Is Integer Linear Programming All You Need for Deletion Propagation?[*]

A Unified and Practical Approach for Generalized Deletion Propagation

Neha Makhija
Northeastern University, USA
makhija.n@northeastern.edu

Wolfgang Gatterbauer
Northeastern University, USA
w.gatterbauer@northeastern.edu

ABSTRACT

Deletion Propagation (DP) refers to a family of database problems rooted in the classical view-update problem: how to propagate intended deletions in a view (query output) back to the source database while satisfying constraints and minimizing side effects. Although studied for over 40 years, DP variants, their complexities, and practical algorithms have been typically explored in isolation.

This work presents a unified and generalized framework for DP with several key benefits: (1) It *unifies and generalizes* all previously known DP variants, effectively subsuming them within a broader class of problems, including new, well-motivated variants. (2) It comes with a practical and general-purpose algorithm that is "*coarse-grained instance-optimal*": it runs in PTIME for all known PTIME cases and can *automatically exploit structural regularities* in the data, i.e. it does not rely on hints about such regularities as part of the input. (3) It is *complete*: our framework handles all known DP variants in all settings (including those involving self-joins, unions, and bag semantics), and allows us to provide new complexity results. (4) It is *easy to implement* and, in many cases, outperforms prior variant-specific solutions, sometimes by orders of magnitude. We provide the first experimental results for several DP variants previously studied only in theory.

PVLDB Reference Format:
Neha Makhija and Wolfgang Gatterbauer. Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach for Generalized Deletion Propagation. PVLDB, 18(8): 2667 - 2680, 2025.
doi:10.14778/3742728.3742756

PVLDB Artifact Availability:
The source code, data, and other artifacts have been made available at https://github.com/northeastern-datalab/generalized-deletion-propagation.

1 INTRODUCTION

Deletion Propagation (DP) was proposed as early as 1982 [11] and corresponds to a basic view-update problem: *Given a tuple we want to delete from a view, which tuples from the source database should we to delete to accomplish this goal?* Because multiple view tuples may depend on the same base tuple in the source database, deleting that base tuple can result in unintended side effects beyond the requested deletion. The challenge is then to delete just enough tuples from the source to achieve the intended view deletion while avoiding unnecessary side effects. This forms a combinatorial optimization problem. Different optimization goals and different choices about what constitutes a side effect lead to several well-motivated variants of DP that have been studied over the last 40+ years. Some variants are used for query explainability, where both *contrastive* or *abductive* explanations [38] can be obtained with appropriate choices of side effects and optimization goals.

"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 [7, 11] and view side effects [7, 31, 32]: *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 [27] 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) [26, 44], 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 is 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 paper shows is that these 3 challenges can be largely addressed by casting the existing problems as special cases of a unified "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 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 4).

Example 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*

[*]Inspired by the 2017 attention paper [55], an increasing number of research papers promise that "X is all you need (for Y)." Similarly, our conjecture is that *Integer Linear Programs can be designed to solve all PTIME cases of deletion propagation in guaranteed PTIME* and, hence, there is no more need for specialized combinatorial algorithms. We give strong evidence of this conjecture by showing that it holds for all currently known tractable cases. However, since it is a conjecture, we phrase our title as a question.

Proceedings of the VLDB Endowment, Vol. 18, No. 8 ISSN 2150-8097.
doi:10.14778/3742728.3742756

airport fee at the locations they operate at. While cutting costs, the airline wants 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 any extension in many ways (discussed further in Section 5). 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. 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. We are inspired by recent work [35] that showed that for a particular DP variant called *resilience* (i.e., DP with source side effects for a Boolean query), such a unified framework exists and is guaranteed to terminate in PTIME for all known PTIME cases. In contrast, we propose a unified "*coarse-grained instance-optimal*" framework[1] which 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 [32]. 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 [15, 31], unions of conjunctive queries [5], and queries under bags semantics [35]. The overall tractability criterion for queries for such "real-world" queries is overall ill-understood.

Contributions and Outline. 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.

❶ We define Generalized Deletion Propagation (GDP) in Section 4. 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**.

❷ We present an Integer Linear Programming (ILP) formulation for the GDP problem in Section 5. 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**.

❸ 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 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.

❹ 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 challenge of capturing the exact tractability boundary. Concretely, we prove in an online appendix [36] that the ILP formulation of a query with union and self-join that can be solved in PTIME under bag semantics.

❺ We experimentally evaluate the efficiency of our contributions in Section 7. 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.

Due to lack of space, we only provide a proof intuition for each theorem in the main text, and make full proofs, additional examples and experiments available in an online appendix [36]. Our code is available online as well [37].

2 PRELIMINARIES

Standard database notations. A *conjunctive query* (CQ) is a first-order formula $Q(\mathbf{y}) = \exists \mathbf{x}\,(g_1 \wedge \ldots \wedge g_m)$ where the variables $\mathbf{x} = \langle x_1, \ldots, x_\ell \rangle$ 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 $\mathbf{x}_i \subseteq \mathbf{x} \cup \mathbf{y} \cup U$, with U being a universe of constant values. $\mathtt{var}(X)$ denotes the variables in a given relation/atom. Notice that

[1]Notice that we use *instance-optimal* in a more *coarse-grained* sense than is more common in complexity theory [48]. Our focus is on solving all known PTIME cases in PTIME, but not necessarily using the fastest possible specialized algorithm in each case. In other words, we ignore *fine-grained* complexity that differentiates between different classes within PTIME. We discuss this distinction further in Section 3.

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. A *self-join-free CQ* (SJ-free CQ) is one where no relation symbol occurs more than once and thus every atom represents a different relation. A *union over conjunctive queries* (UCQ) is given by $Q(\mathbf{y}) :- \bigcup_{i \in [1,l]} \exists \mathbf{x}_i (g_1^i \wedge \ldots \wedge g_m^i)$ where for each i, $Q(\mathbf{y}) :- \mathbf{x}_i (g_1^i \wedge \ldots \wedge g_m^i)$ is a CQ. We write $\mathcal{D}$ for the database, i.e. the set of tuples in the relations. When we refer to bag semantics, we allow $\mathcal{D}$ to be a multiset of tuples in the relations. Unless otherwise stated, a query in this paper refers to a UCQ, and a database instance $\mathcal{D}$ can be considered to a multiset. However, we may fudge notation and represent $\mathcal{D}$ as a set of tuples if all the multiplicities are 1.

We write $[\mathbf{w}/\mathbf{x}]$ as a valuation (or substitution) of query variables $\mathbf{x}$ by $\mathbf{w}$. A view tuple or an output tuple v is a valuation of the head variables $\mathbf{y}$ that is permitted by $\mathcal{D}$. Similarly, a *witness* w is a valuation of all variables $\mathbf{x}$ that is permitted by $\mathcal{D}$[2]. We can alternately describe a witness as an output tuple for the *full version* of the query Q, which is obtained by making all the free variables existential. We denote the set of views tuples obtained by evaluating a query Q over a database $\mathcal{D}$ simply as $Q(\mathcal{D})$. For example, consider the 2-chain query $Q_2^\infty(x) :- R(x, y), S(y, z)$ over the database $\mathcal{D} = \{r_{12}: R(1, 2), r_{2,2}: R(2, 2), s_{2,3}: S(2, 3)\}$. Then $Q(\mathcal{D}) = \{Q(1), Q(2)\}$ and $\texttt{witnesses}(Q_2^\infty, D) = \{(1, 2, 3), (2, 2, 3)\}$.

