A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations

NEHA MAKHIJA, Northeastern University, USA
WOLFGANG GATTERBAUER, Northeastern University , USA

What is a minimal set of tuples to delete from a database in order to eliminate all query answers? This problem is called "the resilience of a query" and is one of the key algorithmic problems underlying various forms of reverse data management, such as view maintenance, deletion propagation and causal responsibility. A long-open question is determining the conjunctive queries (CQs) for which resilience can be solved in PTIME.

We shed new light on this problem by proposing a unified Integer Linear Programming (ILP) formulation. It is unified in that it can solve both previously studied restrictions (e.g., self-join-free CQs under set semantics that allow a PTIME solution) and new cases (all CQs under set or bag semantics). It is also unified in that all queries and all database instances are treated *with the same approach*,yet the algorithm is *guaranteed to terminate in* PTIME for all known PTIME cases. In particular, we prove that for all known easy cases, the optimal solution to our ILP is identical to a simpler Linear Programming (LP) relaxation, which implies that standard ILP solvers return the optimal solution to the original ILP in PTIME.

Our approach allows us to explore new variants and obtain new complexity results. 1) It works under bag semantics, for which we give the first dichotomy results in the problem space. 2) We extend our approach to the related problem of causal responsibility and give a more fine-grained analysis of its complexity. 3) We recover easy instances for generally hard queries, including instances with read-once provenance and instances that become easy because of Functional Dependencies *in the data*. 4) We solve an open conjecture about a unified hardness criterion from PODS 2020 and prove the hardness of several queries of previously unknown complexity. 5) Experiments confirm that our findings accurately predict the asymptotic running times, and that our universal ILP is at times even quicker than a previously proposed dedicated flow algorithm.

CCS Concepts: • **Theory of computation → Database theory**; • **Information systems → Data management systems**.

Additional Key Words and Phrases: Resilience, Causal Responsibility, Reverse Data Management, Query Explanation, Dichotomy, Linear Programming Relaxation

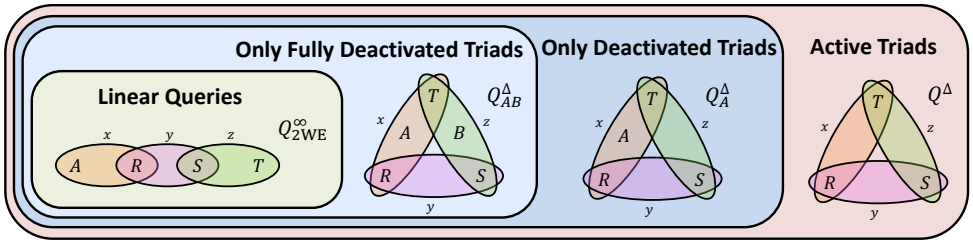
ACM Reference Format:
Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *Proc. ACM Manag. Data* 1, 4 (SIGMOD), Article 228 (December 2023), 27 pages. https://doi.org/10.1145/3626715

1 INTRODUCTION

What is a minimum set of changes to a database in order to produce a certain change in the output of a query? This question underlies many problems of practical relevance, including explanations [36, 62], algorithmic fairness [32, 63], and diagnostics [71, 72]. Arguably, the simplest formulation of such reverse data management [56] questions is "resilience": *What is the minimal number of*

Authors' addresses: Neha Makhija Northeastern University, USA, makhija.n@northeastern.edu; Wolfgang Gatterbauer Northeastern University, USA, w.gatterbauer@northeastern.edu.

2836-6573/2023/12-ART228
https://doi.org/10.1145/3626715



RES (Sets)	PTIME (Thm. 8.6)	PTIME (Thm. 8.7)		NPC (Thm. 7.5)
RSP (Sets)	PTIME (Thm. 8.11)	PTIME (Thm. 8.12)	PTIME: dominating (A) (Thm. 8.13)	NPC (Thm. 8.15)
			NPC: non-dominating (R,S,T) (Thm. 8.14)	
RES (Bags)	PTIME (Thm. 8.6)	NPC (Thm. 8.8)		
RSP (Bags)	PTIME (Thm. 8.11)	NPC (Thm. 8.15)		

Table 1. Overview of complexity results for self-join-free conjunctive queries (SJ-free CQs) that follow from our unified framework in this paper. Results highlighted with yellow background are new. RES stands for resilience and RSP for causal responsibility. Not shown are additional results we give for queries with self-joins.

tuples to delete from a database in order to eliminate all query answers?[1] An early variant of the problem was formulated 40 years ago in the context of view-maintenance [23] and has been studied over the years in various forms. The problem has received considerable attention in the context of provenance and deletion propagation [8–10]. *Deletion propagation* seeks a set of tuples that can be deleted from the database to delete a particular tuple from the view. A variation we study in this paper is *causal responsibility*, which involves finding a minimum subset of tuples to remove to make a given input tuple "counterfactual." [53, 55].

