved efficiently if its natural LP relaxation is tight with the ILP polytope in the direction of the objective (i.e. the ILP and its LP relaxation have the same optimal f^* and share the same “face” perpendicular to the objective vector). Now, it suffices to show that the LP relaxation of the smoothened ILP has the same optimum objective values f^* and preserves at least one optimal integral solution. We see experimentally in Figure 6.9 in Section 6.6 a speedup of 2 orders of magnitude in the ILP solving time simply by adding the smoothing constraints. This is completely justified by our claim that ILP solvers are able to solve ILPs efficiently when the LP relaxation is tight.

6.4 Recovering Existing Tractability Results

In this section we focus on self-join-free queries under set semantics, which is the only case in which complexity dichotomies are known for the DP variants of DP-SS, DP-VS, ADP-SS, SWP. We have shown previously that the GDP framework naturally captures all these problems as special cases. In this section, we show that the LP relaxation of the GDP problem also naturally recovers the optimal, integral solutions for these problems for self-join queries that are known to be tractable under set semantics.

DP-SS. A DP-SS problem on a query Q can be converted to a resilience problem on the existential version of the query Q^E which is obtained by removing all head variables from Q (both in the head and the body). Since a dichotomy result for self-join-free conjunctive queries is known for

resilience both under set and bag semantics, it follows that a complexity dichotomy is also known for DP-SS.

Let $\text{ILP}[\text{GDP}_{\text{DP-SS}}(Q, \mathcal{D}, t)]$ be the ILP obtained when we pose the DP-SS problem over a query Q , database \mathcal{D} and target tuple in view t , as a GDP problem via the method described in Subsection 6.2.2. We now claim that the LP relaxation $\text{LP}[\text{GDP}_{\text{DP-SS}}]$ of such an ILP $\text{ILP}[\text{GDP}_{\text{DP-SS}}]$, is always equivalent to the solution of the optimization problem DP-SS for all known queries Q for which DP-SS can be solved in PTIME, and thus $\text{ILP}[\text{GDP}]$ can be used to solve DP-SS in PTIME for such queries.

Theorem 6.4.1 (Integrality of $\text{LP}[\text{GDP}]$ for tractable instances of DP-SS)
 $\text{LP}[\text{GDP}_{\text{DP-SS}}(Q, \mathcal{D})] = \text{DP-SS}(Q, \mathcal{D})$ for all database instances \mathcal{D} under set semantics if the existential query Q^E does not contain a triad. $\text{LP}[\text{GDP}_{\text{DP-SS}}(Q, \mathcal{D})] = \text{DP-SS}(Q, \mathcal{D})$ for all database instances \mathcal{D} under bag semantics if Q^E is a linear query.

Proof Intuition. We show that $\text{ILP}[\text{GDP}_{\text{DP-SS}}]$ is identical to a specialized ILP that has been proposed [110] for resilience (discussed in Chapter 4). Since that result also shows that for all tractable queries, the LP relaxation of the ILP is integral, the results naturally carry over.

Proof Theorem 6.4.1. Prior work has shown the construction of an ILP $\text{ILP}[\text{RES}^*]$ that has the property that $\text{LP}[\text{RES}^*(Q, D)] = \text{RES}(Q, D)$ for all Q for which $\text{RES}(Q, D)$ is known to be in PTIME - both under set and bag semantics. We simply reuse this result, and show that $\text{LP}[\text{GDP}_{\text{DP-SS}}]$ is identical to $\text{LP}[\text{RES}^*]$ for the same query Q and database \mathcal{D} . We know that in the formulation of $\text{LP}[\text{GDP}_{\text{DP-SS}}]$, both \mathcal{Q}_{max} and $\mathcal{Q}_{\text{pres}}$ are empty. We also know that all the output tuples must be deleted, thus we can replace all variables $X[v]$ for $v \in \mathcal{Q}_{\text{del}}(\mathcal{D})$ with the constant 1 without changing the optimal solution of the ILP. Since each output tuple in $\mathcal{Q}_{\text{min}}(\mathcal{D})$ has a 1-to-1 correspondence with the input tuples in \mathcal{D} , the minimization objective directly corresponds to the number of input tuples that are deleted. Thus, in this case $\text{ILP}[\text{GDP}]$ simplifies to look like:

$$\begin{aligned} & \min \sum_{t \in \mathcal{D}} X[t] \\ & \text{subject to} \\ & \sum_{t \in w} X[t] \geq 1 \quad \forall w \in Q^F(\mathcal{D}) \end{aligned}$$

We observe that this ILP is identical to the ILP $\text{ILP}[\text{RES}^*]$ that has been proposed for the resilience problem [110], and thus the LP relaxation of $\text{ILP}[\text{GDP}_{\text{DP-SS}}]$ is integral for all queries Q for which DP-SS is known to be in PTIME [110, Theorem 8.6 and 8.7.]. \square

DP-VS. It is known that DP-VS is PTIME for self-join-free conjunctive queries if and only if they have the head domination property [102]. We prove that for such queries that have the head domination property, if we pose $\text{DP-VS}(Q, \mathcal{D}, t)$ in the GDP framework, then the LP relaxation $\text{LP}[\text{GDP}_{\text{DP-VS}}(Q, \mathcal{D}, t)]$ is equivalent to the solution of $\text{DP-VS}(Q, \mathcal{D}, t)$.

Theorem 6.4.2 (Integrality of $\text{LP}[\text{GDP}]$ for tractable instances of DP-VS)
 $\text{LP}[\text{GDP}_{\text{DP-VS}}(Q, \mathcal{D}, t)] = \text{DP-VS}(Q, \mathcal{D}, t)$ for all database instances \mathcal{D} under

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[110]: Makhija and Gatterbauer (SIGMOD, 2023), ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. doi:10.1145/3626715

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

set semantics and any tuple v in $Q(D)$ if Q has the head domination property.

Proof Intuition. If a query has the head domination property, it is known that the optimal solution for DP-VS is side effect free [102]. Thus, the optimal value of the ILP objective is 0, and LP relaxation cannot take on a negative value and hence must be equal and integral.

Proof Theorem 6.4.2. DP-VS can be solved in PTIME for a query Q over an arbitrary database \mathcal{D} if and only if it has the head domination property [102]. It is also known that for queries that have the head domination property, the optimal solution for DP-VS is side effect free [102, Proposition 3.2.]. Thus, in our formulation ILP[GDP] always has an optimal solution of 1 - since the only output tuple that is deleted is the one specified by the user. Since we know that $LP[GDP_{DP-VS}]$ is always a lower bound for the true optimal solution, it suffices for us to show that the LP relaxation of $LP[GDP_{DP-VS}]$ is not < 1 . We know that the objective function is a sum of $X[v]$ variable where each v is an output tuple in $\mathcal{Q}_{\max}(\mathcal{D})$ and $X[v]$ takes on value 0 or 1 and thus can never be negative. We also know that for a v that the user would like to delete via DP-VS, $X[v]$ must be set to 1. Thus, the solution to $LP[GDP_{DP-VS}]$ is never < 1 and never more than the optimal solution (which is always 1), and thus the LP relaxation is integral and equal to the optimal solution. \square

SWP. It is known that SWP is PTIME for self-join-free queries if and only if they have the *head clustering property* [93], which is a restriction of the head domination property. We are again able to show that for such queries that have the head clustering property, if we pose $SWP(Q, \mathcal{D})$ in the GDP framework, then the LP relaxation $LP[GDP_{SWP}(Q, \mathcal{D})]$ is equivalent to the solution of $SWP(Q, \mathcal{D})$.

Theorem 6.4.3 (Integrality of $LP[GDP]$ for tractable instances of SWP) $LP[GDP_{SWP}(Q, \mathcal{D})] = SWP(Q, \mathcal{D})$ for all database instances \mathcal{D} under set and bag semantics if Q is a self-join-free conjunctive query with the head clustering property.

Proof Intuition. We first simplify the ILP and phrase it in terms of variables $Y[t] = 1 - X[t]$. We next show that due to the head clustering property, the ILP can be decomposed into multiple independent ILPs, corresponding to different existentially connected components of the query. For each such component, the correct solution can be obtained by preserving an arbitrary witness for each projection, and hence the LP relaxation must be tight.

Proof Theorem 6.4.3. In order to construct SWP as a $ILP[GDP]$, we set $\mathcal{V}_{del} = \mathcal{V}_{min} = \emptyset$ since there are no views from which output must be deleted, or deletions must be minimized. We set $\mathcal{V}_{pres} = \{Q(\mathcal{D})\}$, and $\mathcal{V}_{max} = [R \in D]$. k_{pres} is set to $|Q(\mathcal{D})|$ as we want to preserve all tuples in $Q(\mathcal{D})$, and k_{del} is simply 0.

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

[102]: Kimelfeld, Vondrák, and Williams (TODS, 2012), ‘Maximizing Conjunctive Views in Deletion Propagation’. doi:10.1145/2389241.2389243

[93]: Hu and Sintos (ICDT, 2024), ‘Finding Smallest Witnesses for Conjunctive Queries’. doi:10.4230/LIPICs.ICDT.2024.24

We can then construct the ILP $\text{ILP}[\text{GDP}_{\text{SWP}}]$ as follows:

$$\begin{aligned}
& \min - \sum_{v \in \mathcal{V}_{\max}} X[v] \\
& \text{subject to} \\
& \quad \sum_{v \in Q(\mathcal{D})} X[v] \leq |Q(\mathcal{D})| - k_{\text{pres}} \\
& \quad X[t] \leq X[w], t \in w \\
& \quad X[w] \leq \sum_{t \in w} X[t] \\
& \quad X[v] \leq X[w], v \subseteq w \\
& \quad 1 + \sum_{v \subseteq w} (X[w] - 1) \leq X[v] \\
& \quad X[t], X[w], X[v] \in \{0, 1\} \quad \forall t, w, v
\end{aligned}$$

We know that each output tuple in \mathcal{V}_{\max} has a 1-to-1 correspondence with the input tuples in \mathcal{D} (since \mathcal{V}_{\max} is simply a union of all input relations). Thus, we can replace all view variables $X[v]$ where $v \in \mathcal{V}_{\max}$ with simply the corresponding input tuple variables $X[t]$. Now the function of the propagation constraints is simply to propagate the deletions of output tuples in $\mathcal{V}_{\text{pres}}$ to the input tuples in \mathcal{D} . Since the user constraint provides an upper bound for $X[v]$ for $v \in \mathcal{V}_{\text{pres}}$, the propagation constraints must only provide a lower bound for $X[v]$ via $X[w]$ and then a lower bound for $X[w]$ via $X[t]$.

Thus, we can greatly simplify the ILP to:

$$\begin{aligned}
& \min - \sum_{t \in \mathcal{D}} X[t] \\
& \text{subject to} \\
& \quad \sum_{v \in Q(\mathcal{D})} X[v] \leq |Q(\mathcal{D})| - |Q(\mathcal{D})| \\
& \quad 1 + \sum_{v \subseteq w} (X[w] - 1) \leq X[v] \\
& \quad X[t] \leq X[w], t \in w \\
& \quad X[t], X[w], X[v] \in \{0, 1\} \quad \forall t, w, v
\end{aligned}$$

Since SWP primarily deals with preservations rather than deletions, we can introduce a variable $Y[v]$ that captures if a tuple v is preserved, i.e., it is set to 1 if the tuple is preserved, and 0 otherwise. We can see that $Y[v] = 1 - X[v]$ and build a corresponding ILP with Y variables instead of X variables. This resulting ILP is identical to the one with X variables in that it will have the same optimal solution, and any solution of an ILP or LP relaxation from one can be mapped to another by simply applying the equality $X[v] = 1 - Y[v]$.

$$\begin{aligned}
& \min - \sum_{t \in \mathcal{D}} 1 - Y[t] \\
& \text{subject to} \\
& \quad \sum_{v \in Q(\mathcal{D})} 1 - Y[v] \leq |Q(\mathcal{D})| - |Q(\mathcal{D})| \\
& \quad 1 - Y[t] \leq 1 - Y[w], t \in w \\
& \quad 1 + \sum_{v \subseteq w} (1 - Y[w] - 1) \leq 1 - Y[v] \\
& \quad Y[t], Y[w], Y[v] \in \{0, 1\} \quad \forall t, w, v
\end{aligned}$$

Simplifying this ILP, we get:

$$\begin{aligned}
& \min - |D| + \sum_{t \in \mathcal{D}} Y[t] \\
& \text{subject to} \\
& \quad |Q(D)| \leq \sum_{v \in Q(\mathcal{D})} Y[v] \\
& \quad \sum_{v \subseteq w} (-Y[w]) \leq -Y[v] \\
& \quad Y[w] \leq Y[t], t \in w \\
& \quad Y[t], Y[w], Y[v] \in \{0, 1\} \quad \forall t, w, v
\end{aligned}$$

We can see through the constraint $|Q(D)| \leq \sum_{v \in Q(\mathcal{D})} Y[v]$ that every $Y[v]$ value must necessarily be set to 1 to satisfy the constraint, whether in the ILP or in the LP relaxation. We can thus simplify further to:

$$\begin{aligned}
& \min \sum_{t \in \mathcal{D}} Y[t] \\
& \text{subject to} \\
& \quad 1 \leq \sum_{v \subseteq w} (Y[w]) \\
& \quad Y[w] \leq Y[t], t \in w \\
& \quad Y[t], Y[w], Y[v] \in \{0, 1\} \quad \forall t, w, v
\end{aligned}$$

Another way to see this simplified linear program is that it constrains that at least one witness must be preserved for every output tuple, and that if a witness is preserved, then all tuples in it must be preserved.

We now show that for queries that have the head cluster property, the solution of $\text{LP}[\text{GDP}_{\text{SWP}}]$ is always integral. We first restate the definition of the head cluster property.

Definition 6.4.1 (Existential Connectivity Graph G_Q^\exists) *The existential-connectivity graph of a query Q is a graph G_Q^\exists where each relation $R_i \in Q$ with $\text{attr}(R_i) - \text{head}(Q) \neq \emptyset$ is a vertex, and there is an edge between R_i and R_j if $\text{attr}(R_i) \cap \text{attr}(R_j) - \text{head}(Q) \neq \emptyset$.*

We can find the connected components of G_Q^\exists by applying search algorithm on G_Q^\exists , and finding all connected components for G_Q^\exists . Let $E_1, E_2, \dots, E_k \subseteq \text{rels}(Q)$ be the connected components of G_Q^\exists , each corresponding to a subset of relations in Q .

Definition 6.4.2 (Head Cluster Property) *A query Q has the head cluster property if for every pair of relations $R_i, R_j \in Q$ with $\text{head}(R_i) \neq \text{head}(R_j)$, it must be that R_i and R_j are in different connected components of G_Q^\exists .*

In other words, the head cluster property ensures that all relations in the same connected component of G_Q^\exists have exactly the same head variables.

First let's assume we have a query with a single connected component in G_Q^\exists . The head variables in this case are the same for all relations in the query, and are the same as the head variables of the query. Due to this, each input tuple contributes to a single output tuple, and thus we can treat each projection to be preserved independently. In other words, we can reduce our problem to preserving a single projection. Consider that the query has m relations. In a self-join-free query, at least one tuple from each relation must be preserved, and a solution can be obtained that preserves exactly m tuples by preserving anyone witness arbitrarily. Thus, the optimal solution to the ILP for a preserving a single projection independently is always to preserve exactly m tuples. We need to prove that the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$ never has an optimal value smaller than m . For every relation R_i in the query, we show that the sum of $X[t]$ variables for $t \in R_i$ is always at least 1. This is because for each witness we associate a corresponding R_i input tuple that has at least the fractional value assigned to $X[w]$ (if a tuple of R_i corresponds to multiple w then it takes on sum of the fractional values of $X[w]$). Since the sum of $X[w]$ variables is at least 1, the sum of $X[t]$ variables for $t \in R_i$ is at least 1. Repeating this argument for all relations in the query, we can see that the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$ is at least m , and thus never better than the optimal solution of the ILP.

Now let's consider a query with multiple connected components in G_Q^\exists . We argue that we can treat each connected component independently, and the optimal solution of the ILP is the sum of the optimal solutions of the ILP for each connected component. Due to the fact that we deal with self-join-free queries, no relations participate in multiple connected components, and thus we can partition the input to the ILP into disjoint sets of input relations.

We can show that the optimal value of the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$ is equal to the sum of the optimal values of $\text{LP}[\text{GDP}_{\text{SWP}}]$ for each connected component (which we have shown above to be integral). First we see naturally that the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$ naturally decomposes into the sum of the LP relaxations of $\text{LP}[\text{GDP}_{\text{SWP}}]$ for each connected component - since if the overall output is preserved, each connected component must be preserved. In the other direction, we want to show that if we add up the solutions for each connected component, we get a feasible solution for the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$. This follows from the fact that relations in different connected components share only head variables and thus the preserved tuples in one connected component always join with the preserved tuples in another connected component since they share the same head variables. Thus the sum of the solutions for each connected component is a feasible solution for the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$, and

thus the LP relaxation is integral and equal to the optimal solution of the ILP. \square

ADP-SS. A complexity dichotomy for the ADP-SS problem for self-join-free queries under set semantics is known. However, the complexity criterion [94] is much more involved. In particular, ADP-SS for a self-join-free query is PTIME if and only if (1) The query is Boolean and does not have a triad, (2) The query has a *singleton* relation, (3) Repeated application of decomposition by removing head variables that are present in all atoms, and treating disconnected components of a query independently, results in queries that are tractable. We show that no matter the reason for tractability, if we pose $\text{ADP-SS}(Q, \mathcal{D}, k)$ in the GDP framework, then the LP relaxation $\text{LP}[\text{ADP-SS}]$ is equivalent to the solution of $\text{ADP-SS}(Q, \mathcal{D}, k)$.

Theorem 6.4.4 (Integrality of $\text{LP}[\text{GDP}]$ for tractable instances of ADP-SS) $\text{LP}[\text{GDP}_{\text{ADP-SS}}(Q, \mathcal{D})] = \text{SWP}(Q, \mathcal{D})$ for all database instances \mathcal{D} if Q is a self-join-free for which $\text{ADP-SS}(Q)$ is known to be tractable under set semantics.

Proof Intuition. The proof of optimality of the LP relaxation for ADP-SS is similar to the proof for tractability in ADP-SS[94] in terms of the base cases and how queries are decomposed. For the base case of boolean queries without a triad, we use the proof of Theorem 6.4.1 as an argument, while for the base case of singleton relations, we use the proof similar to that of Theorem 6.4.2. We also show that the value of the LP relaxation is preserved even when the query is decomposed into multiple parts.

Proof Theorem 6.4.4. The proof of tractability of ADP-SS [94] is divided into two base cases and two types of recursive decompositions. The proof of the integrality of the LP relaxation of $\text{LP}[\text{ADP-SS}]$ follows the same structure as the original proof of tractability of ADP-SS [94].

The two bases cases are for (1) Boolean queries and (2) queries with a *singleton* relation i.e. a relation whose variables are either a subset of the relations of all other relations in the query, or a subset or superset of the head variables of the query. For base case (1), we see that this reduces exactly to the resilience problem, and we can simply use Theorem 6.4.1 to show the integrality of $\text{ILP}[\text{GDP}_{\text{ADP-SS}}]$. For base case (2), we observe that there is always an optimal solution of ADP-SS that deletes tuples only from the singleton relation. In the language of the resilience problem [59], the singleton relation dominates all other relations in the query. The tuples of the singleton relations can then be removed from the optimization problem. The resulting problem now contains only one tuple per witness. The constraint matrix of this ILP never contains a cycle - since no two input tuples share a witness, and thus do not form a cycle in the constraint matrix. The constraint matrix is thus balanced [144], and through well known results in optimization theory, is known to have an integral solution for any integral objective function.

The two types of recursive decompositions are for (1) queries with a universal attribute i.e., an attribute that appears in all relations of the query, and (2) disconnected queries. We apply that this decomposition in a similar manner to the proof of Theorem 6.4.3, and show that an optimal solution of $\text{LP}[\text{ADP-SS}]$ can be obtained by treating each decomposition of the query independently. Thus, the overall LP relaxation is always equal to the sum of the LP relaxations of the decomposed queries, and by applying these decompositions recursively and using the integral base

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[94]: Hu, Sun, Patwa, Panigrahi, and Roy (PVLDB, 2020), ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. doi:10.14778/3425879.3425892

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

[144]: Schrijver (1998), *Theory of linear and integer programming*. doi:10.1137/1030065

cases, we can show that the LP relaxation of $\text{LP}[\text{ADP-SS}]$ is integral for all queries for which ADP-SS is known to be in PTIME.

However, a difference in the decomposition performed here vs the proof of [Theorem 6.4.3](#) is that the value of k in the decomposition of ADP-SS must also be split across the decomposed queries. We do not actually need to run the decomposed ILP over all possible splits of k , however we simply prove that for any possible split of k , the solutions of the decomposed ILPs leads to a feasible solution of the original ILP that is (recursively) known to be integral. The optimal split of k across the decomposed queries (whatever it may be) is also then always recovered by the original LP relaxation and hence the LP relaxation is integral and equal to the optimal solution of the ILP for all queries for which ADP-SS is known to be in PTIME (for all queries for which applying these decompositions leads to one of the two base cases). \square

6.5 New Tractability Results

In this section, we show an example of a query that contains self-joins and unions, and is tractable under bag semantics for the DP-VS and SWP problems. We also show that this tractability can be recovered by posing the problems in the GDP framework. Bag semantics and queries with self-joins and unions are known to be challenging to analyze for deletion propagation problems, and papers have been written with the sole purpose of making progress on understanding the tractability landscape in this complicated settings [\[17, 60\]](#). This section (and the one query presented in it) are meant to act as a proof of concept that the $\text{ILP}[\text{GDP}]$ framework can be an invaluable tool to help understand and recover tractability results for various DP problems in these complicated settings.

We show that DP-SS, DP-VS, SWP and ADP-SS are tractable for Q^{Δ^-} [\(6.2\)](#) under bag semantics. We know from prior work that this query is hard for RES [\[60\][†]](#), and hence can infer that it is hard for DP-SS and ADP-SS as well - since they are both generalizations of RES.

$$Q^{\Delta^-}(a) := R(x, a, b), R(x, b, c), R(x, c, a) \cup R(x, e, f), R(x, f, g) \quad (6.2)$$

Proposition 6.5.1 (New tractable deletion propagation case via $\text{LP}[\text{GDP}]$)

For all database instances \mathcal{D} under bag semantics: (1) $\text{LP}[\text{GDP}_{\text{DP-VS}}(Q^{\Delta^-}, \mathcal{D})] = \text{DP-VS}(Q^{\Delta^-}, \mathcal{D}, t)$ for an arbitrary t in the view (2) $\text{LP}[\text{GDP}_{\text{SWP}}(Q^{\Delta^-}, \mathcal{D})] = \text{SWP}(Q^{\Delta^-}, \mathcal{D})$

Proof [Proposition 6.5.1](#). We look at each problem in turn.

For (1) DP-VS, we simply show that like in [Theorem 6.4.2](#), the optimal solution of $\text{ILP}[\text{GDP}_{\text{DP-VS}}]$ is always 1 and the LP relaxation thus cannot take a lower value. In other words, a solution can be obtained that is side effect free. If we simply delete all facts that contribute to the output tuple, we will obtain a side effect free solution since every relation contains a super set of the head variables of Q^{Δ^-} . Thus, every tuple that is consistent with the output tuple to be deleted, simply cannot be consistent with or contribute to any other output tuple, and thus this solution is side effect free. The rest of the proof is identical to [Theorem 6.4.2](#).

[†] It reduces to the boolean SJ-chain query $Q() := R(x, y), R(y, z)$, which is shown to be hard

[\[17\]](#): Bodirsky, Semanisinová, and Lutz (LICS, 2024), ‘The Complexity of Resilience Problems via Valued Constraint Satisfaction Problems’. doi:10.1145/3661814.3662071

[\[60\]](#): Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

[\[60\]](#): Freire, Gatterbauer, Immerman, and Meliou (PODS, 2020), ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. doi:10.1145/3375395.3387647

For (2) SWP, we show that like in [Theorem 6.4.3](#), each input tuple contributes to a single output tuple and thus each projection can be treated independently. We also observe that the first sub-query of Q^{Δ^-} i.e. $Q_1^{\Delta^-}(x) := R(x, a, b), R(x, b, c), R(x, c, a)$ is *dominated by* the second sub-query i.e., $Q_2^{\Delta^-}(x) := R(x, e, f), R(x, f, g)$. This means that preserving all output tuples of $Q_2^{\Delta^-}$ will also preserve all output tuples of $Q_1^{\Delta^-}$. In other words, the constraints enforced to preserve the output tuples of $Q_2^{\Delta^-}$ automatically imply the constraints that are enforced to preserve the output tuples of $Q_1^{\Delta^-}$, and it suffices to reason about the preservation of the output tuples of $Q_2^{\Delta^-}$.

We then also notice that each input tuple contributes to a single output tuple, and thus we can treat each projection independently (just like in [Theorem 6.4.3](#)). The difference here is that due to the self-joins in the query, a single input tuple can contribute to a witness “multiple times”. However, this does not make a difference in the phrasing of Propagation Constraint 4, since the constraint looks at each input tuple independently. It may be a witness has fewer tuples than atoms in the query (since tuples may join with themselves), however, in terms of the ILP this just means that there are fewer constraints to enforce, and the ILP is still integral. Due to this fact, the remainder of the proof is identical to [Theorem 6.4.3](#), and we can show that the LP relaxation of $\text{LP}[\text{GDP}_{\text{SWP}}]$ is integral for Q^{Δ^-} . \square

6.6 Experiments

The goal of our experiments is to evaluate the performance of our unified $\text{ILP}[\text{GDP}]$ (which is our short form for $\text{ILP}_S[\text{GDP}]$) by answering the following 4 questions: (Q1) Is the performance of $\text{ILP}[\text{GDP}]$ comparable to previously proposed specialized algorithms tailored to PTIME cases of particular DP problems? (Q2) Can our unified $\text{ILP}[\text{GDP}]$ indeed efficiently solve new tractable cases with self-joins, unions, and bag semantics that we proved to be in PTIME? (Q3) What, if any, is the performance benefit we obtain via smoothing constraints as discussed in [Section 6.3](#)? (Q4) What is the scalability of solving completely novel DP problems that fall into our unified GDP framework on real-world data? (Q5) What is the scalability of $\text{ILP}[\text{GDP}]$ for PTIME cases with respect to different input parameters such as the number of tuples, the number of relations, and the maximum domain size? (Q6) What is the memory usage of $\text{ILP}[\text{GDP}]$ for various PTIME cases of DP problems?

Algorithms. $\text{ILP}[\text{GDP}]$ denotes our ILP formulation for the generalized deletion propagation. DPVS-S, SWP-S, ADP-S denote prior specialized algorithms (“-S”) for the three problems V DP-VS [102], S SWP [93], and A ADP-SS [94], respectively. Recall that these are dedicated algorithms proposed for a PTIME cases of particular problems. We were not able to find open source code for any of these problems and implemented them based on the pseudocode provided in the respective papers that proposed them: V [102], S [93], A [94]. To the best of our knowledge, no experimental evaluation has ever been undertaken for some of these algorithms [93, 102]. We do not include experimental comparison for DP-SS, as for this problem the $\text{ILP}[\text{GDP}]$ produced is exactly the same as a prior specialized approach [110], and hence there is no difference in performance.

Data. For most experiments we generate synthetic data by fixing the max domain size to 1000, and sampling randomly from all possible tuples. For experiments under bag semantics, each tuple is duplicated by a random

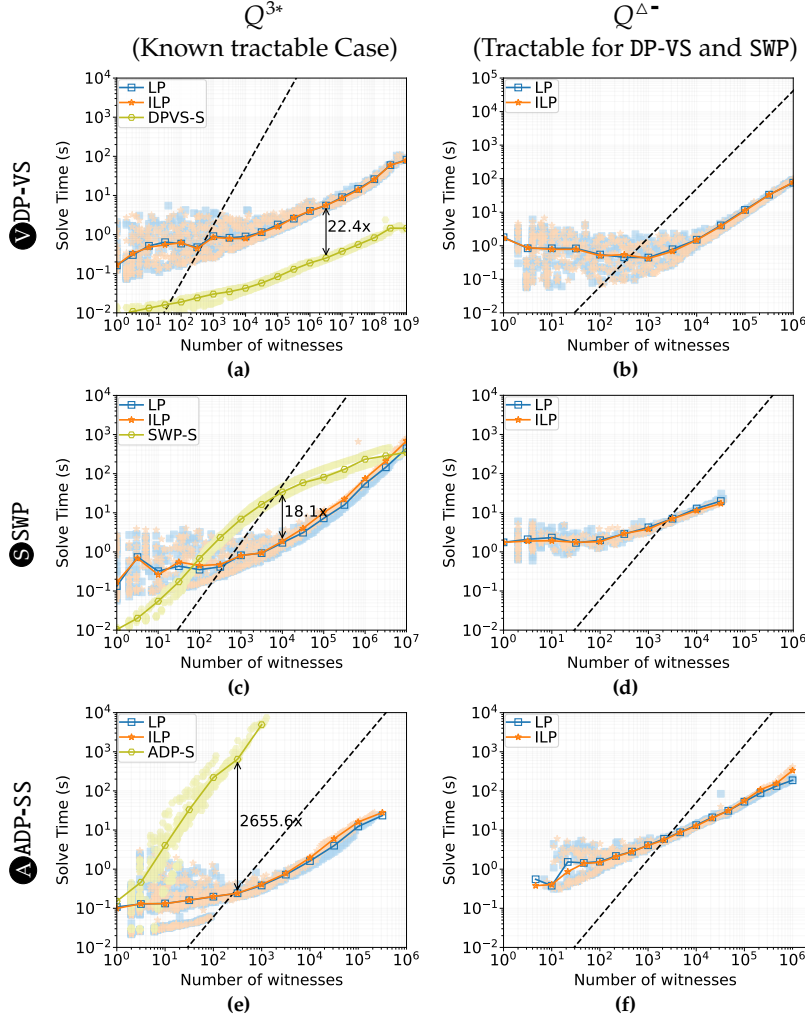


Figure 6.8: (Q1)/(Q2): Performance of ILP[GDP] on a previously known (left column compared against prior specialized algorithms) and a newly discovered tractable query (right column with no prior known specialized algorithm) for three prior studied problems **V** DP-VS, **S** SWP, and **A** ADP-SS. In all cases, ILP[GDP] scales well under the theoretical worst case complexity of ILPs and LPs.

number that is smaller than a pre-specified max bag size of 10. For answering (Q4) in Figure 6.10, we use an existing flights' database [136] that shows flights operated by different airlines in Jan 2019 as real world data case study on a novel problem.

Software and Hardware. The algorithms are implemented in Python 3.8.8 and solve the optimization problems with Gurobi Optimizer 10.0.1. Experiments are run on an Intel Xeon E5-2680v4 @2.40GH machine available via the Northeastern Discovery Cluster.

Experimental Protocol. For each plot we run 3 runs of logarithmically and monotonically increasing database instances. We plot all obtained data points with a low saturation, and draw a trend line between the median points from logarithmically increasing sized buckets. All plots are log-log, and we include a dashed line to show linear scalability as reference in the log-log plot.

(Q1) Known tractable cases. Is the performance of ILP[GDP] over PTIME instances comparable to specialized algorithms studied in prior work? We pick the 3-star query $Q^{3*}(a) :- R(a, b), S(a, c), R(a, d)$, for which all three problems can be solved in PTIME. We run all three problems on this query and compare the performance of ILP[GDP] against specialized algorithms. The ILP and the LP have worse worst-case complexity than the specialized algorithms, however we see that ILP[GDP] is at times even faster than the specialized algorithms, due to the better heuristics used in

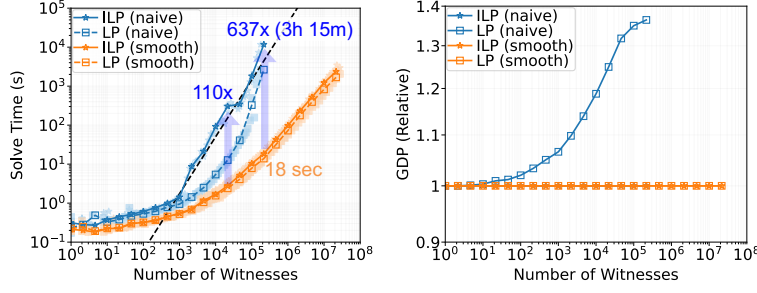


Figure 6.9: (Q3): Experiment showing the power of Smoothing Constraints in ILP[GDP]: we observe that ILP[GDP] is orders-of-magnitude faster than the naive ILP formulation $ILP_N[GDP]$, while also guaranteeing the optimality of its LP relaxation. Contrast with the LP relaxation of $ILP_N[GDP]$ which can have an over 30% higher optimal objective (GDP) value.

the ILP solver. In Figure 6.8a, we see that ILP[GDP] is about 20 times worse than the specialized algorithms for **D** DP-VS. But this is expected for this case, since the PTIME cases of DP-VS are only those where there are *no side effects* and any tuple that contributes to the answer can be deleted, thus making the problem solvable via a trivial algorithm. However, in Figures 6.8c and 6.8e, we see that ILP[GDP] is about 2 and 3 orders of magnitude *faster than the specialized algorithms* for **S** SWP and **A** ADP-SS, both of which use decomposition based techniques, with the specialized algorithm for **A** ADP-SS also requiring dynamic programming. Thus, although the specialized algorithms have better asymptotic fine-grained complexity and will perform better on adversarially chosen instances, over random instances the LP solver (using heuristics) is able to find a solution faster.

(Q2) Newly discovered Tractable cases. We evaluate the performance of $ILP[GDP_{DP-VS}]$, $ILP[GDP_{SWP}]$ and $ILP[GDP_{ADP-SS}]$ for Q^{Δ^+} , a query with self-joins and a union that we run under bag semantics. We showed in Section 6.5 that the **D** DP-VS, **S** SWP, and **A** ADP-SS problems are tractable for this query under bag semantics. There are no known specialized algorithms for these problems, and thus we only compare the performance of ILP[GDP] with the PTIME LP Relaxation. We see in Figures 6.8b, 6.8d and 6.8f that the ILP is as fast as its LP Relaxation and shows linear scalability even for this complicated setting.

(Q3) The Power of Smoothing Constraints. Figure 6.9 shows two orders of magnitude speedup in the ILP solving time after adding our Smoothing Constraint (SC). We run $ILP[GDP_{SWP}]$ for the 3-star query Q^{3*} , contrasting ILP[GDP] with and without smoothing constraints. Since we show in Section 6.4 that the LP relaxation of ILP[GDP] shares the same optimum objective value, this surprising speed-up is completely justified by our the fact that ILP solvers can solve ILPs efficiently when the LP relaxation is tight. Moreover, we see that the LP relaxation of $ILP[GDP_{SWP}]$ is always tight for the Q^{3*} query. This is notably not true for the naive ILP formulation, which can have an over 30% higher optimal objective (GDP) value.[‡]

(Q4) General performance. We use the flights' database [136] that shows 500K+ flights operated by 17 different airlines in Jan 2019. We take the use case of Example 6.0.1 and solve the GDP with the following requirements: (1) Cut 2% of total costs of the airline, (2) Let the connection network of an airline contain all pairs of places that have a direct (0-hop) or 1-hop flight between them. Minimize the effect of deletions on the connection network i.e., minimize the number of location pairs that are removed from the connection network. (3) A subset of pairs of places are

[‡] Due to SWP being a maximization problem being posed as a minimization problem through GDP_{SWP} , the optimal values of GDP_{SWP} are negative, and hence the magnitude of the LP relaxation is higher than the ILP (despite the LP being a lower bound).

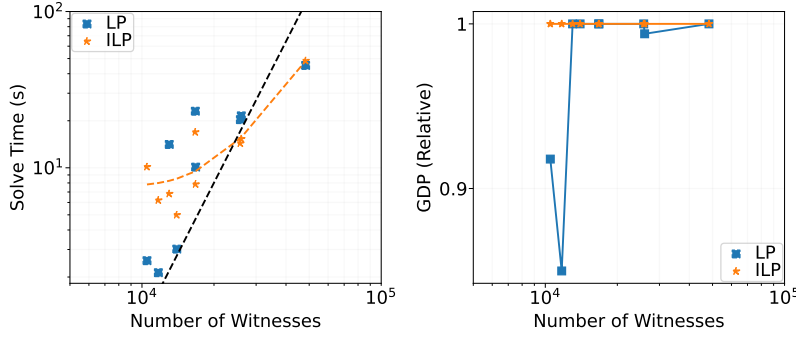


Figure 6.10: (Q4): Performance Evaluation of the Generalized Deletion Propagation over a real-world dataset shows fast solve times (within a minute) and comparable to linear-time (black dashed line) scalability.

“popular connections”. Ensure that such connections are preserved in the connection network. For our case study, we assign a random expense value to each airport as an airport fee and each flight as a fuel cost, since the dataset does not contain any cost information. However, such information or cost factors if known can be easily incorporated into the ILP formulation by simply changing the randomly assigned costs to the actual costs. We assume if an airline uses an airport, it must pay its fee and assume that there are no additional costs. We show in Figure 6.10 the time to solve the GDP for this use case via ILP[GDP], and compare it to the LP relaxation. We see that even for this real-world dataset, where there are no guarantees on the properties of ILP[GDP], most instances are solved in well under a minute, and the optimal solutions of the ILP and LP relaxation coincide in many cases.[§]

(Q5) Scalability of ILP[GDP] for changing parameters. Figure 6.11 show the scalability of ILP[GDP] for DP-VS, SWP and ADP-SS over different maximum domain sizes ($10^2, 10^3, 10^4, 10^5$), and Figure 6.12 for k -star queries with increasing number of joins ($Q^{3*}, Q^{4*}, Q^{5*}, Q^{6*}$). We see that the scalability in both cases is very well predicted by the number of witnesses, irrespective of the domain size or number of tuples. Increasing the maximum domain size leads to a less dense instance that has fewer witnesses and is thus easier to solve than a more dense instance with the same number of tuples. Similarly, for changing the number of joins in the query.

[§] We observed that in some cases, ILP is faster than LP. This is a known observation and may be due to numerical and floating-point issues [82].