Linear Programs (LP). Linear Programs are standard optimization problems [1, 51] in which the objective function and the constraints are linear. A standard form of an LP is $\min \mathbf{c}^\intercal \mathbf{x}$ s.t. $\mathbf{W}\mathbf{x} \geq \mathbf{b}$, where $\mathbf{x}$ denotes the variables, the vector $\mathbf{c}^\intercal$ 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. The objective function $f = \mathbf{c}^\intercal \mathbf{x}$ may also be referred to as a *soft constraint*. We use f^* to denote the optimal value of the objective function. If the variables are constrained to be integers, the resulting program is called an Integer Linear Program (ILP). The *LP relaxation* of an ILP program is obtained by removing the integrality constraint for all variables.

3 RELATED WORK

We will discuss in Section 4 related work on problems that fit within the umbrella of DP. This section covers additional related work concerning broader themes that are discussed in this paper.

Reverse Data Management (RDM). DP can be seen as a type of reverse data management problem [41]. RDM problems search for optimal interventions in the input data that would lead to a desired output. RDM problems are useful in many applications, such as intervention-based approaches for explanations [20, 25, 49, 57], fairness [16, 50], causal inference [17], and data repair [56]. The Tiresias system [42] 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 MIP, and not on building a unified method that can recover tractable cases.

Intervention-Based Explanations. Formal Explainability in AI (FXAI) [38] distinguishes between two types of explanations: *Abductive* explanations (or locally sufficient reasons [4]) 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) [26, 44] as an instance of abductive explanation, and the Resilience Problem (RES) [35] as a contrastive explanation. Generalized Deletion Propagation (GDP) subsumes both SWP and RES and can give *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 [39, 40]: '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).

Linear Optimization Solvers. A key practical advantage of modeling problems as ILPs is that there are many highly-optimized ILP solvers, both commercial [24] and free [45] which can obtain exact results efficiently, in practice. ILP formulations are standardized, and thus programs can easily be swapped between solvers. Any advances made over time by these solvers can automatically make implementations of these problems better over time. For our experimental evaluation we use Gurobi[3] which uses an LP based branch-and-bound method to solve ILPs [22]. 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 whose objective is equal to the LP relaxation optimum, then the solver has found a guaranteed optimal solution and is done. In other words, if we can *prove that the LP relaxation of our given ILP formulation has an integral optimal solution*, then we are guaranteed that our original ILP formulation will terminate in PTIME, even without changing the ILP formulation or letting the solver know about the theoretical complexity.

Complexity of solving ILPs. Solving ILPs is NPC [28], while LPs can be solved in PTIME with Interior Point methods [9, 21]. The specific conditions under which ILPs become tractable is an entire field of study. It is known that if there is an optimal integral assignment to the LP relaxation, then the original ILP can be solved in PTIME as well. There are many structural characteristics that define when the LP is guaranteed to have an integral minimum, and thus where ILPs are in PTIME. For example, if the constraint matrix of an ILP is *Totally Unimodular* [51] then the LP always has the same optima. Similarly, if the constraint matrix is *Balanced* [10], several classes of ILPs are PTIME. We do not use any of these techniques in this paper, but we believe future research in this area may help *automatically* identify more tractable cases of DP problems.

ILPs and Constraint Optimization in Databases. Integer Linear Programming has been used in databases for problems such as in solving package queries [6], query optimization [54], and general optimization applications [53]. However, other than our

[2]Note that this differs from an alternate notion of witness [7, 26] which defines the witness of an output tuple as a subset of input tuples such that running a query over the subset produces the output tuple

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

recent work on the resilience problem [35], we are unaware of any work in databases that uses ILPs to automatically recover tractable cases by proving that the condition ILP = LP holds for the PTIME cases, i.e. that the LP relaxation has an optimal integral value 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 3 and Fig. 6). In Sections 5.2 and 5.3 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 Fig. 8 with a reduction from over 3 hours to under 20 seconds.

Instance Optimal Algorithms. Our notion of "*coarse-grained instance optimality*" is inspired by the notion of instance optimality in complexity theory [48]. 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 [13], and join computation [2, 30, 46].

4 GENERALIZED DELETION PROPAGATION

We introduce Generalized Deletion Propagation (GDP) which generalizes all prior variants of deletion propagation, and 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 possible side effects; one could care about side effects on *another view* that is different from the one in which the deletion occurs. (2) It is natural to enforce constraints or optimize side effects over multiple views. (3) Prior variants focus on a specific type of constraint (either deletion or preservation). In practice, one might want to combine these constraints (e.g., minimizing deletions from one view while maximizing deletions from another). These extensions are motivated with examples in Section 4.3.

We observe that with 4 different sets of views, we can model all existing problems and can also combine individual constraints in arbitrary ways. Definition 4.1 thus defines *generalized deletion propagation* as a constraint optimization problem over four set of views. These sets of views correspond to four primitive operations (or requirements) that typically occur in deletion propagation variants - a requirement to delete tuples from a view, preserve tuples in a view, minimize side effects on a view, or to maximize side effects on a view. Section 4.2 discusses how the GDP definition encapsulates all past variants of DP as special cases (also depicted in Fig. 1), while Section 4.3 motivates DP new variants that are captured by GDP.

4.1 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). $\boldsymbol{Q}$ represents an ordered set of queries, and Q^i represents the i^{th} query in $\boldsymbol{Q}$. $|Q(\mathcal{D})|$ is defined as the number of output tuples in $Q(\mathcal{D})$ and $|\boldsymbol{Q}(\mathcal{D})|= \sum_{Q\in\boldsymbol{Q}}|Q(\mathcal{D})|$ as the number of output tuples across all views in $\boldsymbol{Q}$. 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 $\Delta\boldsymbol{Q}(\mathcal{D},\Gamma)$ as the set of tuples deleted from all views in $\boldsymbol{Q}$:

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

Definition 4.1 (Generalized Deletion Propagation (GDP)). Given four ordered sets of monotone queries $\boldsymbol{Q}_{\texttt{del}}, \boldsymbol{Q}_{\texttt{pres}}, \boldsymbol{Q}_{\texttt{min}}$ and $\boldsymbol{Q}_{\texttt{max}}$ over a database $\mathcal{D}$, and vectors of positive integers $\mathbf{k}_{\texttt{del}}$ and $\mathbf{k}_{\texttt{pres}}$ of size equal to the number of views in $\boldsymbol{Q}_{\texttt{del}}$ and $\boldsymbol{Q}_{\texttt{pres}}$ respectively, the GDP problem is the task of determining a set of input tuples $\Gamma\subseteq\mathcal{D}$ such that

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

is minimized and the following hard constraints are satisfied:

(1) Deleting Γ from the database $\mathcal{D}$ deletes at least $k^i_{\texttt{del}}$ output tuples from the i^{th} view defined by $\boldsymbol{Q}_{\texttt{del}}$ i.e.,
$$|Q^i_{\texttt{del}}(\mathcal{D}\setminus\Gamma)|\leq |Q^i_{\texttt{del}}(\mathcal{D})|-k^i_{\texttt{del}}$$
(2) Deleting Γ from the database $\mathcal{D}$ preserves at least $k^i_{\texttt{pres}}$ output tuples from the i^{th} view defined by $\boldsymbol{Q}_{\texttt{pres}}$ i.e.,
$$|Q^i_{\texttt{pres}}(\mathcal{D}\setminus\Gamma)|\geq k^i_{\texttt{pres}}$$

4.2 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.

*4.2.1 Deletion Propagation with Source Side Effects (*DP-SS*) [7, 11] and Resilience (RES) [5, 14, 15, 35, 43, 43].* Deletion Propagation with source side effects (DP-SS) is one of the two originally formulated variants of the deletion propagation problem [7].