The problems of resilience and causal responsibility have practical applications in helping users better understand transformations of their data and to explain surprising query results. They are both based on the idea of *minimal interventions*, which aims to find the simplest possible satisfying explanations. Intuitively, the resilience of a query provides a minimal set of tuples (i.e. a minimal explanation) without which a Boolean query would not return true. In addition, it is known that a solution to resilience immediately also provides an answer to the deletion propagation with source-side effects problem [30], which seeks a minimal intervention, or a minimal set of input tuples to be deleted to perform deletion propagation (delete a tuple from the view).

The problem of causal responsibility uses the same idea of minimal interventions to provide explanations at a more fine-grained tuple level. For any desired input tuple, users can calculate the "responsibility" of that tuple based on formal, mathematical notions of causality adapted to databases [53]. Then one can derive explanations by ranking input tuples using their responsibilities: tuples with a high degree of responsibility are better explanations for a particular query result. This makes causal responsibility an invaluable tool for query explanations and debugging [36].

Our goal is to understand the complexity of solving resilience and causal responsibility. The first result by Buneman et al. [8] showed that the problem is NP-complete (NPC) for conjunctive queries (CQs) with projections. Later work under the topic of causal responsibility [56] and the simpler notion of resilience [30] showed that a large fraction of self-join-free CQs (triad-free queries) can be solved in PTIME, solving the complexity of self-join-free (SJ-free) queries. However, few results

[1]While the formal definition (which we give later) applies only to Boolean queries, the above more intuitive formulation can be easily transformed into the Boolean variant.

are known for the cases of CQs *with self-joins* [31]. This state is similar to other database problems where establishing complexity results for self-joins is often considerably more involved than for self-join-free queries (e.g., compare the results on probabilistic databases for either self-join-free queries [19] with those for self-joins [20]). Moreover, all these problems have been studied only for set semantics, whereas relational databases actually use bag semantics i.e., they allow duplicate tuples [13]. Like self-joins, bags usually make problems harder to analyze [2, 46, 74], and few complexity results for bag semantics exist.

This paper gives the first dichotomy results under bag semantics for problems in reverse data management (Table 1). We also give a simple-to-verify sufficient hardness criterion for all conjunctive queries (including queries with self-joins and under set or bag semantics). Based on this criterion, we build an automatic hardness certificate finder, that, a given query Q and a fixed domain size d, finds a hardness certificate for Q of domain size $\leq d$, whenever such a certificate exists. We use this construction to find hardness certificates for 5 previously open queries with self-joins.

Our attack on the problem is unconventional: Rather than deriving a dedicated PTIME algorithm for certain queries (and proving hardness for the rest), we instead propose a unified Integer Linear Program (ILP) formulation for all problem variants (self-joins or not, sets or bags, Functional Dependencies or not). We then show that, for all PTIME queries, *the Linear Program (LP) relaxation of our ILP has the same optimal value*, thereby proving that existing ILP solvers are *guaranteed* to solve problems for those queries in PTIME.

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

1) *Unified ILP framework*: We propose an ILP formulation for the problems of resilience and causal responsibility that can not only encode all previously studied variants of the problem, but can also encode *all* formulations of the problem, including self-joins and bag semantics (Sections 4 and 5). This unified encoding allows us to model and solve problems for which currently no algorithm (whether easy or hard) has been proposed. It also allows us to study LP relaxations (Section 6) of our formulation, which form the basis of several of our theoretical results.

2) *Unified hardness criterion*: We prove a variant of an open conjecture from PODS 2020 [31] by defining a structural certificate called Independent Join Path (IJP) and proving that it implies hardness (Section 7). Most interestingly, we give a Disjunctive Logic Program (DLP) formulation that can computationally derive such certificates. We use this certificate to both (*i*) prove hardness for all hard queries in our dichotomies, and (*ii*) obtain computationally derived hardness certificates for 5 previously open queries with self-joins. While solving such programs is general in Σ_p^2 (i.e. on the 2nd level of the polynomial hierarchy) a modern ASP solver *clingo* [34] allowed us to obtain all the new, easy-to-verify proofs in under two hours, including some obtained in seconds.

3) *First results for resilience and responsibility under bag semantics*: We give full dichotomy results for both resilience and causal responsibility under bag semantics for the special case of SJ-free CQs (Section 8). We show that under bag semantics, the PTIME cases for resilience and responsibility are exactly the same (Table 1).