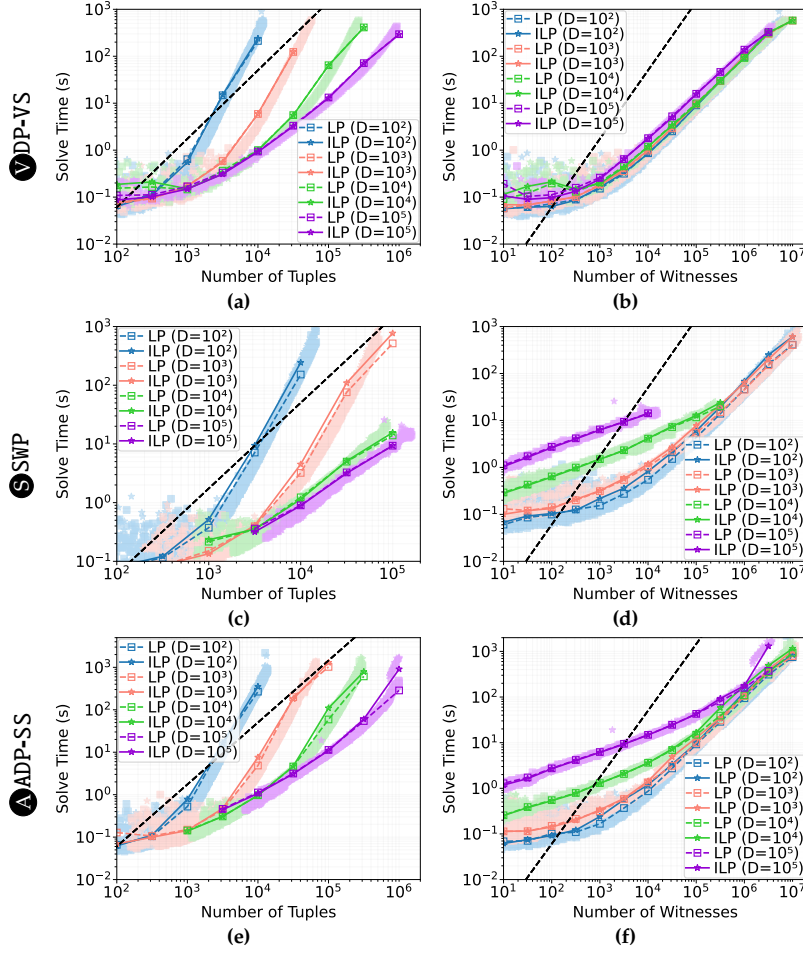


Figure 6.11: (Q5a): Scalability of ILP[GDP] for different domain sizes: we show the scalability with respect to the number of tuples (left) and the number of witnesses (right) for DP-VS (top), SWP (middle), and ADP-SS (bottom). We find that scalability of ILP[GDP] is very well predicted by the number of witnesses, irrespective of the number of tuples or the domain size used.

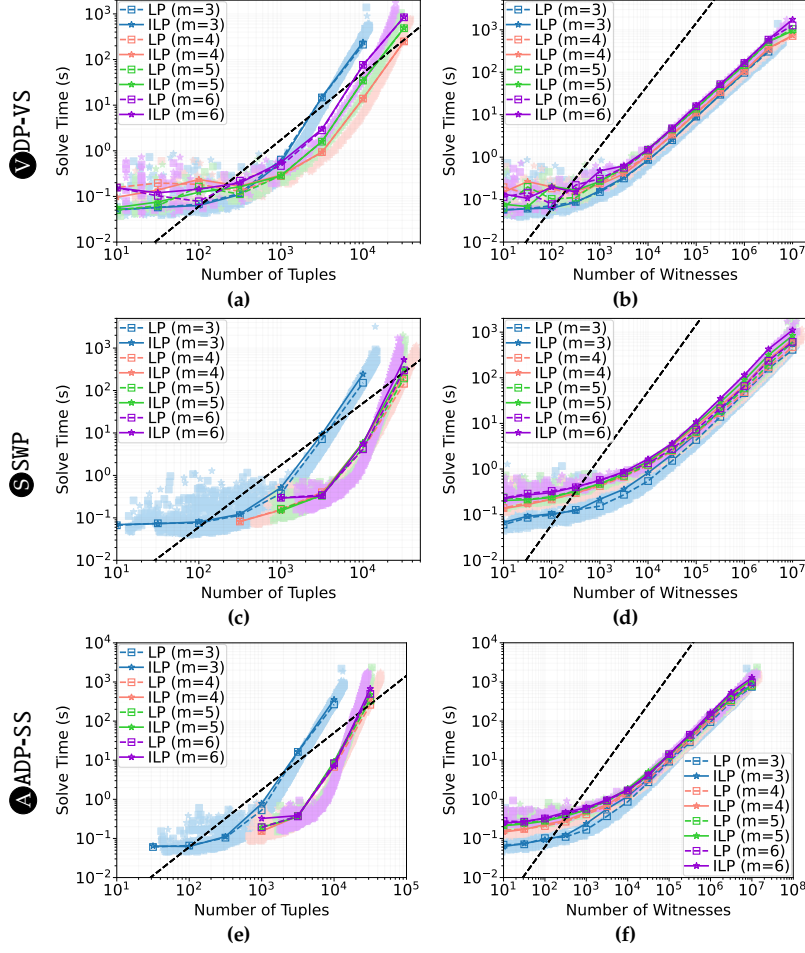


Figure 6.12: (Q5b): Scalability of ILP[GDP] for different query size: we show the scalability with respect to the number of tuples (left) and the number of witnesses (right) for DP-VS (top), SWP (middle), and ADP-SS (bottom). We find that scalability of ILP[GDP] is very well predicted by the number of witnesses, irrespective of the number of tuples or the number of joins in the query (given by m).

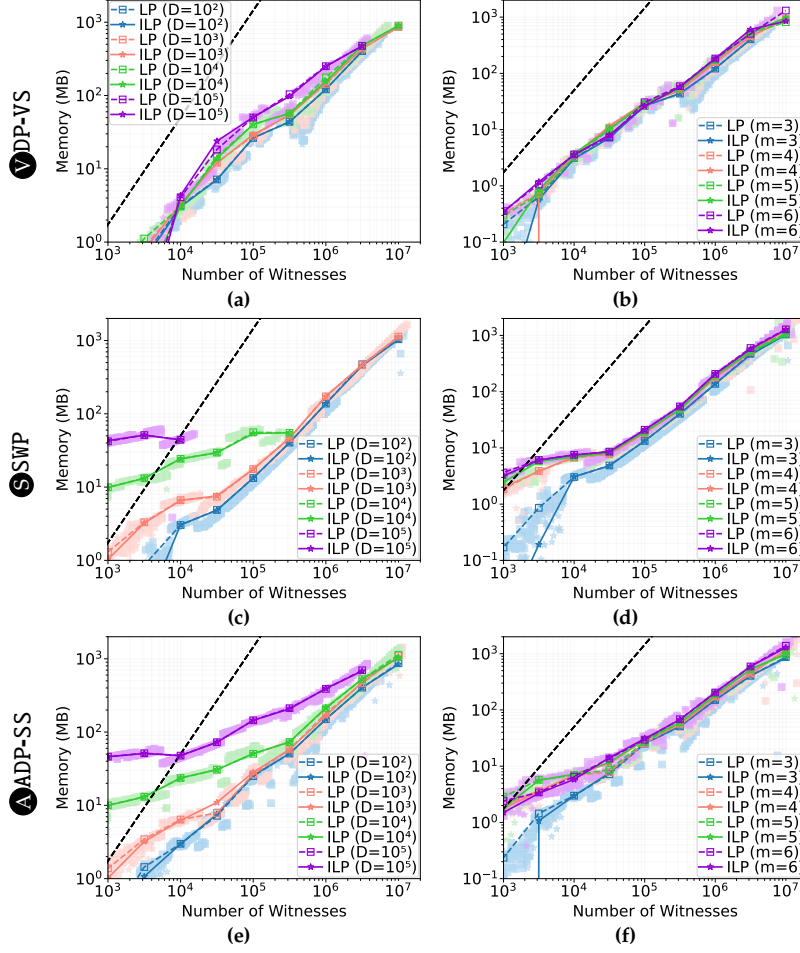


Figure 6.13: (Q6): Memory consumption of ILP[GDP] for three prior studied problems **V** DP-VS, **S** SWP, and **A** ADP-SS. The left column shows the memory consumption over different instances with increasing domain sizes, and the right column shows the memory consumption over different queries with increasing number of joins. In all cases, ILP[GDP] has sublinear memory consumption w.r.t. the number of witnesses.

(Q6) Memory consumption of ILP[GDP]. We conducted an additional experiment [Figure 6.13](#) that shows the measured space consumption of ILP[GDP] using the psutil library[¶]. [Figure 6.13](#) shows the memory consumption of ILP[GDP] for a DP-VS, SWP and ADP-SS problems as a function of the number of witnesses, parameterized for various maximum domain sizes and queries with increasing number of joins. We see that although the memory consumption is not guaranteed to be sublinear, in practice it is sublinear w.r.t. the number of witnesses.

6.7 A Note on System Implementation

Our current proof of concept (see [Section 6.6](#)) uses Python to compute the provenance of query results, and to translate this into an ILP formulation. This part could be more tightly integrated with existing database systems by leveraging existing efforts in our community that have been investigating how to repurpose provenance functionality during query execution into PostgreSQL, such as Perm [70, 71], GProM [10], or ProvSQL [148]. Furthermore, the extensibility features of today’s database systems could be used to add such an ILP solver to the database systems, just user defined functions can be written in programming languages other than the native SQL [20, 26, 109]. Such integrations of highly sophisticated solvers for solving important database problems have been previously proposed

[¶]<https://pypi.org/project/psutil/>

for consistent query answering [47] and query optimization [153], and we believe will become more common.

We believe our unified approach provides an easier integration into existing database infrastructure than prior solutions for several reasons: ① Prior solutions to individual problems use different solution approaches (e.g., ADP-SS uses dynamic programming, whereas DP-SS uses reduction to flow). Thus an integration of all prior work would require *several adaptations*, one for each method. ② For an approach to be natively supported by a relational database (thus without user defined functions written in a programming language), the approach would have to be first-order rewritable. Among the prior solutions, the only cases we know of that are first-order rewritable are the few PTIME cases for DP-VS [102]. All other approaches require writing functionality in programming languages other than SQL, just as ours. ③ All prior exact methods (except [110]) are incomplete in that they work only for those conjunctive queries which can be solved in guaranteed PTIME. Only some prior work propose approximation algorithms for the hard cases (such as the one for DP-VS), yet implementing those require yet other methods. ④ In addition, to our approach being the only one that is complete, it also has a desirable anytime property: ILP solvers can produce solutions of increasing quality as optimization progresses and are able to provide bounds for how far the current solution is from the optimum.

6.8 Chapter Summary

In this chapter, we have introduced a new framework for deletion propagation problems that it is practical, efficient, and complete. We leave with an interesting conjecture: *All queries for which deletion propagation problems are tractable can be solved in PTIME via our approach.* We show that *this conjecture is true for all currently known tractable cases*, and also show an example of a query with self-joins and unions that was not previously studied and that is tractable for our approach, even under bag semantics. If proven correct, this would be another practically appealing reason for using our approach now: we would get guaranteed PTIME performance already today, even if the proof happens in the future. We leave open the question of whether this conjecture is true for all tractable cases, and the question of the complexity dichotomies of various deletion propagation problems (including the ones introduced in this dissertation).

Minimal Factorization of Provenance Formulas

7

Given the provenance formula for a Boolean query, what is its *minimal size equivalent formula*? And under what conditions can this problem be solved efficiently? This chapter investigates the complexity of minFACT, i.e. the problem of finding a minimal factorization for the provenance of self-join-free conjunctive queries (sj-free CQs). While the general Boolean formula minimization is Σ_2^P -complete [21], several important tractable subclasses have been identified, such as read-once formulas [76]. In this chapter, we identify additional tractable cases by identifying a large class of queries for which the minimal factorization of any provenance formula can be found in PTIME.

We focus on provenance formulas for two key reasons: 1) Provenance computation and storage is utilized in numerous database applications. The issue of storing provenance naturally raises the question: How can provenance formulas be represented minimally? This problem has previously been investigated in this context [131], where algorithms were described for factorizations with asymptotically optimal sizes, leading to work on factorized databases. However, finding instance optimal factorizations i.e. factorizations that are guaranteed to be the smallest possible, for any arbitrary input, remains an open challenge, and is the focus of our work.

2) Minimal factorizations of provenance formulas can be used to obtain probabilistic inference bounds. Prior approaches for approximate probabilistic inference are either incomplete i.e. focus on just PTIME cases [141], or do not solve all PTIME cases exactly [39, 66]. As we show, using minimal factorization as a preprocessing step achieves the best of both worlds: It is complete (i.e. it applies to easy and hard cases) while recovering all known PTIME cases exactly.

In this chapter, we prove that the minimal factorization problem is NP-Complete (NPC) for provenance formulas, and give two algorithms for all sj-free CQs that are unified algorithms in the sense that they solve all known tractable cases in PTIME, and provide approximations for hard cases. We further place the set of tractable queries firmly between the tractable queries for two other related problems: resilience [59] and probabilistic query evaluation [38] (Figure 7.1).

7.1	Chapter Overview and Contributions	79
7.2	Problem Definition	80
7.3	Related Work	80
7.4	Search Space for minFACT	83
7.5	ILP Formulation for minFACT	98
7.6	PTIME Algorithms	102
7.7	Recovering Read-Once instances	110
7.8	Tractable Queries for minFACT	112
7.9	Hard Queries for minFACT	119
7.10	Application: A complete approach for Approximate Probabilistic Inference	122
7.11	Optimizations for computing minFACT	126
7.12	Experiments	128
7.13	Chapter Summary	132

This chapter is based on: Neha Makhija and Wolfgang Gatterbauer. 2024. Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries. *Proc. ACM Manag. Data* 2, 2, Article 104 (May 2024), 24 pages. <https://doi.org/10.1145/3651605> [111].

[21]: Buchfuhrer and Umans (JCSS, 2011), ‘The complexity of Boolean formula minimization’. doi:10.1016/j.jcss.2010.06.011

[76]: Golumbic, Mintz, and Rotics (JDAM, 2008), ‘An improvement on the complexity of factoring read-once Boolean functions’. doi:10.1016/j.dam.2008.02.011

[131]: Olteanu and Závodný (ICDT, 2012), ‘Factorised representations of query results: size bounds and readability’. doi:10.1145/2274576.2274607

[141]: Roy, Perduca, and Tannen (ICDT, 2011), ‘Faster query answering in probabilistic databases using read-once functions’. doi:10.1145/1938551.1938582

[39]: Dalvi and Suciu (VLDBJ, 2007), ‘Efficient query evaluation on probabilistic databases’. doi:10.1007/s00778-006-0004-3

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

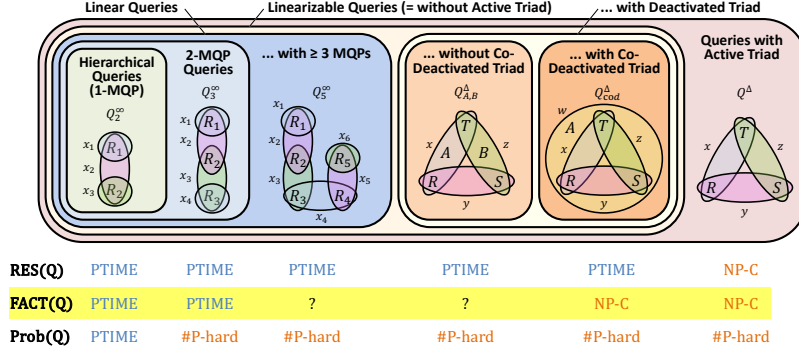


Figure 7.1: This chapter gives hardness results, identifies PTIME cases, and gives exact and approximate algorithms for self-join-free conjunctive queries. We prove that the tractable queries for minFACT reside firmly between the tractable cases for probabilistic query evaluation (PROB) = the hierarchical queries with one minimal query plan, and those for resilience (RES) = queries without active triads. The open cases are linear queries with ≥ 3 minimal query plans (though we know that Q_4^∞ is in PTIME), and linearizable queries with deactivated triads and without co-deactivated triads (though we know that the triangle unary query Q_A^Δ is in PTIME).

7.1 Chapter Overview and Contributions

- ❶ The minFACT problem has strong ties to the diverse problems of Boolean factorization, factorized databases, probabilistic inference, and resilience, among others. Section 7.3 explains these connections after Section 7.2 formalizes the problem.
- ❷ Section 7.4 describes connections between provenance factorizations, variable elimination orders (VEOs) and query plans. These connections allow us to reformulate minFACT as the problem of assigning each witness to one of several “minimal VEOs.”
- ❸ Section 7.5 develops an ILP encoding to solve minFACT for any sj-free CQ exactly. We are not aware of any prior ILP formulation for minimal-size encodings of propositional formulas, for restricted cases like for monotone formulas.
- ❹ Section 7.6 describes our two unified PTIME algorithms that are exact for *all known* PTIME cases, and approximations otherwise. The first one encodes the problem in the form of a “factorization flow graph” s.t. a *minimal cut* of the graph corresponds to a valid factorization of the instance. We refer to this algorithm as the MFMC (Max-Flow Min-Cut) based algorithm. The second is an LP relaxation of our ILP encoding, for which we also prove a guaranteed constant factor approximation for hard queries.
- ❺ Section 7.7 proves that both our unified algorithms can solve the minFACT problem *exactly* if the database instance allows a *read-once factorization*. This implies that our algorithms recover and generalize prior approaches [141] that are limited to read-once formulas.
- ❻ Section 7.8 provides a large class of queries for which our PTIME algorithms can solve the minFACT problem exactly over any database instance. This class includes hierarchical queries as a strict subset, proving that the tractable queries for minFACT are a strict superset of those for probabilistic query evaluation (PROB) [39].
- ❼ Section 7.9 proves that the decision variant of minFACT is NPC for a set of queries that form a strict superset of queries that contain “active triads”. This result proves that the intractable queries for minFACT are a strict superset of those that are intractable for resilience (RES) [59],

[141]: Roy, Perduca, and Tannen (ICDT, 2011), ‘Faster query answering in probabilistic databases using read-once functions’. doi:10.1145/1938551.1938582

[39]: Dalvi and Suciu (VLDB, 2007), ‘Efficient query evaluation on probabilistic databases’. doi:10.1007/s00778-006-0004-3

[59]: Freire, Gatterbauer, Immerman, and Meliou (PVLDB, 2015), ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. doi:10.14778/2850583.2850592

thereby bounding the tractable queries for our problem firmly between those tractable for PROB and those tractable for RES.

⑧ [Section 7.10](#) shows that using minimal factorization can lead to more accurate probabilistic inference.

⑨ [Section 7.12](#) contains experiments evaluating the performance and results of the ILP encoding, LP relaxation, and MFMC-based algorithm.

7.2 Problem Definition

For a provenance $\varphi_p = \text{Prov}(Q, D)$ as DNF, we want to find an equivalent formula $\varphi' \equiv \varphi_p$ with the minimum number of literals.

Definition 7.2.1 (FACT) *Given a query Q and database \mathcal{D} , we say that $(D, k) \in \text{FACT}(Q)$ if there is a formula φ' of length $\text{len}(\varphi') \leq k$ that is equivalent to the expression $\varphi_p = \text{Prov}(Q, D)$.*

Our focus is to determine the difficulty of this problem in terms of data complexity [156], i.e., we treat the query size $|Q|$ as a constant. We are interested in the optimization version of this decision problem: given Q and \mathcal{D} , find the *minimum* k such that $(D, k) \in \text{FACT}(Q)$. We refer to this optimization variant as the minFACT problem and use $\text{minFACT}(Q, D)$ to refer to the length of the minimal size factorization for the provenance of database \mathcal{D} under query Q .

We focus on Boolean queries (i.e., where $\mathbf{y} = \emptyset$), since the problem of finding the minimal factorization of the provenance for one particular output tuple of a non-Boolean query immediately reduces to the Boolean query case (see e.g. [150]).* We also focus on self-join-free conjunctive queries (sj-free CQs) as a first step.

[156]: Vardi (STOC, 1982), ‘The Complexity of Relational Query Languages (Extended Abstract)’. doi:10.1145/800070.802186

[150]: Suciu, Olteanu, Ré, and Koch (2011), *Probabilistic Databases*. doi:10.2200/s00362ed1v01y201105dtm016

Example 7.2.1 (FACT) Consider the provenance of Q_2^* over the modified database \mathcal{D} with tuple s_{13} from [Figure 3.1](#). It has no read-once form and a minimal size formula is

$$\varphi'' = r_1(s_{11}t_1 \vee s_{12}t_2 \vee s_{13}t_3) \vee (r_2s_{23} \vee r_3s_{33})t_3$$

We see that $\text{len}(\varphi'') = 12$. It follows that $(D, 12) \in \text{FACT}(Q_2^*)$. At the same time, $(D, 11) \notin \text{FACT}(Q_2^*)$ and thus $\text{minFACT}(Q_2^*, D) = 12$.

7.3 Related Work

While the related work discussed in [Chapter 2](#) is still relevant in this chapter, there are some additional factorization-specific connections we point out here.

* A solution to Boolean queries immediately also provides an answer to a non-Boolean query $Q(\mathbf{y})$: For each output tuple $t \in Q(D)$, solve the problem for a Boolean query Q' that replaces all head variables \mathbf{y} with constants of the output tuple t .

7.3.1 Boolean Factorization

Minimum Equivalent Expression (MEE) is the problem of deciding whether a given Boolean formula φ (note that we use the terms expressions and formulas interchangeably) has a logically equivalent formula φ' that contains $\leq k$ occurrences of literals. It was known to be at least NP-hard for over 40 years [64, Section 7.2] and was shown to be Σ_p^2 -complete only 10 years ago [21]. The problem is more tractable for certain restrictions like Horn formulas [88] as input, or if allowing arbitrary Boolean functions as connectors [90], or if posed as the Minimum Formula Size Problem (MFSP) that takes the uncompressed truth table as input [6, 96]. There is a lot of work on approximate Boolean function factorization [114, 122], however efficient, exact methods are limited to classes such as read-once [76] and read-polarity-once formulas [24] (see [35, Section 10.8] for a detailed historical overview). Our problem restricts the formula to be minimized to the provenance of a sj-free conjunctive query (i.e. a monotone, m -partite DNF that follows join dependencies), with the goal of uncovering important classes that permit a PTIME exact evaluation.

Figure 7.2 illustrates the landscape of known results for the problem of Minimum Equivalent Expressions (MEE) applied to formulas. The general problem of MEE has been long known to be NP-hard [64]. However, only relatively recently it has been proved to be Σ_p^2 -complete [21]. Various important classes of this problem have been studied, a fundamental one being the factorization of DNF expressions. The MinDNF problem [155], deals with finding the minimum equivalent DNF expression of an input DNF formula, and is also known to be Σ_p^2 -complete. However, if the input to the MinDNF is the truth table (or set of all true assignments of the formula) then the problem is NPC [6]. If we take away the restriction that the factorized formula must be a DNF, then the problem of finding the minimum factorization of an input table is known as the Minimum Formula Size Problem (MFSP) and is shown to be in NP and (ETH)-hard [96].

Another important class of restrictions is over monotone formulas (thus we do not allow negatives in input or output formulas). Surprisingly, we do not know of any work that proves the complexity of the general monotone boolean factorization problem. However, there are many interesting and important restrictions for which complexity results are known. One such important sub-class is that of read-once formulas, which can be factorized in PTIME [76]. For Monotone formulas with DNF input and output restrictions, the problem can be solved in logspace by eliminating monomials [73]. Interestingly the problem monotone formula factorization of an arbitrary formula with a DNF restriction on the output only has differing complexity based on the input encoding of the length of the factorization. Checking if the minimum size of a DNF for a monotone formula is at most k is PP-complete, but for k in unary, the complexity of the problem drops to coNP [73]. The intuition is that in this problem, (which can be seen as “dual” of minFACT since it has a DNF output restriction instead of a DNF input restriction), the optimal output (a DNF) can be exponentially larger than the input (any monotone formula).

Our problem of minFACT is a further restriction on the MEE problem applied to a monotone DNF. Provenance formulas for sj-free CQs are k -partite monotone formulas that satisfy join dependencies. We prove in this paper that the problem is NPC, in general, and further identify interesting PTIME subcases.

[64]: Garey and Johnson (1979), *Computers and intractability*.

[21]: Buchfuhrer and Umans (JCSS, 2011), ‘The complexity of Boolean formula minimization’. doi:10.1016/j.jcss.2010.06.011

[88]: Hammer and Kogan (AI, 1993), ‘Optimal compression of propositional Horn knowledge bases: complexity and approximation’. doi:10.1016/0004-3702(93)90062-G

[90]: Hemaspaandra and Schnoor (IJCAI, 2011), ‘Minimization for generalized boolean formulas’.

[6]: Allender, Hellerstein, McCabe, Pitassi, and Saks (SICOMP, 2008), ‘Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table’. doi:10.1137/060664537

[96]: Ilango (FOCS, 2022), ‘The Minimum Formula Size Problem is (ETH) Hard’. doi:10.1109/FOCS52979.2021.00050

[114]: Martins, Rosa, Rasmussen, Ribas, and Reis (ICCD, 2010), ‘Boolean factoring with multi-objective goals’. doi:10.1109/ICCD.2010.5647772

[122]: Mintz and Golumbic (JDAM, 2005), ‘Factoring Boolean functions using graph partitioning’. doi:10.1016/j.dam.2005.02.007

[76]: Golumbic, Mintz, and Rotics (JDAM, 2008), ‘An improvement on the complexity of factoring read-once Boolean functions’. doi:10.1016/j.dam.2008.02.011

[24]: Callegaro, Martins, Ribas, and Reis (SBCCI, 2013), ‘Read-polarity-once Boolean functions’. doi:10.1109/SBCCI.2013.6644862

[35]: Crama and Hammer (2011), *Boolean Functions: Theory, Algorithms, and Applications*. doi:10.1017/cbo9780511852008.003

[64]: Garey and Johnson (1979), *Computers and intractability*.

[21]: Buchfuhrer and Umans (JCSS, 2011), ‘The complexity of Boolean formula minimization’. doi:10.1016/j.jcss.2010.06.011

[155]: Umans (JCSS, 2001), ‘The minimum equivalent DNF problem and shortest implicants’. doi:10.1109/sfcs.1998.743506

[6]: Allender, Hellerstein, McCabe, Pitassi, and Saks (SICOMP, 2008), ‘Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table’. doi:10.1137/060664537

[96]: Ilango (FOCS, 2022), ‘The Minimum Formula Size Problem is (ETH) Hard’. doi:10.1109/FOCS52979.2021.00050

[76]: Golumbic, Mintz, and Rotics (JDAM, 2008), ‘An improvement on the complexity of factoring read-once Boolean functions’. doi:10.1016/j.dam.2008.02.011

[73]: Goldsmith, Hagen, and Mundhenk (JIC, 2008), ‘Complexity of DNF minimization and isomorphism testing for monotone formulas’. doi:10.1016/j.ic.2008.03.002

[73]: Goldsmith, Hagen, and Mundhenk (JIC, 2008), ‘Complexity of DNF minimization and isomorphism testing for monotone formulas’. doi:10.1016/j.ic.2008.03.002

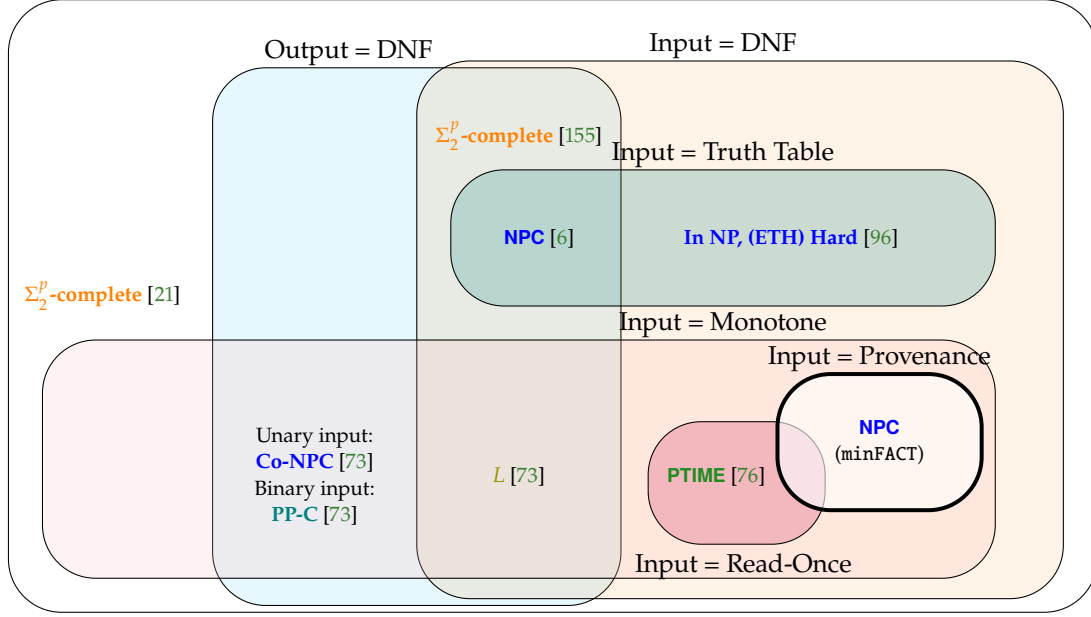


Figure 7.2: An overview of related work on the Exact, Minimal Equivalent Expression (MEE) problem applied to formulas.

Factorized Databases. Our problem has been studied before in the context of Factorized Databases (FDBs) [129–132]. Five key differences in focus are: (i) The tight bounds provided through that line of work are on “readability” i.e. the lowest k such that each variable in the factorized formula is repeated at most k times. The work shows that the class of queries with bounded readability is strictly that of hierarchical queries [131]. In contrast, we focus on the minimal number of variable repetitions and show this can be calculated in PTIME for a strict superset of hierarchical queries. (ii) For bounds on the minimal length (as is our focus), FDBs focus only *asymptotic* bounds on the size of query result representations [132] whereas we focus on minimizing the exact number of variables (e.g. whether a provenance is read-once or has a factor 2 bigger size is of no relevance in the asymptotic analysis of FDBs). (iii) variants of FDBs permit the reuse of intermediate results, i.e. they focus on the corresponding *circuit* size, while we focus on *formulas*. (iv) Intuitively, FDBs study the trade-offs between applying one of several factorizations (or query plans or variable elimination orders) to the entire query results at once, whereas we may *factorize each witness in different ways*. (v) Except for [130], the work on FDBs focuses on factorizations in terms of *domain values* whereas provenance formulas are defined in terms of tuple variables (e.g. a tuple from an arity-3 relation has 3 different domains, but is still represented by a single tuple variable). These discrepancies lead to different technical questions and answers. Also related is the very recently studied problem of finding a factorized representation of all the homomorphisms between two finite relational structures [14]. Similar to FDBs, that work also differs from ours in that it focuses on the asymptotic factorization size (and proves lower bounds and allows a circuit factorization instead of a formula). Our problem is also different from the problem of calculating a “ p -minimal query” for a given query [9]: The solution to our problem depends on the database instance and factorizes a given provenance formula, whereas the latter problem is posed irrespective of any given database, chooses among alternative polynomials, and becomes trivial for queries without self-joins.

[129]: Olteanu and Schleich (SIGMOD Rec. 2016), ‘Factorized Databases’. doi:10.1145/3003665.3003667

[130]: Olteanu and Závodný (TaPP, 2011), ‘On Factorisation of Provenance Polynomials’.

[131]: Olteanu and Závodný (ICDT, 2012), ‘Factorised representations of query results: size bounds and readability’. doi:10.1145/2274576.2274607

[132]: Olteanu and Závodný (TODS, 2015), ‘Size Bounds for Factorised Representations of Query Results’. doi:10.1145/2656335

[14]: Berkholz and Vinnik (ICALP, 2023), ‘A Dichotomy for Succinct Representations of Homomorphisms’. doi:10.4230/LIPIcs.ICALP.2023.113

[9]: Amsterdamer, Deutch, Milo, and Tannen (TODS, 2012), ‘On Provenance Minimization’. doi:10.1145/2389241.2389249

7.3.2 Probabilistic Inference, Read-Once Formulas, and Dissociation

Probabilistic query evaluation (PROB) is #P-hard in general [39]. However, if a provenance formula φ can be represented in *read-once form* then its marginal probability $\mathbb{P}[\varphi]$ can be computed in linear time in the number of literals. Olteanu and Huang [127] showed that the previously known tractable queries called hierarchical queries lead to read-once factorizations. A query Q is called hierarchical [39] iff for any two existential variables x, y , one of the following three conditions holds: $\text{at}(x) \subseteq \text{at}(y)$, $\text{at}(x) \supseteq \text{at}(y)$, or $\text{at}(x) \cap \text{at}(y) = \emptyset$, where $\text{at}(x)$ is the set of atoms of Q in which x participates. Roy et al. [141] and Sen et al. [147] independently proposed algorithms for identifying read-once provenance for non-hierarchical queries in PTIME. Notice that finding the read-once form of a formula (if it exists) is just an extreme case of representing a Boolean function by a minimum length (\vee, \wedge) -formula. Our solution is a *natural generalization* that is *guaranteed* to return a read-once factorization in PTIME should there be one. We give an interesting connection by proving that the tractable queries for our problem are a *strict superset* of hierarchical queries and thus the tractable queries for probabilistic query evaluation.

Given a provenance that is not read-once, one can still upper and lower bound its probability efficiently via dissociation [65]: Let φ and φ' be two Boolean formulas with variables \mathbf{x} and \mathbf{x}' , respectively. Then φ' is a *dissociation* of φ if there exists a substitution $\theta : \mathbf{x}' \rightarrow \mathbf{x}$ s.t. $\varphi'[\theta] = \varphi$. If $\theta^{-1}(x) = \{x'_1, \dots, x'_d\}$, then variable x dissociates into d variables x'_1, \dots, x'_d . Every provenance expression has a unique read-once dissociation up to renaming of variables. One application of compiling provenance polynomials into their smallest representation is motivated by the following known results on “oblivious bounds” [65]: (i) lower and upper bounds for intractable expressions can be found very efficiently; and (ii) those bounds work better the fewer times variables are repeated. Similarly, anytime approximation schemes based on branch-and-bound provenance decomposition methods [56, 92] give tighter bounds if Shannon expansions need to be run on fewer variables.

7.4 Search Space for minFACT

Before we create a unified algorithm to find minimal factorization for provenance formulas, it is important to reason about and formalize the space of possible factorizations and how we can encode them.

7.4.1 Factorizations and Variable Elimination Orders

Factorizations. In order to find the minimal factorization of a provenance formula, we first define a search space of all permissible factorizations. Each factorized formula can be represented as a *factorization tree* (or FT), where each literal of the formula corresponds to a leaf node,[†] and internal nodes denote the \oplus and \otimes operators of the commutative provenance semiring. The length (size) of a FT is the number of leaves. We allow the semiring operations to be k -ary (thus even unary) and use *prefix notation* for the operators when writing FTs in linearized text. Notice that the

[39]: Dalvi and Suciu (VLDBJ, 2007), ‘Efficient query evaluation on probabilistic databases’. doi:10.1007/s00778-006-0004-3

[127]: Olteanu and Huang (SUM, 2008), ‘Using OBDDs for efficient query evaluation on probabilistic databases’. doi:10.1007/978-3-540-87993-0_26

[39]: Dalvi and Suciu (VLDBJ, 2007), ‘Efficient query evaluation on probabilistic databases’. doi:10.1007/s00778-006-0004-3

[141]: Roy, Perduca, and Tannen (ICDT, 2011), ‘Faster query answering in probabilistic databases using read-once functions’. doi:10.1145/1938551.1938582

[147]: Sen, Deshpande, and Getoor (PVLDB, 2010), ‘Read-Once Functions and Query Evaluation in Probabilistic Databases’. doi:10.14778/1920841.1920975

[65]: Gatterbauer and Suciu (TODS, 2014), ‘Oblivious bounds on the probability of Boolean functions’. doi:10.1145/2532641

[65]: Gatterbauer and Suciu (TODS, 2014), ‘Oblivious bounds on the probability of Boolean functions’. doi:10.1145/2532641

[56]: Fink, Huang, and Olteanu (VLDBJ, 2013), ‘Anytime approximation in probabilistic databases’. doi:10.1007/s00778-013-0310-5

[92]: Heuvel, Ivanov, Gatterbauer, Geerts, and Theobald (SIGMOD, 2019), ‘Anytime Approximation in Probabilistic Databases via Scaled Dissociations’. doi:10.1145/3299869.3319900

[†] A variable may appear in multiple leaves just as it can in a factorized formula.

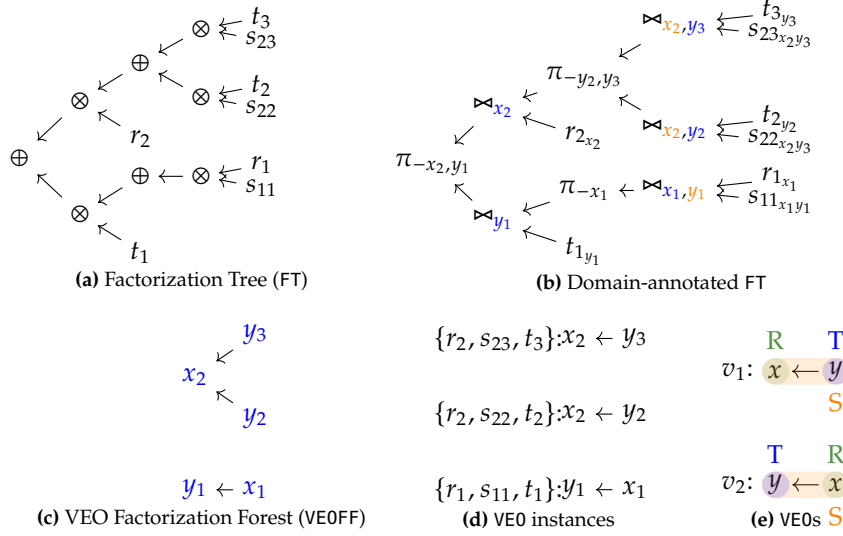


Figure 7.3: Representation of a factorization as a mapping of witnesses to VEOs for an example database under query Q_2^* . [Theorem 7.4.1](#) shows the correspondence of (a) via (b) to (c). [Theorem 7.4.2](#) shows the correspondence of (c) via (d) to (e) for some minimal factorization tree.

space of FTs is strictly larger than the space of factorized expressions: E.g., the FT $\otimes(r_1, s_1, t_1)$ is *not* equivalent to $\otimes(r_1, \otimes(s_1, t_1))$, although they represent the same formula $r_1 s_1 t_1$. We consider FTs as equivalent under commutativity i.e. we treat $\otimes(r_1, s_1, t_1)$ as equivalent to $\otimes(s_1, t_1, r_1)$. Furthermore, w.l.o.g., we only consider trees in which the operators \oplus, \otimes *alternate*: E.g., $\otimes(r_1, \otimes(s_1, t_1))$ is not alternating but represents the same formula as the alternating tree $\otimes(r_1, \oplus(\otimes(s_1, t_1)))$ using unary \oplus . Henceforth, we use factorization trees or FTs as a short form for *alternating factorization trees*.