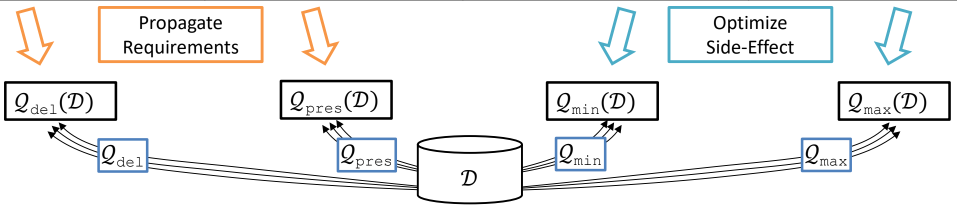
*Definition 4.2 (*DP-SS*).* Given a view defined by a query Q over a database $\mathcal{D}$, and an output tuple $t\in Q(\mathcal{D})$, the deletion propagation with source side effects problem is the task of determining a set of input tuples $\Gamma\subseteq\mathcal{D}$ such that $|\Gamma|$ is minimized and t is not contained in $Q(\mathcal{D}\setminus\Gamma)$. In other words,

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

DP-SS is a special case of GDP - we can solve a DP-SS problem by setting $Q^1_{\texttt{del}}$ to be a query with constants selecting for the values of t, $k^1_{\texttt{del}} = 1$ and setting $\boldsymbol{Q}_{\texttt{min}}$ 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 $\boldsymbol{Q}_{\texttt{min}}$, 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 [14], and a large amount of literature has been dedicated to understanding its complexity [5, 14, 15, 35, 43]. The complexity results for the RES problem also imply complexity results for the DP-SS problem [14]. Existing work has shown a complexity dichotomy for self-join free conjunctive queries, both

Problem instance	Hard Constraints		Soft Constraints	
	DELETE:	PRESERVE:	MINIMIZE:	MAXIMIZE:
DP-SS	All tuples in $Q_{\mathtt{del}}(\mathcal{D})$	–	Deletions in $\mathcal{Q}_{\mathtt{min}} = \{R \in \mathcal{D}\}$	–
DP-VS	All tuples in $Q_{\mathtt{del}}(\mathcal{D})$	–	Deletions in $\mathcal{Q}_{\mathtt{min}}(\mathcal{D}) \supseteq \mathcal{Q}_{\mathtt{del}}(\mathcal{D})$	–
ADP-SS	$\geq k$ tuples in $Q_{\mathtt{del}}(\mathcal{D})$	–	Deletions in $\mathcal{Q}_{\mathtt{min}} = \{R \in \mathcal{D}\}$	–
SWP	–	All tuples in $\mathcal{Q}_{\mathtt{pres}}(\mathcal{D})$	–	Deletions in $\mathcal{Q}_{\mathtt{max}} = \{R \in \mathcal{D}\}$
GDP	$\geq k^i_{\mathtt{del}}$ tuples in $\mathcal{Q}^i_{\mathtt{del}}(\mathcal{D})$	$\geq k^i_{\mathtt{pres}}$ tuples in $\mathcal{Q}^i_{\mathtt{pres}}(\mathcal{D})$	Deletions in $\mathcal{Q}_{\mathtt{min}}$	Deletions in $\mathcal{Q}_{\mathtt{max}}$



Propagate Requirements
Optimize Side-Effect
$\mathcal{Q}_{\mathtt{del}}(\mathcal{D})$
$\mathcal{Q}_{\mathtt{pres}}(\mathcal{D})$
$\mathcal{Q}_{\mathtt{min}}(\mathcal{D})$
$\mathcal{Q}_{\mathtt{max}}(\mathcal{D})$
$\mathcal{Q}_{\mathtt{del}}$
$\mathcal{Q}_{\mathtt{pres}}$
$\mathcal{D}$
$\mathcal{Q}_{\mathtt{min}}$
$\mathcal{Q}_{\mathtt{max}}$

Figure 1: 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) [11], Deletion Propagation with View Side Effects (DP-VS) [33], Aggregated Deletion Propagation with Source Side Effects (ADP-SS) [27], and Smallest Witness Problem (SWP) [26]. 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.

under set [14] and bag semantics [35], yet only few tractability results for queries with self-joins and unions are known [5, 15, 35]. The RES problem can be modelled as a special case of GDP similarly as DP-SS, with the added restriction that $Q^1_{\mathtt{del}}$ is a boolean query.

4.2.2 Deletion Propagation: View Side Effect (DP-VS) [7, 31–33]. 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 4.3 (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 $Q(D \setminus \Gamma)$. 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 $\mathcal{Q}_{\mathtt{del}}(\mathcal{D})$ contains a single query whose output is the output tuple t (just like in DP-SS), $k^1_{\mathtt{del}} = 1$, and $\mathcal{Q}_{\mathtt{min}}$ 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 [32].

4.2.3 Aggregated Deletion Propagation with Source Side effect (ADP-SS) [27]. 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 4.4 (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,

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

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 [27].

4.2.4 Smallest Witness Problem (SWP) [44]. 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 [26]. 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):

Definition 4.5 (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 $\Gamma \subseteq D$ 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 [26]. 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.

4.3 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. These extensions can be combined in arbitrary ways, leading to a rich set of new problems.

Extension 1: New types of side effects. Existing DP variants focus on minimizing source side effects or view side effects. However, one can easily imagine a user wanting to delete tuples from one view while minimizing side effects on another view. For instance, in Example 1, where the output view that deletion constraints very defined on (the connectivity network), was different from the view side effects were measured on (view of popular connections).

Extension 2: Constraints over multiple views. Existing DP variants focus on a single view from which deletions are propagated. However, one can imagine a scenario where tuples from multiple views are to be deleted. As we saw in Example 1, the airline wanted to cut down on multiple costs such as fuel costs and 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[4].

Extension 3: Combination of Deletion and Preservation Constraints. Current DP variants focus on either deletion constraints (DP-SS, DP-VS, ADP-SS) or preservation constraints (SWP) exclusively. However, one may want to enforce both deletion and preservation constraints simultaneously - like in Example 1 where it matters to cut down on costs but also preserve the popular routes.

5 ILP FRAMEWORK FOR GDP

This section specifies an Integer Linear Program (ILP) ILP[GDP] which returns an optimal solution for GDP for any instance supported by Definition 4.1. 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 later investigates how known tractable cases can be solved in PTIME, despite the problem being NPC in general. The input to the ILP[GDP] are the four sets of view-defining queries $\mathcal{Q}_{\texttt{del}}, \mathcal{Q}_{\texttt{pres}}, \mathcal{Q}_{\texttt{min}}, \mathcal{Q}_{\texttt{max}}$ over a database $\mathcal{D}$. Note that any of these sets can be empty as well.[5] 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 ILP[GDP].

In a slight abuse of notation we write $v \in \mathcal{Q}(\mathcal{D})$ for $v \in \bigcup_{Q \in \mathcal{Q}} Q(\mathcal{D})$ and similarly, $w \in \mathcal{Q}^F(\mathcal{D})$ for $w \in \bigcup_{Q \in \mathcal{Q}} Q^F(\mathcal{D})$. 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 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 $v_1 \in Q^i_{\texttt{del}}(\mathcal{D})$ and $v_2 \in Q^j_{\texttt{pres}}(\mathcal{D})$ with $Q^i_{\texttt{del}} = Q^j_{\texttt{pres}}$, 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 theorectically 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.

5.1 A basic ILP Formulation for GDP

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

5.1.1 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 $\mathcal{Q}_{\texttt{del}}(\mathcal{D}), \mathcal{Q}_{\texttt{pres}}(\mathcal{D}), \mathcal{Q}_{\texttt{min}}(\mathcal{D}), \mathcal{Q}_{\texttt{max}}(\mathcal{D})$, and $X[w]$ for each witness w in the full version of those queries.