4) *Recovering* PTIME *cases*: We prove that for all prior known and newly found PTIME cases of SJ-free queries (under both set and bag semantics), our ILP is solved in guaranteed PTIME by standard solvers (Section 8). This means that our formulation is unified not only in being able to model all cases but also in that it is guaranteed to *recover all known* PTIME *cases by terminating in* PTIME. In addition, we uncover more tractable cases for causal responsibility, due to obtaining more fine-grained complexity results (Section 8.3). Our new way of modeling the problem opens up a new route for solving various open problems in reverse data management: by proposing a universal algorithm for solving all variants, future development does not depend on finding new

dedicated PTIME algorithms, but rather on proving that the universal method terminates in PTIME (in similar spirit to proofs in this paper).

5) *Novel approximations*: We show 3 different approximation algorithms for both resilience and causal responsibility. The first approach based on LP-rounding provides a guaranteed m-factor approximation (where m is the number of atoms in the query) for all queries (*including self-joins and bag semantics*). The other two are new flow-based approximation techniques designed for hard queries without self-joins (Section 9).

6) *Experimental Study*: We compare all approaches proposed in this paper on different problem instances: easy or hard, for set or bag semantics, queries with self-joins, and Functional Dependencies. Our results establish the accuracy of our asymptotic predictions, uncover novel practical trade-offs, and show that our approach and approximations create an end-to-end solution (Section 10).

We make all code and experiments available online [51]. We provide a *proof intuition* for each theorem in the main text, and full proofs are available in the full paper [52]. The full paper also contains additional examples and details, and discusses some additional results as well. Our approach can solve resilience and causal responsibility for otherwise hard queries in PTIME for database instances such as *read-once* instances, or instances that obey certain *Functional Dependencies* (not necessarily known at the query level). We show these *instance-based tractability results* in the full paper [52].

2 RELATED WORK

Resilience and Causal Responsibility. Foundational work by Halpern, Pearl, et al. [15, 40, 41] defined the concept of *causal responsibility* based *minimal interventions* in the input. Meliou et al. [55] adapted this concept to define causal responsibility for database queries and proposed a flow algorithm to solve the tractable cases. Freire et al. [30] defined a simpler notion of *resilience* and gave a dichotomy of the complexity for both resilience and responsibility for SJ-free queries under set semantics. While the tractability frontier for self-join case remains open to this day, Freire et al. [31] gave partial complexity results for resilience for queries with self-joins and conjectured that the notion of Independent Join Paths (IJPs) could imply hardness for resilience. We prove one direction of this conjecture (with a slight fix of the original statement). After acceptance of this paper, an interesting preprint was published on arXiv [5] that formulates resilience as a Valued Constraint Satisfaction problem (VCSP) and applies results from an earlier VCSP dichotomy [47]. Interestingly, it also ends with a dichotomy conjecture (not proof) for resilience, notably for bag semantics but not set semantics. We discuss these connections in more detail in the full paper [52].

Other Problems in View Maintenance. There are several variants to resilience such as destroying a pre-specified fraction of witnesses from the database instead of all witnesses [43]. They all are instances of reverse data management [56] and deletion propagation [9, 23]. *Deletion propagation* seeks to delete a set of input tuples in order to delete a particular tuple from the view. Intuitively, this deletion should be achieved with minimal side effects, where side effects are defined with one of two objectives: (a) deletion propagation with *source side effects* seeks a minimum set of input tuples in order to delete a given output tuple; whereas (b) deletion propagation with *view side effects* seeks a set of input tuples that results in a minimum number of output tuple deletions in the view, other than the tuple of interest [9]. The dichotomies for self-join queries remain open for the problems in this space. We believe that our core ideas can be applied to many such problems.

Explanations and fairness. Data management research has recognized the need to derive *explanations* for query results and surprising observations [36]. Existing work on explanations use many approaches [50], including modifying the input (i.e. performing *interventions*) [42, 44, 54, 55, 62, 73], which is our focus as well. Recent approaches show that explanations benefit a variety of applications, such as ensuring or testing fairness [32, 60, 63] or finding bias [75]. We believe our

unified framework of solving both easy and hard cases with one algorithm can also be useful for these applications.

Bag semantics. Real-world databases consist of bags instead of sets. This gap between database theory and database practice has been pointed out years ago [13]. However, studying properties of CQs under bag semantics is often considerably harder. For example, the connection between local and global consistency has only been recently solved for bags [2, 74], and the fundamental problems of query containment of CQs under bag semantics remain open despite recent progress [46, 48]. Our paper gives the first dichotomy result for reverse data management problems under bag semantics.