Variable Elimination Order (VEO). FTs describe tuple-level factorizations, however, they fail to take into account the structure (and resulting join dependencies) of the query producing the provenance. For this purpose, we define query-specific *Variable Elimination Orders* (VEOs). They are similar to VEOs in general reasoning algorithms, such as bucket elimination [45] and VEOs defined in FDBs [129] for the case of no caching (i.e. corresponding to formulas, not circuits). However, our formulation allows each node to have a *set of variables* instead of a single variable (see e.g. [Example 7.4.8](#) and [Figure 7.7](#) in the appendix). This allows VEOs to have a 1-to-1 correspondence to the sequence of variables *projected away* in an “alternating” query plan, in which projections and joins alternate, just as in our FTs (details in [Subsection 7.4.2](#)). Furthermore, we show VEOs can be “annotated” with a data instance and “merged” to form forests that describe a minimal factorization tree of any provenance formula.

Definition 7.4.1 (Variable Elimination Order (VEO)) *A VEO v of a query Q is a rooted tree whose nodes are labeled with non-empty sets of query variables s.t. (i) each variable of Q is assigned to exactly one node of v , and (ii) all variables x for any atom $R(x)$ in Q must occur in the prefix of some node of v .*

Definition 7.4.2 (VEO instance) *Given a VEO v and witness \mathbf{w} , a VEO instance $v(\mathbf{w})$ is the rooted tree resulting from annotating the variables x in v with the domain values of \mathbf{w} .*

[45]: Dechter (AI, 1999), ‘Bucket elimination: A unifying framework for reasoning’. doi:10.1016/S0004-3702(99)00059-4

[129]: Olteanu and Schleich (SIGMOD Rec. 2016), ‘Factorized Databases’. doi:10.1145/3003665.3003667

In order to refer to a VEO in-text, we use a linear notation with parentheses representing sets of children. To make it a unique serialization, we need to assume an ordering on the children of each parent. For notational convenience, we leave out the parentheses for nodes with singleton sets. For example, $x \leftarrow y$, (instead of $\{x\} \leftarrow \{y\}$) and $\{x, y\}$ are two valid VEOs of Q_2^* . We refer to the unique path of a node to the root as its *prefix*.

Example 7.4.1 (VEO and VEO instance) Consider the 3-chain Query $Q_3^\infty := R(x, y), S(y, z), T(z, u)$. An example VEO is $v = z \leftarrow (u, y \leftarrow x)$ (Figure 7.6a). To make it a unique serialization, we need to assume an ordering on the children of each parent. Notice that our definition of VEO also allows sets of variables as nodes. As an extreme example, the legal query plan $P' = \pi_{xyz u} \bowtie (R(x, y), S(y, z), T(z, y))$ corresponds to a VEO v' with one single node containing all variables. In our short notation, we denote nodes with multiple variables in brackets without commas between the variables to distinguish them from children: $v' = \{xyz u\}$.

Now consider a witness $\mathbf{w} = (1, 2, 3, 4)$ for (x, y, z, u) , which we also write as $\mathbf{w} = (x_1, y_2, z_3, u_4)$. The VEO instance of \mathbf{w} for v is then $v(\mathbf{w}) = z_3 \leftarrow (u_4, y_2 \leftarrow x_1)$. Notice our notation for domain values arranged in a tree: In order to make the underlying VEO explicit (and avoiding expressions such as $v(\mathbf{w}) = 3 \leftarrow (4 \leftarrow 2, 1)$ which would become quickly ambiguous) we include the variable names explicitly in the VEO instance. We sometimes refer to them as “domain-annotated variables.”

Definition 7.4.3 (VEO table prefix) *Given an atom R in a query Q and a VEO v , the table prefix v^R is the smallest prefix in v that contains all the variables $\mathbf{x} \in \text{var}(R)$.*

Similarly to $v(\mathbf{w})$ denoting an instance of a given VEO v for a specific witness \mathbf{w} , we also define a *table prefix instance* $v^R(\mathbf{w})$ for a given table prefix v^R and witness \mathbf{w} .

Example 7.4.2 (VEO table prefix and VEO table prefix instance) Consider again the VEO $v = z \leftarrow (u, y \leftarrow x)$ in Example 7.4.1. The table prefix of table $S(y, z)$ on v is $v^S = z \leftarrow y$. Assume a set of two witnesses $W = \{(x_1, y_1, z_1, u_1), (x_1, y_1, z_1, u_2)\}$. Then for both witnesses \mathbf{w}_1 and \mathbf{w}_2 , the table prefix instances for S are identical: $v^S(\mathbf{w}_1) = v^S(\mathbf{w}_2) = z_1 \leftarrow y_1$ (Figure 7.6c).

Definition 7.4.4 (VEO factorization forest (VEOFF)) *A VEOFF \mathcal{V} of provenance φ_p of database \mathcal{D} over query Q is a forest whose nodes are labeled with non-empty sets of domain-annotated variables, such that: (1) For every $\mathbf{w} \in \text{witnesses}(Q, \mathcal{D})$ there exists exactly one subtree in \mathcal{V} that is a VEO instance of \mathbf{w} and Q ; (2) There is no strict sub-forest of \mathcal{V} that fulfills condition (1).*

Example 7.4.3 (VEO Factorization Forest) Continuing with the Q_3^∞ query, and the witnesses $W = \{(x_1, y_1, z_1, u_1), (x_1, y_1, z_1, u_2)\}$ as in Example 7.4.2, we illustrate several valid and invalid VEOFFs. We represent a forest of VEO instances in-text as a set of trees $\{t_1, t_2, \dots\}$.

The forest $\mathcal{V}_1 = \{x_1 \leftarrow (u_1, u_2, (y_1 \leftarrow z_1))\}$ (Figure 7.6d) is a valid VE0FF since (1) for both w_1 and w_2 there is exactly one subtree each in \mathcal{V} that is a VE0 instance. These subtrees are $x_1 \leftarrow (u_1, (y_1 \leftarrow z_1))$ and $x_1 \leftarrow (u_1, (y_1 \leftarrow z_1))$. This VE0FF also satisfies property (2) removing any variable would lead to a VE0FF that does not satisfy property (1).

The forest $\mathcal{V}_2 = \{x_1 \leftarrow (u_1, (y_1 \leftarrow z_1)), y_1 \leftarrow (z_1, (x_1 \leftarrow u_1))\}$ (Figure 7.6e) is also a valid VE0FF since (1) for both w_1 and w_2 there is exactly one subtree each in \mathcal{V} that is a VE0 instance. These subtrees are $x_1 \leftarrow (u_1, (y_1 \leftarrow z_1))$ and $y_1 \leftarrow (z_1, (x_1 \leftarrow u_1))$. This VE0FF also satisfies property (2) removing any variable would lead to a VE0FF that does not satisfy property (1).

The forest $\mathcal{V}_3 = \{x_1 \leftarrow (u_1, (y_1 \leftarrow z_1)), y_1 \leftarrow (z_1, (x_1 \leftarrow u_1 \leftarrow u_2))\}$ (Figure 7.6f) is not a valid VE0FF, although it satisfies property (1) with the same subtrees above. It does not satisfy property (2) since removing u_2 in the second tree would lead to a VE0FF that still satisfies property (1).

Theorem 7.4.1 (Factorizations and VE0s) *There exist transformations from FTs to VE0FFs and back such that the transformations can recover the original FT for at least one minimal size FT φ' of any provenance formula φ_p .*

Proof Intuition. We describe a transformation from FTs to VE0FFs via domain-annotated FTs as intermediate step (Figure 7.3). A domain-annotated FT is constructed as follows: We first replace the \otimes operator with a join (\bowtie) and the \oplus operator with a projection (π) and label the leaves with the domain-annotated variables. We then recursively label each join and projection bottom-up as follows: (1) label each \bowtie by the union of variables of its children, and (2) label each π with the subset of variables of its children that are not required for subsequent joins (this can be inferred from the query). To get the VE0FF instance, we remove all variables on joins that appear in ancestor joins. We remove the leaves and absorb all non-join (projection) nodes into their parents (eliminating the root projection node).

We show that if this transformation succeeds then it is a bijection and can be reversed. The only case when this transformation fails is when it results in an empty annotation for a node, i.e. when there is a join after which no variable is projected away (since by design VE0s do not permit empty nodes). In that case, the FT can always be simplified by removing a \oplus node and merging two \otimes nodes.

Proof Theorem 7.4.1. We first describe in Algorithm 1 a partial mapping from a factorization tree (FT) to a VE0 factorization forest (VE0FF). Note that such a transformation may result in an error if it does not succeed. We show that this process is always reversible if it succeeds. We then show that for at least one minimal FT of any provenance formula φ_p , such a transformation succeeds, and is thus also reversible.

Algorithm 1 returns a structure consistent with the definition of a VE0 factorization forest since (1) for each witness there is a unique VE0 instance in the VE0FF corresponding to sequence of \oplus and \otimes operations that evaluated to the witness in the FT, (2) taking away any variable from the structure would remove at least once VE0 instance (corresponding to a witness of the literal the variable is derived from), hence the VE0FF is minimal in line with the required definition.

If the Algorithm 1 does indeed return a VE0FF, then we can show that this transformation is always reversible by applying the transformation detailed in Algorithm 2. We can prove step-by-step that the Algorithm 2 recovers the same FT as the one that was used to construct the VE0FF. We see that the relationship between \otimes nodes in the factorization trees is never modified in the transformation to the VE0FF, and hence substituting the nodes of the VE0FF with \otimes nodes directly recovers the \otimes relationships. Due to the alternating operators in the factorization trees, we can also recover the \oplus nodes by placing them between \otimes nodes. The leaves can be recovered from the table prefix instances. Since in the transformation to VE0FFs we projected away variables “as soon as possible”, the VE0 table prefix instances can recover losslessly where each tuple was joined in the original factorization tree.

However, consider that Algorithm 1 returns an error for a given FT. Then there exists a join node n in the “reduced” annotated FT built during Algorithm 1 that is not annotated. This happens when all the variables at that join node are used in subsequent joins. Like in Algorithm 1, let \mathcal{D} be the relations of the descendant leaves of n and Q' be the query obtained by removing atoms from Q that correspond to relations in \mathcal{D} . We also define Q'' as the query obtained by keeping only atoms from Q that correspond to relations in \mathcal{D} . Thus, Q is split into Q' and Q'' . If we have an error condition, there is a set of relations \mathbf{R} in Q' that is a superset of $\text{var}(Q'')$. We see that the factorization length cannot increase if we merge the joins of the relations \mathbf{R} with the relations of Q'' . We can now try the transformation again on the new FT with the merged joins. We can repeat this process until we have a FT that does not result in an error. We know that we always make progress since the merging of joins reduces the number of internal nodes of the FT. \square

Example 7.4.4 (Instance where a FT cannot be transformed to a VE0FF) Consider the factorization tree $\otimes(s_{11}, \oplus(\otimes(r_1, t_1)))$ for the Q_2^* query over the simple database with 3 tuples $R(1), S(1, 1), T(1)$.

The join node that is the parent of r_1 and s_1 is annotated with x_1 and y_1 . However, both these variables are also required at the join node with s_{11} and so cannot be projected away. Thus, in the step where we reduce the join annotations in Algorithm 1, both x_1 and y_1 are removed from the deeper join node, leading to an empty annotation. The transformation thus fails.

We can see intuitively that this is due to the fact a join is performed on two relations whose variables are subsumed by the variables of another relation. The joins can thus be reordered without increasing the size of the factorization tree.

7.4.2 Bijection Between VE0s and Query Plans

Intuitively, VE0s capture the sequence of variables *projected away* in a query plan. To make this bijection clear, we define notation around query plans and give an example.

Query plans. A query plan P is given by the grammar $P ::= R_i(\mathbf{x}) \mid \pi_{-\mathbf{x}}P \mid \bowtie(P_1, \dots, P_k)$ where $R_i(\mathbf{x})$ is an atom containing the variables \mathbf{x} , $\pi_{-\mathbf{x}}$ is the *project-away operator* with duplicate elimination (\mathbf{x} here is the set of variables being removed), and $\bowtie(\dots)$ is the *natural join* in prefix notation, which we allow to be k -ary ($k \geq 2$).

Algorithm 1: Transformation of a factorization tree to a VE0 factorization forest

Input: A factorization tree FT over provenance formula φ_p of query Q
Result: A VE0FF or an error

```

/* Build annotated factorization tree: */
1 Replace  $\otimes$  nodes with a join ( $\bowtie$ ) and  $\oplus$  nodes with a projection ( $\pi$ )
/* Annotating the nodes from the leaves to root */
2 while  $\exists$  a node  $n$  that is not annotated do
3   if  $n$  is a leaf node then
4     Annotate with annotated-domain variables of the tuple at the leaf
5   if Join node then
6     Annotate with the union of the annotations of the children of  $n$ 
7   if Projection node then
8      $v$  = union of annotations of the children of  $n$ 
9      $\mathcal{D}$  = set of all relations of descendants of  $n$ 
10     $Q'$  = query obtained by removing atoms from  $Q$  that correspond to
        relations in  $\mathcal{D}$ 
11     $v'$  = The annotated-domain variables in  $v$  that do not correspond to
        variables of  $\text{var}(Q')$  /* These are the variables that can be
        projected away since they do not feature in any subsequent
        joins  $Q'$  */
12    Annotate  $n$  with  $v'$ 
/* Reduce the join annotations: */
13 if  $v$  is a variable that is part of the annotation of a join node  $n$  then
14   if  $\exists n'$  that is an ancestor of  $n$  and contains  $v$  then
15     Remove  $v$  from  $n$ 
16 if A join node has no annotation after minimization then
17   return Error
/* Extract the join annotations: */
18 From the annotated FT, build a forest of join nodes where the parent of a
    join node is its grandparent in the FT (thus skipping over  $\pi$  nodes)
19 Replace join nodes with the annotations obtained after the minimization
    step to obtain a VE0FF
20 return The constructed VE0FF

```

Algorithm 2: Transformation of a factorization tree to a VE0 factorization forest

Input: A VE0FF built from a FT via [Algorithm 1](#)
Result: The original FT

```

/* Re-add literals to the VE0FF */
1 Identify the paths in the VE0FF that correspond to table prefix instances
2 if Node  $n$  is the leaf of a table prefix instance then
3   Add a child to  $n$  corresponding to the tuple represented by the table
    prefix instance
/* Re-add  $\oplus$  and  $\otimes$  nodes */
4 Replace each node in the VE0FF with a  $\otimes$  node
5 Add a  $\oplus$  node as a child for every  $\otimes$  node
6 Replace the parent of each  $\otimes$  node with the  $\oplus$  child of the original parent
7 if there were multiple trees in the VE0FF then
8   Add an  $\oplus$  node as the parent of all the root  $\otimes$  nodes
9 return The reconstructed FT

```

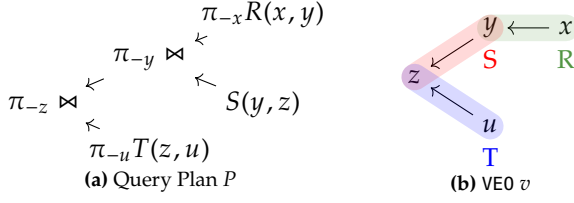


Figure 7.4: Example 7.4.5: A query plan P and the corresponding VE0 v .

We require that joins and projections alternate in every plan. This is w.l.o.g. because nested joins, such as $\bowtie(\bowtie(R, S), T)$ or $\bowtie(R, \bowtie(S, T))$, can be rewritten into $\bowtie(R, S, T)$ while keeping the same provenance by associativity, e.g., $(r_1 s_2) t_3 = r_1 (s_2 t_3)$. For the same reason, we do not distinguish between different permutations in the joins, called join orders [124]. Thus, in contrast to standard query plans, our query plans do not necessarily form binary trees.

An important type of query plan is the *hierarchical plan*. If a query has a hierarchical plan, then this plan can be used to always produce read-once factorized provenance polynomials [127] (and thus the minimum size factorization). Only *hierarchical queries* have hierarchical plans.

Notation Related to Query Plans. We write $\text{var}(P)$ for all variables in a plan P and $\text{HVar}(P)$ for its *head variables*, which are recursively defined as follows: (1) if $P = R_i(\mathbf{x})$, then $\text{HVar}(P) = \mathbf{x}$; (2) if $P = \pi_{\mathbf{x}}(P')$, then $\text{HVar}(P) = \mathbf{x}$; and (3) if $P = \bowtie(P_1, \dots, P_k)$ then $\text{HVar}(P) = \bigcup_{i=1}^k \text{HVar}(P_i)$. The *existential variables* $\text{EVar}(P)$ are then defined as $\text{var}(P) - \text{HVar}(P)$. Every plan P represents a query Q_P defined by taking all atoms mentioned in P as the body and setting $\text{HVar}(Q_P) = \text{HVar}(P)$. A plan is called Boolean if $\text{HVar}(P) = \emptyset$. For notational convenience, we also use the “*project-away operator*” $\pi_{\mathbf{y}}P$ instead of $\pi_{\mathbf{x}}P$, where \mathbf{y} are the variables being projected away, i.e. $\mathbf{x} = \text{HVar}(\pi_{\mathbf{y}}P) = \text{HVar}(P) - \mathbf{y}$.

Legal Query Plans. We assume the usual sanity conditions on plans to be satisfied: for a projection $\pi_{\mathbf{x}}P$ we assume $\mathbf{x} \subseteq \text{HVar}(P)$, and each variable y is projected away at most once in a plan, i.e. there exists at most one such operator $\pi_{\mathbf{x}}$ s.t. $y \in \mathbf{x}$. We also assume that at least one variable is projected away at each projection operator, i.e. for all $\pi_{\mathbf{x}}$ s.t. $\mathbf{x} \neq \emptyset$. We call any such plan *legal*.

Example 7.4.5 (Query Plans and Variable Elimination Orders) Consider the 3-chain Query $Q_3^\infty := R(x, y), S(y, z), T(z, u)$ and the legal query plan $P = \pi_{\mathbf{z}} \bowtie (\pi_{\mathbf{y}} \bowtie (\pi_{\mathbf{x}} R(x, y), S(y, z)), \pi_{\mathbf{u}} T(z, u))$, also shown in Figure 7.4a. The corresponding VE0 v is shown in Figure 7.4b. In order to refer to a particular VE0, we use a linear notation with parentheses representing sets of children i.e. Figure 7.4b can be written as $v = z \leftarrow (u, y \leftarrow x)$. To make it a unique serialization, we only need to assume an ordering on the children of each parent, which we achieve by using the alphabetic ordering on the variables. Notice that our definition of VE0 also allows sets of variables as nodes. As an extreme example, the legal query plan $P' = \pi_{\mathbf{x}yzu} \bowtie (R(x, y), S(y, z), T(z, y))$ corresponds to a VE0 v' with one single node containing all variables. In our short notation, we denote nodes with multiple variables in brackets without commas between the variables in order to distinguish them from children: $v' = \{xyzuz\}$.

Notice that also minimal VE0s can have multiple variable in a node (e.g. $\{yz\} \leftarrow x$ for query Q^Δ).

[124]: Moerkotte (2009), *Building Query Compilers*.

[127]: Olteanu and Huang (SUM, 2008), ‘Using OBDDs for efficient query evaluation on probabilistic databases’. doi:10.1007/978-3-540-87993-0_26

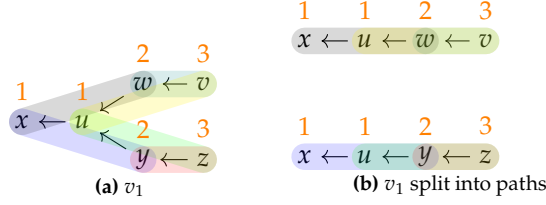


Figure 7.5: The VE0 split operation in [Example 7.4.7](#)

7.4.3 Details and Examples: VE0 Instances, Table Prefixes and Factorization Forests

In this section, we include [Figure 7.6](#), which illustrates the VE0s and VE0FFs mentioned in [Examples 7.4.1 to 7.4.3](#).

We can also define intuitive “merging” and “splitting” operations on VE0s. The merge operation ([Example 7.4.6](#)) takes two VE0 instance forests and combines any trees in the forests if they share path(s) from the root.

Definition 7.4.5 (VE0 Merge) A merge operation on one or more VE0 instances or VE0 instance forests \mathcal{V}_1 and \mathcal{V}_2 outputs a new VE0 instance forest \mathcal{V} such that: (1) All VE0 instances in \mathcal{V}_1 and \mathcal{V}_2 are also present in \mathcal{V} . (2) There are no two trees $t_1, t_2 \in \mathcal{V}$ such that t_1 and t_2 share a prefix for any variable that is in t_1 and t_2 .

Example 7.4.6 (Merging VE0 instances) Consider the two VE0 instances $v_{w_1} = x_1 \leftarrow y_1 \leftarrow z_1$ and $v_{w_2} = x_1 \leftarrow y_1 \leftarrow z_2$. Then merging v_{w_1} and v_{w_2} would result in the instance $x_1 \leftarrow y_1 \leftarrow (z_1, z_2)$, since the common rooted path $x_1 \leftarrow y_1$ in both VE0s can be combined.

On the other hand, $v_{w_3} = x_1 \leftarrow y_1 \leftarrow z_1$ and $v_{w_4} = x_2 \leftarrow y_1 \leftarrow z_1$ have no common rooted path, and their merger results in simply the forest $\{x_1 \leftarrow y_1 \leftarrow z_1, x_2 \leftarrow y_1 \leftarrow z_1\}$.

Another example is merging the VE0 instances $v_{w_5} = y_1 \leftarrow (x_1, z_1)$ and $v_{w_6} = y_1 \leftarrow (x_2, z_2)$ results in combining the shared path $y_1 \leftarrow ((x_1, x_2), (z_1, z_2))$. Notice that this includes the VE0 instances $y_1 \leftarrow (x_2, z_1)$ and $y_1 \leftarrow (x_1, z_2)$, which must be present in the data anyway due to join dependencies.

We can also merge VE0 instances from different VE0s. The merger of $v_{w_7} = x_1 \leftarrow y_1 \leftarrow z_1$ and $v_{w_8} = x_1 \leftarrow z_2 \leftarrow y_2$ leads to $x_1 \leftarrow (y_1 \leftarrow z_1, z_2 \leftarrow y_2)$.

The splitting operation ([Example 7.4.7](#) and [Figure 7.5](#)) splits a VE0 with more than one leaf into nested paths from the root to the leaves. This property is useful to define nested prefix orderings in [Subsection 7.6.1](#).

Definition 7.4.6 (VE0 Split) A split operation on a VE0 instance outputs the set of root-to-leaf paths in the VE0 instance.

Example 7.4.7 (Splitting VE0 instances) The VE0 $x_1 \leftarrow (y_1, z_1)$ can be split into nested paths $x_1 \leftarrow y_1$ and $x_1 \leftarrow z_1$.

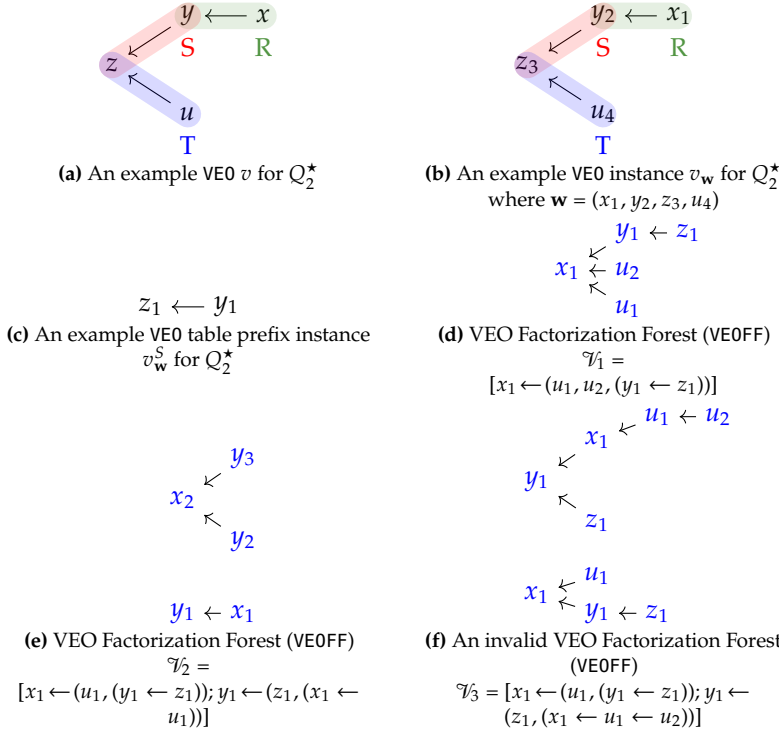


Figure 7.6: Examples 7.4.1 to 7.4.3: Examples of VE0 instances, VE0 table prefix instances and VE0 factorization forests

7.4.4 Connections of VE0s to Related Work

Connection to FBDs. Our definition of VE0s parallels the definition of variable orders defined in Factorized Databases [129, 132], however has two key differences: (i) we do not allow any caching in the variable order, and (ii) we allow multiple variables in each node of the VE0.

Thus, we allow the $\{yz\} \leftarrow x$, while an FDB style definition would force every factorization to choose between $y \leftarrow z \leftarrow x$ and $z \leftarrow y \leftarrow x$.

We see that using multiple variables in each node of the VE0 allows us to build a more restricted search space, since some VE0s with multiple variables in the nodes are minimal VE0s (defined through notions of minimal query plans [65], later described in Subsection 7.4.6). An example of a query with such a minimal VE0 is the triangle unary query Q_A^Δ (Example 7.4.8).

Connection to Knowledge Representation. We can draw a parallel between VE0s and VE0 instances as the notion of v-trees and DNNF formulas in knowledge compilation [133], as the v-trees and VE0s both capture the *structure* of the data in the VE0 instance or DNNF, respectively.

Theorem 7.4.2 (minFACT with VE0s) *There exists a transformation that constructs FTs of a provenance φ_p from mappings of each witness of φ_p to a VE0 of Q , and there exists a mapping that is transformed into a minimal size factorization tree φ' of φ_p under this transformation.*

Proof Intuition. From Theorem 7.4.1, we know that for every provenance formula φ_p there exists a minimal size FT that has a reversible transformation to a VEOFF. We show all such VEOFFs can be constructed by assigning a VE0 to each witness of φ_p . This is constructed by defining a merge operation on VE0s that greedily merges common prefixes.

[129]: Olteanu and Schleich (SIGMOD Rec. 2016), ‘Factorized Databases’. doi:10.1145/3003665.3003667
 [132]: Olteanu and Závodný (TODS, 2015), ‘Size Bounds for Factorised Representations of Query Results’. doi:10.1145/2656335

[65]: Gatterbauer and Suciu (TODS, 2014), ‘Oblivious bounds on the probability of Boolean functions’. doi:10.1145/2532641

[133]: Pipatsrisawat and Darwiche (AAAI, 2008), ‘New Compilation Languages Based on Structured Decomposability’.

Proof Theorem 7.4.2. Through Theorem 7.4.1, we know that for any provenance formula φ_p there exists a minimal size FT that has a correspondence to a VE0FF. We show all such VE0FFs can be constructed by assigning a VE0 to each witness of φ_p . From the assignment of VE0s to each witness, we are able to construct the corresponding VE0 instances. We use the merge operation (Definition 7.4.5) on VE0 instances greedily to merge common prefixes. The merge operation is both commutative and associative, and no matter the order of merging, all $|\mathbf{w}|$ VE0 instances will produce exactly the same forest after merging. This resulting forest is the VE0FF corresponding to the minimal FT, since (1) it contains a VE0 instance for each witness of φ_p (2) it cannot be made smaller since the merge operation maximally merges all shared prefixes. Since we can build a VE0FF corresponding to a minimal formula of φ_p from the assignment of VE0s to witnesses, we can also FT corresponding to a minimal formula of φ_p through the reverse transformation Algorithm 2. \square

Minimal Variable Elimination Orders (mveo). By reducing the problem of finding the minimal factorization to that of assigning a VE0 to each witness, we have so far shown that FACT is in NP with respect to data complexity.¹ However, we can obtain a more practically efficient result by showing that we need not consider all VE0s, but only the *Minimal Variable Elimination Orders* of a query or $\text{mveo}(Q)$. We can define a partial order \leq on VE0s of a query Q as follows: $v_1 \leq v_2$ if for every relation $R_i \in Q$ the variables in the R_i table prefix of v_1 are a subset of the variables of the R_i table prefix of v_2 , i.e. $\forall R_i \in Q : \text{var}(v_1^{R_i}) \subseteq \text{var}(v_2^{R_i})$. $\text{mveo}(Q)$ then is the set of all VE0s of Q that are minimal with respect to this partial order \leq . For Q_2^* , there are only two minimal variable elimination orders $x \leftarrow y$ and $y \leftarrow x$, but not $\{x, y\}$, and for Q_3^* , there are only 6, despite 13 possible VE0s in total. Interestingly, $\text{mveo}(Q)$ corresponds exactly to Minimal Query Plans as defined in work on probabilistic databases [66], and we can use this connection to leverage prior algorithms for computing $\text{mveo}(Q)$.

1: This follows from the fact that the number of witnesses is polynomial in the size of the database, and the number of VE0s only depends on the query size.

Theorem 7.4.3 (minFACT with mveos) *There exists a transformation that constructs FTs of a provenance φ_p from mappings of each witness of φ_p to a VE0 $v \in \text{mveo}(Q)$, and there exists a mapping that is transformed into a minimal size factorization tree φ' of φ_p under this transformation.*

Proof Theorem 7.4.3. We know from Theorem 7.4.2 that a minimal factorization tree can be constructed by assigning VE0s to each witness. It now remains to be shown that using non-minimal VE0s (corresponding to non-minimal query plans) cannot make the factorization smaller than if just minimal VE0s are used. Assume that the factorization uses a non-minimal query plan P' , which corresponds to the dissociation Δ' . Since it is not minimal there must exist a dissociation Δ where $\Delta \leq \Delta'$. This means that the provenance (of \mathcal{D} , or simply a subset of witnesses) calculated with P' is a dissociation of the provenance calculated with P , which means that each variable occurs at least as many times in $\varphi(P^{\Delta'})$ as in $\varphi(P^{\Delta})$ [65]. \square

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

[65]: Gatterbauer and Suciu (TODS, 2014), ‘Oblivious bounds on the probability of Boolean functions’. doi:10.1145/2532641

7.4.5 End to End Example: Query Plans, VE0s and Factorizations

Example 7.4.8 gives an end-to-end example that shows a query, its minimal hierarchical dissociations [66], the corresponding minimal query

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

plans, the VE0s, and the factorizations that each VE0 corresponds to. This example is also interesting since it shows minimal VE0s that have *multiple variables in a node*; this is not permitted in prior definitions of VE0s such as [45, 129].

Example 7.4.8 (Triangle unary Q_A^Δ) Consider the triangle unary query $Q_A^\Delta := U(x), R(x, y), S(y, z), T(z, x)$. Notice that this query is not hierarchical: $\text{at}(y) = \{R, S\}$ and $\text{at}(z) = \{S, T\}$, which is not allowed in a hierarchical query. Thus, it does not have a hierarchical plan, and we know that a database instance under this query is not guaranteed to have a read-once factorization.

Thus, to obtain the minimal factorization, we consider all its minimal hierarchical dissociations [66], which correspond to the minimal query plans and minimal VE0s. This query has 3 such minimal dissociations, shown in Figure 7.7 as incidence matrices, query plans, mveos, and factorizations. The objective is now to assign each witness to exactly one of these 3 minimal dissociations. Here the incidence matrix is a simple visual representation of the variables, atoms, and variables contained in each atom. Recall the intuition for dissociations of adding variables to atoms.

Notice that $v_1 = \{y, z\} \leftarrow x$ would not be permitted in an FDB-style variable ordering, which requires a single variable at each level of the tree [129, Definition 3.2]. Thus it would require v_1 to be split into 2 different query plans $y \leftarrow z \leftarrow x$ and $z \leftarrow y \leftarrow x$. Our approach takes into account concurrent variable elimination and hence can reduce the search space for minimal factorizations in this case.

[45]: Dechter (AI, 1999), ‘Bucket elimination: A unifying framework for reasoning’. doi:10.1016/S0004-3702(99)00059-4

[129]: Olteanu and Schleich (SIGMOD Rec. 2016), ‘Factorized Databases’. doi:10.1145/3003665.3003667

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

[129]: Olteanu and Schleich (SIGMOD Rec. 2016), ‘Factorized Databases’. doi:10.1145/3003665.3003667

7.4.6 Details about Minimal VE0s

We show a connection between minimal VE0s and minimal query plans from [65]. To do this, we leverage a correspondence of VE0s to query plans and the previously defined notion of dissociations from probabilistic databases [65].

Intuitively, the goal of dissociations in probabilistic inference [65] is to obtain better inference bounds, by converting it to a hierarchical query, for which inference is easy. Our purpose in using dissociations is in the same spirit - we know that hierarchical queries have read-once factorizations, and hence we would like to use dissociations to obtain a dissociated database on which we can obtain the original provenance by using the dissociated query.

We here recap the key notions of dissociations from [65] and provide examples. But first, we add some more detail on provenance computation from query plans and hierarchical query plans.

Provenance computation from Query Plans. We slightly adapt our previous definition of query plans to “provenance query plans.” Each sub-plan P now returns an intermediate relation of arity $|\text{HVar}(P)|+1$. The extra *provenance attribute* stores an expression φ for each output tuple $t \in P(D)$ returned from a plan P .

Given a database \mathcal{D} and a plan P , φ is defined inductively on the structure of P as follows: (1) If $t \in R_i(x)$, then $\varphi(t) = \varphi_t$, i.e. the provenance token of t in \mathcal{D} ; (2) if $t \in \bowtie(P_1(D), \dots, P_k(D))$ where $t = \bowtie(t_1, \dots, t_k)$, then $\varphi(t) = \bigwedge_{i=1}^k \varphi(t_i)$; and (3) if $t \in \pi_{\mathbf{x}} P(D)$, and $t_1, \dots, t_n \in P(D)$ are all the

[65]: Gatterbauer and Suciu (TODS, 2014), ‘Oblivious bounds on the probability of Boolean functions’. doi:10.1145/2532641

tuples that project onto t , then $\varphi(t) = (\bigvee_{i=1}^n \varphi(t_i))$ (notice the required outer parentheses). For a Boolean plan P , we get one single expression, which we denote $\varphi(P, D)$.

Hierarchical Query Plans. An important type of plan is the *hierarchical plans*. Intuitively, if a query has a hierarchical plan, then this plan can be used to always produce read-once factorized provenance polynomial [127] (and thus the minimum size factorization). The only queries that have hierarchical plans are the *hierarchical queries*.

[127]: Olteanu and Huang (SUM, 2008), ‘Using OBDDs for efficient query evaluation on probabilistic databases’. doi:10.1007/978-3-540-87993-0_26

Definition 7.4.7 (Hierarchical query [39]) *A query Q is called hierarchical iff for any two existential variables x, y , one of the following three conditions holds: $\text{at}(x) \subseteq \text{at}(y)$, $\text{at}(x) \supseteq \text{at}(y)$, or $\text{at}(x) \cap \text{at}(y) = \emptyset$ where $\text{at}(x)$ is the set of atoms of Q in which x participates.*

Definition 7.4.8 (Hierarchical plan) *A plan P is called hierarchical iff, for each join $\bowtie (P_1, \dots, P_k)$, the head variables of each sub-plan P_i contain the same existential variables of plan P : $\text{HVar}(P_i) \cap \text{EVar}(P) = \text{HVar}(P_j) \cap \text{EVar}(P), \forall i, j \in [k]$*

Example 7.4.9 (Hierarchical Queries) The 2-chain query $Q_2^\infty := R(x, y), S(y, z)$ is a hierarchical query since for each pair of variables a, b either $\text{at}(a)$ and $\text{at}(b)$ are disjoint (like the variables x and z , since $\text{at}(x) = \{R\}$ and $\text{at}(z) = \{S\}$) or have a subset relationship (like $\text{at}(x) = \{R\}$ and $\text{at}(y) = \{R, S\}$).

However, the 3-chain query $Q_3^\infty := R(x, y), S(y, z), T(z, u)$ is not hierarchical since $\text{at}(y) = \{R, S\}$ and $\text{at}(z) = \{S, T\}$ and these are overlapping sets with neither being a subset of the other.

Example 7.4.10 (Hierarchical Plan) Consider two query plans for the query Q_2^∞ in Figure 7.8.

The query plan P_2 is *not* a hierarchical plan since for the sub-plan $\bowtie (R(x, y), S(y, z))$ the head variables for $R(x, y)$ are $\{x, y\}$ and for $S(y, z)$ they are $\{y, z\}$. Thus, they contain different sets of existential variables of P_2 .

However, in P_1 , at the only join $\bowtie (\pi_{-x}R(x, y), \pi_{-z}S(y, z))$, the head variables of both subplans $\pi_{-x}R(x, y)$ and $\pi_{-z}S(y, z)$ are the same set $\{y\}$, so the plan is hierarchical.

Query Dissociation [66, Section 3.1]. We leverage the previously defined idea of query dissociation to generate query plans that “come as close as possible” to hierarchical plans, to help find the minimal factorization for queries that are not read-once.

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

A query dissociation is a rewriting of both the data and the query by adding variables to atoms in the query. In this paper, we care only about hierarchical dissociations, i.e. dissociations s.t. the rewritten dissociated query plan is hierarchical. Dissociations can be compared with a partial order, and for our problem, we only care about minimal hierarchical dissociations and their corresponding plans, known as *minimal query plans*. There exists an algorithm that can find all such minimal query plans [66], and we assume them as input.

Definition 7.4.9 (Query dissociation) *Given a query $Q(\mathbf{z}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$. Let $\Delta = (\mathbf{y}_1, \dots, \mathbf{y}_m)$ be a collection of sets of variables with $\mathbf{y}_i \subseteq \text{EVar}(Q) - \mathbf{x}_i$ for every relation R_i . The “dissociated query” defined by Δ is then $Q^\Delta(\mathbf{z}) :- R_1^{\mathbf{y}_1}(\mathbf{x}_1, \mathbf{y}_1), \dots, R_m^{\mathbf{y}_m}(\mathbf{x}_m, \mathbf{y}_m)$ where each $R_i^{\mathbf{y}_i}(\mathbf{x}_i, \mathbf{y}_i)$ is a new relation of arity $|\mathbf{x}_i| + |\mathbf{y}_i|$.*

Example 7.4.11 (Query Dissociation) Consider the non-hierarchical query of $Q_3^\infty :- R(x, y), S(y, z), T(z, u)$ again. Consider the dissociation $\Delta = (\{z\}, \{\}, \{\})$. Then the dissociated query is $Q_3^{\star\Delta} :- R(x, y, z), S(y, z), T(z, u)$.

Intuitively, the goal of dissociations in probabilistic inference [66] is to obtain better inference bounds, by converting it to a hierarchical query, for which inference is easy. Our purpose in using dissociations is in the same spirit - we know that hierarchical queries have read-once factorizations, and hence we would like to use dissociations to obtain a dissociated database on which we can obtain the original provenance by using the dissociated query.