5.1.2 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 4.1, we thus count the side effects as the number of output tuples deleted from $\mathcal{Q}_{\texttt{min}}(\mathcal{D})$ plus the number of tuples preserved in $\mathcal{Q}_{\texttt{max}}(\mathcal{D})$, 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:

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

5.1.3 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 $\mathcal{Q}_{\texttt{del}}$ and $\mathcal{Q}_{\texttt{pres}}$, respectively. The deletion constraints specify that at least $k^i_{\texttt{del}}$ tuples must be deleted from each view $Q^i_{\texttt{del}} \in \mathcal{Q}_{\texttt{del}}$, while the preservation constraints specify that at least $k^i_{\texttt{pres}}$ tuples must be preserved in each view $Q^i_{\texttt{pres}} \in \mathcal{Q}_{\texttt{pres}}$

[4]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. Cutting 2% of total costs is not necessarily the same as cutting 1% of fuel costs and 1% of airport lease costs.

[5]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.

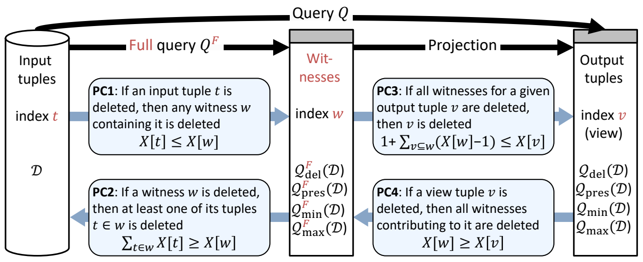

Query Q
Input tuples
index t
$\mathcal{D}$
Full query Q^F
Wit-nesses
index w
$\mathcal{Q}^F_{\text{del}}(\mathcal{D})$
$\mathcal{Q}^F_{\text{pres}}(\mathcal{D})$
$\mathcal{Q}^F_{\text{min}}(\mathcal{D})$
$\mathcal{Q}^F_{\text{max}}(\mathcal{D})$
Projection
Output tuples
index v (view)
$\mathcal{Q}_{\text{del}}(\mathcal{D})$
$\mathcal{Q}_{\text{pres}}(\mathcal{D})$
$\mathcal{Q}_{\text{min}}(\mathcal{D})$
$\mathcal{Q}_{\text{max}}(\mathcal{D})$
PC1: If an input tuple t is deleted, then any witness w containing it is deleted $X[t] \leq X[w]$
PC2: If a witness w is deleted, then at least one of its tuples $t \in w$ is deleted $\sum_{t \in w} X[t] \geq X[w]$
PC3: If all witnesses for a given output tuple v are deleted, then v is deleted $1+\sum_{v \subseteq w}(X[w]-1) \leq X[v]$
PC4: If a view tuple v is deleted, then all witnesses contributing to it are deleted $X[w] \geq X[v]$

Figure 2: Propagation constraints in our ILP formulation $\texttt{ILP}_{\texttt{N}}[\texttt{GDP}]$, 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.

(which is equivalent to deleting at most $|Q^i_{\texttt{pres}}(\mathcal{D})| - k^i_{\texttt{pres}}$ tuples):

$$\sum_{v \in Q^i_{\texttt{del}}(\mathcal{D})} X[v] \geq k^i_{\texttt{del}} \qquad \forall Q^i_{\texttt{del}} \in \mathcal{Q}_{\texttt{del}}$$

$$\sum_{v \in Q^i_{\texttt{pres}}(\mathcal{D})} X[v] \leq |Q^i_{\texttt{pres}}(\mathcal{D})| - k^i_{\texttt{pres}} \qquad \forall Q^i_{\texttt{pres}} \in \mathcal{Q}_{\texttt{pres}}$$

(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 [7].

Figure 2 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:** ($\longrightarrow$) If an input tuple t is deleted, then any witness w containing it is deleted. ($\longleftarrow$) 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:** ($\longleftarrow$) If a witness w is deleted, then at least one of its tuples $t \in w$ is deleted. ($\longrightarrow$) Alternatively, if all tuples $t \in w$ are not deleted, then the witness w is not deleted.
$$\sum_{t \in w} X[t] \geq X[w] \tag{1}$$
- **PC3:** ($\longrightarrow$) If all witnesses for a given output tuple v are deleted, then v is deleted. ($\longleftarrow$) 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:** ($\longleftarrow$) If a view tuple v is deleted, then all witnesses contributing to it are deleted. ($\longrightarrow$) If a witness w is not deleted, then any view tuple $v \subseteq w$ is not deleted.
$$X[w] \geq X[v], v \subseteq w$$

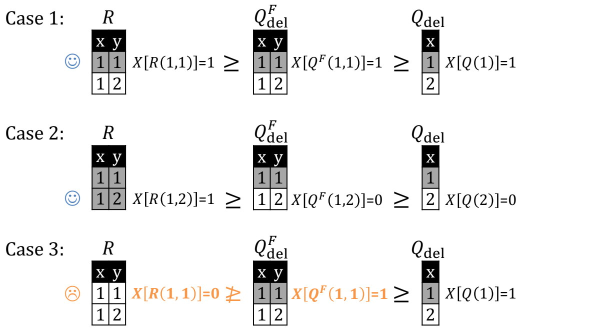


Figure 3: Example 2: The true state of deletions in the database $\mathcal{D}$ is always faithfully represented by database variables (e.g., $R(1,2)$ is deleted and thus $X[R(1,2)] = 1$ and is grayed out). However, deletions in the views defined by a query in $\mathcal{Q}_{\texttt{del}}$ need to provide 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).

5.1.4 Naive ILP. We define $\texttt{ILP}_{\texttt{N}}[\texttt{GDP}]$ as the program resulting from our definitions of ILP variables, objective function, and constraints, and will sometimes refer to it as the "naive ILP".

Theorem 5.1. *[Naive ILP] The interventions given by an optimum solution of* $\texttt{ILP}_{\texttt{N}}[\texttt{GDP}]$ *for any* $\mathcal{D}, \mathcal{Q}_{\texttt{del}}, \mathcal{Q}_{\texttt{pres}}, \mathcal{Q}_{\texttt{min}}, \mathcal{Q}_{\texttt{max}}, \mathbf{k}_{\texttt{del}}, \mathbf{k}_{\texttt{pres}}$ *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.

5.2 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 $v_1 \in Q^i_{\texttt{del}}(\mathcal{D})$ means it is necessarily deleted, and setting $X[v_2] = 0$ for $v_2 \in Q^i_{\texttt{pres}}(\mathcal{D})$ 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.

Example 2 (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 Fig. 3 some possible variable assignments 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

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

Figure 4: 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.

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 2 showed that for a variable $X[w]$ for a witness w in $\mathcal{Q}^F_{\texttt{del}}(\mathcal{D})$, 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 $w \notin \mathcal{Q}^F_{\texttt{del}}(\mathcal{D})$. 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 $\mathcal{Q}^F_{\texttt{del}}$ 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 $\mathcal{Q}_{\texttt{del}}(\mathcal{D})$ provide a lower bound on the deletions of tuples in the database (Fig. 3). The exact same reasoning applies to the $X[v]$ variables for view tuples in $\mathcal{Q}_{\texttt{del}}(\mathcal{D})$ as well.

Similarly, witness and view variables for $\mathcal{Q}^F_{\texttt{max}}$ and $\mathcal{Q}_{\texttt{max}}$ 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 witness/view tuple may or may not be deleted.