Linear Optimization and Data Management. Ideas from the two fields have been connected in the past, both to solve data management problems efficiently [7, 57], and to use the factorized nature of data to solve linear optimization problems more efficiently [11]. The Tiresias system [57] implements how-to queries by translating them to MILPs in order to solve them efficiently. Package queries [7] allow users to define constraints over multiple tuples with extensions of SQL, and also leverage ILP solvers in the background. Recent work by Capelli at al [11] provides an approach to solve a specific class of linear programs (LP(CQ)), whose variables correspond to answers of a CQ. They show that such LPs have PTIME query complexity for CQs with bounded fractional hypertreewidth, by leveraging the factorized structure of the data. Our work similarly leverages the structure of data, but focuses on *data* complexity of *Integer* Linear Programs to investigate the tractability of reverse data management problems and solve them efficiently when possible.

3 PRELIMINARIES

3.1 Formal Problem Setup

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 $\mathbf{x} = (x_1, \ldots, x_\ell)$ are called existential variables, $\mathbf{y}$ are called the head or free variables, and each atom g_i represents a relation $g_i = R_{j_i}(\mathbf{x}_i)$ where $\mathbf{x}_i \subseteq \mathbf{x} \cup \mathbf{y}$.[2] $\mathtt{var}(X)$ denotes the variables in a given relation/atom. Notice that a query has at least one output tuple iff the Boolean variant of the query (obtained by making all the free variables existential) is true. Unless otherwise stated, a query in this paper denotes a Boolean CQ, i.e. $\mathbf{y} = \emptyset$. We write Q to denote that that query $D \models Q$ to denote that query Q evaluates to `true` over database instance D, and $D \not\models Q$ to denote it evaluates to `false`.

Queries are interpreted as hypergraphs with edges formed by atoms and nodes by variables. Two hyperedges are connected if they share at least one node. We use concepts like paths and reachable nodes on the hypergraph of a query in the usual sense [6]. A query Q is minimal if for every other equivalent conjunctive query Q' has at least as many atoms as Q [31]. WLOG we discuss only connected queries in the rest of the paper.[3] 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.

We write D for the database, i.e. the set of tuples in the relations. When we refer to bag semantics, we allow D to be a multiset of tuples in the relations. We write $[\mathbf{w}/\mathbf{x}]$ as a valuation (or substitution) of query variables $\mathbf{x}$ by $\mathbf{w}$. A *witness* $\mathbf{w}$ is a valuation of $\mathbf{x}$ that is permitted by D and that makes Q `true` (i.e. $D \models Q[\mathbf{w}/\mathbf{x}]$).[4] The set of witnesses is then

$$\mathtt{witnesses}(Q, D) = \left\{\mathbf{w} \;\middle|\; D \models Q[\mathbf{w}/\mathbf{x}]\right\}.$$

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

[3]Results for disconnected queries follow by treating each of the components *independently*.

[4]Note that our notion of witness slightly differs from the one used in provenance literature where a "witness" refers to a subset of the input database records that is sufficient to ensure that a given output tuple appears in the result of a query [14].

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

Resilience, Responsibility, and related terminology.

Definition 3.1 (Resilience [30]). Given a query Q and database D, we say that $k \in \mathsf{RES}(Q, D)$ if and only if $D \models Q$ and there exists some contingency set $\Gamma \subseteq D$ with $|\Gamma| \leq k$ such that $D - \Gamma \not\models Q$.

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

Definition 3.2 (Responsibility [55]). Given query Q and an input tuple t, we say that $k \in \mathsf{RSP}(Q, D, t)$ if and only if $D \models Q$ and there is a contingency set $\Gamma \subseteq D$ with $|\Gamma| \leq k$ such that $D - \Gamma \models Q$ but $D - (\Gamma \cup \{t\}) \not\models Q$.

In other words, causal responsibility aims to determine whether *a particular input tuple* t (the *responsibility tuple*) can be made "counterfactual" by deleting a set of other input tuples Γ of size k or less. Counterfactual here means that the query is true with that input tuple present, but false if it is also deleted. In contrast to resilience, the problem of responsibility is defined for *a particular tuple* t in D, and instead of finding a Γ that will leave no witnesses for $D - \Gamma \models q$, we want to preserve only witnesses that involve t, so that there is no witness left for $D - (\Gamma \cup \{t\}) \models Q$. Responsibility measures the *degree* of causal contribution of a particular tuple t to the output of a query as a function of the size of a minimum contingency set (the *responsibility set*). We are again interested in the optimization version of this problem: $\mathsf{RSP}^*(Q, D, t)$.[5]

Definition 3.3 (Exogenous / Endogenous tuples). A tuple is exogenous if it must not or need not participate in a contingency set, and endogenous otherwise.