Hierarchical Dissociation.

Definition 7.4.10 (Hierarchical dissociation) *A dissociation Δ of a query Q is called hierarchical if the dissociated query Q^Δ is hierarchical. We denote the corresponding hierarchical plan applied over the original relations P^Δ .*

For every sf-free CQ, there is an isomorphism between the set of query plans and the set of hierarchical dissociations [66]. Thus we can restrict the dissociations we consider to hierarchical dissociations.

Example 7.4.12 (Hierarchical Dissociation) Consider again the non-hierarchical query of Q_3^∞ and its dissociation $\Delta = (\{z\}, \{\}, \{\})$ from [Example 7.4.11](#). Then the dissociated query $Q_3^{\star\Delta} :- R(x, y, z), S(y, z), T(z, u)$ is hierarchical as all pairs of variables satisfy the constraint for hierarchical queries. Thus Δ is a hierarchical dissociation.

Partial Dissociation Orders. The number of dissociations to be considered can be further reduced by exploiting an ordering on the dissociations.

Definition 7.4.11 (Partial dissociation order) *Given two dissociations $\Delta = (\mathbf{y}_1, \dots, \mathbf{y}_m)$ and $\Delta' = (\mathbf{y}'_1, \dots, \mathbf{y}'_m)$. We define the partial order on the dissociations of a query as: $\Delta \leq \Delta' \Leftrightarrow \forall i : \mathbf{y}_i \subseteq \mathbf{y}'_i$.*

A minimal hierarchical dissociation is one such that no smaller dissociation is hierarchical.

Example 7.4.13 (Partial Dissociation Order) For Q_3^∞ , the dissociation $\Delta = (\{z\}, \{\}, \{\})$ is a minimal hierarchical dissociation while $\Delta' = (\{z\}, \{\}, \{y\})$ is hierarchical but not minimal.

Additionally, we can say that $\Delta \leq \Delta'$.

Equivalence of $\text{mveo}(Q)$ and Minimal Query Plans. In [Section 7.4](#), we defined minimal VE0s based on a partial order \leq defined via the size of table prefixes. We show that this partial order corresponds to the partial

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’.
doi:10.1007/s00778-016-0434-5

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’.
doi:10.1007/s00778-016-0434-5

dissociation order, and hence we can obtain a direct correspondence between $\text{mveo}(Q)$ and Minimal Query Plans (due to the fact that all VEOs correspond to query plans through the bijection explained in [Subsection 7.4.2](#)). The variables of table-prefix of R always contain the variables of R , plus additional variables that are used in joins with R . If these “additional” variables of each table-prefix are added to the respective relations, then the resulting query is hierarchical since every atom already contains all the variables that are used in joins with them. In other words, the additional variables in the table-prefixes correspond exactly to a hierarchical dissociation of the query. We can see then that the partial order on VEOs is exactly the same as the partial dissociation order since both look at the variables in the table-prefixes / dissociations.

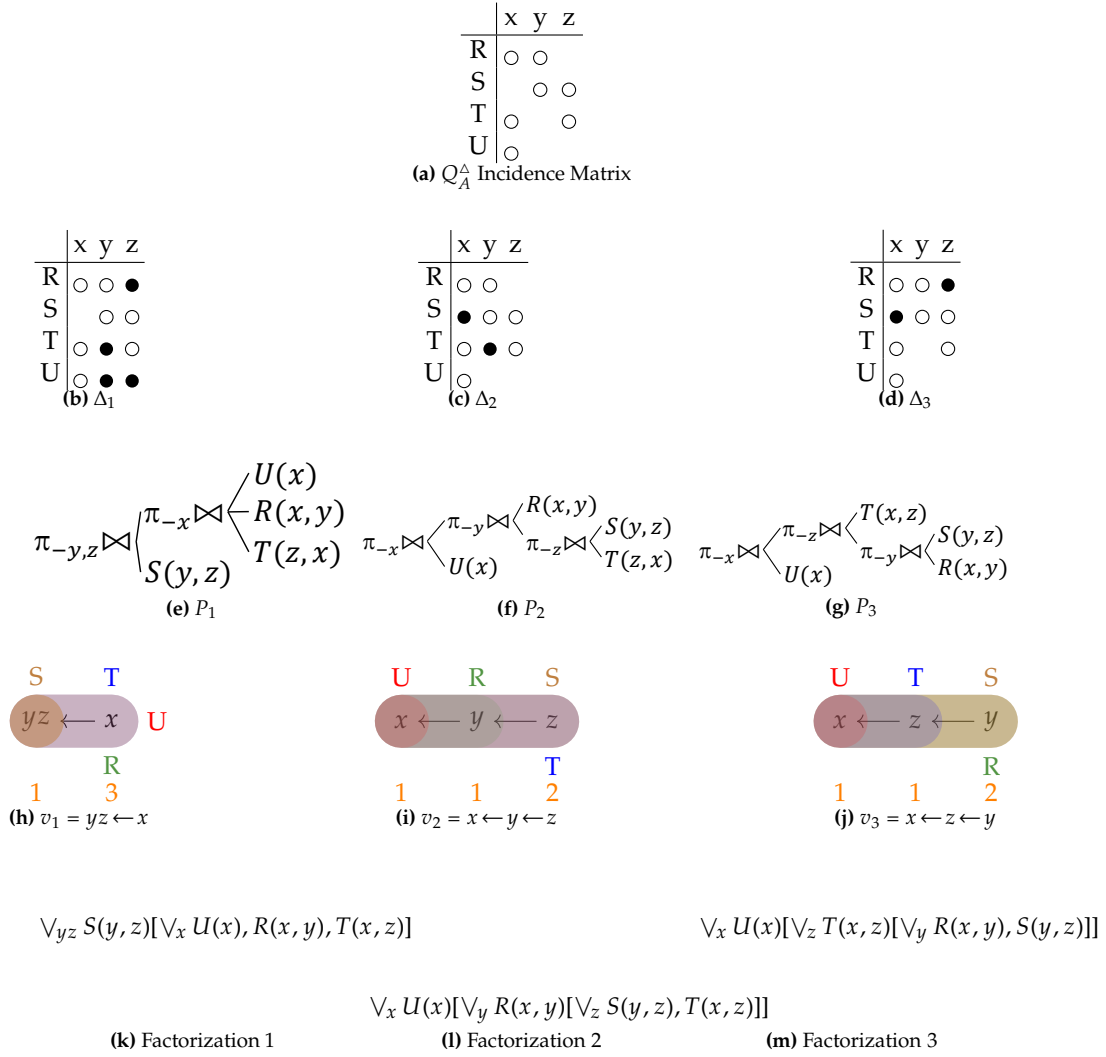


Figure 7.7: Example 7.4.8: For the triangle unary query $Q_A^\Delta := U(x), R(x, y), S(y, z), T(z, x)$, we show the incidence graph matrix (a), the three minimal dissociations (b)-(d), corresponding three minimal query plans (e)-(g), three minimal VE0s (h)-(j), and factorizations (k)-(m).

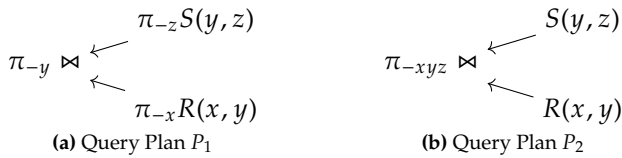


Figure 7.8: Example 7.4.10: Two legal query plans for Q_2^∞ . P_1 is a hierarchical minimal plan while P_2 is not.

7.5 ILP Formulation for minFACT

Given a set of witnesses $W = \text{witnesses}(Q, D)$ for a query Q over some database \mathcal{D} , we can use the insight of [Theorem 7.4.3](#) to describe a 0-1 Integer Linear Program (ILP) $\text{ILP}[\text{minFACT}]$ (7.2) that chooses a $v \in \text{mveo}$ or equivalently a minimal query plan, for each $\mathbf{w} \in W$, s.t. the resulting factorization is of minimal size. The size of the ILP is polynomial in $n = |D|$ and exponential in the query size.

ILP Decision Variables. The ILP is based on two sets of binary variables: Query Plan Variables (QPV) q use a one-hot encoding for the choice of a minimal VE0 (or equivalently minimal query plan) for each witness, and Prefix Variables (PV) p encode sub-factorizations that are a consequence of that choice. Intuitively, shared prefixes encode shared computation through factorization.

(1) *Query Plan Variables (QPV):* For each witness $\mathbf{w} \in W$ and each minimal VE0 $v \in \text{mveo}(Q)$ we define a binary variable $q[v\langle\mathbf{w}\rangle]$, which is set to 1 iff VE0 v is chosen for witness \mathbf{w} .[‡]

(2) *Prefix Variables (PV):* All witnesses must be linked to a set of prefix variables, by creating instances of VE0 prefixes that are in a query-specific set called the *Prefix Variable Format (PVF)*. This set PVF is composed of all table prefixes of all minimal VE0s $v \in \text{mveo}(Q)$. Notice that prefix variables can be shared by multiple witnesses, which captures the idea of joint factorization. Additionally, we define a *weight* (or *cost*)[§] $c(v^R)$ for each table prefix $v^R \in PVF$; this weight is equal to the number of tables that have the same table prefix for a given VE0. From that PVF set, then binary prefix variables $p[v^R\langle\mathbf{w}\rangle]$ are defined for each table prefix $v^R \in PVF$ and $\mathbf{w} \in W$.

ILP Objective. The ILP should minimize the length of factorization len , which can be calculated by counting the number of times each tuple is written. If a tuple is a part of multiple witnesses, it may be repeated in the factorization. However, if the tuple has *the same table prefix instance across different witnesses* (whether as part of the same VE0 or not), then those occurrences are factorized together and the tuple is written just once in the factorization. Thus, len is the weighted sum of all selected table prefix instances. The weight accounts for tuples of different tables that have the same table prefix under the same VE0. Since $p[v^R\langle\mathbf{w}\rangle] = 0$ for unselected table prefixes, we can calculate len as:

$$\text{len} = \sum_{v^R\langle\mathbf{w}\rangle \in PV} c(v^R) \cdot p[v^R\langle\mathbf{w}\rangle] \quad (7.1)$$

$$\begin{aligned} \min \quad & \sum_{v^R\langle\mathbf{w}\rangle \in PV} c(v^R) \cdot p[v^R\langle\mathbf{w}\rangle] \\ \text{s.t.} \quad & \sum_{v \in \text{mveo}(Q)} q[v\langle\mathbf{w}\rangle] \geq 1, & \forall \mathbf{w} \in W \\ & p[v^R\langle\mathbf{w}\rangle] \geq \sum_{v^R\langle\mathbf{w}\rangle \text{ prefix of } v\langle\mathbf{w}\rangle} q[v\langle\mathbf{w}\rangle], & \forall p[v^R\langle\mathbf{w}\rangle] \in PV \\ & p[v^R\langle\mathbf{w}\rangle] \in \{0, 1\}, & \forall p[v^R\langle\mathbf{w}\rangle] \in PV \\ & q[v\langle\mathbf{w}\rangle] \in \{0, 1\}, & \forall q[v\langle\mathbf{w}\rangle] \in QPV \end{aligned} \quad (7.2)$$

Figure 7.9: ILP Formulation for minFACT

[‡] Notice that we use indexing in brackets $q[v\langle\mathbf{w}\rangle]$ instead of the more common subscript notation $q_{v\langle\mathbf{w}\rangle}$ since each $v\langle\mathbf{w}\rangle$ can depict a tree. Our bracket notation is more convenient.

[§] We write c for weight (or cost) to avoid confusion with witnesses \mathbf{w} .

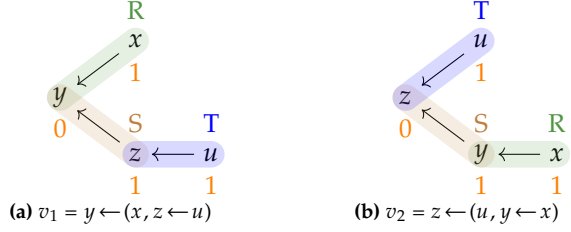


Figure 7.10: Example 7.5.1: mveo for 3-chain query Q_3^∞ .

ILP Constraints. A valid factorization of W must satisfy three types of constraints:

- (1) *Query Plan Constraints:* For every witness $\mathbf{w} \in W$, some $v \in \text{mveo}(Q)$ must be selected.[¶] For example, for $\mathbf{w} = (x_1, y_1)$ under Q_2^\star , we enforce that: $q[x_1 \leftarrow y_1] + q[y_1 \leftarrow x_1] \geq 1$.
- (2) *Prefix Constraints:* For any given table prefix p , it must be selected if any one of the VE0s that has it as a prefix is selected. Since (under a minimization optimization) only one VE0 is chosen per witness, we can say that the value of $p[v^R(\mathbf{w})]$ must be at least as much as the sum of all query plan variables $q[v(\mathbf{w})]$ such that $v^R(\mathbf{w})$ is a prefix of $v(\mathbf{w})$. For example, we enforce that $p[x_1]$ must have value at least as much as $q[x_1 \leftarrow y_1 \leftarrow z_1] + q[x_1 \leftarrow z_1 \leftarrow y_1]$. But we cannot enforce $p[x_1] \geq q[x_1 \leftarrow y_1 \leftarrow z_1] + q[x_1 \leftarrow z_1 \leftarrow y_2]$ (as both VE0s do not belong to the same witness.)
- (3) *Boolean Integer Constraints:* Since a VE0 is either selected or unselected, we set all variables in PV and QPV to 0 or 1.

Theorem 7.5.1 (ILP[minFACT] correctness) *The objective of ILP[minFACT] for a query Q and database \mathcal{D} is always equal to $\text{minFACT}(Q, D)$.*

Corollary 7.5.2 (FACT is in NP) *FACT, the decision variant of minFACT, is in NP.*

Size of ILP. For query Q with m relations and $|\text{mveo}(Q)| = k$, and a set of n witnesses under Q : the resulting ILP (without any optimizations) has $n(1 + km)$ constraints. Thus the size of the ILP is linear in the number of witnesses.

7.5.1 Example minFACT ILPs

Example 7.5.1 (ILP formulation for 3-chain query) Consider the 3-Chain query $Q_3^\infty := R(x, y), S(y, z), T(z, u)$ with a set of 2 witnesses $W = \{(x_1, y_1, z_1, u_1), (x_1, y_1, z_1, u_2)\}$ and provenance in DNF of $r_{11}s_{11}t_{11} + r_{11}s_{11}t_{12}$. Using the dissociation based algorithm [66], we see that this query has 2 minimal query plans corresponding to the two VE0s shown in Figure 7.10. We use these VE0s to first build the set QPV (Query Plan Variables) and enforce a query plan constraint for

[66]: Gatterbauer and Suciu (VLDBJ, 2017), ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. doi:10.1007/s00778-016-0434-5

[¶] We wish to have exactly one query plan or minimal VE0 per witness, but in a minimization problem, it suffices to say that at least one $v \in \text{mveo}$ is selected - if multiple are selected, either one of them arbitrarily still fulfills all constraints.

each of the 2 witnesses:

$$\begin{aligned} q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] + q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] &\geq 1 \\ q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] + q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] &\geq 1 \end{aligned}$$

Then, we calculate the elements of the set *PVF* (Prefix Variable Format) as well as their weights. For the two VEOs from Figure 7.10, and the three tables *R*, *S*, *T*, we get 6 distinct table prefixes:

VEO v_1	VEO v_2
$v_1^R = y \leftarrow x$	$v_2^R = z \leftarrow y \leftarrow x$
$v_1^S = y \leftarrow z$	$v_2^S = z \leftarrow y$
$v_1^T = y \leftarrow z \leftarrow u$	$v_2^T = z \leftarrow u$

We add all these table-prefixes to the *PVF*. Since no table prefix is repeated, they all are assigned weight $c = 1$. Notice that prefixes y for v_1 and z for v_2 are no table prefixes (and thus have weight $c = 0$ and do not participate in the objective).

From the set of table prefixes *PVF*, we then create the set of prefix variables *PV*, one for each table prefix and each witness $\mathbf{w} \in W$, and define their prefix constraints. The prefix constraints necessary for witness $\mathbf{w}_1 = (x_1, y_1, z_1, u_1)$ are as follows:

$$\begin{aligned} p[y_1 \leftarrow x_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] & p[y_1 \leftarrow z_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] \\ p[y_1 \leftarrow z_1 \leftarrow u_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] & p[z_1 \leftarrow y_1 \leftarrow x_1] &\geq q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] \\ p[z_1 \leftarrow y_1] &\geq q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] & p[z_1 \leftarrow u_1] &\geq q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] \end{aligned}$$

The prefix constraints for witness $\mathbf{w}_2 = (x_1, y_1, z_1, u_2)$ are:

$$\begin{aligned} p[y_1 \leftarrow x_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] & p[y_1 \leftarrow z_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] \\ p[y_1 \leftarrow z_1 \leftarrow u_2] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] & p[z_1 \leftarrow y_1 \leftarrow x_1] &\geq q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] \\ p[z_1 \leftarrow y_1] &\geq q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] & p[z_1 \leftarrow u_2] &\geq q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] \end{aligned}$$

Notice that we have 12 constraints (one for each pair of witness and table prefix), yet only 8 distinct prefix variables due to common prefixes across the two witnesses (which intuitively enables shorter factorizations). For this query, for every witness, there are 6 prefix variables in the objective (some of which are used by multiple witnesses), 1 Query Plan constraint, and 6 Prefix constraints.

Finally, we define the objective to minimize the weighted sum of all 8 prefix variables in *PV* (here all weights are 1):

$$\text{len} = p(y_1 \leftarrow x_1) + p[y_1 \leftarrow z_1] + p[y_1 \leftarrow z_1 \leftarrow u_1] + p[z_1 \leftarrow y_1 \leftarrow x_1] + p[z_1 \leftarrow y_1] + p[z_1 \leftarrow u_1] + p[y_1 \leftarrow z_1 \leftarrow u_2] + p[z_1 \leftarrow u_2]$$

In our given database instance, *len* has an optimal value of 4 when the prefixes $p[z_1 \leftarrow y_1 \leftarrow x_1]$, $p[z_1 \leftarrow y_1]$, $p[z_1 \leftarrow u_1]$ and $p[z_1 \leftarrow u_2]$ are set to 1. This corresponds to the minimal factorization $r_{11}s_{11}(t_{11} + t_{12})$.

This next example highlights some optimizations to the ILP.

Example 7.5.2 (ILP formulation for 3-Star query) Consider the 3-Star query $Q_3^* := R(x), S(y), T(z), W(x, y, z)$. This query has 6 minimal

query plans corresponding to the 6 VEOs shown in Figure 7.11. We use these VEOs to build the set QPV .

Note that since all query plans are linear (they all form paths), they can alternatively be specified by the first two variables in the VEO (the last one is implied). We use this shorter notation as it also simplifies constraints later. Hence, the query plan constraint for an example witness $\mathbf{w}_1 = (x_1, y_1, z_1)$ would be:

$$q[x_1 \leftarrow y_1] + q[x_1 \leftarrow z_1] + q[y_1 \leftarrow x_1] \\ + q[y_1 \leftarrow z_1] + q[z_1 \leftarrow x_1] + q[z_1 \leftarrow y_1] \geq 1$$

Then, we calculate the table-prefixes needed to build set PVF , illustrated here for all 6 VEOs:

VEO v_1	VEO v_2	VEO v_3
$v_1^R = x$	$v_2^R = x$	$v_3^R = x \leftarrow y$
$v_1^S = x \leftarrow y$	$v_2^S = x \leftarrow z \leftarrow y$	$v_3^S = y$
$v_1^T = x \leftarrow y \leftarrow z$	$v_2^T = x \leftarrow z$	$v_3^T = y \leftarrow x \leftarrow z$
$v_1^W = x \leftarrow y \leftarrow z$	$v_2^W = x \leftarrow z \leftarrow y$	$v_3^W = y \leftarrow x \leftarrow z$

VEO v_4	VEO v_5	VEO v_6
$v_4^R = y \leftarrow z \leftarrow x$	$v_5^R = z \leftarrow x$	$v_6^R = z \leftarrow y \leftarrow x$
$v_4^S = y$	$v_5^S = z \leftarrow x \leftarrow y$	$v_6^S = z \leftarrow y$
$v_4^T = y \leftarrow z$	$v_5^T = z$	$v_6^T = z$
$v_4^W = y \leftarrow z \leftarrow x$	$v_5^W = z \leftarrow x \leftarrow y$	$v_6^W = z \leftarrow y \leftarrow x$

Notice that we need to count the paths of length 3 twice, once for the table W , and once for the last of the unary tables in the respective VEO. In total, we have $15 = 3 + 6 + 6$ unique prefixes (3 of length 1, 6 of length 2, 6 of length 3).

Let T_i be the set of all table prefix instances of length i . The objective can be written as

$$\text{len} = \min \sum_{\pi \in T_1} p(\pi) + \sum_{\pi \in T_2} p(\pi) + 2 \sum_{\pi \in T_3} p(\pi)$$

We can simplify this objective by observing that the prefixes of length 3 are unique to each witness (this is so as those prefixes contain all variables, and every witness is unique under set semantics). Thus, the sum of all total orders of length 3 from the set $\{x, y, z\}$ over all n witnesses is n (one for each witness). We can thus replace all projection variables with table-prefix of length 3 with a constant and simplify the objective to

$$\text{len} = \min \sum_{\pi \in T_1} p(\pi) + \sum_{\pi \in T_2} p(\pi) + 2n$$

The projection constraints for the witness $w_1 = (x_1, y_1, z_1)$ are:

$$p[x_1] \geq q[x_1 \leftarrow y_1] + q[x_1 \leftarrow z_1] \\ p[y_1] \geq q[y_1 \leftarrow x_1] + q[y_1 \leftarrow z_1] \\ p[z_1] \geq q[z_1 \leftarrow x_1] + q[z_1 \leftarrow y_1]$$

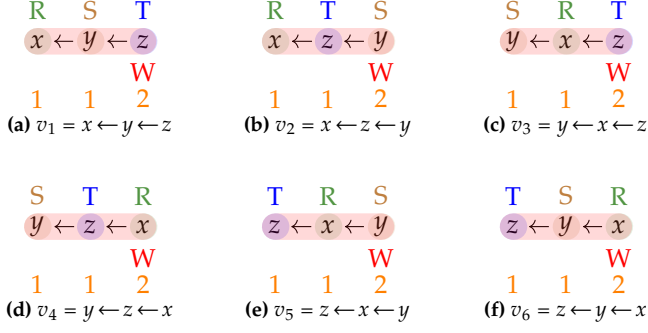


Figure 7.11: Example 7.5.2: mveo for 3-star query Q_3^* .

In this query, for every witness, there must be 9 corresponding variables in the objective, 1 Query Plan constraint, and 3 Projection constraints.

7.5.2 Value of structural knowledge

A key underlying quest in complexity theory and knowledge compilation is determining the *value of knowledge of the underlying structure* by which a problem decomposes. Consider an arbitrary m -partite positive DNF and the problem of finding its minimal factorization. What is the value of knowing the query that produced that provenance polynomial?

If we don't have additional information, then each clause (or witness) could be factored in any of the possible factorization orders. The number of those is equal to the number of labeled rooted trees with m vertices, which is m^{m-1} [126]: A000169: 1, 2, 9, 64, 625, 7776, 117649, . . . However, if we know that the DNF is the provenance of a query Q with k variables over a database \mathcal{D} , then we only need to consider the query-specific number $|\text{mveo}(Q)|$ of minimal VE0s.

[126]: (), OEIS A000169: The On-Line Encyclopedia of Integer Sequences.

For example, for the k -star query Q_k^* , which has the highest number of minimal VE0s (given fixed k), the number of minimal VE0s is $k!$. Contrast this with knowing the query and just having a $k + 1$ -partite positive DNF (its provenance has $k + 1$ partitions). The number partitions alone would imply $(k + 1)^k$ possible factorizations for each witness. To illustrate the difference, we contrast the numbers for increasing $k = 2, 3, \dots$: $\{(k : k! | (k + 1)^k)\} = \{(2 : 2 | 9), (3 : 6 | 64), (4 : 24 | 625), (5 : 120 | 7776), (6 : 720 | 117649), \dots\}$.

7.6 PTIME Algorithms

We provide two PTIME algorithms: (1) a Max-Flow Min-Cut (MFMC) based approach and (2) an LP relaxation from which we obtain a rounding algorithm that gives a guaranteed $|\text{mveo}|$ -approximation for all instances. Interestingly, Section 7.8 will later show that both algorithms (while generally just approximations) give *exact answers* for all currently known PTIME cases of minFACT.

7.6.1 MFMC-based Algorithm for minFACT

Given witnesses W and $\text{mveo}(Q)$, we describe the construction of a *factorization flow graph* F s.t. any *minimal cut* of F corresponds to a factorization of W . A minimal cut of a flow graph is the smallest set

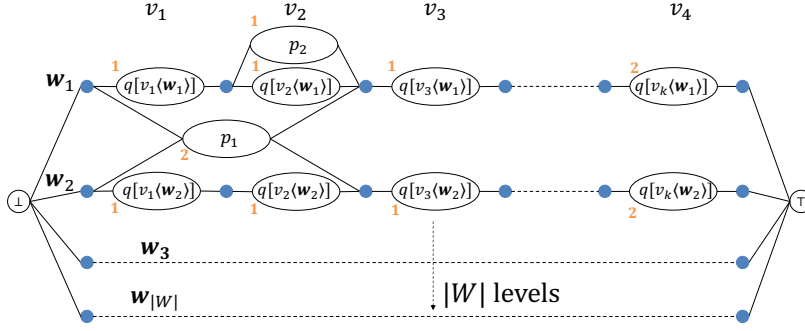


Figure 7.12: A flow graph F for minFACT . The goal is to disconnect the source and the target nodes with minimum cuts. White q and p nodes can be cut and have capacities (in orange) equal to the weights of the corresponding variables in the ILP objective. Edges and connector nodes (in blue) have infinite capacity and cannot be cut.

of nodes whose removal disconnects the source (\perp) and target (\top) nodes [160]. Since minimal cuts of flow graphs can be found in PTIME [42], we obtain is a PTIME approximation for minFACT .

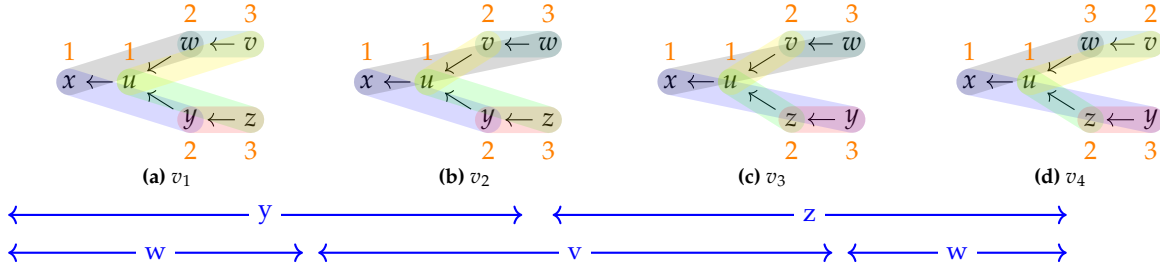
Construction of a factorization flow graph

We construct a flow graph F s.t. there exists a valid factorization of W of length $\leq c$ if the graph has a cut of size c . F is constructed by transforming decision variables in the ILP into "cut" nodes in the flow graph that may be cut at a penalty equal to their weight. Any valid cut of the graph selects nodes that fulfill all constraints of the ILP. We prove this by describing the construction of F (Figure 7.12).

- (1) *mveo order*: We use Ω to describe a total order on mveo (i.e. a total order on the set of minimal query plans). In Figure 7.12, mveo is ordered by $\Omega = (v_1, v_2, \dots, v_k)$ where $k = |\text{mveo}|$.
- (2) *QPV*: For each witness \mathbf{w} , connect the query plan variables (QPV) as defined by Ω . Since all paths from source to target must be disconnected, at least one mveo must be cut from this path. Thus, F enforces the *Query Plan Constraints*.
- (3) *PV*: For each witness \mathbf{w} and prefix variable p , identify the first and last query variable for which p is a prefix and connect the corresponding prefix variable node to connector nodes before and after these query variables. For example, in Figure 7.12, for \mathbf{w}_2 , p_1 starts at $q[v_1(\mathbf{w}_2)]$ and ends at $q[v_2(\mathbf{w}_2)]$ implying that p_1 is a prefix for $q[v_1(\mathbf{w}_2)]$ and $q[v_2(\mathbf{w}_2)]$, but no query plan after that. Now if either of $q[v_1(\mathbf{w}_2)]$ or $q[v_2(\mathbf{w}_2)]$ are in the minimal cut, the graph is not disconnected until p_1 is added to the cut as well. These nodes guarantee that F enforces *Prefix Constraints*.
- (4) *Weights*: Assign each q and p node in F the same weight as in the ILP objective. Recall that this weight is the number of tables with the same table prefix under the same VEO and that it helps calculate the correct factorization length.

Thus, a min-cut of F contains at least one plan for each witness, along with all the prefixes that are necessary for the plan. This guarantees a valid (although not necessarily minimal) factorization.^{||}

^{||} If a min-cut contains more than one plan for a witness, then one can pick either of the plans arbitrarily to obtain a valid factorization.

Figure 7.13: Example 7.6.2: 4 mveo for Q_{6WE}^o that prevent an RP-ordering

When is the MFMC-based algorithm optimal?

In the previous subsection we saw that a min-cut of F always represents a valid factorization. However, the converse is not true: there can be factorizations that do not correspond to a cut. The reason is that *spurious constraints* might arise by the interaction of paths; those additional constraints no longer permit the factorization. There are two types of spurious paths:

(1) Spurious Prefix Constraints. Spurious prefix constraints arise when a prefix node p is in parallel with a query node q of which it is not a prefix. This happens when a q is not prefixed by p , but other query plans before and after are. To avoid this, the ordering Ω must be a *Running-Prefixes (RP) ordering*.^{**}

Definition 7.6.1 (Running-Prefixes (RP) ordering) An ordering $\Omega = (q_1, q_2, \dots, q_k)$ is an RP Ordering and satisfies the RP-Property iff for any p that is a prefix for both q_i and q_j ($i < j$), p is a prefix for all q_k with $i \leq k \leq j$.

Example 7.6.1 (RP ordering) Assume $\text{mveo} = \{(x \leftarrow y \leftarrow z), (x \leftarrow z \leftarrow y), (z \leftarrow y \leftarrow x)\}$. Then $\Omega_1 = ((x \leftarrow y \leftarrow z), (z \leftarrow y \leftarrow x), (x \leftarrow z \leftarrow y))$ is not an RP ordering since the 1st and 3rd VEO share prefix x , however the 2nd starts with z . In contrast, $\Omega_2 = ((x \leftarrow y \leftarrow z), (x \leftarrow z \leftarrow y), (z \leftarrow y \leftarrow x))$ is an RP ordering.

It turns out that for some queries RP-Orderings are impossible (such as Q_{6WE}^o , see Figure 7.13). However, we are able to adapt our algorithm for such queries with a simple extension called *nested orderings*. We first define two query plans as *nestable* if each query plan can be “split” into paths from root to leaf such that they have an equal number of resulting paths, and that the resulting paths can be mapped to each other satisfying the property that corresponding paths use the same set of query variables. *Nested orderings* then are partial orders of query plans such that the pair of query plans may be uncomparable iff they are nestable. Finally, we define Nested RP-orderings as those such that all paths in the partial nested order satisfy the RP property. Intuitively, nested orderings add parallel paths for a single witness to model independent decisions. We can now prove that there always is an ordering that avoids spurious prefix constraints.

^{**} Notice that this concept is reminiscent of the *running intersection* property [13, 15] and the *consecutive ones* property [29].

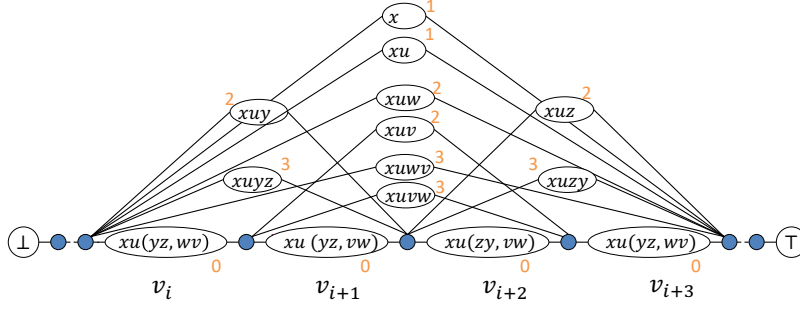


Figure 7.14: Example 7.6.2 incorrect: Part of factorization flow graph showing 4 min VE0s in Non-RP ordering. Notice that prefixes xuw and $xuvw$ span query variables for which they are not prefixes.

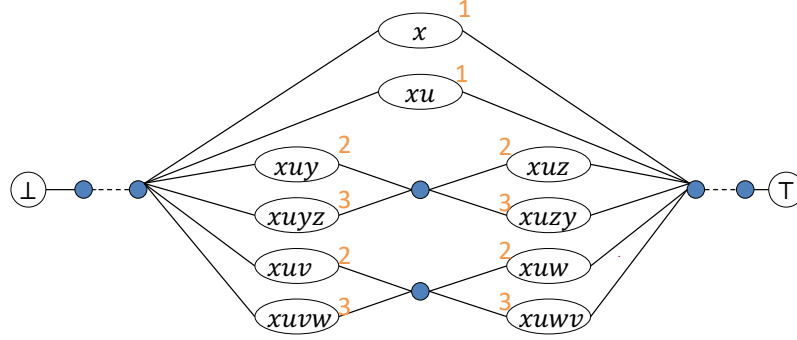


Figure 7.15: Example 7.6.2 corrected: Nested RP-ordering that treats each subtree $\{y, z\}$ and $\{v, w\}$ under $\{x, u\}$ as independent subproblem.

Example 7.6.2 (6-cycle) Consider the 6-cycle query with endpoints shown in Figure 7.16 and the subset $\{v_1, v_2, v_3, v_4\}$ of mveo of this query shown in Figure 7.13. Notice that there is no ordering for these 4 VE0s to allow the running prefix property. For example, if the prefixes of y, z , and v satisfy the property, then we cannot have the prefix of w adjacent as well in a linear ordering. Figure 7.14 shows a partial factorization flow graph focused on these four VE0s in an arbitrary ordering. Notice that the nodes such as $x \leftarrow u \leftarrow w$ may be selected even if their corresponding plans are not used.

Our modification to generate an RP-Ordering relies on the observation that the two children of node a in the VE0 may be *independently* chosen. The left sub-child may be $y \leftarrow z$ or $z \leftarrow y$ and the right sub-child can independently be $v \leftarrow w$ or $w \leftarrow v$. We encode these independent decisions as parallel paths and write the nested ordering as $\Omega = [x \leftarrow u \leftarrow ((z \leftarrow y, y \leftarrow z) \times (v \leftarrow w, w \leftarrow v))]$. In the new flow graph Figure 7.15, each independent decision is a parallel path in the flow graph. Thus, the choice of $xuyz$ vs. $xuzv$ is independent of choosing $xuvw$ vs. $xuvw$. More formally, $\{v_1, v_2, v_3, v_4\}$ are all nestable with each other. Thus, we can build a nested ordering where any pair of these are uncomparable. Figure 7.15 shows the factorization flow graph for Ω .

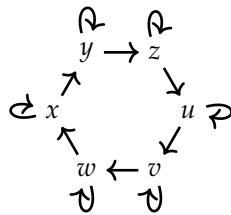


Figure 7.16: Example 7.6.2: 6-cycle query with endpoints $Q_{6WE}^o := A(x), R(x, y), B(y), S(y, z), C(z), T(z, u), D(u), U(u, v), E(v), V(v, w), F(w), W(w, x)$.

Theorem 7.6.1 (Running Prefixes (RP) Property) *For any query, there is a simple or nested ordering Ω that satisfies the RP Property.*

Proof Theorem 7.6.1. We prove the theorem by outlining a method of construction of Ω and showing that this Ω is guaranteed to satisfy the RP property.

To construct an RP-Ordering Ω we first group the mveo by their root variable r . Since the root variables are present in any prefix we know that no two VE0s in different groups can share prefixes. Thus arranging the roots in a pre-determined order e.g. a lexicographical order can lead to no violation of the Running Prefixes property. Within a group of VE0s with root r , we must again decide the ordering. If r disconnects the query incidence matrix, it will have as many children in the VE0s as the number of disconnected components. Since r is a min-cut of the query matrix, there will be no node that is shared by its children and the ordering of each component can be decided independently. With these new groups of independent prefixes, we again look to order them using the predetermined order, splitting into nested paths when necessary. Hence, we obtain a nested RP-Ordering Ω for any Q . \square

(2) Spurious Query Constraints. Query Plan constraints are enforced by paths from source to target such that at least one node *from each path* must be chosen for a valid factorization. Due to sharing of prefix variables, these paths can interact and can lead to additional *spurious paths* that place additional *spurious constraints* on the query nodes. The existence of spurious query constraints does not necessarily imply that the algorithm is not optimal. In fact, Section 7.8 shows that Q_A^Δ and Q_4^∞ have spurious query paths, yet the min-cut is guaranteed to correspond to the minimal factorization. However, if the presence of spurious query paths prevents any of the minimal factorizations to be a min-cut for F , then we say that the factorization flow graph F has “leakage” (Example 7.6.3). Since all paths along one witness are cut by construction, a leakage path must contain nodes from at least two different witnesses.

Definition 7.6.2 (Leakage) *Leakage exists in a factorization flow graph if no minimal factorization is a valid cut of the graph. A leakage path is a path from source to target such that a valid minimal factorization is possible without using any node on the path.*

A Flow Graph With Leakage

The factorization flow graphs are an approximation and can have *leakage* i.e. if no minimal factorization forms a min-cut of the flow graph.