Symmetrically, for $\mathcal{Q}_{\texttt{pres}}(\mathcal{D})$ and $\mathcal{Q}_{\texttt{min}}(\mathcal{D})$, 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 $\mathcal{Q}_{\texttt{pres}}$ and $\mathcal{Q}_{\texttt{min}}$ 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 4 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 Section 5.1.1 to obtain

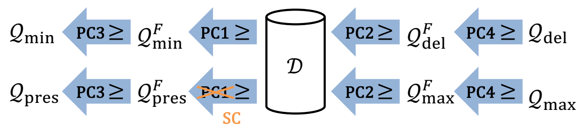

$\mathcal{Q}_{\text{min}}$
PC3 ≥
$\mathcal{Q}^F_{\text{min}}$
PC1 ≥
$\mathcal{D}$
PC2 ≥
$\mathcal{Q}^F_{\text{del}}$
PC4 ≥
$\mathcal{Q}_{\text{del}}$
$\mathcal{Q}_{\text{pres}}$
PC3 ≥
$\mathcal{Q}^F_{\text{pres}}$
~~PC1~~ ≥
SC
PC2 ≥
$\mathcal{Q}^F_{\text{max}}$
PC4 ≥
$\mathcal{Q}_{\text{max}}$

Figure 5: 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 (Section 5.2). Also shown is how our Smoothing Constraints (SC) replace PC1 for $\mathcal{Q}^F_{\texttt{pres}}(\mathcal{D})$ (Section 5.3). It is that replacement that gives us a powerful PTIME guarantee for PTIME problems (see later Fig. 8 from the experiments).

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 $\mathcal{Q}_{\texttt{pres}}$ and $\mathcal{Q}_{\texttt{min}}$, but do not to $\mathcal{Q}_{\texttt{del}}$ and $\mathcal{Q}_{\texttt{max}}$. 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 $\mathcal{Q}_{\texttt{del}}$ and $\mathcal{Q}_{\texttt{max}}$, but not to $\mathcal{Q}_{\texttt{pres}}$ and $\mathcal{Q}_{\texttt{min}}$. Figure 5 summarizes the selective application of PCs to the different views.

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

Theorem 5.2. *[Wildcard ILP] The interventions suggested by an optimum solution of* $\texttt{ILP}_{\texttt{W}}[\texttt{GDP}]$ *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).

5.3 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 ingredient in the results of Section 6 where we show that an ILP with smoothing constraints $\texttt{ILP}_{\texttt{S}}[\texttt{GDP}]$ can be solved in PTIME for all known tractable cases. Hence, we also refer to $\texttt{ILP}_{\texttt{S}}[\texttt{GDP}]$ as simply $\texttt{ILP}[\texttt{GDP}]$.

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 [29] 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 $Q_{\texttt{pres}}$. Recall that for $Q_{\texttt{pres}}$, 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 $Q_{\texttt{pres}}$, 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 $w_1, w_2, \ldots, w_k$ 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 $v \in Q_{\texttt{pres}}(\mathcal{D})$:

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

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

Theorem 5.3. *[Smoothened ILP] The interventions suggested by an optimum solution of* $\texttt{ILP}_{\texttt{S}}[\texttt{GDP}]$ *with wildcard semantics and smoothened constraints form an optimum solution to* GDP.

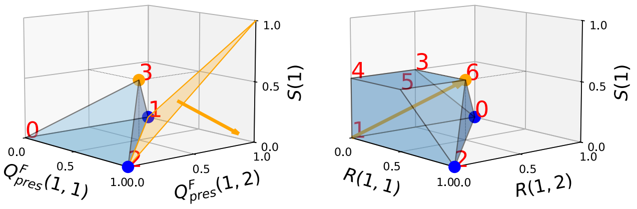
Proof Intuition. Adding the smoothing constraint to the wildcard ILP always preserves at least one optimal solution - this follows also from the argument above the the smoothing constraint can be derived by logically following the wildcard semantics.

An interesting asymmetry. We notice an interesting asymmetry at play. We could apply a symmetric smoothing constraint in the case for PC4 on $Q_{\texttt{del}}$. Interestingly, such an additional smoothing constraint would *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 $Q_{\texttt{del}}$.

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 3 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 Proposition 6.3, we show that this is the case for all prior known PTIME cases of SWP.

Example 3 (Power of smoothing). *Consider again the $\mathcal{D}$ from Example 2: with $R(1,1)$, $R(1,2)$, and $S(1)$. We want to solve the smallest witness problem $\texttt{SWP}(Q_{\texttt{pres}}, \mathcal{D})$ for $Q_{\texttt{pres}}(x) :\!- R(x,y), S(x)$. To model it as* GDP, *we set $Q_{\texttt{pres}}$ to be $\langle Q_{\texttt{pres}}\rangle$ and $\mathbf{k}_{\texttt{pres}} = \langle k_{\texttt{pres}}\rangle$ with $k_{\texttt{pres}} = 1$, which is the number of output tuples in $Q_{\texttt{pres}}(\mathcal{D})$.*



(a) Projection on variables from SC. **(b) Projection on database tuples.**

Figure 6: Example 3: Our Smoothing Constraint (SC) acts as a cutting plane, removing a non-integral optimal point from the LP relaxation of our ILP formulation (see details in text).

We also set $Q_{\texttt{max}} = \langle Q^1_{\texttt{max}}(x,y) :\!- R(x,y), Q^2_{\texttt{max}}(x) :\!- S(x)\rangle$ and $Q_{\texttt{del}} = Q_{\texttt{min}} = \emptyset$. Our GDP *formulation is as follows:*

$$f(\mathbf{X}) = -(X[Q^1_{\texttt{max}}(1,1)] + X[Q^1_{\texttt{max}}(1,2)] + X[Q^2_{\texttt{max}}(1)])$$

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

$$X[Q_{\texttt{pres}}(1)] \leq 0 \qquad \text{(UC)}$$
$$X[Q^F_{\texttt{pres}}(1,1)] + X[Q^F_{\texttt{pres}}(1,2)] - 1 \leq X[Q_{\texttt{pres}}(1)] \qquad \text{(PC3)}$$
$$X[R(1,1)] \leq X[Q^F_{\texttt{pres}}(1,1)] \qquad \text{(PC1)}$$
$$X[S(1)] \leq X[Q^F_{\texttt{pres}}(1,1)] \qquad \text{(PC1)}$$
$$X[R(1,2)] \leq X[Q^F_{\texttt{pres}}(1,2)] \qquad \text{(PC1)}$$
$$X[S(1)] \leq X[Q^F_{\texttt{pres}}(1,2)] \qquad \text{(PC1)}$$
$$X[R(1,1)] \leq X[Q^{1F}_{\texttt{max}}(1,1)] \qquad \text{(PC2)}$$
$$X[R(1,2)] \leq X[Q^{1F}_{\texttt{max}}(1,2)] \qquad \text{(PC2)}$$
$$X[S(1)] \leq X[Q^{2F}_{\texttt{max}}(1)] \qquad \text{(PC2)}$$
$$X[Q^{1F}_{\texttt{max}}(1,1)] \leq X[Q^1_{\texttt{max}}(1,1)] \qquad \text{(PC4)}$$
$$X[Q^{1F}_{\texttt{max}}(1,2)] \leq X[Q^1_{\texttt{max}}(1,2)] \qquad \text{(PC4)}$$
$$X[Q^{2F}_{\texttt{max}}(1)] \leq X[Q^2_{\texttt{max}}(1)] \qquad \text{(PC4)}$$

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 $X[Q^F_{\texttt{pres}}(1,1)]$ and $X[Q^F_{\texttt{pres}}(1,2)]$ can take values 0.5, which is why $X[Q^1_{\texttt{pres}}(1)]$ can be set to 0 while fulfilling all constraints.