Prior work [55] has defined relations (or atoms) to be exogenous or endogenous, i.e. when all tuples in any relation (or relation of the atom) are either exogenous or endogenous. We use but also generalize this notation to allow *individual tuples* to be declared exogenous (but keep them endogenous by default). We will see later in Section 7 that this generalization allows us to formulate resilience and responsibility with a simple universal hardness criterion.[6] The set of exogenous tuples $E \subset D$ can be provided as an additional input parameter as in $\mathsf{RES}(Q, D, E)$ and $\mathsf{RSP}(Q, D, t, E)$. We assume a database instance has no exogenous tuples unless explicitly specified, and we omit the parameter for simplicity.

Our focus. We are interested in the *data complexity* [69] of $\mathsf{RES}(Q, D)$ and $\mathsf{RSP}(Q, D, t)$, i.e. the complexity of the problem as D increases but Q remains fixed. We refer to $\mathsf{RES}(Q)$ and $\mathsf{RSP}(Q)$ to discuss the complexity of the problems of query Q over an arbitrary data instance (and arbitrary responsibility tuple).

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

[6]In more detail, we will formulate hardness of responsibility via an Independent Join Path which is only possible because one specified tuple is exogenous, e.g. Theorem 8.14.

3.2 Tools and Techniques

We use *Integer Linear Programs* and their *relaxations* to model and solve resilience and causal responsibility. *Disjunctive Logic Programs*, which can solve problems higher in the polynomial hierarchy, are used to find certificates for hard cases.

Linear Programs (LP). Linear Programs are standard optimization problems [1, 64] 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. If the variables are constrained to be integers, the resulting program is called an Integer Linear Program (ILP), while a program with some integral variables is referred to as a Mixed Integer Linear Program (MILP). The *LP relaxation* of an ILP program is obtained by removing the integrality constraint for all variables.

Complexity of solving ILPs. ILPs are NPC and part of Karp's 21 problems [45], while LPs can be solved in PTIME with Interior Point methods [16, 37]. The complexity of MILPs is exponential in the number of integer variables. However, there are conditions under which ILPs become tractable. In particular, if there is an optimal integral assignment to the LP relaxation, then the original ILP can be solved in PTIME as well. A lot of work studies conditions when this property holds [18, 29, 49, 64]. A famous example is the max-flow min-cut problem which can be solved with LPs despite integrality constraints. The *max-flow Integrality Theorem* [29] states that for every flow graph with all capacities as integer values, there is an optimal maximum flow such that all flow values are integral. Therefore, in order to find an integral max-flow for such a graph, one need not solve an ILP but rather an LP relaxation suffices to get the same optimal value. There are many other 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* [64] then the LP always has the same optima. Similarly, if the constraint matrix is *Balanced* [17], several classes of ILPs are PTIME.

We use the results of Balanced Matrices to show that the resilience and responsibility of any read-once data instances can be found in PTIME (as an additional result in the full paper [52]). For other PTIME cases, we have ILP constraint matrices that do not fit into any previous tractability characterization. Despite this, we are able to use these results indirectly (via an intermediate flow representation) to show that the *LP relaxation* has the same objective as the original ILP and thus the ILP can be solved in PTIME.

Linear Optimization Solvers. A key advantage of modeling problems as ILPs is practical. There are many highly-optimized ILP solvers, both commercial [39] and free [58] which can obtain exact results fast in practice. ILP formulations are standardized, and thus programs can easily be swapped between solvers. Any advances made over time by these solvers (improvements in the presolve phase, heuristics, and even novel techniques) can automatically make implementations of these problems better over time.

For our experimental evaluation we use Gurobi.[7] Gurobi uses an LP based branch-and-bound method to solve ILPs and MILPs [38]. 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 that 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 formulation or letting the solver know anything about the theoretical complexity.

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

Disjunctive Logic Programs (DLPs). Disjunctive Logic Programs are Logic Programs that allow disjunction in the head of a rule [21, 61]. DLPs have been shown to be Σ_p^2-complete [25, 26], and are more expressive than Logic Programs without disjunctions that are NPC. The key to higher expressivity is the non-obvious *saturation* technique that can check if *all* possible assignments satisfy a given property [24]. Logic Programs have been used for database repairs [35] and to determine the responsibility of tuples in a database [4]. We go beyond this to build a DLP that searches for a certificate that proves that solving the resilience/responsibility problem is NPC for a given query. We represent our DLP as an Answer Set Program (ASP) [27] and use *clingo* [59] to solve it.