Example 7.6.3 (Leakage in Q^Δ flow graph) Consider a set of 4 witnesses $W = \{r_0s_0t_0, r_1s_1t_0, r_2s_1t_1, r_2s_2t_2\}$ for Q^Δ . It has 3 minimal VE0s: $\text{mveo}(Q^\Delta) = \{v_1 = xy \leftarrow z, v_2 = yz \leftarrow x, v_3 = zx \leftarrow y\}$. Notice that any permutation of the mveo is RP. We arrange the query plan variables in order $\Omega = [v_1, v_2, v_3]$ to construct Figure 7.17. The min-cut of the graph is 11. However, the minimal factorization of W $t_0(r_0s_0 \vee r_1s_1) \vee r_2(s_1t_1 \vee s_2t_2)$ has length 10 (corresponding to the nodes in blue). However, this factorization does not cut the graph

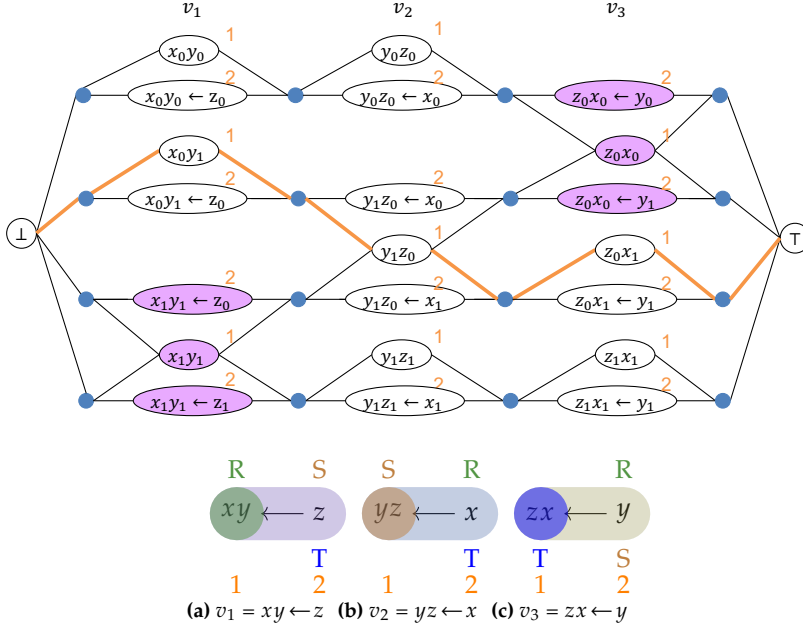


Figure 7.17: Example 7.6.3: A flow graph with leakage for the query Q^Δ .

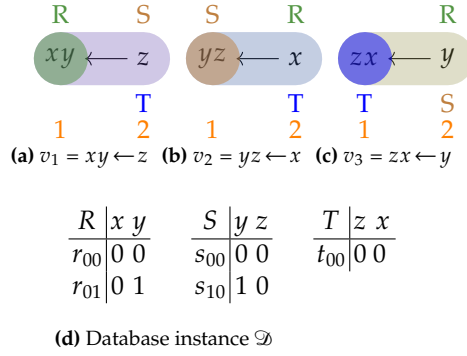


Figure 7.18: Example 7.6.4: Three mveo's for triangle query $Q^\Delta := R(x, y), S(y, z), T(z, x)$ (a)-(c), example database instance \mathcal{D} (d), and constructed flow graph for mveo order $\Omega = [v_1, v_2, v_3]$ (e). Notice that several variables may appear in the same node of a mveo (e.g., x and y in $xy \leftarrow z$).

because the graph encodes an additional spurious constraint (shown as a red path) that enforces $x_0y_1 + y_1z_0 + z_0x_1 \geq 1$ which need not be true.

Optimality of algorithm. Thus, a solution found by the MFMC-based algorithm is guaranteed to be optimal if it has two properties:

1. The ordering Ω is a *Running-Prefixes ordering* or a *nested Running-Prefixes ordering* (always possible).
2. There is no *leakage* in the flow graph (not always possible).

We use these properties in Section 7.7 and Section 7.8 to prove a number of queries to be in PTIME. In fact, all currently known PTIME cases can be solved exactly with the MFMC-based algorithm via a query-dependent ordering of the mveos.

Example 7.6.4 (Flow graph construction for Triangle Query) Consider the triangle query $Q^\Delta := R(x, y), S(y, z), T(z, x)$. The query has 3 minimal Query Plans corresponding to mveos shown in Figures 7.18a to 7.18c. The provenance of Q^Δ over the database shown in Figure 7.18d, has 2 witnesses: $W = \{r_{00}s_{00}t_{00}, r_{01}s_{10}t_{00}\}$. We build a flow graph to

find a factorization. (1) We choose $\Omega = (v_1, v_2, v_3)$ as linear order for the mveo. (2) For each witness, we connect their three query plan variables $q[v\langle\mathbf{w}\rangle]$ in this order serially from source to target. (3) In Q^Δ , each mveo has a single prefix. We attach these variables in parallel to their corresponding query variables. Notice that the prefix z_0x_0 is shared by both \mathbf{w}_1 and \mathbf{w}_2 , and therefore is attached in parallel to both corresponding query variables. (4) Finally, we add weights corresponding to the number of tables having each prefix (see Figures 7.18a to 7.18c).

The resulting flow graph is shown in Figure 7.18e. The min-cut (highlighted in purple) consists of the nodes $\{z_0x_0 \leftarrow y_0, z_0x_0, z_0x_0 \leftarrow y_1\}$. The corresponding factorization using the selected query plans is $t_{00}(r_{00}s_{00} \vee r_{01}s_{10})$. The weighted cut-value (5) is equal to the length of the factorization. This factorization is minimal.

7.6.2 LP relaxation for minFACT and an LP relaxation-based approximation

Linear Programming relaxation and rounding is a commonly-used technique to find PTIME approximations for NPC problems [157]. The LP relaxation for minFACT simply removes the integrality constraints on all the problem variables. The LP relaxation may pick multiple query plans for a given witness, each with fractional values. We present a rounding scheme for minFACT and show it to be a $|\text{mveo}|$ -factor approximation of the optimal solution. The rounding algorithm simply picks the maximum fractional value of query plan variables for each witness, breaking ties arbitrarily. Finally, it counts only the prefix variables necessitated by the chosen query plans.

Algorithm 3: LP rounding Algorithm

Input: List of witnesses W under query Q
Result: A factorization for the instance

```

/* Solving a LP: */
1 Construct the required an ILP for minFACT( $Q$ ) over witnesses  $W$ 
2 Solve an LP relaxation of the problem by treating all decision variables as
  continuous- i.e. allowing values  $[0,1]$  instead of  $\{0,1\}$ .
/* LP rounding: */
3 Set the value of all prefixes variables to 0.
4 for  $\forall \mathbf{w} \in W$  do
5    $qp$  = the query plan variable of  $\mathbf{w}$  with the highest value (break ties
    arbitrarily)
6   Set value of  $qp = 1$ 
7   Set all other query plan variables of  $\mathbf{w}$  to 0
8 for  $\forall$  query plan variables with value = 1 do
9   Set all their prefix variables to 1.
```

The bound provided by Algorithm 3 is the number of minimal query plans, which may be exponential in query size. However, we argue that this simple algorithm provides a useful bound. Let us compare it to the trivial upper bound provided by the DNF provenance. Consider an arbitrary k -chain with intermediates query $\text{:- } R_1(x_1, z_1, x_2), R_2(x_2, z_2, x_3), \dots R_k(x_k, z_k, x_{k+1})$. Assume that all variables have the same domain d . A read-once factorization would have $k * d$ literals, while the DNF could have upto d^k literals, with a worst-case approximation of d^{k-1}/k . On the other hand, the number of minimal query plans for a k -chain query are

given by the Catalan numbers^{††} [66]. The approximation bound does not depend on domain size and gets better in comparison to the *DNF* as domain size is increased. However, even for a small domain size of 10, all chains up to at least $k=30$ have a better approximation bound with the LP rounding.

Theorem 7.6.2 (Constant-factor approximation algorithm for *minFACT*)
*The described rounding scheme gives a PTIME, $|mveo|$ -factor approximation for *minFACT*.*

Proof Theorem 7.6.2. Since for each witness, a valid query plan as well as all associated prefixes are chosen, we see that the approximation algorithm returns a valid factorization.

We see that each query plan is multiplied by at most $|mveo|$ the value it had in the LP. A prefix is chosen only if the associated query plan has a value of at least $1/|mveo|$ in the LP. Due to prefix constraints, any chosen prefix too must have had to have a value of at least $1/|mveo|$ in the LP. Thus all variables are multiplied by at most $|mveo|$ and we can guarantee the algorithm is an $|mveo|$ -factor approximation. \square

When is the LP relaxation optimal?

Experimentally, we observed that the LP solution of many queries are equal to the integral ILP solution. This is surprising since the ILP does not satisfy any of the known requirements for tractable ILPs such as Total Unimodularity, Balanced Matrices, or even Total Dual Integrality [144]. Interestingly, we next prove that the LP relaxation has the same objective value as the original ILP whenever the MFMC-based algorithm is optimal.

Theorem 7.6.3 (Relationship between MFMC and LP based approaches for *minFACT*)
*If all database instances can be solved exactly by the MFMC-based algorithm for a given query Q (i.e. for each database instance there exists an ordering that generates a leakage-free graph), then the LP relaxation of *minFACT* always has the same objective as the original ILP.*

Proof Theorem 7.6.3. We prove that, for such a query, the LP relaxation of the ILP[*minFACT*] always gives the correct value. Concretely, we show that the solution of the LP is the same as that of the MFMC-based algorithm in any leakage-free ordering in [Subsection 7.8.2](#).

First, we notice that the LP relaxation can only provide a lower bound for *minFACT*, while we have the assurance that the MFMC-based algorithm computes the exact *minFACT* value.

Next, we show that any solution obtained by the LP must satisfy the MFMC algorithm in a leakage-free ordering. We know that the LP solution must satisfy the query plan and prefix constraints, and thus the corresponding paths are cut in the factorization flow graph. The only remaining paths can only be leakage paths. However, since we assumed this to be a leakage-free ordering, any additional leakage paths are guaranteed not to increase the value of the min-cut. Thus, due to the leakage-free nature, we need only check that the paths corresponding to

^{††} OEIS sequence [A000108](#)

query plan paths and prefix paths are cut for a valid cut. In other words, a solution that cuts all query plan and prefix paths must have an equivalent solution that cuts all paths. Thus there is no path (or “constraint”) that is not cut (i.e. not satisfied) by the LP solution, and the LP solution can be represented as a cut of the graph.

Since we have shown that the LP solution is both no greater than and no smaller than the min-cut solution, we have also shown that they are identical (in the presence of a leakage-free ordering). \square

This result is important as it exposes cases for which the optimal objective value of ILP[minFACT] is identical to the optimal objective value of a simpler LP relaxation. This follows the same theme that we have seen in the previous chapters, and here as well we can thus ILPs to solve tractable cases in PTIME. We use this knowledge in Section 7.8 to show that ILP solvers can solve all known PTIME cases in PTIME.

7.7 Recovering Read-Once instances

It is known that the read-once factorization of a read-once instance can be found in PTIME with specialized algorithms [75, 141, 147]. We prove that our more general MFMC based algorithm and LP relaxation are *always guaranteed to find read-once formulas* when they exist, even though they are not specifically designed to do so.

Details on read-once instances. A formula is read-once if it does not contain any repeated variables. Read-once provenance formulas are characterized by the absence of a P_4 [35] co-occurrence pattern, which is a pattern $(w_1 - r - w_2 - s - w_3)$ where a witness w_2 shares tuple r with w_1 , and shares tuple s with w_3 , however, w_1 and w_3 do not share r nor s (Figure 7.19).² We prove that for any read-once formula for any query the F under any RP-Ordering does not have leakage.

Theorem 7.7.1 (minFACT of Read-Once Instances is tractable) *minFACT can be found in PTIME by*

- ▶ the MFMC based algorithm, and
- ▶ the LP relaxation,

for any query and database instance that permits a read-once factorization.

2: P_4 patterns are alternatively defined based on the 4 variables that form a path like $(t - r - s - u)$ in Figure 7.19. The second property of “normality” [35, Sect. 10] is always fulfilled by partitioned expressions such as the provenance for sj-free queries.

Proof Theorem 7.7.1 (Part 1). Consider any RP or nested RP-Ordering Ω . Notice that the factorization flow graph F represents a database with a P_4 if and only if we have three witnesses connected as shown in Figure 7.20.

Let us next assume that we have no P_4 , yet have leakage in F involving w_1 and w_2 . For this to be true, w_1 and w_2 must share a node t that is not chosen in the minimal factorization. It is usually not beneficial to choose a node that belongs exclusively to w_1 or w_2 over a shared prefix t .

Thus, a shared prefix t is *not* chosen only if there is some other prefix that is shared between witnesses. Then there are exactly two possibilities:

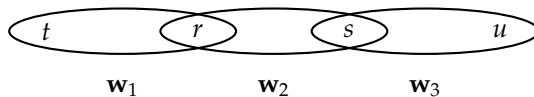


Figure 7.19: Section 7.7: P_4 co-occurrence pattern $(w_1 - r - w_2 - s - w_3)$.

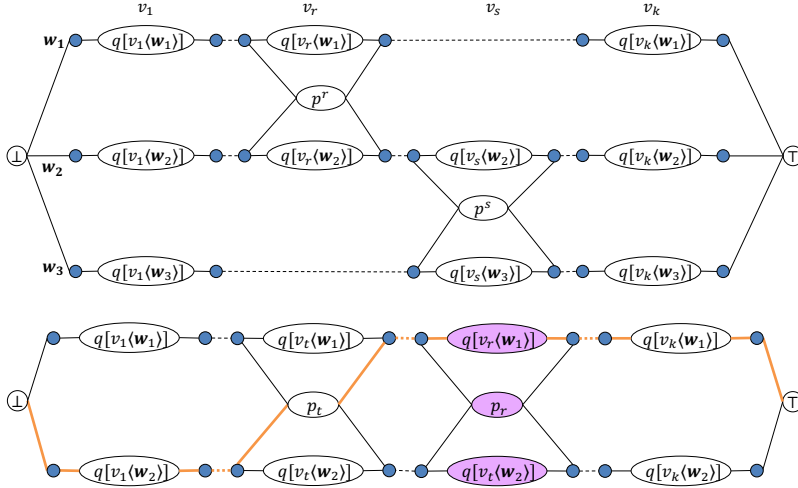


Figure 7.20: Proof [Theorem 7.7.1](#): $P_4(w_1 - p_r - w_2 - p_s - w_3)$ in factorization flow graph.

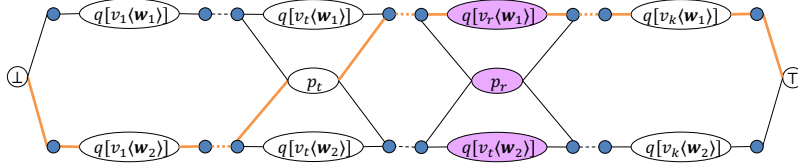


Figure 7.21: Proof [Theorem 7.7.1](#): Impossibility of leakage in read-once Instances (Case 2).

either this additional shared prefix is 1) shared with one of w_1 or w_2 , or 2) shared with both w_1 and w_2 . We explore both these cases:

1. One of the witnesses shares a different prefix with a different witness w_3 . This scenario describes a P_4 , and hence is not possible in a read-once instance.
2. The two witnesses w_1 and w_2 share a different node r as well, and that node is chosen for the minimal factorization. W.l.o.g, consider that the chosen VEO v_r lies to the right of shared v_t node under the ordering Ω ([Figure 7.21](#)). Any leakage path that passes through v_t must pass through v_r or a prefix of v_r as well. Since these nodes must be selected, the path cannot "leak".

Thus, we can never have a leakage path for any read-once instance and the factorization flow graph returns the minimal factorization. In more general terms, our flow graph also correctly identifies the minimal factorization of all known tractable cases of exact probabilistic inference (see [Figure 7.34](#)) \square

Proof [Theorem 7.7.1](#) (Part 2). We show that the LP relaxation of the minFACT ILP is always correct for read-once instances by showing that the solution of the LP is the same as that of the MFMC-based algorithm (which we proved is correct in [Theorem 7.7.1](#)). First, we see that any solution obtained by the MFMC-based algorithm is a valid solution for the LP. Since each path in the factorization flow graph is cut, any solution fulfills all the query plan as well as prefix constraints, thus becoming a valid solution for the LP and thus an upper bound for the optimal LP solution.

We can next see that the reverse is also true, that a valid LP solution also is a cut for the factorization flow graph. We showed in [Theorem 7.7.1](#) that a read-once instance has no leakage path, therefore each path of the flow graph corresponds exactly to a constraint in the LP. Since the flow graph imposes no additional constraints ("paths"), there is no constraint ("path") that is not satisfied (not cut) by the LP solution.

Hence, since valid LP solutions and valid min-cuts have an equivalence for read-once instances, the Linear Program is guaranteed to find the read-once factorization as well. \square

7.8 Tractable Queries for minFACT

We now go beyond cases when PROB is in PTIME (i.e. read-once instances). We first prove that $\text{minFACT}(Q)$ is PTIME for the large class of queries with 2 minimal query plans. We then show examples of queries with 3 and 5 minimal query plans that are PTIME as well. All these newly recovered PTIME cases, along with the previously known read-once cases, can be solved *exactly* with both our PTIME algorithms from [Section 7.6](#). Finally, we hypothesize that minFACT is in PTIME for any linear query.

7.8.1 All queries with ≤ 2 minimal query plans

We prove that our MFMC-based algorithm has no leakage and thus always finds the minimal factorization for queries with at most 2 minimal VEOs (2-MQP) queries such as Q_2^* and Q_3^∞ . We also give an alternative proof that shows that any ILP generated by such a query is guaranteed to have a *Totally Unimodular (TU)* constraint matrix, and thus is PTIME solvable [144].

Theorem 7.8.1 (minFACT of 2-MQP Queries is tractable) *minFACT can be found in PTIME for any query with max 2 minimal VEOs by*

- ▶ the MFMC based algorithm, and
- ▶ the LP relaxation.

The theorem recovers the hierarchical queries which are equivalent to 1-MQP queries since they have one “safe plan” [39]. The PTIME nature of 1-MQP queries also follows from [Theorem 7.7.1](#), as all hierarchical queries have read-once formulations.

We first prove the theorem for the MFMC based algorithm, and then for the LP relaxation. The proof for the LP relaxation alternatively follows from [Theorem 7.6.3](#) given the first part of the proof.

Proof Theorem 7.8.1 (Part 1). Consider an arbitrary query Q with $\text{mveo} = \{v_1, v_2\}$. There are two possible orderings of QPV . We can trivially see they are both Running-Prefixes ordering since both query nodes either share prefixes or do not. For there to be leakage in the flow graph, there must be a path that connects two cut nodes that are not associated with the same witness. Since the query has two minimal plans, every path from source to target passes through at most two cut nodes. The two nodes in the path are connected if and only if they are associated with a common witness. Thus, no leakage is possible in this graph.

Since the flow graph for any query with mveo of size 2 always has an RP ordering and can never have leakage, our algorithm is guaranteed to always return the minimal factorization. \square

Proof Theorem 7.8.1 (Part 2). If a query has 2 min-VEOs, we first show that if it has “equal roots” then the constraint matrix for the ILP generated by this query is Totally Unimodular, and then show how this applies to any query with 2 minimal plans.

Definition 7.8.1 (VEO roots) *The root of a VEO is the set of variables in the VEO that are projected away last in the corresponding query plan.*

Notice that the roots of two VEO are considered equal if and only if the sets are identical.

Example 7.8.1 (VEO roots) The VEOs $x \leftarrow y \leftarrow z$ and $x \leftarrow z \leftarrow y$ have equal root $\{x\}$, while $xy \leftarrow z$ and $xz \leftarrow y$ do not.

Proposition 7.8.2 (Total Unimodularity of 2-MQP Queries) *Let Q be a query with precisely two minimal query plans $\{v_1, v_2\} = \text{mveo}(Q)$ where v_1 and v_2 have unequal roots. Let C be the constraint matrix of an ILP to find the minimal factorization of any instance with Q constructed as defined in (7.2). Then C is a Totally Unimodular (TU) matrix.*

Proof [Proposition 7.8.2](#). The following [Lemma 7.8.3](#) describes a set of sufficient conditions for Total Unimodularity. We show that C^T meets all conditions. Since the transpose of a TU matrix is also TU, we hence prove that C is a TU matrix as well.

Lemma 7.8.3 (Sufficient conditions for Total Unimodularity [89]) *Let A be an m by n matrix whose rows can be partitioned into two disjoint sets B and C , with the following properties:*

1. *every entry in A is 0, +1, or -1*
2. *every column of A contains at most two non-zero entries;*
3. *if two non-zero entries in a column of A have the same sign, then the row of one is in B , and the other in C*
4. *if two non-zero entries in a column of A have opposite signs, then the rows of both are in B , or both in C .*

Then every minor determinant of A is 0, +1, or -1 i.e. A is a totally unimodular matrix.

Each column in C^T represents an ILP constraint and each row represents a decision variable.

Condition 1: Recall the query plan constraint, projection constraint, and integrity constraints. For any variable and any constraint, the coefficient is 1 if the variable is on the LHS of the constraint, -1 if it is on the RHS, and 0 if it does not participate in the constraint. The coefficient cannot be any other value.

Condition 2: The query plan constraint enforces a relationship between the variables of each min-VEO. In the case of queries with 2 plans, this constraint involves 2 variables. The project constraints always enforce a relationship between 2 variables- i.e. a query plan instance and a variable-prefix instance, while the integrity constraint always involves a single variable. Hence, there are never more than 2 non-zero coefficients in a single constraint (and hence, a column of C^T).

Condition 3: The only type of constraint that has two coefficients of the same non-zero value is the query plan constraint. Each constraint involves a variable for VEO v_1 , and another for VEO v_2 . We assign all variables corresponding to v_1 to set B and those corresponding to v_2 to set C to satisfy this property.

Condition 4: The only type of constraint that has two coefficients of the different non-zero values is the projection constraint. The constraint involves a query variable and a projection variable. To satisfy condition 3 we have already assigned the query variable to either set B or C . We now

assign the projection variable to the same set to which the query variable is assigned. A projection variable is constrained to a query variable if and only if the projection variable has the same table-prefix instances as the query variable. Since v_1 and v_2 have different roots, no projection variable can be constrained to both the query variables. Hence, every variable is assigned to only one of the two sets, and we fulfill all required properties. \square

We can now use the proven-PTIME case for 2 VE0s with unequal roots to also solve the case where the roots are equal. For query Q with min-VE0s v_1, v_2 such that they have equal roots, we first check which tables have the same table-prefix in both the VE0s. The tuples from these tables can be minimally factorized only in a pre-ordained since they have a single table-prefix. We separately pre-compute the length due to tuples from these tables and remove these table-prefixes from the set PVF . After this simplification, no table-prefix will be associated with both VE0s. We can now satisfy Condition 4 by dividing projection variables like in the unequal-root case. Recall that condition 4 was the only condition where we required the unequal roots property. Thus, we prove that the simplified matrix is TU, and we can obtain PTIME solutions for all queries with 2 minimal query plans. \square

Corollary 7.8.4 (minFACT of Hierarchical Queries is tractable) *minFACT for Hierarchical Queries is in PTIME.*

Corollary 7.8.5 (minFACT more tractable than PQE) *The classes of queries for which minFACT is in PTIME is a strict super-class of those for which probabilistic query evaluation is in PTIME (if $P \neq NP$).*

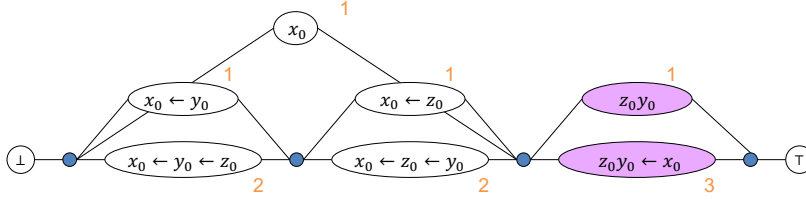
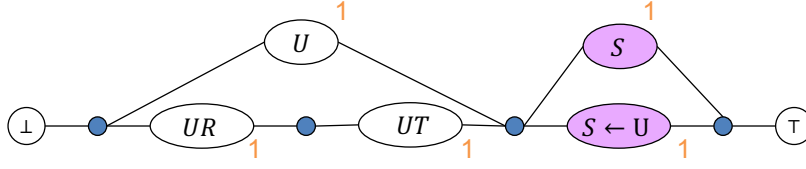
7.8.2 Two queries with ≥ 3 minimal query plans

Triangle-unary Q_A^Δ . We next prove that minFACT for a query that we call “Triangle Unary” $Q_A^\Delta := U(x), R(x, y), S(y, z), T(z, x)$ is in PTIME. This is surprising since the query has 3 minimal VE0s $\text{mveo} = \{x \leftarrow y \leftarrow z, x \leftarrow z \leftarrow y, yz \leftarrow x\}$ and appears very similar to the triangle query $Q_\Delta := R(x, y), S(y, z), T(z, x)$ which also has 3 minimal VE0s $\text{mveo} = \{xy \leftarrow z, xz \leftarrow y, yz \leftarrow x\}$. Both queries are illustrated in Figure 7.30. However Q_Δ contains an active triad, which proves $\text{FACT}(Q)$ to be NPC, whereas Q_A^Δ does not have an active triad (neither $R(x, y)$ nor $T(z, x)$ are independent due to the relation $U(x)$). Furthermore, the constraint matrix of the Q_A^Δ ’s ILP is *not guaranteed to be Totally Unimodular (TU)*, thus our prior TU argument cannot be used anymore for it to be in PTIME. Then we can use Theorem 7.6.3 to show that the LP relaxation is always easy as well.

Theorem 7.8.6 (minFACT is tractable for Q_A^Δ) *minFACT(Q_A^Δ, D) can be found in PTIME for any database \mathcal{D} by*

- the MFMC based algorithm, and
- the LP relaxation.

Theorem 7.8.6 (Part 1). Figure 7.22 shows the flow graph for a single witness under order $\Omega = [x \leftarrow y \leftarrow z, x \leftarrow z \leftarrow x, yz \leftarrow x]$. However,


 Figure 7.22: Q_A^Δ graph for a single witness

 Figure 7.23: Simplified Q_A^Δ graph for a single witness

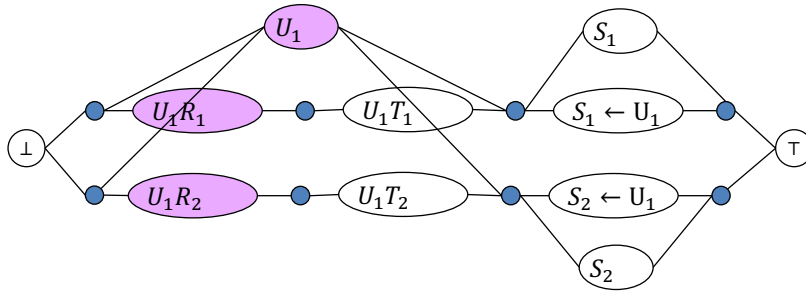
this proof can be applied to any Ω of $\text{mveo}(Q_A^\Delta)$ that satisfies the Running Prefix property. Notice that no matter what mveo is chosen, a weight of at least 2 is assigned to the full witness. We can preprocess the length due to these nodes as $2 * |W|$ and remove the nodes that correspond to the full witness from the graph. In addition, we rename the mveo to names based on the tables that distinguish them i.e. Ω can now be represented as $UR, UT, S \leftarrow U$. The optimized graph for a single witness is shown in Figure 7.23. Since all the weights are = 1, we leave out the weights in subsequent figures.

We see from the graph that Ω is a Running-Prefixes Ordering, as the only shared prefix U , is shared between adjacent nodes.

We now show that $F(Q, \Omega)$ is leakage-free for any W . A leakage path must involve at least two witnesses since a path with nodes from one witness only will be cut in any minimal factorization. Let w_1 and w_2 be two witnesses whose nodes lie on the leakage path.

We divide the proof into 7 parts based on the relationship between w_1 and w_2 . If w_1 and w_2 share:

1. *No common variable values*: There will be no common nodes between the two witnesses, and leakage is not possible.
2. *Only y or z values*: There are still no common nodes between the two witnesses (since there are no y or z nodes), and hence leakage is not possible.
3. *Only x values*: In this case (Figure 7.24), the U node is shared. For the shared node to be on the leakage path, it is not selected in a minimal factorization - thus the witnesses may not choose any UR or UT plans \rightarrow thus the witnesses must choose the $S \leftarrow U$ plans. Since all paths through U pass through S or $S \leftarrow U$ nodes, leakage is not possible.


 Figure 7.24: Q_A^Δ instance, with shared x (Case 3)

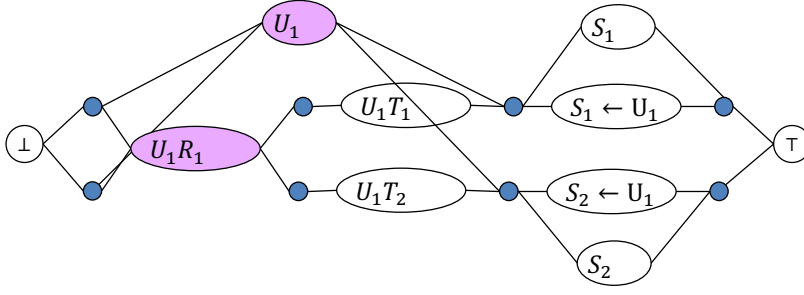


Figure 7.25: Q_A^Δ instance, with shared xy (Case 4)

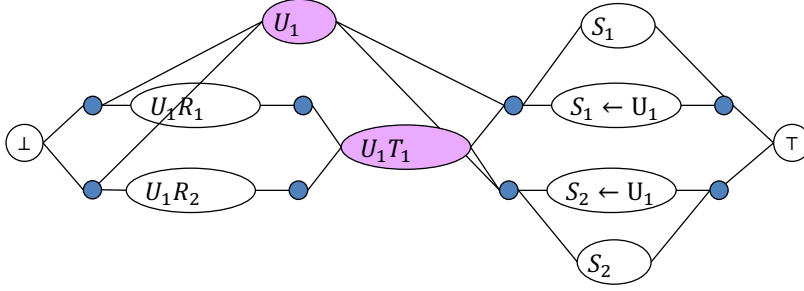


Figure 7.26: Q_A^Δ instance, with shared xz (Case 5)

4. x and y values: In this case (Figure 7.25), we can see that since the shared node UR is at the side of the graph, we see the paths passing through UR must pass through nodes corresponding with the same witness only (either w_1 or w_2). Thus, no leakage is possible.
5. x and z values: In this case (Figure 7.26), a leakage is possible if and only if one witness uses the UR plan while the other uses the $S \leftarrow U$ plan. However, if an R or T tuple is repeated, then the minimal factorization will never use the $S \leftarrow U$ plan, thus preventing the only potential leakage path.
6. y and z values: This case (Figure 7.27), is similar to case 4 - since the shared node S is at the side of the graph, we see that all paths through it pass through nodes corresponding with either only w_1 or only w_2 , thus not allowing leakage.
7. x , y and z values: This case is not possible under set semantics. \square

We show an RP and leakage-free ordering for Q_A^Δ , hence proving that the algorithm meets conditions for optimality described in Subsection 7.6.1.

Theorem 7.8.6 (Part 2). As we have shown in Theorem 7.6.3 that the LP relaxation of minFACT is correct when it is solved by the MFMC-based algorithm, this result follows given Theorem 7.8.6. \square

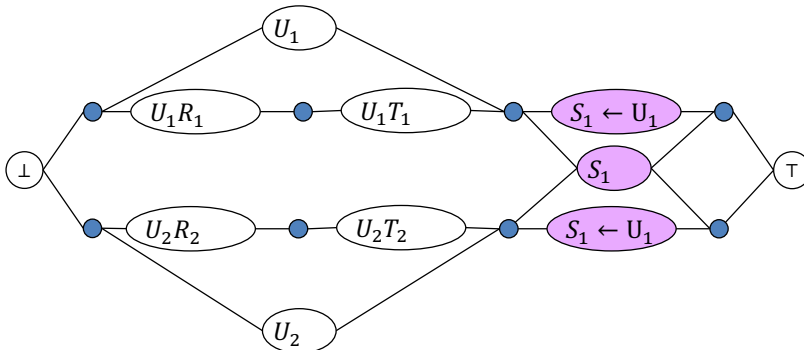


Figure 7.27: Q_A^Δ instance, with shared yz (Case 6)

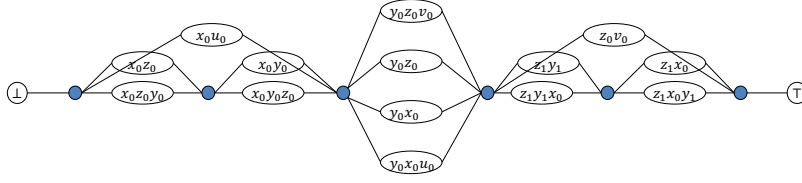


Figure 7.28: Flow graph with a single witness of Q_4^∞ under ordering Ω .

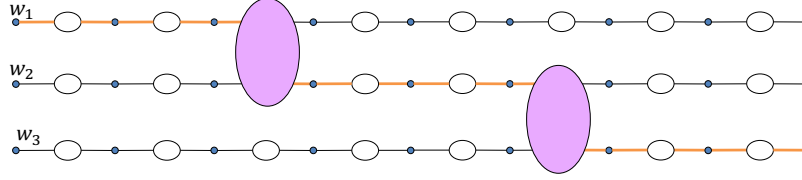


Figure 7.29: Lemma 7.8.8: Showing that if there is leakage in the graph, there must be two witnesses that can together form a leakage path.

4-chain Q_4^∞ . This is arguably the most involved proof in the paper. Q_4^∞ has $|\text{mveo}| = 5$. Yet in a similar proof to Q^Δ , we can show that the MFMC-based algorithm and the LP are both optimal. This surprising result leads to the conjecture that minFACT for longer chains, and all linear queries are in PTIME.

Just like in the previous subsections, we show the PTIME complexity by showing that the MFMC-based algorithm and LP relaxation are optimal. The query has five minimal VE0s and in the first proof, we show that one RP-ordering is leakage-free and optimal. We then use this factorization flow graph to show that the Linear Program is always correct for any Q_4^∞ instance as well.

Theorem 7.8.7 (minFACT is tractable for Q_4^∞) $\text{minFACT}(Q_4^\infty, D)$ can be found in PTIME for any database D by

- the MFMC based algorithm, and
- the LP relaxation.

Proof Theorem 7.8.7 (Part 1). Q_4^∞ has 5 minimal VE0s. Let us define an ordering $\Omega = [x \leftarrow (u, y \leftarrow z \leftarrow v), x \leftarrow (u, z \leftarrow (y, v)), y \leftarrow (x \leftarrow u, z \leftarrow v), z \leftarrow (x \leftarrow (u, y), v), z \leftarrow (y \leftarrow x \leftarrow u, v)]$

Notice that there are 2 plans each with x or z as a prefix, 1 plan with y as a prefix, and no plan with u or v as the prefix. Notice as well that in the ordering Ω , the y plan is in the center and the $x \leftarrow z$ and $z \leftarrow x$ plans are at the sides. Figure 7.28 shows a single witness under this ordering.

We first check that this ordering has the Running Prefixes Property. This is easily verified from the figure, no projection is parallel to a node it is not a prefix of.

Next, we want to show that no flow graph under this ordering will have leakage. For this, we first prove a lemma that if there is leakage in a flow graph then there exists a leakage path comprised of nodes from just 2 witnesses.

Lemma 7.8.8 (Necessary condition for leakage) *If there exists leakage in a factorization flow graph F then there must exist two witnesses w_1, w_2 such that if we only consider a graph with cut nodes corresponding to these witnesses then the plans they use any minimal factorization do not cut the graph.*

Lemma 7.8.8. Assume that the lemma is false and any leakage path in F must traverse through 3 witnesses. We know that for a leakage path to exist, the witnesses must have some shared nodes. Let us the minimal paths needed for such a set of witnesses w_1, w_2, w_3 in Figure 7.29. For a leakage path to exist, they must not choose any of the shared computations in the path. W.l.o.g., assume w_1 chooses a plan to the right of the shared computation. Notice that if w_2 chooses a plan in between the two shared computations then there isn't any more leakage. So the only remaining case is if w_2 chooses a plan to the left of the shared computation with w_3 , in which case there is a leakage path between w_2 and w_3 . \square

Now that we know that if there is leakage, there must be a leakage path between two witnesses, let us assume there is an instance of Q_4^∞ under Ω where witnesses w_1, w_2 have a leakage path.

We do a case-wise analysis to show how this cannot exist, based on the variables that w_1 and w_2 share.

1. They share no variables: No leakage is possible
2. They share u, x, y, z , or v variables: In this case to the two witnesses share no nodes so no leakage is possible
3. They share ux or zv or xz variables: In this case, the two witnesses share a variable - however, the variable is at the sides of the flow graph and hence cannot create a leakage path (there cannot be a crossover in the leakage path from w_1 to w_2)
4. They share $uxzv$: In this case, they share multiple nodes, but again, the shared nodes as on the sides.
5. They share xy (or yz): In this case, we can see there will be leakage if the w_1 chooses an x plan and w_2 chooses a different z . If the two witnesses share only xy and no other variables, then xy must be a part of at least 4 witnesses because of join dependencies (every u that x is connected to must also connect to z and vice versa). In this case, you would always choose a y plan since y acts as the root of the Cartesian product.
However, if they share other variables along with xy , the argument differs. The choosing of alternate plans could only happen if x and z were both connected to different y as well. In this case, the xy tuple would be repeated at least once, and since they use different prefixes and share tuples, all of the tuples ux, yz, yv would be repeated. Meanwhile, a factorization where they both choose the same plan could only incur additional repeats on one of ux or zv .

Notice that the above cases are exhaustive since they cover all possible prefix node combinations that can be shared between two witnesses. That is, they cover the possibility of two witnesses sharing nodes ux, xz (equivalently zx , since the variables shared are the same), zv (in Case 3); xy (equivalently yx), yz (equivalently zy), xzy (equivalently xyz), yzv , yxu, zyx (equivalently zxy) (in Case 5). Notice that case 5 also includes when any other additional variables are shared alongside xy or yz . Thus, the above cases suffice to cover all possible prefix node sharing between two witnesses. Note that two witnesses can't share all 5 variables or all prefix nodes, as that would make them the same witness. \square

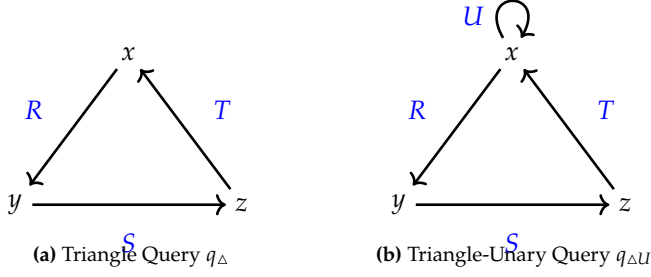


Figure 7.30: We show that Q^{Δ} is hard in Section 7.9 because it contains an “active triad.” Surprisingly, for Q_A^{Δ} , a query that differs by a single unary relation, the minimal factorization can always be found in PTIME by either using our MFMC based algorithm from Subsection 7.6.1 or our LP relaxation from Subsection 7.6.2.

7.8.3 Conjecture for Linear Queries

A query is acyclic if it has a join tree, i.e. it permits a placement of its atoms into a tree s.t. for any two atoms, the intersection of variables is contained in the union of the variables of the atoms on the unique path between them.^{‡‡} A query is *linear* if it permits a join path.^{§§} We have spent a lot of time trying to prove the hardness of such queries without success. Based on our intuition we hypothesize that *all linear queries are in PTIME*. Our intuition is strengthened by the fact that over many experimental evaluations, the LP relaxation of the ILP[minFACT] was always integral and optimal, thus being able to solve the problem in PTIME.