Our smoothing constraint for this example is the following

$$X[S(1)] \leq X[Q^F_{\texttt{pres}}(1,1)] + X[Q^F_{\texttt{pres}}(1,2)] - 1 \qquad \text{(SC)}$$

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

The SC ensures that if $X[S(1)]$ is set to 0.5, then $X[Q^F_{\texttt{pres}}(1,1)] + X[Q^F_{\texttt{pres}}(1,2)] \geq 1.5$, thus violating PC4, and thereby effectively removing the non-integer solution. To gain more intuition, Fig. 6 shows the polytope of the LP relaxation of our wildcard formulation projected on either the variables involved in SC (Fig. 6a), or the three input tuples (Fig. 6b). The optimal LP solution corresponds to the orange point (point 3 in Fig. 6a, point 6 in Fig. 6b), and the two optimal ILP solutions correspond to the two blue points (points 1 and

2 in Fig. 6a, points 0 and 2 in Fig. 6b). Notice how our SC (shown as yellow cutting plane in Fig. 6a) 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 Fig. 6b), 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 Fig. 8 in Section 7 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 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 $\mathtt{ILP}[\mathtt{GDP}_{\mathtt{DP\text{-}SS}}(Q, \mathcal{D}, t)]$ 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 Section 4.2.1. We now claim that the LP relaxation $\mathtt{LP}[\mathtt{GDP}_{\mathtt{DP\text{-}SS}}]$ of such an ILP $\mathtt{ILP}[\mathtt{GDP}_{\mathtt{DP\text{-}SS}}]$, 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 ILP[GDP] can be used to solve DP-SS in PTIME for such queries.

Proposition 6.1. $\mathtt{LP}[\mathtt{GDP}_{\mathtt{DP\text{-}SS}}(Q, \mathcal{D})] = \mathtt{DP\text{-}SS}(Q, \mathcal{D})$ *for all database instances $\mathcal{D}$ under set semantics if the existential query Q^E does not contain a triad.* $\mathtt{LP}[\mathtt{GDP}_{\mathtt{DP\text{-}SS}}(Q, \mathcal{D})] = \mathtt{DP\text{-}SS}(Q, \mathcal{D})$ *for all database instances $\mathcal{D}$ under bag semantics if Q^E is a linear query.*

Proof Intuition. We show that $\mathtt{ILP}[\mathtt{GDP}_{\mathtt{DP\text{-}SS}}]$ is identical to a specialized ILP that has been proposed [35] for resilience. Since that paper also shows that for all tractable queries, the LP relaxation of the ILP is integral, the results naturally carry over.

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 [32]. We prove that for such queries that have the head domination property, if we pose DP-VS$(Q, \mathcal{D}, t)$ in the GDP framework, then the LP relaxation $\mathtt{LP}[\mathtt{GDP}_{\mathtt{DP\text{-}VS}}(Q, \mathcal{D}, t)]$ is equivalent to the solution of DP-VS$(Q, \mathcal{D}, t)$.

Proposition 6.2. $\mathtt{LP}[\mathtt{GDP}_{\mathtt{DP\text{-}VS}}(Q, \mathcal{D}, t)] = \mathtt{DP\text{-}VS}(Q, \mathcal{D}, t)$ *for all database instances $\mathcal{D}$ under 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 optimal solution for DP-VS is side effect free [32]. 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.

SWP. It is known that SWP is PTIME for self-join free queries if and only if they have the *head clustering property* [26], 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 $\mathtt{LP}[\mathtt{GDP}_{\mathtt{SWP}}(Q, \mathcal{D})]$ is equivalent to the solution of SWP$(Q, \mathcal{D})$.

Proposition 6.3. $\mathtt{LP}[\mathtt{GDP}_{\mathtt{SWP}}(Q, \mathcal{D})] = \mathtt{SWP}(Q, \mathcal{D})$ *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.

ADP-SS. A complexity dichotomy for the ADP-SS problem for self-join free queries under set semantics is known. However, the complexity criterion [27] 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 ADP-SS$(Q, \mathcal{D}, k)$ in the GDP framework, then the LP relaxation LP[ADP-SS] is equivalent to the solution of ADP-SS$(Q, \mathcal{D}, k)$.

Proposition 6.4. $\mathtt{LP}[\mathtt{GDP}_{\mathtt{ADP\text{-}SS}}(Q, \mathcal{D})] = \mathtt{SWP}(Q, \mathcal{D})$ *for all database instances $\mathcal{D}$ if Q is a self-join free for which* 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[27] 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 Proposition 6.1 as an argument, while for the base case of singleton relations, we use the proof similar to that of Proposition 6.2. We also show that the value of the LP relaxation is preserved even when the query is decomposed into multiple parts.

7 EXPERIMENTS

The goal of our experiments is to evaluate the performance of our unified ILP[GDP] (which is our short form for $\mathtt{ILP}_{\mathtt{S}}[\mathtt{GDP}]$) by answering the following 4 questions: (Q1) Is the performance of

ILP[GDP] comparable to previously proposed specialized algorithms tailored to PTIME cases of particular DP problems? (Q2) Can our unified ILP[GDP] indeed efficiently solve new tractable cases with self-joins, unions, and bag semantics that we proved to be in PTIME in the full paper [36]? (Q3) What, if any, is the performance benefit we obtain via smoothing constraints as discussed in Section 5? (Q4) What is the scalability of solving completely novel DP problems that fall into our unified GDP framework on real-world data?

Algorithms. ILP[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 Ⓥ DP-VS [32], Ⓢ SWP [26], and Ⓐ ADP-SS [27], 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: Ⓥ [32], Ⓢ [26], Ⓐ [27]. To the best of our knowledge, no experimental evaluation has ever been undertaken for some of these algorithms [26, 32]. We do not include experimental comparison for DP-SS, as for this problem the ILP[GDP] produced is exactly the same as a prior specialized approach [35], 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 number that is smaller than a pre-specified max bag size of 10. For answering (4) in Fig. 9, we use an existing flights' database [47] 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 the ILP solver. In Fig. 7a, we see that ILP[GDP] is about 20 times worse than the specialized algorithms for Ⓓ 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 Figs. 7c and 7e, we see that ILP[GDP] is about 2 and 3 orders of magnitude *faster than the specialized algorithms* for Ⓢ SWP and Ⓐ ADP-SS, both of which use decomposition based techniques, with the specialized algorithm for Ⓐ ADP-SS also requiring

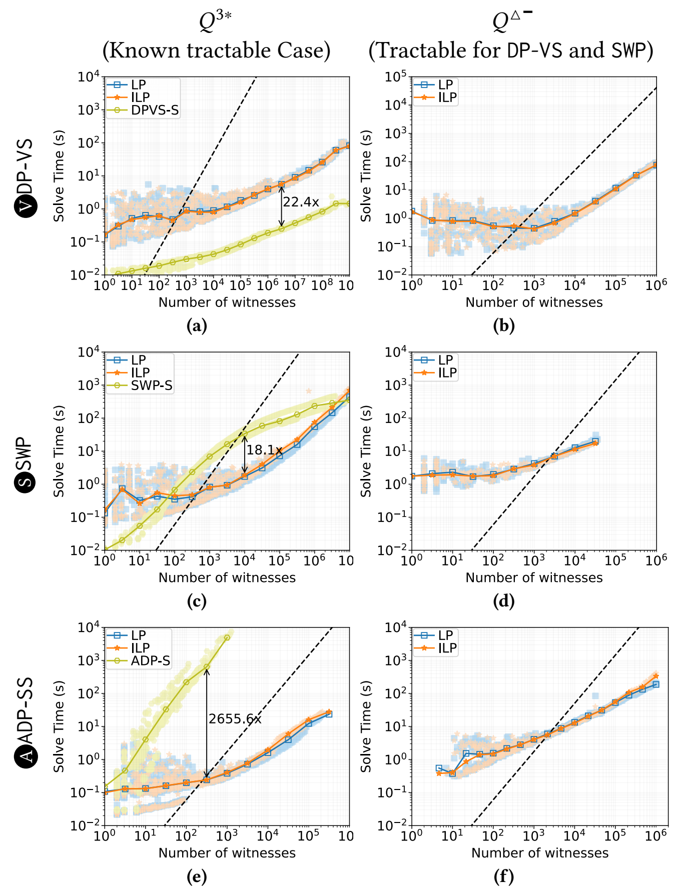
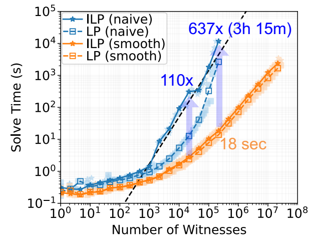