4 ILP FOR RESILIENCE

We construct an Integer Linear Program $\texttt{ILP}[\texttt{RES}^*(Q, D)]$ from a CQ Q and a database D which returns the solution to the optimization problem $\texttt{RES}^*(Q, D)$ for any Boolean CQ (even with self-joins) under either set or bag semantics.[8] This section focuses on the correctness of the ILP. Section 6 later investigates how easy cases can be solved in PTIME, despite the problem being NPC in general.

To construct the ILP, we need to specify the decision variables, constraints and objective. As input to the ILP, we first run the query on the database instance to compute all the witnesses. This can be achieved with a modified witness query, a query that returns keys for each table, and thus each returned row is a *set* of tuples from each of the tables.[9]

1. Decision Variables. We create an indicator variable $X[t] \in \{0, 1\}$ for each tuple t in the database instance D. A value of 1 for $X[t]$ means that t is included in a contingency set, and 0 otherwise. For bag semantics, Lemma 4.1 shows that it suffices to define a single variable for a set of duplicate tuples (intuitively, an optimal solution chooses either all or none).

2. Constraints. Each witness must be destroyed in order to make the output false for a Boolean query (or equivalently, to eliminate all output tuples from a non-Boolean query). A witness is destroyed, when at least one of its tuples is removed from the input. Thus, for each witness, we add one constraint enforcing that at least one of its tuples must be removed. For example, for a witness $\mathbf{w} = \{r_i, r_j, r_k\}$ we add the constraint that $X[r_i] + X[r_j] + X[r_k] \geq 1$.[10]

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

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

x	y	z
1	1	1
2	3	4

$\mathbf{w}_1 = \{r_{11}\}$
$\mathbf{w}_2 = \{r_{23}, r_{34}\}$

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

$$X[r_{11}] \geq 1 \qquad X[r_{23}] + X[r_{34}] \geq 1$$

[8] Notice that we also write ILP[problem] for the optimal value of the program
[9] Duplicate tuples have the same key.
[10] Notice that for SJ-free queries, the number of tuples in each constraint is exactly equal to the number of atoms in the query. But for queries with self-joins, the number of tuples in each constraint is not fixed (is lower when a tuple joins with itself).

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

Example 2 (RES ILP: Bag Semantics). *Assume the same problem as Example 1, but we allow duplicates in the input. Concretely assume* r_{23} *appears twice:* $R' = \{(1,1) : 1, (2,3) : 2, (3,4) : 1\}$. *The variables and constraints stay the same, only the objective function changes now to*

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

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

Before we prove the correctness of $\mathtt{ILP}[\mathtt{RES}^*(Q, D)]$ in Theorem 4.2, we will justify our decision to use a single decision variable per unique tuple with the help of Lemma 4.1.

Lemma 4.1. *There exists a resilience set where for each unique tuple in D, either all occurrences of the tuple are in the resilience set, or none are.*

Proof Intuition (Lemma 4.1). We show that if a tuple t is in a contingency set Γ but a duplicate tuple t' is not, then removing t leads to a now smaller contingency set Γ'. This is due to the fact that since t and t' they are identical, they form witnesses with the same set of tuples. If t' is not in the contingency set, there must be another tuple in the contingency set for every witness of t'. This implies that all the witnesses t participates in are already covered, and t need not be in the contingency set. □

Theorem 4.2. *[*$\mathtt{RES}$ *ILP correctness]* $\mathtt{ILP}[\mathtt{RES}^*(Q, D)] = \mathtt{RES}^*(Q, D)$ *for any CQ* Q *and database* D *under set or bag semantics.*

Proof Intuition. We prove validity by showing that any satisfying solution would necessarily destroy all witnesses i.e. make the query false. Thus, if we consider any invalid solution i.e. one in which not all witnesses have been destroyed, we can see that there is an unsatisfied constraint in $\mathtt{ILP}[\mathtt{RES}^*]$. Hence, all $\mathtt{ILP}[\mathtt{RES}^*]$ are valid. Next we prove optimality by showing that any valid resilience set would be a valid solution for the ILP. This is equivalent to showing that any valid contingency set is a solution to $\mathtt{ILP}[\mathtt{RES}^*]$, since they must satisfy all constraints. Since $\mathtt{ILP}[\mathtt{RES}^*]$ always gives a valid, optimal solution, it is correct. □

We would like to stress to the reader that changing from sets to bags affects only the objective function, not the constraint matrix. Later in Section 8, we will prove that for queries such as $Q_A^{\triangle}$, the problem of finding resilience becomes NPC under bag semantics, while it is solvable in PTIME under set semantics. This observation is significant because most literature on tractable cases in ILP focuses exclusively on analyzing the constraint matrix. For example, if an ILP has a constraint matrix that is Totally Unimodular it is PTIME no matter the objective function [65, Section 19].