Conjecture 7.8.9 (PTIME conjecture) *If Q is a linear query, then $\text{minFACT}(Q, D)$ can be found in PTIME for any database D .*

We think that additional insights from optimization theory are needed to explain the integrality of the solution to the LP relaxation and to thus prove this conjecture. We leave open the structural criterion that separates the easy and hard cases.

7.9 Hard Queries for minFACT

In this section, we first prove that all queries that contain a structure called “an active triad” (e.g. Q_3^* and Q^{Δ}) are NPC. We then prove another query to be NPC that does not contain an active triad, but a “co-deactivated triad.” We thus show that while active triads are sufficient for FACT of a query to be NPC, they are not necessary, and minFACT is a strictly harder problem than RES.

Queries with Active triads. We repeat here the necessary definitions introduced in the context of resilience under bag semantics (Chapter 4). A *triad* is a set of three atoms, $\mathcal{T} = \{R_1, R_2, R_3\}$ s.t. for every pair $i \neq j$, there is a path from R_i to R_j that uses no variable occurring in the third atom of \mathcal{T} . Here a *path* is an alternating sequence of relations and variables $R_1 - x_1 - R_2 - \dots - x_{p-1} - R_p$ s.t. all adjacent relations R_i, R_{i+1} share variables x_i . In a query Q with atoms R and S , we say R *dominates* S iff $\text{var}(R) \subset \text{var}(S)$. We call an atom g in a query *independent* iff there is no other atom in the query that contains a strict subset of its variables

^{‡‡} The concept is alternatively called coherence, the running intersection property, connected subgraph property [13, 15, 151], and is used in the definition of the junction tree algorithm [107] and tree decompositions [46, 139].

^{§§} This definition, introduced in Chapter 4, is more restrictive than linear queries defined in the original work on resilience [59] as it does not allow *linearizable queries* (those that can be made linear by “making dominated atoms exogenous”).

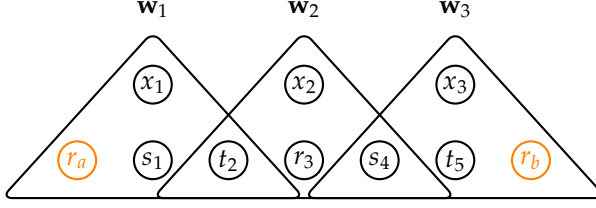


Figure 7.31: Theorem 7.9.1: Hardness gadget for a query with triad $\{R, S, T\}$.

(and hence it is not dominated). A triad is *active* iff none of its atoms are dominated.

Theorem 7.9.1 (minFACT is hard for queries with Active Triads) $\text{FACT}(Q)$ for a query Q with an active triad is NPC.

We notice that this proof is inspired by a result in [60]. It can easily be adapted to also prove the hardness of triads for resilience and is considerably simpler than the original proof in [59]. The key idea is that in our proof construction, the chosen roots correspond to the minimum vertex cover, or equivalently, the minimum number of input tuples to cover, in order to remove all witnesses.

Proof Theorem 7.9.1. Let Q be a query with triad $\mathcal{T} = \{R, S, T\}$. We construct a hardness gadget Figure 7.31 that can be used to build a reduction from IndSet to any query Q .

The hardness gadget is built with 3 witnesses. We assume that no variable is shared by all three elements of \mathcal{T} (we can ignore any such variable by setting it to a constant). For the first 2 witnesses, only the variables of atom T are equal in both witnesses. Because T is an independent atom, its tuple will be the only common tuple between w_1 and w_2 . The nodes x_1, x_2, x_3 represent all the tuples of tables that are not part of the triad \mathcal{T} . Hence any query with a triad can create an instance of this gadget.

We see that the gadget has 2 minimal factorizations: $r_a s_1 t_2 x_1 \vee s_4 (r_3 t_2 x_2 \vee t_5 x_3 r_b)$ or as $t_2 (r_a s_1 x_1 \vee r_3 s_4 x_2) \vee r_b t_5 s_4 x_3$, both of which incur a penalty of 1 and use one of r_a or r_b as root. We can represent these factorizations with gadget orientations, such that each gadget is oriented from the root of the factorization to the sink. Either of these factorizations must repeat some variable (thus have at least a penalty of 1), for t_2 or s_4 respectively. Since r_a and r_b may be connected to the other edges of the graph as well, they may incur further penalties if they are not root nodes in more than one factorization term.

To prove the problem is NPC, we reduce the Independent Set (IS) problem to $\text{FACT}(Q)$. An independent set (IndSet) of an undirected Graph $G(V, E)$ is a subset of vertices $V' \subseteq V$, such that no two vertices in V' are adjacent. The IndSet problem asks, for a given graph G and a positive integer $k \leq |V|$, whether G contains an independent set V' having $|V'| \geq k$, which we write as $(G, k) \in \text{IndSet}$.

$$\text{IndSet} \leq \text{FACT}(Q_3^*)$$

Proposition 7.9.2 (Connection of factorization and independent set: Direction 1) \mathcal{D} has a factorization with a penalty $2|E| - k$ if G has an independent set of size k .

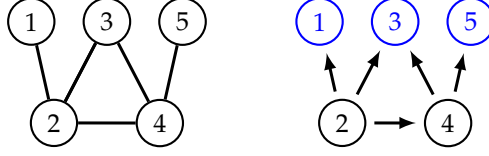


Figure 7.32: Theorem 7.9.1: Example graph with an independent set $\{1, 3, 5\}$. By orienting all edge directions towards the independent sets (and thus making them sink nodes), we achieve a factorization with penalty $2|E| - 3 = 7$. The penalties at endpoints 3 and 4 are 1, elsewhere 0. Additionally, each of the 5 edges is encoded with the path gadget and has an internal penalty of 1 each.

Proof Proposition 7.9.2. If G has an independent set of size k then a factorization can be constructed corresponding to an edge orientation that points all edges towards nodes in the independent set. A minimum penalty of $|E|$ is guaranteed to be incurred as there is a penalty either on t_3 or s_5 for every gadget. What we need to reason about is the additional penalties that are incurred by the nodes at the end of the gadget. For every node, it is counted in the factorization only once as root, and as many times as a non-root as it has incoming edges. If a node ever acts as a root, then its penalty is equal to the number of incoming edges, however, if it is never a root, then the penalty is the number of incoming edges -1 . We construct an edge orientation such that any member of a maximum independent set never acts as the root, as shown in Figure 7.32. This construction is always possible since no members of the independent set are connected. The total factorization penalty then is the total incoming edges over all nodes - the size of the independent set. Since all $|E|$ edges are an incoming edge for some node, the penalty due to k nodes is then $|E| - k$. The total penalty is hence $|E| + |E| - k$ or $2|E| - k$. \square

Proposition 7.9.3 (Connection of factorization and independent set: Direction 2) *If G has no independent set of size k then \mathcal{D} cannot have a factorization with penalty $2|E| - k$.*

Proof Proposition 7.9.3. For \mathcal{D} to have a factorization with penalty $2|E| - k$ there would have to be k gadgets with penalty exactly equal to 1 thus implying the presence of k sink nodes. Since k sink nodes form an independent set of size k , \mathcal{D} cannot have a factorization with penalty $2|E| - k$. \square

Propositions 7.9.2 and 7.9.3 together finish the reduction. \square

Separation between RES and minFACT. A triad is *deactivated* if any of the three atoms is dominated. A triad is *co-deactivated* if all three atoms are dominated only by the same (non-empty) set of atoms. The co-deactivated triangle query $Q_{\text{cod}}^\Delta := A(w), R(w, x, y), S(w, y, z), T(w, z, x)$ contains no active triads: notice that the tables R, S and T are not independent and have no independent paths to each other. Thus, $\text{RES}(Q_{\text{cod}}^\Delta)$ is PTIME. However, Q_{cod}^Δ contains a co-deactivated triad since R, S and T are all dominated only by atom A . We next prove that $\text{FACT}(Q_{\text{cod}}^\Delta)$ is NPC, thus showing a strict separation in the complexities of the two problems.

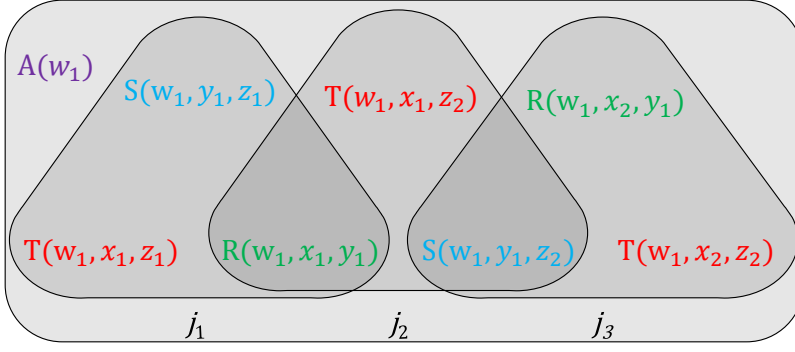


Figure 7.33: Theorem 7.9.4: Hardness gadget for Q_{cod}^{Δ}

Theorem 7.9.4 (minFACT is hard for queries with Co-Deactivated Triads) *FACT(Q) for a query Q with a co-deactivated triad is NPC.*

A slight modification of the gadget from Theorem 7.9.1 can be used to show that this class of queries is hard for minFACT, although it is easy for RES.

Proof Theorem 7.9.4. We construct a hardness gadget Figure 7.33 that can be used to build a reduction from IndSet. The hardness gadget is built with 3 witnesses. It is essentially the same hardness as Theorem 7.9.1, with the addition of a table that dominates all tables in the co-deactivated triad $A(w)$. We keep the value of the variable w fixed to w_1 in all gadgets. Then w_1 must be the root in the minFACT, and the endpoints now denote the root of a factorization assuming $A(w_1)$ has been factored out. We see that the endpoints $T(w, x, z)$ are independent of all other tables in the gadgets excluding $A(w)$. Since only a single $A(w)$ tuple will be used through all the partitions, joining these gadgets will not lead to any unexpected witnesses.

We see that the gadget has 2 minimal factorizations: both of which incur a penalty of 1 and use one of $T(w_1, x_1, z_1)$ or $T(w_1, x_2, z_2)$ as root. We can use these factorizations as graph orientations as in Theorem 7.9.1 and the rest of the reduction is identical. \square

7.10 Application: A complete approach for Approximate Probabilistic Inference

Definition 7.10.1 (Probabilistic query evaluation) *Given a Boolean query Q, a database \mathcal{D} , and a function p that assigns an independent probability to each tuple. Probabilistic query evaluation $\text{PROB}(Q, \mathcal{D}, p)$ computes the marginal probability $\mathbb{P}[Q, \mathcal{D}, p]$ (i.e., the probability that Q evaluates to true in a randomly chosen world).*

An important contribution of the database community to probabilistic inference has been the identification of tractable queries and database instances that allow exact evaluation in PTIME. Approaches towards identifying tractable cases are either at the query level (thus only look at the query and ignore the database [39, 40]) or at the data-level i.e. from the actual provenance polynomial [141, 147]. A practical concern with approaches that focus on tractable cases is that they provide only “partial” solutions; they offer none for the non-tractable cases which then

	Query-level	Data-level
Partial solution	[39]: dichotomy that has a PTIME solution iff the query is hierarchical	[141, 147]: PTIME solution iff the provenance polynomial is read-once (which includes [39] as special case)
Complete solution	[66]: PTIME approximation for any query (recovers the PTIME cases of [39], but approximates some cases of [141])	this paper : finding the minimum factorization includes all other three solutions [39, 66, 141] as special cases

Figure 7.34: Connection of the problem formulation and solutions in this chapter with prior results on exact or approximate solutions for probabilistic evaluation of sj-free conjunctive queries.

need separate approximate methods, usually based on Monte Carlo approximations [19, 39, 137] or anytime approximation schemes based on branch-and-bound provenance decomposition methods [56, 57, 92, 128]. A different line of work is “complete” in that it can answer all queries, but it is query-centric [65, 66] and thus *unnecessarily approximates* cases that would allow an exact PTIME solution if one looked at the concrete database instance. Interestingly, a solution to the “minimal provenance factorization” problem combines the best of both worlds (Figure 7.34): *a complete approach that includes all known tractable cases for exact inference as special cases*. We illustrate this in Example 7.10.1.

Example 7.10.1 (Provenance) Consider again the Boolean 2-star query $Q_2^* := R(x), S(x, y), T(y)$ over the database in Figure 3.1 (ignore tuple s_{13} for now) as in Example 3.2.1. Each tuple is annotated with a Boolean variable r_1, r_2, \dots , representing the independent event that the tuple is present in the database. The provenance φ is the Boolean expression that states which tuples need to be present for Q_2^* to be true:

$$\varphi = r_1 s_{11} t_1 \vee r_1 s_{12} t_2 \vee r_2 s_{23} t_3 \vee r_3 s_{33} t_3 \quad (7.3)$$

Here, the color blue indicates tuple variables that are repeated in the expression. Evaluating the probability of Q_2^* is #P-hard according to the dichotomy by Dalvi and Suciu for SJ-free CQs [39]. This means that for database instances of increasing sizes, the evaluation becomes infeasible, in general. However, the dichotomy does not take the actual database into account, and Roy et al. [141] and Sen et al. [147] later independently proposed a PTIME solution for particular database instances (including the one from this example) that allowed a read-once factorization. This is a factorized representation of the provenance polynomial in which every variable occurs once, and which can be found in PTIME in the size of the database:

$$\varphi' = r_1 (s_{11} t_1 \vee s_{12} t_2) \vee (r_2 s_{23} \vee r_3 s_{33}) t_3$$

Another approach called query dissociation [66] can evaluate it only approximately with an upper bound that is calculated with a “probabilistic query plan”

$$P = \pi_{\mathcal{C}_x}^p \bowtie^p [R(x), \pi_{\mathcal{C}_y}^p \bowtie^p [S(x, y), T(y)]] \quad (7.4)$$

where the probabilistic join operator $\bowtie^p[\dots]$ in prefix notation and the probabilistic project-away operator with duplicate elimination π^p compute the probability assuming that their input probabilities are independent [61]. This approach intuitively leads to a read-once upper-bound expression in which formerly repeated variables are now

replaced by fresh copies (indicated by the new indices):

$$\varphi'' = r_1(s_{11}t_1 \vee s_{12}t_2) \vee r_2s_{23}t'_{3,1} \vee r_3s_{33}t'_{3,2} \quad (7.5)$$

The exact approaches listed above are important milestones as they delineate which cases can be calculated efficiently or not. However, an important problem is that even the data-level approaches [141, 147] are only “partial”, i.e. when we make a minor modification to the database, they do not work anymore, as illustrated next.

Example 7.10.2 (continued) Consider again adding one tuple s_{13} to the database as in Example 7.2.1: it is shown as a red dashed line in the join graph of Figure 3.1. After this addition, the exact approaches based on query level [39] or on instance read-once formulas [141] do not work anymore.

An approach based on query dissociation with minimal query plans [66] would choose one of two query-level factorizations with either all tuples from R or all tuples from T as “root variables.” The solution with R ’s as root is:

$$\phi = r_1(s_{11}t_1 \vee s_{12}t_2 \vee s_{13}t_3) \vee r_2(s_{23}t_3) \vee r_3(s_{33}t_3)$$

This would lead to variable t_3 being repeated 3 times and expression size 13. Similarly, approaches based on Shannon expansion [128] need to start from such a factorization before repeatedly applying the expansion until arriving at a read-once expression.

However, there is an optimal factorization that repeats one variable only 2 times and has a total size of 12. Importantly, this factorization needs to use *different* root variables for different witnesses:

$$\phi' = r_1(s_{11}t_1 \vee s_{12}t_2 \vee s_{13}t_3) \vee (r_2s_{23} \vee r_3s_{33})t_3$$

Our idea is to try to come as close as possible to a read-once factorization in the general case by using dissociation-based bounds with the fewest number of repeated variables necessary. This achieves a complete solution (covering all queries and database instances). The problem is related to various topics in databases and is best understood in the modern formulation of provenance semirings [78, 79]. A minimal factorization can also be used as input to anytime or exact algorithms relying on repeated application of Shannon expansion [56]. Our problem generalizes the problem of determining whether a propositional formula is read-once to that of finding an expression of minimal size.

Example 7.10.3 (Example 7.10.1 continued) Consider the following plan

$$P' = \pi_{x,y}^P \bowtie^p [R(x), S(x, y), T(y)] \quad (7.6)$$

It corresponds to a query dissociation $\Delta' = (\{y\}, \emptyset, \{x\})$ whereas the plan shown in (7.4) corresponds to a dissociation $\Delta = (\emptyset, \emptyset, \{x\})$. Thus, $\Delta \leq \Delta'$ and we know from Theorem 7.4.3 that $\text{len}(\varphi(P^\Delta)) \leq \text{len}(\varphi(P^{\Delta'}))$ over any database. Let’s verify for the case of the database in Figure 3.1 without the red tuple. $\varphi(P^\Delta)$ corresponds to (7.5) with length 11 whereas $\varphi(P^{\Delta'})$ corresponds to (7.3) with length 12.

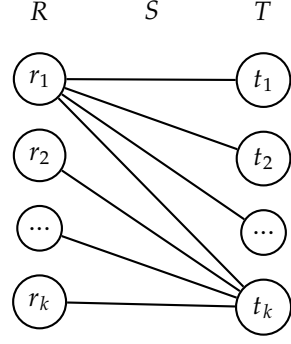


Figure 7.35: Subsection 7.10.1: Parameterized database instance for evaluating probabilistic inference with $2k$ probabilistic variables.

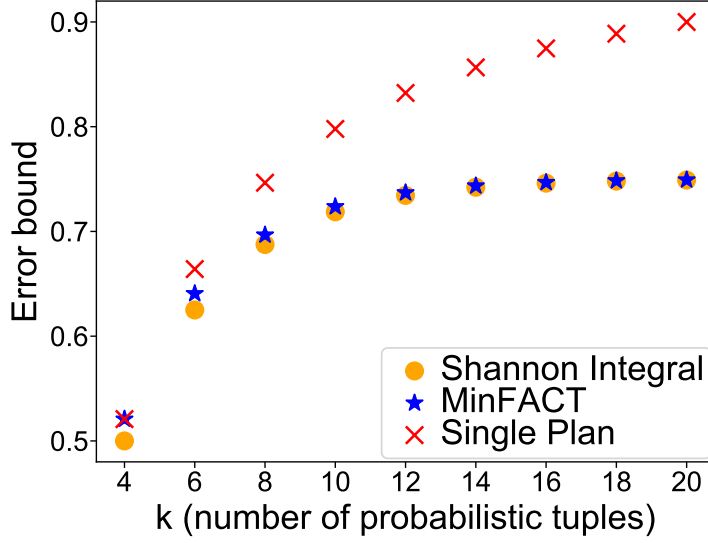


Figure 7.36: Subsection 7.10.1: Expected value of exact and approximate probabilistic inference comparison for an increasing number of tuples (x-axis is k).

7.10.1 A Proof-of-Concept for Improving Probabilistic Inference

Setup. With Figure 7.36, we next generalize our setup from Figure 3.1b: We again consider a database of format where one of the r tuples is connected to all t tuples, and vice versa (Figure 7.35). But instead of 3 tuples, we now have k tuples in tables R and T , respectively (and $2k - 1$ in S). We consider the query Q_3^∞ , which has 2 minimal query plans, and allow the R and T tables to take on probabilistic values (whereas tuples in S are deterministic, i.e. have probabilities 1). We investigate if the minimal factorization indeed leads to better probabilistic bounds as k increases.

Methods. Exact probabilistic inference is hard for this case, and we use Shannon expansion to obtain the exact probabilities as a baseline. As comparison, we use the minimal factorization of this expression (which is not read-once) and the expression obtained by evaluating the database under a single dissociated query plan.

Evaluation. For all three expressions, we calculate the expected probability assuming that each of the $2k$ tuples can take on a probability independently sampled from a uniform random probability in $[0, 1]$. To calculate that expected value, we integrate over all variables. Thus, for an instance with just $k = 2$ tuples in R and $2 T$, respectively, we perform a quadruple integral. We performed these integrations in Wolfram Mathematica [138] and were only able to scale to 20 repeated integrals.

Results. We plot the integral of the expected probability over the whole space on the y-axis, and the x-axis shows the number k of probabilistic variables. We see that with increasing variables k , minFACT converges to the exact probabilistic inference given by Shannon expansion, while a single plan dissociation gives increasingly worse approximations.

7.11 Optimizations for computing minFACT

We think of minFACT as the problem of assigning VEOs to each witness to optimize len. Each witness chooses between $|\mathbb{mveo}|$ VEOs. However, it is not necessary that each must choose from the same set. For some witnesses, some VEOs can be directly eliminated (or pruned) and thus their set of choices for those witnesses is reduced. The pruning is done based on “degree statistics” which summarize information about how many witnesses share a given subset of the domain values of a given witness. This optimization can be thought of as a preprocessing step for both the ILP and the MFMC-based algorithm.

Example 7.11.1 (VEO Pruning) Consider query Q_3^* query and a database instance that contains 2 witnesses: $\mathbf{w}_1 = (r_1, s_1, t_1, w_{111})$ and $\mathbf{w}_2 = (r_1, s_2, t_2, w_{122})$ (or $\mathbf{w}_1 = (x_1, y_1, z_1)$ and $\mathbf{w}_2 = (x_1, y_2, z_2)$ in the domain values perspective). The only tuple they have in common is r_1 and the only variable they have in common is x_1 . We say that the degree (or count) of x_1 in \mathbf{w}_1 is 2. We can intuitively see that a minimal factorization would have r_1 as a “root” i.e. would use a VEO in which the x_1 is a prefix.

Degree Statistics. We leverage degree statistics of tuples participating in a witness to filter its potential query plan assignments. Given a witness \mathbf{w} and strict subset of attributes $\{x_1, x_2, \dots, x_k\}$, we define $\text{count}(x_1, x_2, \dots, x_k)$ as the number of witnesses that have the same valuation of these attributes. Each degree statistic can be calculated by a simple group-by and aggregation. We calculate the degree statistics for all sets in the powerset of $\text{var}(Q)$. The number of such statistics is exponential in the size of the query, but polynomial in the data size, and in practice the time to compute all such statistics is small (see experimental figures).

Example 7.11.2 (Example 7.11.1 continued) We compute the degree statistics over the example: \mathbf{w}_1 has $c_x = 2, c_y = 1, c_z = 1, c_{xy} = 1, c_{yz} = 1$ and $c_{xz} = 1$. \mathbf{w}_2 has the exact same statistics.

Pruning Rule. Given two VEOs v_1 and v_2 with the roots of the trees being composed of the set of variables \mathbf{r}_1 and \mathbf{r}_2 , we can apply the following rules:

If $\text{count}(\mathbf{r}_1) == \text{count}(\mathbf{r}_1 + \mathbf{r}_2)$ and $\text{count}(\mathbf{r}_1) < \text{count}(\mathbf{r}_2)$:
 Eliminate Plan v_1
 Elif $\text{count}(\mathbf{r}_2) == \text{count}(\mathbf{r}_1 + \mathbf{r}_2)$ and $\text{count}(\mathbf{r}_2) < \text{count}(\mathbf{r}_1)$:
 Eliminate Plan v_2
 Elif $\text{count}(\mathbf{r}_1) == \text{count}(\mathbf{r}_2) == \text{count}(\mathbf{r}_1 + \mathbf{r}_2)$ and
 $\text{count}(\mathbf{r}_2) = \text{count}(\mathbf{r}_1)$:
 Assign Equivalence Class $\{v_1, v_2\}$

Example 7.11.3 (Pruning Rule) Continuing our setup, we see that for witness $x_1y_1z_1$, $\text{count}(y_1) = \text{count}(x_1y_1)$ and that $\text{count}(y_1) < \text{count}(x_1)$. Thus, we can eliminate plans with the root y_1 , i.e. the two plans $y \leftarrow x \leftarrow z$ and $y \leftarrow z \leftarrow x$.

We can apply the pruning rule again in the same manner to eliminate two plans with z_1 as the root. The same pruning will also apply to w_2 .

Recursive Application of Pruning Rule. Consider 2 VEOs v_1 and v_2 with the same root r , but then we compare all the children subtrees of r and apply the pruning rule recursively to further prune between plans with same root.

Example 7.11.4 (Recursive Pruning) After the pruning in the last step, we are left with two plans $x \leftarrow y \leftarrow z$ and $x \leftarrow z \leftarrow y$. We now prune at tree level 2. We see that $\text{count}(xy) = \text{count}(xz) = \text{count}(xyz)$. Thus, we assign an equivalence class $\{xy, xz\}$

Equivalence Classes. At the end of all the pruning, we look at our collected equivalence classes. If two VEOs are in an equivalence class, this implies that all else being equal, both plans will lead to equal len. Thus, if we have multiple plans from the same equivalence class, we remove all but one arbitrarily.

Example 7.11.5 (Equivalence Classes) Since $\{xy, xz\}$ are in an equivalence class, and we have finished our pruning, we arbitrarily remove plans with the prefix xz . Thus, we are left with sole plan $x \leftarrow y \leftarrow z$. We were able to reduce both witnesses from 6 to 1 plan in this very easy example. (Note that in other cases, different witnesses may have different degree statistics and hence reach a different set of pruned plans).

Proof of Correctness of Optimization.

1. **Correctness of the Pruning Rule:** Consider the case where with VEOs v_1 and v_2 where $\text{count}(r_1) = \text{count}(r_1 + r_2)$ and $\text{count}(r_1) < \text{count}(r_2)$. Let us assume that a witness w_1 necessarily needs v_1 for a minimal factorization of the instance. We take all witnesses that share r_1 and r_2 and if they use v_1 as their VEO, we switch them to v_2 . Notice that $\text{count}(r_1) = \text{count}(r_1 + r_2)$ implies that whichever witnesses share r_1 , also share r_2 . So switching a subset of these witnesses over from v_2 to v_1 can repeat no additional variables since it is never worse to factor out v_2 first (which they all share anyway). For witnesses that contain r_1 and r_2 but did not use v_1 plan, they could possibly incur penalties on a subset of r_1 that they used as prefixes. However, as $\text{count}(r_1) < \text{count}(r_2)$, there is a greater prevalence of r_2 variables that can be repeated and so they are never negatively affected by the switch. Thus switching to v_2 can only lead to better or equal factorization hence the pruning rule is always correct.
2. **Correctness of Equivalence Classes:** We can use the same logic as that of the pruning rule to show that when $\text{count}(r_1) = \text{count}(r_2) = \text{count}(r_1 + r_2)$ and $\text{count}(r_2) = \text{count}(r_1)$, then neither v_1 nor v_2 can be worse than each other.
3. **Correctness of Repeated Application:** If each pruning rule application is correct, then for every plan that is pruned, there exists an unpruned plan that leads to a better or equal len. Since the rules

are applied linearly and not in parallel, we can safely repeatedly apply the rule until applicable.

7.12 Experiments

We implemented our ILP, LP relaxation, and the MFMC-based algorithm as LP^{¶¶} by using Python 3.8.5 for the pre-processing of the provenance and creating the problem encoding, and then Gurobi Optimizer 8.1.0 [83] for solving the respective optimization problems. Our goals are to evaluate 1) the running times, and 2) the size of resulting factorizations. In this section we illustrate our results and show three interesting takeaways: 1) ILP is, as expected, exponential in the size of the program, in general. The MFMC-based algorithm is not and can thus speed up the evaluation quite drastically. While it comes with no guarantees for hard cases, it approximates the minimum size quite well. 2) For queries that we have shown are in PTIME, the ILP solver is comparably fast as the LP and all algorithms provide the correct solution. 3) The optimizations described in Section 7.11 further speed up the evaluations in a way that Gurobi cannot.

For both the ILP and MFMC-based algorithms, the *optimizations* are applied during a preprocessing step and simplify the resulting ILP and MFMC formulations (Section 7.11). The intuition is that based on how each witness interacts with other witnesses (i.e. based on some *degree statistics* obtained by simple group-bys), we can mark certain minimal query plans as unnecessary for a minimal factorization.^{***} Being able to find such optimization is notable since it implies we are *identifying and leveraging structural properties of the problem* based on the minimal query plans that are difficult to extract in the same time by the state-of-the-art ILP solver Gurobi.

7.12.1 Experimental Setup

Software and Hardware. We implement the algorithms using Python 3.8.5. and solve the respective optimization problems with Gurobi Optimizer 8.1.0 [83], a commercial and highly optimized solver. The experiments are run on an Intel[®] Core[™] i7-1065G7 CPU @ 1.30GHz machine with 132 GB RAM.

Experimental Protocol. We focus on 5 queries: the two hard queries Q_3^* and Q_4^Δ , the two easy queries Q_A^Δ , Q_4^∞ , and finally query Q_5^∞ which we hypothesize to be easy. We first create a random database instance by fixing a number of tuples to sample, and sampling attribute values independently for each attribute (and removing duplicates) and parameterized by total tuples in the instance. We then run a provenance query and store the resulting provenance (and its size). On this database instance, we then run our five algorithms: ILP, ILP (OPT), MIN-CUT, MIN-CUT (OPT), and LP. Here (OPT) refers to executing an algorithm after applying preprocessing optimizations described in Section 7.11.

^{¶¶} The MFMC-based algorithm can be encoded as an LP since the min-cut problem itself can be encoded as an LP

^{***} We also believe that a similar pruning algorithm will be instrumental in achieving a PTIME factorization flow graph algorithm for other linear queries by eliminating nodes that cause leakage.

7.12.2 Experimental Results

In this section, we empirically evaluate 5 queries discussed throughout the paper and not only gather some empirical support about the complexities we show in the paper but also gain some insight into the effectiveness of the optimizations and if advanced ILP Solvers can detect our PTIME Cases.

The “Penalty” of a factorization is the length (len) minus the number of different tuple variables. The Approximation Ratio (AR) measures the relative increase of the penalty of an algorithm to the minimal penalty. The ILP and ILP (OPT) algorithms are guaranteed to find the optimal solution, i.e. +0%. While the MFMC-based algorithm provides no guarantees for NPC queries, we show that it provides a good approximation.

We can see that the MFMC-based algorithm contains more constraints and variables than the ILP problem (we must add a flow conservation constraint for each node in the graph, which includes nodes for corresponding ILP variables, plus connector nodes). However, since it is solved as an LP optimization problem instead of an ILP optimization, it is much faster. We also notice that the optimized algorithm decreases the number of constraints in the MFMC-based algorithm by 15%. The number of constraints does not decrease in the ILP, but because the Query Plan Constraints involve fewer variables, the constraints are less complex and the ILP can be solved faster.

Q_3^* **Figure 7.37a**. The 3-Star query is a hard query with an active triad and $|mveo| = 6$. For this query, the provenance computation time and problem creation time are small compared to the time needed to solve the ILP. The *optimized preprocessing step* reduces the time for the ILP solution by a tenth, while the MFMC-based algorithm is ≈ 60 times faster than the ILP. This is expected as the min-cut algorithm is PTIME while the ILP is not. We see here that the Linear Program is faster than the Flow algorithm, however, it has a worse approximation ratio (+2.781% instead of +0.632% - but still within the proved bound of 6 for this query).

Q^Δ **(Figure 7.37b)**. The triangle query is a hard query with $|mveo| = 3$. We see in the figure that since there are just 3 minimal VEOs, the effect of the optimization is not very much. In fact, for the MFMC-based algorithm, the longer time to create the optimized version is not paid off in the solve time. However, as expected for a hard query, we see that the PTIME MFMC-based algorithm is much faster than the exact ILP. The MFMC-based algorithm enables us to get a very close approximation in less than one-fourth of the time. In this case, the Linear Program, also a PTIME algorithm, is slower than the MFMC-based algorithm but gives an exact solution!

Q_A^Δ **(Figure 7.37c)**. Next, we look at the very structurally similar but easy query, Q_A^Δ with $|mveo| = 3$. Here we see surprisingly that the ILP is faster than the MFMC-based algorithm! Notice that for this query we have shown that both algorithms are exact. This is because an optimized ILP Solver like Gurobi can leverage the fact that LP polytope has integral optimal vertices and solve the ILP in PTIME (even though the ILP constraints do not follow traditionally PTIME structures like Total Unimodularity, it leverages some structure in the matrix that makes it PTIME solvable). Since there are indeed more variables and constraints in the MFMC-based algorithm, the ILP turns out to be faster.

Q_4^∞ (Figure 7.37d). The 4 chain query is PTIME like Q_A^Δ but with $|mveo|=5$ and has a similar graph to Q_A^Δ . Here again, the guarantee of the MFMC-based algorithm being exact is fulfilled, the pruning offers some benefit, and the ILP is faster than the MFMC-based algorithm due to the smaller problem size and some innate structure discovered by the solver. The Linear Program, which has always returned the optimal solution, is significantly faster.

Q_5^∞ (Figure 7.37e). The 5 chain query has $|mveo|=14$, and we believe it to be easy, but do not yet have proof for the same. In this case, too, we believe that the Gurobi solver has some tricks that make the ILP run faster than exponential. (Notice that a bigger instance with more than double the size of $mveo$ is solved faster than Q_3^\star in Figure 7.37a). In addition, the Linear Program is optimal. This backs up our hypothesis that the query is in PTIME. While the MFMC-based algorithm is not optimal in this case, we see that applying our pruning rules helps eliminate leakage paths and gives us a better approximation as well. We hypothesize that there exists a set of pruning rules that make the MFMC-based algorithm optimal for all Linear Queries (which we hypothesize are all in PTIME).

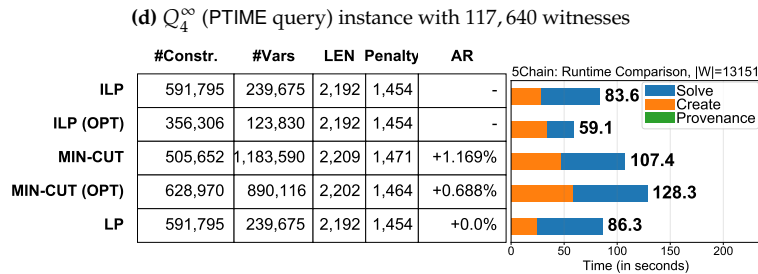
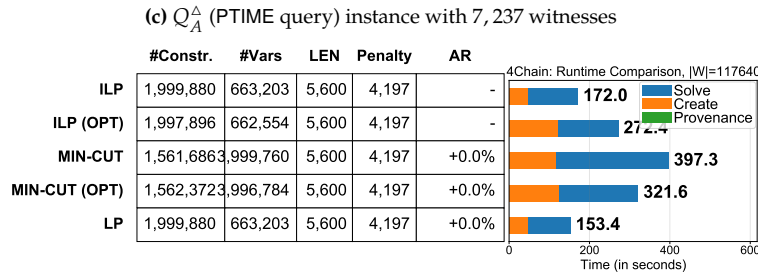
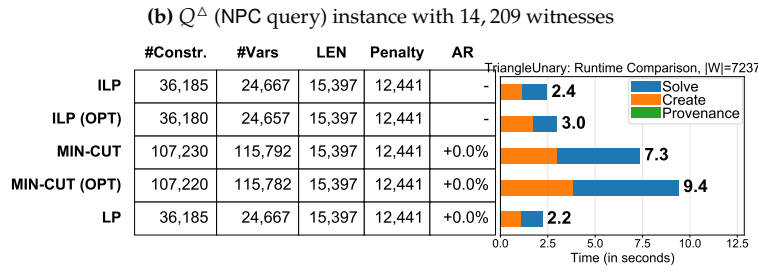
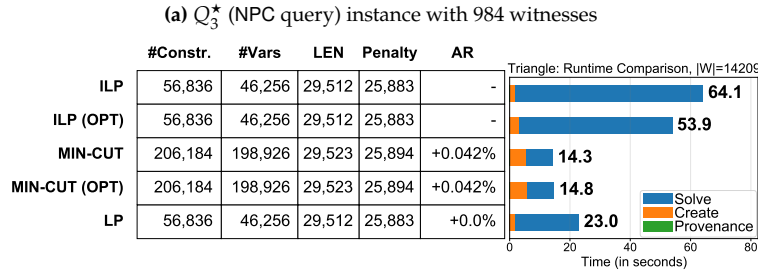
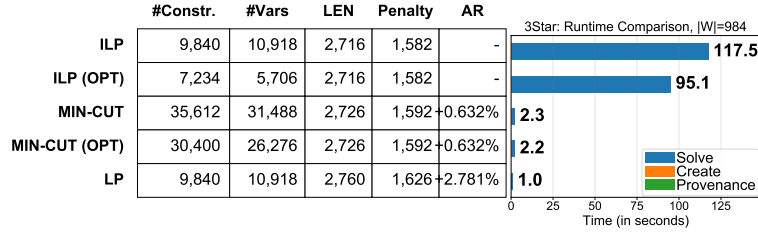


Figure 7.37: Experimental study of time and factorization length for 5 queries over all algorithms.

7.13 Chapter Summary

We propose a unified algorithm for minimizing the size of provenance polynomials for sj-free CQs. We show that our problem is NPC and thus in a lower complexity class than the general Minimum Equivalent Expression (MEE) problem. Key to our formulation is a way to systematically constrain a space of possible minimum factorizations thus allowing us to build an ILP, and connecting minimal variable elimination orders to minimal query plans developed in the context of probabilistic databases. We complement our hardness results with two unified PTIME algorithms that can recover *exact solutions* to a strict superset of *all prior known tractable cases*.

Automatic Hardness Gadgets

8

This chapter deals with a question that is tangential to the other parts of the dissertation, but a question we believe is interesting in its own right. Even though unified algorithms offer theoretical guarantees for all known tractable cases, and one does not hope for tractable algorithms for NP-Hard cases - there still is usually left the challenge of showing that a problem is NP-Hard. This is usually a difficult, creative endeavor, and it can be a roadblock in proving complexity dichotomies. Our goal in this chapter is to use computational tools to automate this process, and thus discover new complexity reductions.

This chapter focuses again on the problem of resilience explored in Chapter 4, but we believe that the ideas in this chapter can be naturally extended beyond this specific problem.

8.1 Related Work

Prior work in Computationally Generated Hardness Gadgets. Although it is a very natural idea to use computational tools to help the often challenging task of finding hardness reductions, we are aware of very few works in this direction. The earliest approach we are aware of is from 2010 [36]. They use SAT solvers (unlike our approach with DLP solvers), which leads to multiple independent solver calls (since the SAT solvers have lower expressibility). More recent work [77] adopts a formalism called “cookbook reductions”, which are in some ways more general than the reductions in this chapter (we focus only on edge gadgets), however are limited to decision problems. The focus of their work is on gadget verification and use in a classroom, rather than coming up with new complexity reductions (as we do).