Figure 7: (Q1)/(Q2): Performance of ILP[GDP] on a previously known (left column compared against prior specialized algorithms) and a newly discovered tractable query (right column with no prior known specialized algorithm) for three prior studied problems Ⓥ DP-VS, Ⓢ SWP, and Ⓐ ADP-SS. In all cases, ILP[GDP] scales well under the theoretical worst case complexity of ILPs and LPs. This is due to the nice algorithmic properties of ILP[GDP], which ensure that the optimal LP solution is integral for all known tractable queries.

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[$\text{GDP}_{\text{DP-VS}}$], ILP[$\text{GDP}_{\text{SWP}}$] and ILP[$\text{GDP}_{\text{ADP-SS}}$] for $Q^{\triangle^-}$, a query with self-joins and a union that we run under bag semantics. We showed in the online appendix that the Ⓓ DP-VS, Ⓢ SWP, and Ⓐ 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 Figs. 7b, 7d and 7f 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 8 shows two orders of magnitude speedup in the ILP solving time after adding our Smoothing Constraint (SC). We run ILP[GDP_{SWP}] for the



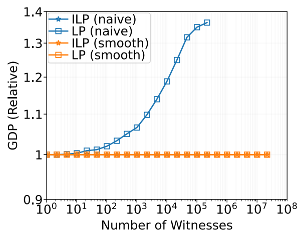


Figure 8: (Q3): Experiment showing the power of Smoothing Constraints in ILP[GDP]: we observe that ILP[GDP] is orders-of-magnitude faster than the naive ILP formulation ILP_N[GDP], while also guaranteeing the optimality of its LP relaxation. Contrast with the LP relaxation of ILP_N[GDP] which can have an over 30% higher optimal objective (GDP) value.

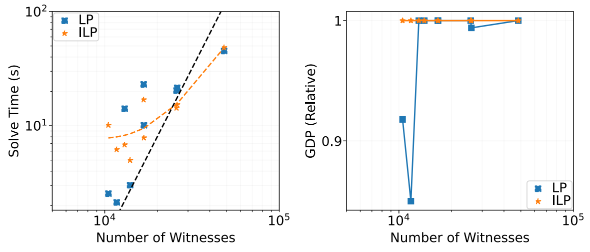


Figure 9: (Q4): Performance Evaluation of the Generalized Deletion Propagation over a real-world dataset shows fast solve times (within a minute) and comparable to linear-time (black dashed line) scalability.

3-star query Q^{3*}, contrasting ILP[GDP] with and without smoothing constraints. Since we show in Section 6 that the LP relaxation of ILP[GDP] shares the same optimum objective value, this surprising speed-up 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 $\text{ILP}[\text{GDP}_\text{SWP}]$ 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.[6]

(Q4) General performance. We use the flights' database [47] that shows 500K+ flights operated by 17 different airlines in Jan 2019. We take the use case of Example 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 "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 Fig. 9 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.[7]

8 A NOTE ON SYSTEM IMPLEMENTATION

Our current proof of concept (see Section 7) 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 [18, 19], GProM [3], or ProvSQL [52]. 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 [6, 8, 34]. Such integrations of highly sophisticated solvers for solving important database problems have been previously proposed for consistent query answering [12] and query optimization [54], 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 [32]. All other approaches require writing functionality in programming languages other than SQL, just as ours. ③ All prior exact methods (except [35]) 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.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation (NSF) under award number IIS-1762268 and IIS-1956096, and conducted in part while the authors were visiting the Simons Institute for the Theory of Computing.

REFERENCES

[1] Karen Aardal, George L Nemhauser, and Robert Weismantel. 2005. *Handbooks in Operations Research and Management Science: Discrete Optimization.* Elsevier.

[6]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).
[7]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 [23].

doi:10.1016/s0927-0507(05)x1200-2

[2] Kaleb Alway, Eric Blais, and Semih Salihoglu. 2021. Box Covers and Domain Orderings for Beyond Worst-Case Join Processing. In *ICDT (LIPIcs, Vol. 186)*. 3:1–3:23. doi:10.4230/LIPICS.ICDT.2021.3

[3] Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. 2018. GProM - A Swiss Army Knife for Your Provenance Needs. *IEEE Data Eng. Bull.* 41, 1 (2018), 51–62. http://sites.computer.org/debull/A18mar/p51.pdf

[4] Shahaf Bassan, Guy Amir, and Guy Katz. 2024. Local vs. Global Interpretability: A Computational Complexity Perspective. In *ICML (PMLR, Vol. 235)*. 3133–3167. https://proceedings.mlr.press/v235/bassan24a.html

[5] Manuel Bodirsky, Zaneta Semanisinová, and Carsten Lutz. 2024. The Complexity of Resilience Problems via Valued Constraint Satisfaction Problems. In *LICS*. ACM, 14:1–14:14. doi:10.1145/3661814.3662071

[6] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2019. Scalable computation of high-order optimization queries. *Commun. ACM* 62, 2 (2019), 108–116. doi:10.1145/3299881

[7] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. 2002. On Propagation of Deletions and Annotations Through Views. In *PODS*. 150–158. doi:10.1145/543613.543633

[8] Surajit Chaudhuri. 2019. To do or not to do: extending SQL with integer linear programming?: technical perspective. *Commun. ACM* 62, 2 (2019), 107. doi:10.1145/3299879

[9] Michael B Cohen, Yin Tat Lee, and Zhao Song. 2021. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)* 68, 1 (2021), 1–39. doi:10.1145/3424305

[10] Michele Conforti, Gérard Cornuéjols, and Kristina Vušković. 2006. Balanced matrices. *Discrete Mathematics* 306, 19-20 (2006), 2411–2437. doi:10.1016/j.disc.2005.12.033

[11] Umeshwar Dayal and Philip A. Bernstein. 1982. On the Correct Translation of Update Operations on Relational Views. *ACM TODS* 7, 3 (1982), 381–416. doi:10.1145/319732.319740

[12] Akhil A. Dixit and Phokion G. Kolaitis. 2022. Consistent Answers of Aggregation Queries via SAT. In *ICDE*. IEEE, 924–937. doi:10.1109/ICDE53745.2022.00074

[13] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *PODS*. 102–113. doi:10.1145/375551.375567

[14] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2015. The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries. *PVLDB* 9, 3 (2015), 180–191. http://www.vldb.org/pvldb/vol9/p180-freire.pdf

[15] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2020. New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins. In *PODS*. 271–284. doi:10.1145/3375395.3387647

[16] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *11th Joint Meeting on Foundations of Software Engineeriing (ESEC/FSE)*. 498–510. doi:10.1145/3106237.3106277