5 ILP FOR RESPONSIBILITY

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

1. Decision Variables. $\mathtt{ILP}[\mathtt{RSP}^*(Q, D, t)]$ has two types of decision variables:

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

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

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

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

Theorem 5.1. $\texttt{ILP}[\texttt{RSP}^*(Q, D, t)] = \texttt{RSP}^*(Q, D, t)$ *of a tuple t in database instance D under CQ Q under set or bag semantics.*

Proof Intuition (Theorem 5.1). Like Theorem 4.2, we prove validity and then optimality. We show that for any responsibility set we can assign values to the ILP variables such that they can form a satisfying solution (this follows from that fact that the responsibility set must preserve at least one witness containing t). Thus the correct solution is captured by $\texttt{ILP}[\texttt{RSP}^*]$, while any invalid contingency set violates at least one constraint. □

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

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

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

$$X[r_{11}] + X[s_{12}] \geq 1$$
$$X[r_{11}] + X[s_{13}] \geq 1$$

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

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

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

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

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

6 LP RELAXATIONS OF ILP[RES*] & ILP[RSP*]

The previous sections introduced unified ILPs to solve for RES and RSP. However, ILPs are NPC in general, and we would like stronger runtime guarantees for cases where RES and RSP can be solved in PTIME. We do this with the introduction of LP relaxations, which generally act as lower bounds for minimization problems. However, in Section 8 we prove that these relaxations LP[RES*] and MILP[RSP*] are actually always equal to the corresponding ILPs for all easy SJ-free queries. Thus, whether easy or hard, exact or approximate, problems can be solved within the same framework, with the same solver, with minimal modification, and with the best-achievable time guarantees.

6.1 LP Relaxation for RES

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

6.2 MILP Relaxation for RSP

For responsibility, the relaxation is more intricate. It turns out that an LP relaxation is not optimal for PTIME cases (Example 4). We introduce a Mixed Integer Linear Program MILP[RSP*], where tuple indicator variables are relaxed and take values in $[0, 1]$ whereas witness indicator variables are restricted to values $\{0, 1\}$. Typically, MILPs are exponential in the number of integer variables i.e. if there are n integer binary variables, a solver explores 2^n possible branches of assignments. However, despite having an integer variable for every witness that contains t (thus up to linear in the size of the database), we show that MILP[RSP*] is in PTIME.

Lemma 6.1. *For any CQ Q and tuple t, MILP[RSP*(Q, D, t)] can be solved in PTIME in the size of database D.*

Proof Intuition. We show that is possible to solve MILP[RSP*] in PTIME by solving a linear number of linear programs. Instead of looking at all possible 0-1 assigments to witness indicator variables - we simply need to select 1 witness indicator variable that is to be set to 0. All witness indicator variables are combined into one counterfactual constraint. This constraint is always satisfied when any one of the variable takes value 0, irrespective of other variable values. Thus, we only need to explore the assignments where exactly 1 variable takes on value 0, thus a linear number of assignments in the size of the database. □

In addition to the above theoretical proof of the PTIME solvability of MILP[RSP*], we see experimentally in Section 10 that a typical ILP solver indeed scales in polynomial time to solve MILP[RSP*].

Example 4. *Consider again the problem in Example 3. The solution of ILP[RSP*] was 2 at $X[s_{12}] = 1, X[s_{13}] = 1\ X[r_{11}] = 0$ and $X[\mathbf{w}_1] = 0$. What happens if we relax the integrality constraints and allow $0 \le X[v] \le 1$ for all variables? We can get a smaller satisfying solution 1.5 at the point $X[s_{12}] = 0.5, X[s_{13}] = 0.5\ X[r_{11}] = 0.5$ and $X[\mathbf{w}_1] = 0.5$. This value is LP[RSP*] and is not guaranteed to be equal to ILP[RSP*]. If we instead create MILP[RSP*] and apply* integrality constraints only for the witness indicator variables*, then $X[\mathbf{w}_1]$ is forced to be in $\{0, 1\}$ while all other variables*

can be fractional. We see that the LP[RSP*] *solution is no longer permitted, and solving* MILP[RSP*] *results in the true* RSP *value of 2. We show in Section 8.3 that* MILP[RSP*] = ILP[RSP*] *for all easy cases like chain queries such as* Q_2^∞ *(Table 1).*

We conjecture that these relaxations are all we need to solve the problems of resilience and causal responsibility efficiently, whenever an efficient solution is possible. In Section 8, we prove that Conjectures 6.2 and 6.3 are true for all self-join free queries.