Disjunctive Logic Programs (DLPs). Disjunctive Logic Programs are Logic Programs that allow disjunction in the head of a rule [41, 135]. DLPs have been shown to be Σ_p^2 -complete [50, 51], and thus are more expressive than Logic Programs without disjunctions that are NPC. The key to higher expressivity is the non-obvious *saturation* technique that can check if *all* possible assignments satisfy a given property [49]. Logic Programs have been used for database repairs [69] and for determining the responsibility of tuples in a database [16]. We go beyond this to build a DLP that searches a certificate that proves that solving the resilience / responsibility problem is NPC for a given query. We represent our DLP as an Answer Set Program (ASP) [52] and use *clingo* [68] to solve it.

8.2 Automatic Hardness Gadgets for Resilience

We introduce a Disjunctive Logic Program DLP[RESIJP] that finds IJPs to prove hardness for RES. Each DLP requires Q , a domain d (which bounds the size of the IJP), and two endpoints \mathcal{S}, \mathcal{T} .^{*} DLP[RESIJP] programs

^{*} Since the number of possible endpoint configurations is polynomial in the query size, we can simply run parallel programs for different endpoints as input. Notice that endpoints $e_1 = \{A(1)\}, e_2 = \{A(2)\}$ is exactly the same as $e_1 = \{A(3)\}, e_2 = \{A(4)\}$ since the actual

8.1 Related Work	133
8.2 Automatic Hardness Gadgets for Resilience . .	133
8.3 Scalability Experiment of newly found hard query .	135
8.4 Chapter Summary	136

This chapter is based on Section 7.2 of: Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *Proc. ACM Manag. Data* 1, 4, Article 228 (December 2023), 27 pages. <https://doi.org/10.1145/3626715> [110].

- [36]: Crouch, Immerman, and Moss (Fields of Logic and Computation, 2010), ‘Finding Reductions Automatically’. doi:10.1007/978-3-642-15025-8_10
- [77]: Grange, Vehlken, Vortmeier, and Zeume (MFCS, 2024), ‘Specification and Automatic Verification of Computational Reductions’. doi:10.4230/LIPICS.MFCS.2024.56
- [41]: Dantsin, Eiter, Gottlob, and Voronkov (CSUR, 2001), ‘Complexity and Expressive Power of Logic Programming’. doi:10.1145/502807.502810
- [135]: Przymusiński (New Generation Computing, 1991), ‘Stable Semantics for Disjunctive Programs’. doi:10.1007/BF03037171
- [50]: Eiter and Gottlob (AMAI, 1995), ‘On the computational cost of disjunctive logic programming: Propositional case’. doi:10.1007/bf01536399
- [51]: Eiter, Gottlob, and Mannila (TODS, 1997), ‘Disjunctive Datalog’. doi:10.1145/261124.261126
- [49]: Eiter and Gottlob (TCS, 1993), ‘Propositional circumscription and extended closed-world reasoning are Π_2^P -complete’. doi:10.1016/0304-3975(93)90073-3
- [69]: Gelfond and Kahl (2014), *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. doi:10.1017/cbo9781139342124
- [16]: Bertossi (Knowledge and Information Systems, 2021), ‘Specifying and computing causes for query answers in databases via database repairs and repair-programs’. doi:10.1007/s10115-020-01516-6
- [52]: Eiter, Ianni, and Krennwallner (2009), *Answer set programming: A primer*. doi:10.1007/978-3-642-03754-2_2
- [68]: Gebser, Kaufmann, Kaminski, Ostrowski, Schaub, and Schneider (AI Communications, 2011), ‘Potassco: The Potsdam answer set solving collection’. doi:10.3233/aic-2011-0491

are generated automatically for a given input, are short (200-300 lines depending on the query) and leverage many key technical insights used to model DLPs.

The goal of DLP[RESIJP] is to find a database that fulfills the conditions of Definition 4.5.2. The search space is a database with all possible tuples given domain d (thus of size $\mathcal{O}(d^a)$ where a is the maximum arity of any relation). Each tuple in the search space must be either “picked” in the target database or not. The constraints of our definition are modeled as disjunctive rules with negation. We solve our DLP with the open-source ASP solver *clingo* [68] which uses an enhancement of the DPLL algorithm [43] (used in SAT solvers) and works far faster in practice than a brute force approach. Here we talk about the overall structure and intuition, and a concrete end to end example is available in Chapter B.

1. **Search Space:** For all relations in Q , we initialize all possible tuples permitted in domain d as input facts and provide them with an additional tuple id (TID). Thus, each relation R has a corresponding relation in the program with arity $(R)^d$ facts.
2. **“Guess” an IJP:** Each tuple either participates in the IJP or not. We follow the Guess-Check methodology [53] and use a relation $indb(R, TID, I)$ to “guess” for each tuple whether it is in the IJP database or not. Here R stands for a relation and together with TID uniquely identifies a tuple. The binary value I is 1 if the tuple is in the IJP, and 0 otherwise.
3. **Enforce JP endpoint conditions:** Since the endpoints are considered “input”, we do not need to check condition (3i) for the JP endpoints (Definition 4.5.1). However, we need to verify condition (3ii) as it depends on the other tuples in the IJP and translate the condition directly into a logic rule.
4. **Calculate Resilience using “Saturation”:** We solve a problem that is NPC (i.e. check that there is a valid contingency set of size c), and a problem that is co-NP-complete (i.e. there is no valid contingency set of size $c - 1$). For solving the NP problem we use the guess-check methodology and to solve the co-NP problem, we use the saturation technique.
5. **Enforce OR-property:** We calculate resilience for 4 databases using the previous step: our original “guess”, and the guess with either or both endpoints removed. The removal of endpoints here simply implies defining a new relation that has all tuples of $indb$ except the removed endpoint tuples.
6. **Enforce non-leaking composition:** We define a mapping relation to create 3 isomorphs of the tuples in $indb$. We combine them into one database and check that computing query Q results in exactly 3 times the number of original witnesses.
7. **(Optional) Minimize the size of the IJP:** To generate smaller certificates that are more human-readable, we simply minimize the number of witnesses in the IJP. We use weak constraints [52] to perform this optimization.

[68]: Gebser, Kaufmann, Kaminski, Ostrowski, Schaub, and Schneider (AI Communications, 2011), ‘Potassco: The Potsdam answer set solving collection’. doi:10.3233/aic-2011-0491

[43]: Davis, Logemann, and Loveland (CACM, 1962), ‘A Machine Program for Theorem-Proving’. doi:10.1145/368273.368557

[53]: Eiter and Polleres (TPLP, 2006), ‘Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications’. doi:10.1017/s1471068405002577

[52]: Eiter, Ianni, and Krennwallner (2009), *Answer set programming: A primer*. doi:10.1007/978-3-642-03754-2_2

Corollary 8.2.1 (Sufficient hardness condition via Hardness Gadgets) *If there is a domain d and endpoints \mathcal{S}, \mathcal{T} such that $\text{DLP}[\text{RESIJP}(Q, d, \mathcal{S}, \mathcal{T})]$ is satisfiable, then $\text{RES}(Q)$ is NPC.*

value does not matter. In practice, we used any subset of endogenous tuples from a canonical database that can be shared across two witnesses without creating another witness.

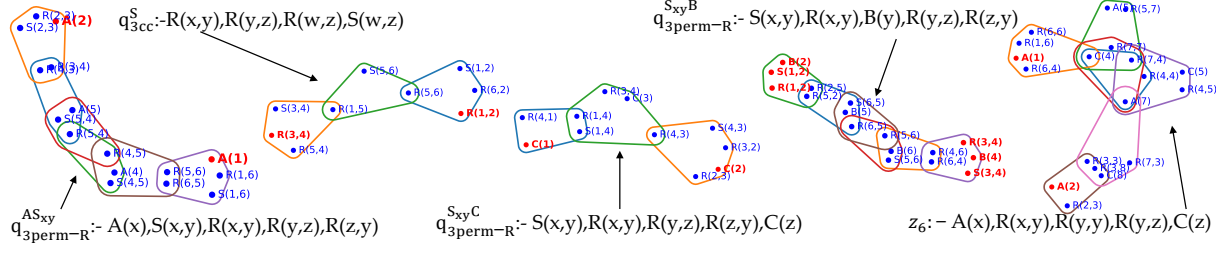


Figure 8.1: Automatically generated and visualized IJPs for 5 previously open queries. The nodes corresponding to tuples in $\mathcal{S} \cup \mathcal{T}$ are in red.

Corollary 8.2.2 (Complexity bound of finding Hardness Gadgets) *It is in Σ_p^2 of d to check if a query Q can form an IJP of domain size d or less.*

The guarantees of our DLP is one-sided: if it finds a certificate, then resilience of the query is guaranteed to be NPC. If it does not provide a certificate, then we have no guarantee. So far we have not found any query that is known to be hard and for which our DLP could not create a certificate for $d = 3 \cdot |\text{var}(Q)|$. We conjecture it is not only a sufficient but also complete algorithm for $d = 7 \cdot |\text{var}(Q)|$ (i.e. if the algorithm does not find a certificate for $d = 7 \cdot |\text{var}(Q)|$, then the query is in PTIME).

Conjecture 8.2.3 (Necessary hardness condition) *If there are no endpoints \mathcal{S}, \mathcal{T} such that $\text{DLP}[\text{RESIJP}(Q, d, \mathcal{S}, \mathcal{T})]$ is satisfiable for domain $d = 7 \cdot |\text{var}(Q)|$, then $\text{RES}(Q)$ is in PTIME.*

8.3 Scalability Experiment of newly found hard query

We run an experiment with synthetic data using the same experimental setup described in Section 7.12. The purpose here is to see if the hardness of the query is obvious from the performance of a unified ILP formulation. Surprisingly, the answer turns out to be no, further motivating the need of counter-intuitive hardness certificates to understand the tractability landscapes of RDM problems.

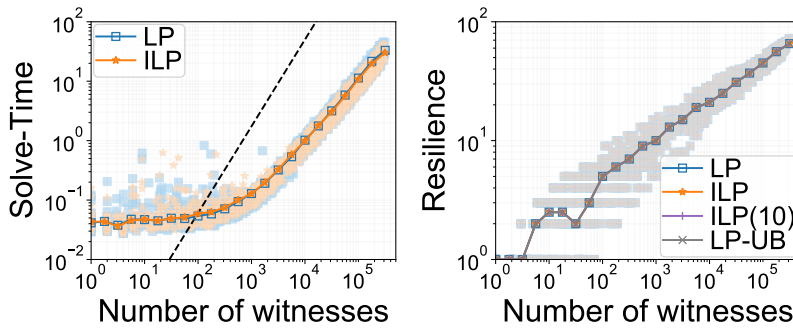


Figure 8.2: RES for a newly proven hard SJ query.

Self-Join Queries with newly founded hardness. Figure 8.2 investigates z_6 whose complexity we proved in to be hard automatically. Although resilience for this query is hard, it is unlikely to create a random database instance where solving resilience is actually difficult. Although the domain is pretty dense and the database instance large, for all experiments

we run, the LP solution is integral and identical to the ILP solution. However, by using our IJP, we could create an artificial synthetic database with 21 witnesses for which the LP solution is fractional.

8.4 Chapter Summary

This chapter showed a sample of an interesting research direction: can we use computational tools like DLP solvers to solve open complexity problems in databases? We notice that the semantic properties we specify in the definition of IJPs and in the DLP, are dependent on the source problem we reduce from (Vertex Cover), but not of the problem we are trying to prove hardness for (resilience for a specific query). Thus, the “OR-property” and “non-leaking composition” are a requirement for any reduction from vertex cover, even for problems different from resilience. This leads us to believe that these ideas are extensible beyond resilience, or even RDM problems, and maybe widely useful in different contexts to discover hardness gadgets.

The results in this dissertation introduce and validate a new paradigm to optimize Reverse Data Management queries in a “coarse-grained instance optimal” manner. We show such unified algorithms by designing ILP formulations such that we are guaranteed that their relaxations are tight for all known tractable classes of inputs. Thus, our unified algorithms are guaranteed to terminate in PTIME for all tractable instances, doing away with the need for specialized algorithms for different tractability criteria. We show that our proposed framework is theoretically grounded, scalable and practical. However, there are still many interesting directions to build on this work, which we outline in the rest of the chapter.

9.1 Future Directions 137

9.1 Future Directions

The dissertation leads to several natural questions and open directions for future research.

Complexity Dichotomies for various RDM problems. Complexity dichotomies for various RDM problems are still only known for the restricted setting of self-join-free conjunctive queries and do not extend to queries with self-joins or unions. While the ILP formalisms proposed in this dissertation are correct and complete even for queries with self-joins and unions, the complexity of these problems is still open. We conjecture that our unified algorithms will be able to solve tractable cases discovered in the future in PTIME as well, and that the LP relaxation will be tight for such tractable cases. We hope that the techniques developed in this dissertation can aid in proving complexity dichotomies for this broader class of queries - both by using our unified algorithms as “computational hints” and by using our new proof techniques to prove the tractability of new cases. A further direction of research could be to automatically search for tractable algorithms (if they exist) by looking for correct flow representations of the problem.

RDM problems for queries beyond Conjunctive Queries. The focus of this dissertation has been limited to Conjunctive Queries, with and without self-joins. In [Chapter 6](#), we showed that our unified algorithms capture unions of conjunctive queries as well. However, in practice, one may want to ask Reverse Questions over more complicated forward transformations. An interesting and useful class of queries to study such problems are *Regular Path Queries*. Although they are more restricted than Conjunctive Queries in that they are restricted to relations of binary arity and path queries, they are more general since they allow disjunctions and infinite queries through the Kleene Star. Recent work studies the complexity of Resilience for Regular Path Queries [8]. It would be interesting to study the suite of RDM problems for Regular Path Queries, or higher order logics, finding how the complexity landscape changes. A more applied direction is to extend these problems to interpretable Machine Learning pipelines and transformations.

Building a system for RDM problems. While the focus of this dissertation is on theoretically and practically efficient algorithms for RDM problems, building a large scale system that can solve RDM problems in practice is

[8]: Amarilli, Gatterbauer, Makhija, and Monet (PODS, 2025), ‘Resilience for Regular Path Queries: Towards a Complexity Classification’. doi:10.1145/3725245

a natural next step. There are many research and engineering challenges in building such a system, and such a system could learn from existing knowledge in building query optimizers to similarly include various heuristics, fast indexing, and other optimizations to make the system efficient in practice. Another significant open question is designing a user interface / query language for RDM problems that is intuitive and easy to use for non-experts.

Minimal-sized Circuits. A natural question is to extend the results of minimal factorization of provenance formulas to minimal-sized circuits. Circuits are a strictly more succinct representation of provenance, and have attracted considerable attention in the database theory community. Some work has been done on the *asymptotic* size of factorized circuits [55], however tight bounds remain unknown and the tractability of finding instance optimal sized circuits remains open. A separate related question is to find the minimum size formula / circuit for a given provenance formula under different semirings (i.e., whether the semi-ring operations are absorptive, idempotent, etc.)

[55]: Fan, Koutris, and Zhao (PACMOD, 2024), ‘Tight Bounds of Circuits for Sum-Product Queries’. doi:10.1145/3651588

Automatic Hardness Gadgets beyond Resilience. The automatic hardness gadget-finder developed in this dissertation is a general method that we believe can be extended beyond resilience problems. This is due to the fact that neither the semantic specification of NP-Hardness nor the Disjunctive Logic Program used to generate the reductions are deeply tied to the resilience problem. Extending this method to other RDM problems could naturally lead to many avenues of future research: leading not only to new hardness results, but also better understanding of linear reductions (which are the kinds of reductions that are found by the gadget-finder) and semantic specifications of hardness gadgets for various problems.

Fine Grained Complexity. This dissertation focused on “coarse-grained instance optimality” as a bridge to true instance optimality, of which few and far results are known. Recent foundational work in theoretical computer science on fine-grained complexity [159] could be used to further specify the complexity results shown in this dissertation. It remains open if all tractable classes for resilience, for example, are within the same fine-grained complexity class or not.

[159]: Williams (ICM, 2019), ‘On some Fine-grained Questions in Algorithms and Complexity’. doi:10.1142/9789813272880_0188

APPENDIX

A Nomenclature

The Notation Table (Table A.1) contains common nomenclature, and Query Table (Table A.2) lists example queries used through the paper.

Table A.1: Notation Table

Symbol	Definition
Q	Conjunctive query (CQ)
R, S, T, U	Relations
x, y, z	Query variables
$\text{var}(R)$	the set of variables in atom / relation / formula X
$\text{at}(x_j)$	set of atoms that contain variable x_j
\mathcal{D}	Database Instance, i.e. a set of tables
r_i, s_i, t_i, u_i	tuple identifiers
N	size of database $ D $
$\mathcal{D} \models Q$	Database \mathcal{D} satisfies query Q
$\mathcal{D} \not\models Q$	Database \mathcal{D} does not satisfy query Q
$Q(D)$	A view representing the evaluation of query Q on database D
$ Q(\mathcal{D}) $	The number of tuples in the view $Q(\mathcal{D})$
\mathbf{w}	Witness
W	Set of witnesses $W = \text{witnesses}(Q, D)$
m	Number of atoms in a query Q
k	Number of tables in Q ($= m$ in a self-join-free query)
Γ	A set of tuples (usually to be deleted from a database <i>i.e.</i> a contingency set)
E	A set of exogenous tuples
\mathbf{x}	unordered set or ordered tuple
\mathbf{a}/\mathbf{x}	substitute values \mathbf{a} for variables \mathbf{x}
$Q[\mathbf{x}]$	indicates that \mathbf{x} represents the set of all existentially quantified variables for Boolean query Q
\mathbb{Q}	An ordered set of queries
Q^i	The i -th query in \mathbb{Q}
Γ	A set of tuples (usually denoted a set of tuples to be deleted from the database)
$ \Delta Q(\mathcal{D}, \Gamma) $	$ Q(\mathcal{D}) - Q(\mathcal{D} - \Gamma) $
$X[v]$	A binary variable in an (Integer) Linear Program corresponding to a variable v
$X[v]$	A variable in an (Integer) Linear Program
\mathcal{S}, \mathcal{T}	Set of start and terminal endpoints of a JP
$\text{DLP}[\text{RESIJP}]$	A DLP to find IJPs for queries
φ, ψ	propositional formulas / expressions
$\text{VE0}(Q)$	set of all legal VE0s for Q
$\text{mveo}(Q)$	set of minimal VE0s for Q
$k = \text{mveo}(Q) $	number of minimal VE0s
$v\langle \mathbf{w} \rangle$	a VE0 instance of VE0 v over witness \mathbf{w}
$\text{var}(g_i)$	set of variables of a query q or atom g_i
P	query plan
\mathcal{P}	set of query plans
F	flow graph
$\bowtie(\dots)$	provenance join operator in prefix notation
$\pi_{\mathbf{x}}, \pi_{\neg \mathbf{y}}$	provenance project operators: onto \mathbf{x} , or project \mathbf{y} away

Continued on next page

Table A.1 – continued from previous page

Symbol	Definition
len	Length of a Factorization
QPV	Query Plan Variables of an ILP
PV	Prefix Variables of an ILP
$q[\dots]$	a ILP decision query plan variable
$p[\dots]$	a ILP decision prefix variable
c	weight (or cost) of variables in the ILP / nodes in the Factorization Flow Graph
Ω	An Ordering of mveo chosen for MFMC based algorithm
(v_1, v_2, \dots, v_k)	An ordered list of VE0s, VE0FFs or any other set of objects
$HVar(P)$	set of head variables of a query q or a plan P
$EVar(q)$	set of existential variables: $EVar(q) = \text{var}(q) - HVar(q)$
$\mathbb{P}[\phi]$	probability of a Boolean expression
Δ	collection of sets of variables $\Delta = (\mathbf{y}_1, \dots, \mathbf{y}_m)$
$R_i^{y_i}$	dissociated relation $R_i(\mathbf{x}_i)$ on variables \mathbf{y}_i : $R_i(\mathbf{x}_i, \mathbf{y}_i)$
q^Δ	dissociated query

Table A.2: Example Queries

Query	Definition
Q_2^∞	2-chain query $R(x, y), S(y, z)$
Q_3^∞	3-chain query $R(x, y), S(y, z), T(z, u)$
Q_4^∞	4-chain query $P(u, x), R(x, y), S(y, z), T(z, v)$
Q_5^∞	5-chain query $L(a, u), P(u, x), R(x, y), S(y, z), T(z, v)$
Q_2^\star	2-star query $R(x)S(y), W(x, y)$
Q_3^\star	3-star query $R(x)S(y), T(z)W(x, y, z)$
Q^Δ	Triangle query $R(x, y)S(y, z), T(z, x)$
Q_A^Δ	Triangle Unary query $A(x)R(x, y)S(y, z), T(z, x)$
Q_{AB}^Δ	Triangle Binary query $A(x)R(x, y)S(y, z), T(z, x), B(z)$
Q_{2-SJ}^∞	Self-Join 2-chain query $R(x, y)R(y, z)$
Q_{2-SJ}^\sim	Self-Join 2-confluence query $A(x)R(x, y)S(y, z), B(z)$
Q_{SJ}^{z6}	Self-Join z6 query $A(x)R(x, y)R(y, y), R(y, z), C(z)$
Q_{6WE}°	6-cycle query with end points $A(x), R(x, y), B(y), S(y, z), C(z), T(z, u) D(u), U(u, v), E(v), V(v, w), F(w), W(w, x)$
Q_{cod}^Δ	Co-dominated triangle query $A(w), R(w, x, y), S(w, y, z), T(w, z, x)$

Table A.3: Common Abbreviations

Term	Full Form
ADP-SS	Aggregated Deletion Propagation
AHG	Automatic Hardness Gadget
CQ	Conjunctive Query
DLP	Disjunctive Logic Program
DP	Deletion Propagation
DP-SS	Deletion Propagation Source Side Effects
DP-VS	Deletion Propagation View Side Effects
GDP	Generalized Deletion Propagation
IJP	Independent Join Path
ILP	Integer Linear Programming or Integer Linear Program, depending on context

Continued on next page

Table A.3 – continued from previous page

Query	Definition
LP	Linear Program
MILP	Mixed Integer Linear Program
minFACT	Minimum Sized Factorization of Provenance Formulas
MFMC	Max Flow Min Cut
RES	Resilience
RSP	Responsibility
RDM	Reverse Data Management
SWP	Smallest Witness Problem

B Disjunctive Logic Program for Automatic Hardness Gadget-finder: An Example

We show an example DLP[RESIJP] for the 2-chain with self-join query $Q_{2-SJ}^{\infty} :- R(x, y), R(y, z)$. Here we are able to show the code in its entirety for $d = 5$ and endpoints $\{R(1, 2)\}$ and $\{R(3, 4)\}$.

The formulation finds a hardness proof in just 0.3 seconds, running on a local Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz with 8 cores.

```
1 % 1. Define Search Space
2 % There is one table R of arity 2 and d = 5. Thus we have 5^2 tuples in relation r.
3 r(1,1,1).
4 r(2,1,2).
5 r(3,1,3).
6 r(4,1,4).
7 r(5,1,5).
8 r(6,2,1).
9 r(7,2,2).
10 r(8,2,3).
11 r(9,2,4).
12 r(10,2,5).
13 r(11,3,1).
14 r(12,3,2).
15 r(13,3,3).
16 r(14,3,4).
17 r(15,3,5).
18 r(16,4,1).
19 r(17,4,2).
20 r(18,4,3).
21 r(19,4,4).
22 r(20,4,5).
23 r(21,5,1).
24 r(22,5,2).
25 r(23,5,3).
26 r(24,5,4).
27 r(25,5,5).
28
29 % 2. "Guess" an IJP
30 % For every tuple we define if it is in the IJP or not.
31 % We also calculate the witnesses and number of witnesses in the IJP
32 indb(r, Tid, 1) | indb(r, Tid, 0) :- r(Tid, -, -).
33 witness(X, Z, Y, T1, T2) :- r(T1, X, Y), r(T2, Y, Z), indb(r, T1, 1), indb(r, T2, 1).
34 number_of_witnesses(K) :- #count{ X, Z, Y, T1, T2 : witness(X, Z, Y, T1, T2) } = K.
35
36 % 3. JP Endpoint Condition
37 range_triangle(1..3).range_domain(1..5).
38 endpoint1_constant(1).
39 endpoint1_constant(2).
40 endpoint2_constant(3).
41 endpoint2_constant(4).
42 endpoint1_witness(T1, T2) :- witness(X, Z, Y, T1, T2), indb(r, T1, 1), r(T1, X, Y), endpoint1_constant(X),
43   endpoint1_constant(Y).
44 endpoint1_witness(T1, T2) :- witness(X, Z, Y, T1, T2), indb(r, T2, 1), r(T2, Y, Z), endpoint1_constant(Y),
45   endpoint1_constant(Z).
46 :- not #count{T1, T2: endpoint1_witness(T1, T2)} = 1.
47 endpoint2_witness(T1, T2) :- witness(X, Z, Y, T1, T2), indb(r, T1, 1), r(T1, X, Y), endpoint2_constant(X),
48   endpoint2_constant(Y).
49 endpoint2_witness(T1, T2) :- witness(X, Z, Y, T1, T2), indb(r, T2, 1), r(T2, Y, Z), endpoint2_constant(Y),
50   endpoint2_constant(Z).
51 :- not #count{T1, T2: endpoint2_witness(T1, T2)} = 1.
52 :- witness(X, Z, Y, T1, T2), endpoint1_constant(X), endpoint1_constant(Z), endpoint1_constant(Y).
53 :- witness(X, Z, Y, T1, T2), endpoint2_constant(X), endpoint2_constant(Z), endpoint2_constant(Y).
54
55 % 4. Calculate Resilience
```



```

52 valid_res2(r, 2, 1).
53 invalid_res2(r, 2, 1).
54 valid_res3(r, 14, 1).
55 invalid_res3(r, 14, 1).
56 valid_res4(r, 2, 1).
57 invalid_res4(r, 2, 1).
58 valid_res4(r, 14, 1).
59 invalid_res4(r, 14, 1).
60
61
62 invalid_res1(r, Tid, 1) | invalid_res1(r, Tid, 0) :- r(Tid, _, _).
63 invalid_res2(r, Tid, 1) | invalid_res2(r, Tid, 0) :- r(Tid, _, _).
64 invalid_res3(r, Tid, 1) | invalid_res3(r, Tid, 0) :- r(Tid, _, _).
65 invalid_res4(r, Tid, 1) | invalid_res4(r, Tid, 0) :- r(Tid, _, _).
66 valid_res1(r, Tid, 1) | valid_res1(r, Tid, 0) :- r(Tid, _, _).
67 valid_res2(r, Tid, 1) | valid_res2(r, Tid, 0) :- r(Tid, _, _).
68 valid_res3(r, Tid, 1) | valid_res3(r, Tid, 0) :- r(Tid, _, _).
69 valid_res4(r, Tid, 1) | valid_res4(r, Tid, 0) :- r(Tid, _, _).
70
71
72 invalid_resilience1 :- witness(X, Z, Y, T1, T2), invalid_res1(r, T1, 0), invalid_res1(r, T2, 0).
73 invalid_resilience1 :- #count{Table, Tid: invalid_res1(Table, Tid, 1)} >= K, res(K).
74 invalid_resilience2 :- witness(X, Z, Y, T1, T2), invalid_res2(r, T1, 0), invalid_res2(r, T2, 0).
75 invalid_resilience2 :- #count{Table, Tid: invalid_res2(Table, Tid, 1)} >= K, res(K).
76 invalid_resilience3 :- witness(X, Z, Y, T1, T2), invalid_res3(r, T1, 0), invalid_res3(r, T2, 0).
77 invalid_resilience3 :- #count{Table, Tid: invalid_res3(Table, Tid, 1)} >= K, res(K).
78 invalid_resilience4 :- witness(X, Z, Y, T1, T2), invalid_res4(r, T1, 0), invalid_res4(r, T2, 0).
79 invalid_resilience4 :- #count{Table, Tid: invalid_res4(Table, Tid, 1)} >= K+1, res(K).
80
81 % Here we are "saturating" the solution
82 invalid_res1(r, Tid, 0) :- invalid_resilience1, r(Tid, _, _).
83 invalid_res1(r, Tid, 1) :- invalid_resilience1, r(Tid, _, _).
84
85 invalid_res2(r, Tid, 0) :- invalid_resilience2, r(Tid, _, _).
86 invalid_res2(r, Tid, 1) :- invalid_resilience2, r(Tid, _, _).
87
88 invalid_res3(r, Tid, 0) :- invalid_resilience3, r(Tid, _, _).
89 invalid_res3(r, Tid, 1) :- invalid_resilience3, r(Tid, _, _).
90
91 invalid_res4(r, Tid, 0) :- invalid_resilience4, r(Tid, _, _).
92 invalid_res4(r, Tid, 1) :- invalid_resilience4, r(Tid, _, _).
93
94 :- not invalid_resilience1.
95 :- not invalid_resilience2.
96 :- not invalid_resilience3.
97 :- not invalid_resilience4.
98
99 % 5. Check for the OR Property
100
101 :- witness(X, Z, Y, T1, T2), valid_res1(r, T1, 0), valid_res1(r, T2, 0).
102 res(K) :- #count{Table, Tid: valid_res1(Table, Tid, 1)} = K.
103 :- witness(X, Z, Y, T1, T2), valid_res2(r, T1, 0), valid_res2(r, T2, 0).
104 :- not #count{Table, Tid: valid_res2(Table, Tid, 1)} = K, res(K).
105 :- witness(X, Z, Y, T1, T2), valid_res3(r, T1, 0), valid_res3(r, T2, 0).
106 :- not #count{Table, Tid: valid_res3(Table, Tid, 1)} = K, res(K).
107 :- witness(X, Z, Y, T1, T2), valid_res4(r, T1, 0), valid_res4(r, T2, 0).
108 :- not #count{Table, Tid: valid_res4(Table, Tid, 1)} = K+1, res(K).
109
110 % 6. Check for non-leaking composition
111
112 isomorph_map(C, 1, C) :- endpoint1_constant(C), range_triangle(I). % endpoint1 gets mapped to itself for edge
113 1
114 isomorph_map(C, 2, X) :- endpoint1_constant(C), range_triangle(I), X = C + 2. %endpoint1 gets mapped to 2 for
115 edge 2 - add endpoint arity
116 isomorph_map(C, 3, C) :- endpoint1_constant(C), range_triangle(I). %endpoint1 gets mapped to itself for edge 3
117 isomorph_map(C, 1, C) :- endpoint2_constant(C), range_triangle(I). %endpoint2 gets mapped to itself for edge 1
118 isomorph_map(C, 2, X) :- endpoint2_constant(C), range_triangle(I), X = C + 2. %endpoint2 gets mapped to 3 for
119 edge 2
120 isomorph_map(C, 3, X) :- endpoint2_constant(C), range_triangle(I), X = C + 2. %endpoint2 gets mapped to 3 for
121 edge 3
122 isomorph_map(C, I, X) :- range_triangle(I), range_domain(C), X = C+(5+1)*I, not endpoint1_constant(C), not
123 endpoint2_constant(C).

```

```

119 ijp_isomorph_1_r(TID, VI0,VI1) :- indb(r, TID, 1), r(TID, V0,V1), isomorph_map(V0,1,VI0), isomorph_map(V1,1,VI1
120 ).
121 ijp_isomorph_2_r(TID, VI0,VI1) :- indb(r, TID, 1), r(TID, V0,V1), isomorph_map(V0,2,VI0), isomorph_map(V1,2,VI1
122 ).
123 ijp_isomorph_3_r(TID, VI0,VI1) :- indb(r, TID, 1), r(TID, V0,V1), isomorph_map(V0,3,VI0), isomorph_map(V1,3,VI1
124 ).
125 ijp_isomorph_triangle_r(TID, V0, V1) :- ijp_isomorph_1_r(TID, V0, V1).
126 ijp_isomorph_triangle_r(TID, V0, V1) :- ijp_isomorph_2_r(TID, V0, V1).
127 ijp_isomorph_triangle_r(TID, V0, V1) :- ijp_isomorph_3_r(TID, V0, V1).
128 ijp_triangle_witness(X, Z, Y) :- ijp_isomorph_triangle_r(T1, X, Y), ijp_isomorph_triangle_r(T2, Y, Z).
129 :- number_of_witnesses(K), not #count{ X, Z, Y : ijp_triangle_witness(X, Z, Y) }= 3*K.
130
131 % 7. (Optional) Minimize the size of the IJP
132 :- witness(Z, Y, X, T1, T2). [1@1, Z, Y, X]
133
134 #show.
135 #show number_of_witnesses(K) : number_of_witnesses(K).
136 #show witness(X, Z, Y) : witness(X, Z, Y, T1, T2).
137 #show res(K) : res(K).

```

The code gives the following output, finding an IJP with 3 witnesses:

```

1 clingo version 5.6.2
2 Reading from ...gen_asp_scripts\ijp_expt_cases-1003.dl
3 Solving...
4 Progression : [1;inf]
5 Progression : [2;inf]
6 Answer: 1
7 res(3) witness(5,2,1) witness(4,3,5) witness(3,5,2) witness(3,5,5) witness(5,5,2) witness(5,5,5)
8   number_of_witnesses(6)
9 Optimization: 6
10 Answer: 2
11 res(2) witness(5,2,1) witness(4,3,5) witness(3,5,2) number_of_witnesses(3)
12 Optimization: 3
13 OPTIMUM FOUND
14 Models      : 2
15   Optimum   : yes
16 Optimization: 3
17 Calls       : 1
18 Time        : 0.392s (Solving: 0.18s 1st Model: 0.11s Unsat: 0.05s)
19 CPU Time    : 1.578s
20 Threads     : 8      (Winner: 4)

```

The IJP can then be automatically visualized as in Figure B.1.

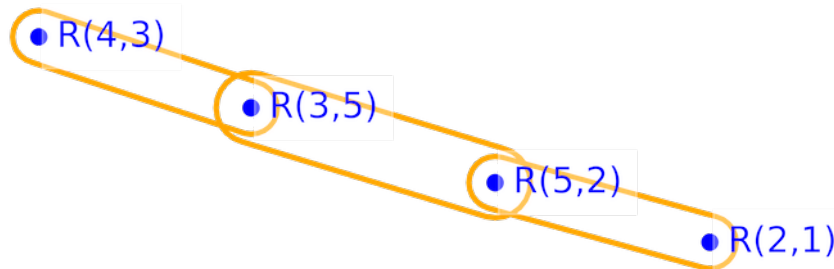


Figure B.1: Automatically generated IJP for Q_{2-SJ}^{∞}

Bibliography

- [1] Karen Aardal, George L Nemhauser, and Robert Weismantel. *Handbooks in Operations Research and Management Science: Discrete Optimization*. Elsevier, 2005 (cited on page 12).
- [2] Ahmad Abdi. ‘Ideal Clutters’. PhD thesis. University of Waterloo, 2018 (cited on page 13).
- [3] Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. ‘Instance Optimal Join Size Estimation’. In: *Procedia Computer Science* 195 (2021), pp. 135–144. doi: [10.1016/j.procs.2021.11.019](https://doi.org/10.1016/j.procs.2021.11.019) (cited on page 9).
- [4] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. ‘What Do Shannon-Type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?’ In: *PODS*. 2017, pp. 429–444. ISBN: 9781450341981. doi: [10.1145/3034786.3056105](https://doi.org/10.1145/3034786.3056105) (cited on page 9).
- [5] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. ‘Presolve reductions in mixed integer programming’. In: *INFORMS Journal on Computing* 32.2 (2020), pp. 473–506. doi: [10.1287/ijoc.2018.0857](https://doi.org/10.1287/ijoc.2018.0857) (cited on page 32).
- [6] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. ‘Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table’. In: *SIAM Journal on Computing* 38.1 (2008), pp. 63–84. doi: [10.1137/060664537](https://doi.org/10.1137/060664537) (cited on pages 81, 82).
- [7] Kaleb Alway, Eric Blais, and Semih Salihoglu. ‘Box Covers and Domain Orderings for Beyond Worst-Case Join Processing’. In: *International Conference on Database Theory*. Vol. 186. LIPIcs. 2021, 3:1–3:23. doi: [10.4230/LIPICS.ICDT.2021.3](https://doi.org/10.4230/LIPICS.ICDT.2021.3) (cited on page 9).
- [8] Antoine Amarilli, Wolfgang Gatterbauer, Neha Makhija, and Mikaël Monet. ‘Resilience for Regular Path Queries: Towards a Complexity Classification’. In: *Proceedings of the ACM on Management of Data* 3.2 (June 2025), pp. 1–18. doi: [10.1145/3725245](https://doi.org/10.1145/3725245) (cited on page 137).
- [9] Yael Amsterdamer, Daniel Deutch, Tova Milo, and Val Tannen. ‘On Provenance Minimization’. In: *ACM Trans. Database Syst.* 37.4 (Dec. 2012). doi: [10.1145/2389241.2389249](https://doi.org/10.1145/2389241.2389249) (cited on page 82).
- [10] Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. ‘GProM - A Swiss Army Knife for Your Provenance Needs’. In: *IEEE Data Engineering Bulletin* 41.1 (2018), pp. 51–62 (cited on page 76).
- [11] Albert Atserias and Phokion G Kolaitis. ‘Structure and complexity of bag consistency’. In: *ACM SIGMOD Record* 51.1 (2022), pp. 78–85. doi: [10.1145/3542700.3542719](https://doi.org/10.1145/3542700.3542719) (cited on pages 9, 15).
- [12] Shahaf Bassan, Guy Amir, and Guy Katz. ‘Local vs. Global Interpretability: A Computational Complexity Perspective’. In: *PMLR* (2024). doi: [10.5555/3692070.3692196](https://doi.org/10.5555/3692070.3692196) (cited on pages 8, 9).
- [13] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. ‘On the Desirability of Acyclic Database Schemes’. In: *Journal of the ACM (JACM)* 30.3 (July 1983), pp. 479–513. doi: [10.1145/2402.322389](https://doi.org/10.1145/2402.322389) (cited on pages 25, 104, 119).
- [14] Christoph Berkholz and Harry Vinall-Smeeth. ‘A Dichotomy for Succinct Representations of Homomorphisms’. In: *ICALP*. Ed. by Kousha Etessami, Uriel Feige, and Gabriele Puppis. Vol. 261. LIPIcs. 2023, 113:1–113:19. ISBN: 978-3-95977-278-5. doi: [10.4230/LIPICS.ICALP.2023.113](https://doi.org/10.4230/LIPICS.ICALP.2023.113) (cited on page 82).
- [15] Philip A. Bernstein and Nathan Goodman. ‘Power of Natural Semijoins’. In: *SIAM Journal on Computing* 10.4 (1981), pp. 751–771. doi: [10.1137/0210059](https://doi.org/10.1137/0210059) (cited on pages 104, 119).
- [16] Leopoldo Bertossi. ‘Specifying and computing causes for query answers in databases via database repairs and repair-programs’. In: *Knowledge and Information Systems* 63.1 (2021), pp. 199–231. doi: [10.1007/s10115-020-01516-6](https://doi.org/10.1007/s10115-020-01516-6) (cited on page 133).
- [17] Manuel Bodirsky, Zaneta Semanisinová, and Carsten Lutz. ‘The Complexity of Resilience Problems via Valued Constraint Satisfaction Problems’. In: *LICS*. ACM, 2024, 14:1–14:14. doi: [10.1145/3661814.3662071](https://doi.org/10.1145/3661814.3662071) (cited on pages 48, 50, 51, 69).
- [18] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 1998 (cited on page 10).