[17] Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. 2022. HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach. In *SIGMOD*. 1598–1611. doi:10.1145/3514221.3526149

[18] Boris Glavic. 2010. *Perm: efficient provenance support for relational databases*. Ph. D. Dissertation. University of Zurich. http://cs.iit.edu/%7edbgroup/assets/pdfpubls/G10a.pdf

[19] Boris Glavic and Gustavo Alonso. 2009. The perm provenance management system in action. In *SIGMOD* (Providence, Rhode Island, USA) *(SIGMOD '09)*. 1055–1058. doi:10.1145/1559845.1559980

[20] Boris Glavic, Alexandra Meliou, and Sudeepa Roy. 2021. Trends in explanations: Understanding and debugging data-driven systems. *Foundations and Trends in Databases* 11, 3 (2021). doi:10.1561/9781680838817

[21] Martin Grötschel, László Lovász, Alexander Schrijver, Martin Grötschel, László Lovász, and Alexander Schrijver. 1993. The ellipsoid method. *Geometric Algorithms and Combinatorial Optimization* (1993), 64–101. doi:10.1007/978-3-642-78240-4_4

[22] LLC Gurobi Optimization. 2021. Mixed-Integer Programming (MIP) – A Primer on the Basics. Retrieved 2025-05-24 from https://www.gurobi.com/resource/mip-basics/

[23] LLC Gurobi Optimization. 2022. Gurobi Guidelines For Numerical Issues. Retrieved 2025-05-24 from https://www.gurobi.com/documentation/10.0/refman/guidelines_for_numerical_i.html

[24] LLC Gurobi Optimization. 2022. Gurobi Optimizer Reference Manual. Retrieved 2025-05-24 from http://www.gurobi.com

[25] Melanie Herschel, Mauricio A. Hernández, and Wang Chiew Tan. 2009. Artemis: A System for Analyzing Missing Answers. *PVLDB* 2, 2 (2009), 1550–1553. doi:10.14778/1687553.1687588

[26] Xiao Hu and Stavros Sintos. 2024. Finding Smallest Witnesses for Conjunctive Queries. In *ICDT (LIPIcs, Vol. 290)*. 24:1–24:20. doi:10.4230/LIPIcs.ICDT.2024.24

[27] Xiao Hu, Shouzhuo Sun, Shweta Patwa, Debmalya Panigrahi, and Sudeepa Roy. 2020. Aggregated Deletion Propagation for Counting Conjunctive Query Answers. *PVLDB* 14, 2 (2020), 228–240. doi:10.14778/3425879.3425892

[28] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103. doi:10.1007/978-1-4684-2001-2_9

[29] J. E. Kelley, Jr. 1960. The Cutting-Plane Method for Solving Convex Programs. *J. Soc. Indust. Appl. Math.* 8, 4 (1960), 703–712. doi:10.1137/0108053

[30] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2016. Joins via Geometric Resolutions: Worst Case and Beyond. *ACM TODS* 41, 4, Article 22 (Nov. 2016), 45 pages. doi:10.1145/2967101

[31] Benny Kimelfeld. 2012. A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies. In *PODS*. 191–202. doi:10.1145/2213556.2213584

[32] Benny Kimelfeld, Jan Vondrák, and Ryan Williams. 2012. Maximizing Conjunctive Views in Deletion Propagation. *ACM TODS* 37, 4, Article 24 (2012), 37 pages. doi:10.1145/2389241.2389243

[33] Benny Kimelfeld, Jan Vondrák, and David P. Woodruff. 2013. Multi-tuple Deletion Propagation: Approximations and Complexity. *PVLDB* 6, 13 (2013), 1558–1569. doi:10.14778/2536258.2536267

[34] Anh L. Mai, Pengyu Wang, Azza Abouzied, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. 2024. Scaling Package Queries to a Billion Tuples via Hierarchical Partitioning and Customized Optimization. *PVLDB* 17, 5 (2024), 1146–1158. doi:10.14778/3641204.3641222

[35] Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *PACMMOD* 1, 4, Article 228 (dec 2023), 27 pages. doi:10.1145/3626715

[36] Neha Makhija and Wolfgang Gatterbauer. 2024. A Unified and Practical Approach for Generalized Deletion Propagation. *arXiv:2411.17603* (2024). https://arxiv.org/abs/2411.17603

[37] Neha Makhija and Wolfgang Gatterbauer. 2025. Generalized Deletion Propogation: Code. https://github.com/northeastern-datalab/generalized-deletion-propagation

[38] Joao Marques-Silva. 2023. *Logic-Based Explainability in Machine Learning*. Springer Nature Switzerland, 24–104. doi:10.1007/978-3-031-31414-8_2

[39] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB* 4, 1 (2010), 34–45. http://www.vldb.org/pvldb/vol4/p34-meliou.pdf

[40] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. Why so? or Why no? Functional Causality for Explaining Query Answers, In 4th International Workshop on Management of Uncertain Data (MUD). *CoRR*, 3–17. http://arxiv.org/abs/0912.5340

[41] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. 2011. Reverse Data Management. *PVLDB* 4, 12 (2011), 1490–1493. http://www.vldb.org/pvldb/vol4/p1490-meliou.pdf

[42] Alexandra Meliou and Dan Suciu. 2012. Tiresias: the database oracle for how-to queries. In *SIGMOD*. 337–348. doi:10.1145/2213836.2213875

[43] Dongjing Miao, Jianzhong Li, and Zhipeng Cai. 2020. The parameterized complexity and kernelization of resilience for database queries. *Theoretical Computer Science* 840 (2020), 199–211. doi:10.1016/j.tcs.2020.08.018

[44] Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. Explaining Wrong Queries Using Small Examples. In *SIGMOD*. 503–520. doi:10.1145/3299869.3319866

[45] Stuart Mitchell, Michael OSullivan, and Iain Dunning. 2011. PuLP: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand* 65 (2011). https://optimization-online.org/?p=11731

[46] Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. 2014. Beyond worst-case analysis for joins with minesweeper. In *PODS*. 234–245. doi:10.1145/2594538.2594547

[47] Chandrasekhar Ramakrishnan. 2020. 2019-01 US Flights. doi:10.7910/DVN/WTZS4K

[48] Tim Roughgarden. 2020. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press. doi:10.1017/9781108637435

[49] Sudeepa Roy and Dan Suciu. 2014. A Formal Approach to Finding Explanations for Database Queries. In *SIGMOD*. 1579–1590. doi:10.1145/2588555.2588578

[50] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional fairness: Causal database repair for algorithmic fairness. In *SIGMOD*. 793–810. doi:10.1145/3299869.3319901

[51] Alexander Schrijver. 1998. *Theory of linear and integer programming*. John Wiley & Sons. doi:10.1137/1030065

[52] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and Probability Management in PostgreSQL. *PVLDB* 11, 12 (2018), 2034–2037. doi:10.14778/3229863.3236253

[53] Laurynas Siksnys and Torben Bach Pedersen. 2016. SolveDB: Integrating Optimization Problem Solvers Into SQL Databases. In *SSDBM*. ACM, 14:1–14:12. doi:10.1145/2949689.2949693

[54] Immanuel Trummer and Christoph Koch. 2017. Solving the Join Ordering Problem via Mixed Integer Linear Programming. In *SIGMOD*. 1025–1040. doi:10.1145/3035918.3064039

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*, Vol. 30. https://arxiv.org/abs/1706.03762

[56] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. 2017. QFix: Diagnosing errors through query histories. In *SIGMOD*. 1369–1384. doi:10.1145/3035918.3035925

[57] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6, 8 (2013), 553–564. doi:10.14778/2536354.2536356