- [19] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. ‘MYSTIQ: a system for finding more answers by using probabilities’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2005, pp. 891–893. doi: [10.1145/1066157.1066277](#) (cited on page [123](#)).
- [20] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. ‘Scalable computation of high-order optimization queries’. In: *Communications of the ACM* 62.2 (2019), pp. 108–116. doi: [10.1145/3299881](#) (cited on pages [14](#), [76](#)).
- [21] David Buchfuhrer and Christopher Umans. ‘The complexity of Boolean formula minimization’. In: *Journal of Computer and System Sciences* 77.1 (2011), pp. 142–153. doi: [10.1016/j.jcss.2010.06.011](#) (cited on pages [78](#), [81](#), [82](#)).
- [22] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. ‘Why and Where: A Characterization of Data Provenance’. In: *International Conference on Database Theory*. 2001, pp. 316–330. doi: [10.1007/3-540-44503-x_20](#) (cited on page [15](#)).
- [23] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. ‘On Propagation of Deletions and Annotations Through Views’. In: *PODS*. 2002, pp. 150–158. ISBN: 1-58113-507-6. doi: [10.1145/543613.543633](#) (cited on pages [2](#), [4](#), [5](#), [7](#), [46](#), [50](#), [51](#), [55](#)).
- [24] Vinicius Callegaro, Mayler GA Martins, Renato P Ribas, and André I Reis. ‘Read-polarity-once Boolean functions’. In: *Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE. 2013, pp. 1–6. doi: [10.1109/SBCCI.2013.6644862](#) (cited on page [81](#)).
- [25] Ashok K. Chandra and Philip M. Merlin. ‘Optimal Implementation of Conjunctive Queries in Relational Data Bases’. In: *STOC*. 1977, pp. 77–90. doi: [10.1145/800105.803397](#) (cited on page [19](#)).
- [26] Surajit Chaudhuri. ‘To do or not to do: extending SQL with integer linear programming?: technical perspective’. In: *Communications of the ACM* 62.2 (2019), p. 107. doi: [10.1145/3299879](#) (cited on page [76](#)).
- [27] Surajit Chaudhuri and Moshe Y Vardi. ‘Optimization of real conjunctive queries’. In: *PODS*. 1993, pp. 59–70. doi: [10.1145/153850.153856](#) (cited on pages [9](#), [15](#)).
- [28] Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. ‘Why Not Yet: Fixing a Top-k Ranking that is Not Fair to Individuals’. In: *Proceedings of the VLDB Endowment* 16.9 (May 2023), pp. 2377–2390. doi: [10.14778/3598581.3598606](#) (cited on page [1](#)).
- [29] Zixuan Chen, Subhodeep Mitra, R Ravi, and Wolfgang Gatterbauer. ‘HITSNDIFFS: From Truth Discovery to Ability Discovery by Recovering Matrices with the Consecutive Ones Property’. In: *International Conference on Data Engineering (ICDE)*. 2024 (cited on page [104](#)).
- [30] James Cheney, Laura Chiticariu, and Wang Chiew Tan. ‘Provenance in Databases: Why, How, and Where’. In: *Foundations and Trends in Databases* 1.4 (2009), pp. 379–474. doi: [10.1561/9781601982339](#) (cited on page [10](#)).
- [31] Hana Chockler and Joseph Y. Halpern. ‘Responsibility and Blame: A Structural-Model Approach’. In: *J. Artif. Intell. Res. (JAIR)* 22 (2004), pp. 93–115. doi: [10.1613/jair.1391](#) (cited on pages [8](#), [34](#)).
- [32] Michael B Cohen, Yin Tat Lee, and Zhao Song. ‘Solving linear programs in the current matrix multiplication time’. In: *Journal of the ACM (JACM)* 68.1 (2021), pp. 1–39. doi: [10.1145/3424305](#) (cited on page [13](#)).
- [33] Michele Conforti, Gérard Cornuéjols, and Kristina Vušković. ‘Balanced matrices’. In: *Discrete Mathematics* 306.19-20 (2006), pp. 2411–2437. doi: [10.1016/j.disc.2005.12.033](#) (cited on page [13](#)).
- [34] Gérard Cornuéjols and Bertrand Guenin. ‘Ideal clutters’. In: *Discrete Applied Mathematics* 123.1-3 (2002), pp. 303–338. doi: [10.1016/S0166-218X\(01\)00344-4](#) (cited on page [13](#)).
- [35] Yves Crama and Peter L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, 2011 (cited on pages [11](#), [42](#), [81](#), [110](#)).
- [36] Michael Crouch, Neil Immerman, and J. Eliot B. Moss. ‘Finding Reductions Automatically’. In: *Fields of Logic and Computation*. Springer Berlin Heidelberg, 2010, pp. 181–200. ISBN: 9783642150258. doi: [10.1007/978-3-642-15025-8_10](#) (cited on page [133](#)).
- [37] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. ‘Tracing the lineage of view data in a warehousing environment’. In: *ACM Transactions on Database Systems (TODS)* 25.2 (2000), pp. 179–227. doi: [10.1145/357775.357777](#) (cited on page [11](#)).

- [38] Nilesh N. Dalvi and Dan Suciu. ‘Efficient Query Evaluation on Probabilistic Databases’. In: *VLDB*. 2004, pp. 864–875. doi: [10.1016/b978-012088469-8.50076-0](https://doi.org/10.1016/b978-012088469-8.50076-0) (cited on page 78).
- [39] Nilesh N. Dalvi and Dan Suciu. ‘Efficient query evaluation on probabilistic databases’. In: *VLDB J.* 16.4 (2007), pp. 523–544. doi: [10.1007/s00778-006-0004-3](https://doi.org/10.1007/s00778-006-0004-3) (cited on pages 15, 78, 79, 83, 94, 112, 122–124).
- [40] Nilesh N. Dalvi and Dan Suciu. ‘The dichotomy of probabilistic inference for unions of conjunctive queries’. In: *Journal of the ACM (JACM)* 59.6 (2012), p. 30. doi: [10.1145/2395116.2395119](https://doi.org/10.1145/2395116.2395119) (cited on pages 15, 122).
- [41] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. ‘Complexity and Expressive Power of Logic Programming’. In: *ACM Computing Surveys* 33.3 (2001), pp. 374–425. doi: [10.1145/502807.502810](https://doi.org/10.1145/502807.502810) (cited on page 133).
- [42] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. Boston: McGraw-Hill Higher Education, 2008. ISBN: 9780073523408 (cited on page 103).
- [43] Martin Davis, George Logemann, and Donald Loveland. ‘A Machine Program for Theorem-Proving’. In: *Communications of the ACM* 5.7 (1962), pp. 394–397. doi: [10.1145/368273.368557](https://doi.org/10.1145/368273.368557) (cited on page 134).
- [44] Umeshwar Dayal and Philip A. Bernstein. ‘On the Correct Translation of Update Operations on Relational Views’. In: *ACM Transactions on Database Systems (TODS)* 7.3 (1982), pp. 381–416. doi: [10.1145/319732.319740](https://doi.org/10.1145/319732.319740) (cited on pages 7, 46, 49, 50).
- [45] Rina Dechter. ‘Bucket elimination: A unifying framework for reasoning’. In: *Artificial Intelligence* 113.1 (1999), pp. 41–85. doi: [10.1016/S0004-3702\(99\)00059-4](https://doi.org/10.1016/S0004-3702(99)00059-4) (cited on pages 84, 93).
- [46] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003 (cited on page 119).
- [47] Akhil A. Dixit and Phokion G. Kolaitis. ‘Consistent Answers of Aggregation Queries via SAT’. In: *International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 924–937. doi: [10.1109/ICDE53745.2022.00074](https://doi.org/10.1109/ICDE53745.2022.00074) (cited on page 77).
- [48] Jack Edmonds and Rick Giles. ‘A Min-Max Relation for Submodular Functions on Graphs’. In: *Studies in Integer Programming*. Elsevier, 1977, pp. 185–204. ISBN: 9780720407655. doi: [10.1016/s0167-5060\(08\)70734-9](https://doi.org/10.1016/s0167-5060(08)70734-9) (cited on pages 13, 26).
- [49] Thomas Eiter and Georg Gottlob. ‘Propositional circumscription and extended closed-world reasoning are Π_2^P -complete’. In: *Theoretical Computer Science* 114.2 (1993), pp. 231–245. doi: [10.1016/0304-3975\(93\)90073-3](https://doi.org/10.1016/0304-3975(93)90073-3) (cited on page 133).
- [50] Thomas Eiter and Georg Gottlob. ‘On the computational cost of disjunctive logic programming: Propositional case’. In: *Annals of Mathematics and Artificial Intelligence* 15 (1995), pp. 289–323. doi: [10.1007/bf01536399](https://doi.org/10.1007/bf01536399) (cited on page 133).
- [51] Thomas Eiter, Georg Gottlob, and Heikki Mannila. ‘Disjunctive Datalog’. In: *ACM Transactions on Database Systems (TODS)* 22.3 (1997), pp. 364–418. doi: [10.1145/261124.261126](https://doi.org/10.1145/261124.261126) (cited on page 133).
- [52] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer set programming: A primer*. Springer, 2009 (cited on pages 133, 134).
- [53] Thomas Eiter and Axel Polleres. ‘Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications’. In: *Theory and Practice of Logic Programming* 6.1-2 (2006), pp. 23–60. doi: [10.1017/s1471068405002577](https://doi.org/10.1017/s1471068405002577) (cited on page 134).
- [54] Ronald Fagin, Amnon Lotem, and Moni Naor. ‘Optimal aggregation algorithms for middleware’. In: *PODS*. 2001, pp. 102–113. ISBN: 1581133618. doi: [10.1145/375551.375567](https://doi.org/10.1145/375551.375567) (cited on page 9).
- [55] Austen Z. Fan, Paraschos Koutris, and Hangdong Zhao. ‘Tight Bounds of Circuits for Sum-Product Queries’. In: *Proceedings of the ACM on Management of Data* 2.2 (May 2024), pp. 1–20. doi: [10.1145/3651588](https://doi.org/10.1145/3651588) (cited on page 138).
- [56] Robert Fink, Jiewen Huang, and Dan Olteanu. ‘Anytime approximation in probabilistic databases’. In: *VLDB J.* 22.6 (2013), pp. 823–848. doi: [10.1007/s00778-013-0310-5](https://doi.org/10.1007/s00778-013-0310-5) (cited on pages 83, 123, 124).
- [57] Robert Fink and Dan Olteanu. ‘On the optimal approximation of queries using tractable propositional languages’. In: *International Conference on Database Theory*. 2011, pp. 174–185. doi: [10.1145/1938551.1938575](https://doi.org/10.1145/1938551.1938575) (cited on page 123).

- [58] Lester Randolph Ford and Delbert R Fulkerson. ‘Maximal flow through a network’. In: *Canadian journal of Mathematics* 8 (1956), pp. 399–404. doi: [10.4153/cjm-1956-045-5](#) (cited on pages [13](#), [26](#)).
- [59] Cibeles Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. ‘The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries’. In: *PVLDB* 9.3 (2015), pp. 180–191. doi: [10.14778/2850583.2850592](#) (cited on pages [2](#), [4](#), [7](#), [15](#), [22–27](#), [30](#), [34](#), [39–41](#), [43](#), [50](#), [51](#), [68](#), [78](#), [79](#), [119](#), [120](#)).
- [60] Cibeles Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. ‘New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins’. In: *PODS*. 2020, pp. 271–284. doi: [10.1145/3375395.3387647](#) (cited on pages [4](#), [5](#), [7](#), [15](#), [16](#), [19](#), [22](#), [33](#), [48](#), [50](#), [51](#), [69](#), [120](#)).
- [61] Norbert Fuhr and Thomas Rölleke. ‘A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems’. In: *ACM Transactions on Information Systems (TOIS)* 15.1 (1997), pp. 32–66. doi: [10.1145/239041.239045](#) (cited on page [123](#)).
- [62] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. ‘Fairness testing: testing software for discrimination’. In: *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*. 2017, pp. 498–510. doi: [10.1145/3106237.3106277](#) (cited on pages [1](#), [7](#), [8](#)).
- [63] Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. ‘Hyper: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2022, pp. 1598–1611. ISBN: 9781450392495. doi: [10.1145/3514221.3526149](#) (cited on pages [1](#), [7](#)).
- [64] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. W. H. Freeman & Co., 1979. ISBN: 978-0-7167-1045-5 (cited on page [81](#)).
- [65] Wolfgang Gatterbauer and Dan Suciu. ‘Oblivious bounds on the probability of Boolean functions’. In: *ACM Transactions on Database Systems (TODS)* 39.1 (2014), pp. 1–34. doi: [10.1145/2532641](#) (cited on pages [83](#), [91–93](#), [123](#)).
- [66] Wolfgang Gatterbauer and Dan Suciu. ‘Dissociation and propagation for approximate lifted inference with standard relational database management systems’. In: *VLDB J.* 26.1 (2017), pp. 5–30. doi: [10.1007/s00778-016-0434-5](#) (cited on pages [29](#), [78](#), [92–95](#), [99](#), [109](#), [123](#), [124](#)).
- [67] Dongdong Ge, Chengwenjian Wang, Zikai Xiong, and Yinyu Ye. ‘From an Interior Point to a Corner Point: Smart Crossover’. In: *INFORMS Journal on Computing* (Mar. 2025). doi: [10.1287/ijoc.2022.0291](#) (cited on page [13](#)).
- [68] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. ‘Potassco: The Potsdam answer set solving collection’. In: *Ai Communications* 24.2 (2011), pp. 107–124. doi: [10.3233/aic-2011-0491](#) (cited on pages [133](#), [134](#)).
- [69] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014 (cited on page [133](#)).
- [70] Boris Glavic. ‘Perm: efficient provenance support for relational databases’. PhD thesis. University of Zurich, 2010 (cited on page [76](#)).
- [71] Boris Glavic and Gustavo Alonso. ‘The perm provenance management system in action’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Ed. by Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul. ACM, 2009, pp. 1055–1058. doi: [10.1145/1559845.1559980](#) (cited on page [76](#)).
- [72] Boris Glavic, Alexandra Meliou, and Sudeepa Roy. ‘Trends in explanations: Understanding and debugging data-driven systems’. In: *Foundations and Trends in Databases* 11.3 (2021). doi: [10.1561/9781680838817](#) (cited on pages [1](#), [7](#), [8](#)).
- [73] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. ‘Complexity of DNF minimization and isomorphism testing for monotone formulas’. In: *Information and Computation* 206.6 (2008), pp. 760–775. doi: [10.1016/j.ic.2008.03.002](#) (cited on pages [81](#), [82](#)).
- [74] Martin Charles Golumbic and Vladimir Gurvich. ‘Read-once functions’. In: *Boolean Functions: Theory, Algorithms and Applications*. Cambridge University Press, 2011. Chap. 10. doi: [10.1017/cbo9780511852008.011](#) (cited on pages [11](#), [42](#)).

- [75] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. ‘Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees’. In: *Discrete Applied Mathematics* 154.10 (2006), pp. 1465–1477. doi: [10.1016/j.dam.2005.09.016](https://doi.org/10.1016/j.dam.2005.09.016) (cited on pages [12](#), [42](#), [110](#)).
- [76] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. ‘An improvement on the complexity of factoring read-once Boolean functions’. In: *Discrete Applied Mathematics* 156.10 (2008), pp. 1633–1636. doi: [10.1016/j.dam.2008.02.011](https://doi.org/10.1016/j.dam.2008.02.011) (cited on pages [78](#), [81](#), [82](#)).
- [77] Julien Grange, Fabian Vehlken, Nils Vortmeier, and Thomas Zeume. ‘Specification and Automatic Verification of Computational Reductions’. en. In: *Mathematical Foundations of Computer Science*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi: [10.4230/LIPICS.MFCS.2024.56](https://doi.org/10.4230/LIPICS.MFCS.2024.56) (cited on page [133](#)).
- [78] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. ‘Provenance semirings’. In: *PODS*. 2007, pp. 31–40. doi: [10.1145/1265530.1265535](https://doi.org/10.1145/1265530.1265535) (cited on pages [8](#), [11](#), [124](#)).
- [79] Todd J. Green and Val Tannen. ‘The Semiring Framework for Database Provenance’. In: *PODS*. 2017, pp. 93–99. ISBN: 978-1-4503-4198-1. doi: [10.1145/3034786.3056125](https://doi.org/10.1145/3034786.3056125) (cited on pages [11](#), [124](#)).
- [80] Martin Grötschel, László Lovász, and Alexander Schrijver. ‘The Ellipsoid Method’. In: *Geometric Algorithms and Combinatorial Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 64–101. ISBN: 978-3-642-78240-4. doi: [10.1007/978-3-642-78240-4_4](https://doi.org/10.1007/978-3-642-78240-4_4) (cited on page [13](#)).
- [81] LLC Gurobi Optimization. *Mixed-Integer Programming (MIP) – A Primer on the Basics*. 2021. URL: <https://www.gurobi.com/resource/mip-basics/> (cited on page [14](#)).
- [82] LLC Gurobi Optimization. *Gurobi Guidelines For Numerical Issues*. 2022. URL: https://www.gurobi.com/documentation/10.0/refman/guidelines_for_numerical_i.html (cited on pages [32](#), [73](#)).
- [83] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2022. URL: <http://www.gurobi.com> (cited on pages [14](#), [30](#), [128](#)).
- [84] LLC Gurobi Optimization. *How does presolve work?* 2025. URL: <https://support.gurobi.com/hc/en-us/articles/360024738352-How-does-presolve-work> (cited on pages [14](#), [26](#)).
- [85] V.A. Gurvich. ‘Repetition-free Boolean functions’. In: *Uspekhi Mat. Nauk* 32 (1977), pp. 183–184 (cited on page [11](#)).
- [86] Joseph Y. Halpern and Judea Pearl. ‘Causes and Explanations: A Structural-Model Approach: Part 1: Causes’. In: *The British Journal for the Philosophy of Science*. 2001, pp. 194–202. doi: [10.1093/bjps/axi147](https://doi.org/10.1093/bjps/axi147) (cited on pages [5](#), [8](#), [34](#)).
- [87] Joseph Y. Halpern and Judea Pearl. ‘Causes and Explanations: A structural-model Approach. Part II: Explanations’. In: *The British Journal for the Philosophy of Science* 56 (2005), pp. 889–911. doi: [10.1093/bjps/axi148](https://doi.org/10.1093/bjps/axi148) (cited on pages [8](#), [34](#)).
- [88] Peter L Hammer and Alexander Kogan. ‘Optimal compression of propositional Horn knowledge bases: complexity and approximation’. In: *Artificial Intelligence* 64.1 (1993), pp. 131–145. doi: [10.1016/0004-3702\(93\)90062-G](https://doi.org/10.1016/0004-3702(93)90062-G) (cited on page [81](#)).
- [89] I. Heller and C. B. Tompkins. ‘14 . An Extension of a Theorem of Dantzig’s’. In: Princeton: Princeton University Press, 1957, pp. 247–254. ISBN: 9781400881987. doi: [10.1515/9781400881987-015](https://doi.org/10.1515/9781400881987-015) (cited on page [113](#)).
- [90] Edith Hemaspaandra and Henning Schnoor. ‘Minimization for generalized boolean formulas’. In: *21st International Joint Conference on Artificial Intelligence (IJCAI)*. 2011, pp. 566–571 (cited on page [81](#)).
- [91] Melanie Herschel, Mauricio A. Hernández, and Wang Chiew Tan. ‘Artemis: A System for Analyzing Missing Answers’. In: *PVLDB* 2.2 (2009), pp. 1550–1553. doi: [10.14778/1687553.1687588](https://doi.org/10.14778/1687553.1687588) (cited on pages [7](#), [8](#)).
- [92] Maarten Van den Heuvel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerts, and Martin Theobald. ‘Anytime Approximation in Probabilistic Databases via Scaled Dissociations’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2019, pp. 1295–1312. doi: [10.1145/3299869.3319900](https://doi.org/10.1145/3299869.3319900) (cited on pages [83](#), [123](#)).
- [93] Xiao Hu and Stavros Sintos. ‘Finding Smallest Witnesses for Conjunctive Queries’. In: *International Conference on Database Theory*. Vol. 290. LIPIcs. 2024, 24:1–24:20. ISBN: 978-3-95977-312-6. doi: [10.4230/LIPICS.ICDT.2024.24](https://doi.org/10.4230/LIPICS.ICDT.2024.24) (cited on pages [4](#), [5](#), [8](#), [9](#), [46](#), [49](#), [52](#), [64](#), [70](#)).

- [94] Xiao Hu, Shouzhao Sun, Shweta Patwa, Debmalya Panigrahi, and Sudeepa Roy. ‘Aggregated Deletion Propagation for Counting Conjunctive Query Answers’. In: *PVLDB* 14.2 (2020), pp. 228–240. doi: [10.14778/3425879.3425892](#) (cited on pages [2](#), [4](#), [5](#), [8](#), [46](#), [49](#), [51](#), [52](#), [68](#), [70](#)).
- [95] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. ‘On the provenance of non-answers to queries over extracted data’. In: *PVLDB* 1.1 (2008), pp. 736–747. doi: [10.14778/1453856.1453936](#) (cited on page [8](#)).
- [96] Rahul Ilango. ‘The Minimum Formula Size Problem is (ETH) Hard’. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 427–432. doi: [10.1109/FOCS52979.2021.00050](#) (cited on pages [81](#), [82](#)).
- [97] Richard M Karp. ‘Reducibility among combinatorial problems’. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103. doi: [10.1007/978-1-4684-2001-2_9](#) (cited on page [13](#)).
- [98] J. E. Kelley Jr. ‘The Cutting-Plane Method for Solving Convex Programs’. In: *Journal of the Society for Industrial and Applied Mathematics* 8.4 (1960), pp. 703–712. doi: [10.1137/0108053](#) (cited on pages [3](#), [59](#)).
- [99] Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. ‘Bag Query Containment and Information Theory’. In: *ACM Transactions on Database Systems (TODS)* 46.3 (2021). doi: [10.1145/3472391](#) (cited on pages [9](#), [15](#)).
- [100] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. ‘Joins via Geometric Resolutions: Worst Case and Beyond’. In: *ACM Transactions on Database Systems (TODS)* 41.4 (Nov. 2016). doi: [10.1145/2967101](#) (cited on page [9](#)).
- [101] Benny Kimelfeld. ‘A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies’. In: *PODS*. 2012, pp. 191–202. ISBN: 978-1-4503-1248-6. doi: [10.1145/2213556.2213584](#) (cited on pages [46](#), [48](#), [51](#)).
- [102] Benny Kimelfeld, Jan Vondrák, and Ryan Williams. ‘Maximizing Conjunctive Views in Deletion Propagation’. In: *ACM Transactions on Database Systems (TODS)* 37.4 (2012), 24:1–24:37. doi: [10.1145/2389241.2389243](#) (cited on pages [2](#), [4](#), [5](#), [46](#), [47](#), [51](#), [63](#), [64](#), [70](#), [77](#)).
- [103] Benny Kimelfeld, Jan Vondrák, and David P. Woodruff. ‘Multi-tuple Deletion Propagation: Approximations and Complexity’. In: *PVLDB* 6.13 (2013), pp. 1558–1569. doi: [10.14778/2536258.2536267](#) (cited on pages [49](#), [51](#)).
- [104] Solmaz Kolahi. ‘Functional Dependency’. In: *Encyclopedia of Database Systems*. Springer US, 2009, pp. 1200–1201. ISBN: 9780387399409. doi: [10.1007/978-0-387-39940-9_1247](#) (cited on page [43](#)).
- [105] George Konstantinidis and Fabio Mogavero. ‘Attacking Diophantus: Solving a Special Case of Bag Containment’. In: *PODS*. 2019, pp. 399–413. doi: [10.1145/3294052.3319689](#) (cited on page [9](#)).
- [106] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press, 2011 (cited on page [13](#)).
- [107] S. L. Lauritzen and D. J. Spiegelhalter. ‘Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems’. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 50.2 (1988), pp. 157–224. doi: [10.1111/j.2517-6161.1988.tb01721.x](#) (cited on page [119](#)).
- [108] Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. ‘Why and why not explanations improve the intelligibility of context-aware intelligent systems’. In: *CHI*. 2009, pp. 2119–2128 (cited on page [8](#)).
- [109] Anh L. Mai, Pengyu Wang, Azza Abouzied, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. ‘Scaling Package Queries to a Billion Tuples via Hierarchical Partitioning and Customized Optimization’. In: *Proc. VLDB Endow.* 17.5 (2024), pp. 1146–1158. doi: [10.14778/3641204.3641222](#) (cited on page [76](#)).
- [110] Neha Makhija and Wolfgang Gatterbauer. ‘A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations’. In: *Proceedings of the ACM on Management of Data* 1.4 (2023). doi: [10.1145/3626715](#) (cited on pages [2–6](#), [8](#), [9](#), [14](#), [15](#), [34](#), [48](#), [50](#), [51](#), [63](#), [70](#), [77](#), [133](#)).
- [111] Neha Makhija and Wolfgang Gatterbauer. ‘Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries’. In: *Proceedings of the ACM on Management of Data* 2.2 (May 2024), pp. 1–24. doi: [10.1145/3651605](#) (cited on pages [2–4](#), [6](#), [78](#)).

- [112] Neha Makhija and Wolfgang Gatterbauer. ‘Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach for Generalized Deletion Propagation’. In: *PVLDB* (2025). doi: [10.14778/3742728.3742756](https://doi.org/10.14778/3742728.3742756) (cited on pages 2–4, 46).
- [113] Joao Marques-Silva. ‘Logic-Based Explainability in Machine Learning’. In: *Reasoning Web. Causality, Explanations and Declarative Knowledge*. Springer Nature Switzerland, 2023, pp. 24–104. ISBN: 9783031314148. doi: [10.1007/978-3-031-31414-8_2](https://doi.org/10.1007/978-3-031-31414-8_2) (cited on page 8).
- [114] Mayler GA Martins, Leomar Rosa, Anders B Rasmussen, Renato P Ribas, and Andre I Reis. ‘Boolean factoring with multi-objective goals’. In: *International Conference on Computer Design (ICCD)*. IEEE. 2010, pp. 229–234. doi: [10.1109/ICCD.2010.5647772](https://doi.org/10.1109/ICCD.2010.5647772) (cited on page 81).
- [115] Nimrod Megiddo. ‘On Finding Primal- and Dual-Optimal Bases’. In: *ORSA Journal on Computing* 3.1 (Feb. 1991), pp. 63–65. doi: [10.1287/ijoc.3.1.63](https://doi.org/10.1287/ijoc.3.1.63) (cited on page 13).
- [116] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. ‘Why so? or Why no? Functional Causality for Explaining Query Answers’. In: *Workshop on Management of Uncertain Data (MUD)*. Vol. abs/0912.5340. 2009, pp. 3–17. doi: [10.48550/arxiv.0912.5340](https://doi.org/10.48550/arxiv.0912.5340) (cited on pages 8, 9).
- [117] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. ‘The Complexity of Causality and Responsibility for Query Answers and non-Answers’. In: *PVLDB* 4.1 (2010), pp. 34–45. doi: [10.14778/1880172.1880176](https://doi.org/10.14778/1880172.1880176) (cited on pages 2, 4, 5, 8, 9, 24, 26, 29, 30, 34, 39).
- [118] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. ‘Reverse Data Management’. In: *PVLDB* 4.12 (2011), pp. 1490–1493. doi: [10.14778/3402755.3402803](https://doi.org/10.14778/3402755.3402803) (cited on pages 1, 4, 7).
- [119] Alexandra Meliou and Dan Suciu. ‘Tiresias: the database oracle for how-to queries’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2012, pp. 337–348. ISBN: 9781450312479. doi: [10.1145/2213836.2213875](https://doi.org/10.1145/2213836.2213875) (cited on pages 1, 7).
- [120] Dongjing Miao, Jianzhong Li, and Zhipeng Cai. ‘The parameterized complexity and kernelization of resilience for database queries’. In: *Theoretical Computer Science* 840 (2020), pp. 199–211. doi: [10.1016/j.tcs.2020.08.018](https://doi.org/10.1016/j.tcs.2020.08.018) (cited on pages 50, 51).
- [121] Zhengjie Miao, Sudeepa Roy, and Jun Yang. ‘Explaining Wrong Queries Using Small Examples’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Amsterdam, Netherlands, 2019, pp. 503–520. ISBN: 9781450356435. doi: [10.1145/3299869.3319866](https://doi.org/10.1145/3299869.3319866) (cited on pages 2, 8, 9, 46, 52).
- [122] Aviad Mintz and Martin Charles Golumbic. ‘Factoring Boolean functions using graph partitioning’. In: *Discrete Applied Mathematics* 149.1 (2005), pp. 131–153. doi: [10.1016/j.dam.2005.02.007](https://doi.org/10.1016/j.dam.2005.02.007) (cited on page 81).
- [123] Stuart Mitchell, Michael OSullivan, and Iain Dunning. *PuLP: a linear programming toolkit for python*. 2011. URL: <https://optimization-online.org/?p=11731> (cited on page 14).
- [124] Guido Moerkotte. *Building Query Compilers*. Draft version 03.03.09. 2009. URL: <http://theo.cs.ovgu.de/lehre/lehre09w/Anfrageoptimierung/querycompiler.pdf> (cited on page 89).
- [125] Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. ‘Beyond worst-case analysis for joins with minesweeper’. In: *PODS*. Snowbird, Utah, USA, 2014, pp. 234–245. ISBN: 9781450323758. doi: [10.1145/2594538.2594547](https://doi.org/10.1145/2594538.2594547) (cited on page 9).
- [126] OEIS A000169: *The On-Line Encyclopedia of Integer Sequences*. URL: <https://oeis.org/A000169> (cited on page 102).
- [127] Dan Olteanu and Jiewen Huang. ‘Using OBDDs for efficient query evaluation on probabilistic databases’. In: *International Conference on Scalable Uncertainty Management (SUM)*. Springer. 2008, pp. 326–340. doi: [10.1007/978-3-540-87993-0_26](https://doi.org/10.1007/978-3-540-87993-0_26) (cited on pages 11, 83, 89, 94).
- [128] Dan Olteanu, Jiewen Huang, and Christoph Koch. ‘Approximate Confidence Computation in Probabilistic Databases’. In: *International Conference on Data Engineering (ICDE)*. 2010, pp. 145–156. doi: [10.1109/icde.2010.5447826](https://doi.org/10.1109/icde.2010.5447826) (cited on pages 123, 124).
- [129] Dan Olteanu and Maximilian Schleich. ‘Factorized Databases’. In: *ACM SIGMOD Record* 45.2 (2016), pp. 5–16. doi: [10.1145/3003665.3003667](https://doi.org/10.1145/3003665.3003667) (cited on pages 82, 84, 91, 93).
- [130] Dan Olteanu and Jakub Závodný. ‘On Factorisation of Provenance Polynomials’. In: *Workshop on the Theory and Practice of Provenance*. USENIX, 2011 (cited on page 82).

- [131] Dan Olteanu and Jakub Závodný. ‘Factorised representations of query results: size bounds and readability’. In: *International Conference on Database Theory*. 2012, pp. 285–298. doi: [10.1145/2274576.2274607](#) (cited on pages [78](#), [82](#)).
- [132] Dan Olteanu and Jakub Závodný. ‘Size Bounds for Factorised Representations of Query Results’. In: *ACM Trans. Database Syst.* 40.1 (2015), 2:1–2:44. doi: [10.1145/2656335](#) (cited on pages [4](#), [82](#), [91](#)).
- [133] Knot Pipatsrisawat and Adnan Darwiche. ‘New Compilation Languages Based on Structured Decomposability’. In: *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1. AAAI’08*. Chicago, Illinois: AAAI Press, 2008, pp. 517–522. ISBN: 9781577353683 (cited on page [91](#)).
- [134] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. ‘Interpretable data-based explanations for fairness debugging’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2022, pp. 247–261. doi: [10.1145/3514221.3517886](#) (cited on page [8](#)).
- [135] Teodor C. Przymusiński. ‘Stable Semantics for Disjunctive Programs’. In: *New Generation Computing* 9.3–4 (1991), pp. 401–424. doi: [10.1007/BF03037171](#) (cited on page [133](#)).
- [136] Chandrasekhar Ramakrishnan. *2019-01 US Flights*. Version V1. 2020. doi: [10.7910/DVN/WTZS4K](#). URL: [10.7910/DVN/WTZS4K](#) (cited on pages [71](#), [72](#)).
- [137] Christopher Ré, Nilesch Dalvi, and Dan Suciu. ‘Efficient top-k query evaluation on probabilistic data’. In: *International Conference on Data Engineering (ICDE)*. 2007, pp. 886–895. doi: [10.1109/icde.2007.367934](#) (cited on page [123](#)).
- [138] Wolfram Research. *Mathematica, Version 13.1*. 2022. URL: <https://www.wolfram.com/mathematica> (cited on page [125](#)).
- [139] Neil Robertson and Paul D. Seymour. ‘Graph Minors. II. Algorithmic Aspects of Tree-Width’. In: *J. Algorithms* 7.3 (1986), pp. 309–322. doi: [10.1016/0196-6774\(86\)90023-4](#) (cited on page [119](#)).
- [140] Tim Roughgarden. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020. ISBN: 9781108494311 (cited on page [9](#)).
- [141] Sudeepa Roy, Vittorio Perduca, and Val Tannen. ‘Faster query answering in probabilistic databases using read-once functions’. In: *International Conference on Database Theory*. 2011, pp. 232–243. doi: [10.1145/1938551.1938582](#) (cited on pages [78](#), [79](#), [83](#), [110](#), [122–124](#)).
- [142] Sudeepa Roy and Dan Suciu. ‘A Formal Approach to Finding Explanations for Database Queries’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2014, pp. 1579–1590. doi: [10.1145/2588555.2588578](#) (cited on pages [1](#), [7](#), [8](#)).
- [143] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. ‘Interventional fairness: Causal database repair for algorithmic fairness’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2019, pp. 793–810. doi: [10.1145/3299869.3319901](#) (cited on pages [1](#), [7](#), [8](#)).
- [144] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998 (cited on pages [12](#), [13](#), [26](#), [43](#), [68](#), [109](#), [112](#)).
- [145] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Algorithms and Combinatorics. Springer, 2003. ISBN: 978-3-540-44389-6 (cited on pages [19](#), [33](#)).
- [146] Daniel A Schult and P Swart. ‘Exploring network structure, dynamics, and function using NetworkX’. In: *Proceedings of the Python in Science Conferences*. Vol. 2008. Pasadena, CA. 2008, pp. 11–16 (cited on page [30](#)).
- [147] Prithviraj Sen, Amol Deshpande, and Lise Getoor. ‘Read-Once Functions and Query Evaluation in Probabilistic Databases’. In: *PVLDB* 3.1 (2010), pp. 1068–1079. doi: [10.14778/1920841.1920975](#) (cited on pages [83](#), [110](#), [122–124](#)).
- [148] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ‘ProvSQL: Provenance and Probability Management in PostgreSQL’. In: *Proc. VLDB Endow.* 11.12 (2018), pp. 2034–2037. doi: [10.14778/3229863.3236253](#) (cited on page [76](#)).
- [149] Laurynas Siksnys and Torben Bach Pedersen. ‘SolveDB: Integrating Optimization Problem Solvers Into SQL Databases’. In: *International Conference on Scientific and Statistical Database Management, SSDBM*. Ed. by Peter Baumann, Ioana Manolescu-Goujot, Luca Trani, Yannis E. Ioannidis, Gergely Gábor Barnaföldi, László Dobos, and Evelin Bányai. ACM, 2016, 14:1–14:12. doi: [10.1145/2949689.2949693](#) (cited on page [14](#)).

- [150] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool, 2011 (cited on page 80).
- [151] Robert Endre Tarjan and Mihalis Yannakakis. ‘Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs’. In: *SIAM Journal on Computing* 13.3 (1984), pp. 566–579. doi: 10.1137/0213035 (cited on page 119).
- [152] TPC-H Benchmark. URL: <http://www.tpc.org/tpch/> (cited on page 44).
- [153] Immanuel Trummer and Christoph Koch. ‘Solving the Join Ordering Problem via Mixed Integer Linear Programming’. In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu. ACM, 2017, pp. 1025–1040. doi: 10.1145/3035918.3064039 (cited on pages 14, 77).
- [154] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 978-0-7167-8162-2 (cited on page 19).
- [155] Christopher Umans. ‘The minimum equivalent DNF problem and shortest implicants’. In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 597–611. doi: 10.1109/sfcs.1998.743506 (cited on pages 6, 81, 82).
- [156] Moshe Y. Vardi. ‘The Complexity of Relational Query Languages (Extended Abstract)’. In: *STOC*. 1982, pp. 137–146. doi: 10.1145/800070.802186 (cited on pages 11, 80).
- [157] Vijay V Vazirani. *Approximation algorithms*. Vol. 1. Springer, 2001 (cited on pages 28, 108).
- [158] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. ‘QFix: Diagnosing errors through query histories’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2017, pp. 1369–1384. doi: 10.1145/3035918.3035925 (cited on pages 1, 7).
- [159] Virginia Vassilevska Williams. ‘On some Fine-grained Questions in Algorithms and Complexity’. In: *Proceedings of the International Congress of Mathematicians (ICM 2018)*. WORLD SCIENTIFIC, May 2019. doi: 10.1142/9789813272880_0188 (cited on page 138).
- [160] David P Williamson. *Network flow algorithms*. Cambridge University Press, 2019 (cited on page 103).
- [161] Eugene Wu and Samuel Madden. ‘Scorpion: Explaining Away Outliers in Aggregate Queries’. In: *PVLDB* 6.8 (2013), pp. 553–564. doi: 10.14778/2536354.2536356 (cited on pages 7, 8).
- [162] Mihalis Yannakakis. ‘Technical Perspective: Structure and Complexity of Bag Consistency’. In: *ACM SIGMOD Record* 51.1 (2022), pp. 77–77. doi: 10.1145/3542700.3542718 (cited on pages 9, 15).
- [163] Brit Youngmann, Michael Cafarella, Yuval Moskovitch, and Babak Salimi. ‘On Explaining Confounding Bias’. In: *International Conference on Data Engineering (ICDE)*. IEEE, Apr. 2023, pp. 1846–1859. doi: 10.1109/icde55515.2023.00144 (cited on page 8).