Conjecture 6.2 (RES is easy ⇒ LP=ILP). *If* RES(Q) *can be solved in* PTIME *under set/bag semantics, then* LP[RES*(Q, D)] = ILP[RES*(Q, D)] *for any database* D *under the same semantics.*

Conjecture 6.3 (RSP is easy ⇒ MILP=ILP). *If* RSP(Q) *can be solved in* PTIME *under set/bag semantics, then* MILP[RSP*(Q, D)] = ILP[RSP*(Q, D)] *for any database* D *under the same semantics.*

7 FINDING HARDNESS CERTIFICATES

Freire et al. [31] conjectured that the ability to construct a particular certificate called "Independent Join Path" is a sufficient criterion to prove hardness of resilience for a query. We prove here that not the original, but a slight variation of that idea is indeed correct.

We also prove that this construction is a *necessary criterion for hardness of self-join free queries* and conjecture it to be also necessary for any query. In addition, we also give a Disjunctive Logic Program (DLP[RESIJP]) that can create hardness certificates and use it to prove hardness for 5 previously open queries with self-joins.

7.1 Independent Join Paths (IJPs)

We slowly build up intuition to define IJPs (Definitions 7.1 and 7.3). Recall the concept of a *canonical database* for a minimized CQ resulting from replacing each variable with a *different constant* [12, 68]. For example $A(1), R(1,2), S(2,3), T(3,1)$ is a canonical database for the triangle query $Q_A^\triangle :- A(x), R(x,y), S(y,z), T(z,x)$. Intuitively, one can think of a *witness* as more general than a canonical database in that several variables may map to the *same constant*. A join path is then a set of witnesses that share enough constants to be connected (this sharing of constants can be best formalized as a *partition of the constants* among a fixed number of witnesses). In addition, join paths are defined with two "*isomorphic*" sets of tuples, the start $\mathcal{S}$ and terminal $\mathcal{T}$ (both together called the "*endpoints*"). We call two sets of tuples *isomorphic* iff there a bijective mapping between the constants of the sets that preserves the sets of shared constants across table attributes. For example, $\mathcal{S}_1 = \{R(1,2), A(2), R(2,2)\}$ is isomorphic to $\mathcal{S}_2 = \{R(3,4), A(4), R(4,4)\}$ but not to $\mathcal{S}_2 = \{R(3,4), A(4), R(4,5)\}$.

Definition 7.1 (Join Path (JP)). A database D (under set or bag semantics) forms a Join Path from a set of tuples $\mathcal{S}$ (start) to a set of tuples $\mathcal{T}$ (terminal), for query Q if

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

Example 5 (Join paths). *Consider again the query* $Q_A^\triangle$. *The following database of 9 tuples (Fig. 1a)* $D = \{A(1), A(4), R(1,2), R(4,2), R(4,5), S(2,3), S(5,3), T(3,1), T(3,4)\}$ *where* $A(1)$ *and* $A(4)$ *are exogenous, forms a join path from* $\{S = \{R(1,2)\}$ *to* $\mathcal{T} = \{R(4,5)\}$. *It has 3 witnesses* $\mathbf{w}_1 = \{A(1), R(1,2), S(2,3), T(3,1)\}$,

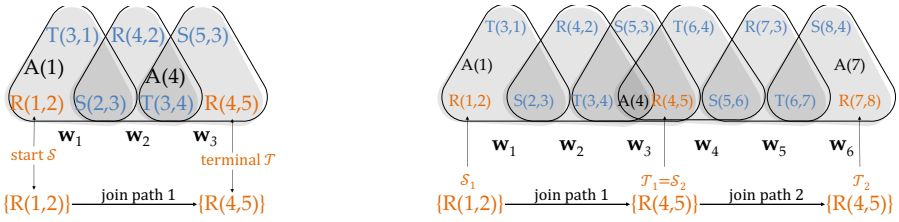


Fig. 1. (a) IJP for triangle query $Q_A^{\triangle}$. (b) IJPs are composed by sharing their endpoints (start or terminal tuples).

$\mathbf{w}_2 = \{A(4),$
$R(4,2), S(2,3), T(3,4)\}$, *and* $\mathbf{w}_3 = \{A(4), R(4,5), S(5,3), T(3,4)\}$. *This join path can also be interpreted as a partition* $\{\{x^1\}, \{x^2, x^3\}, \{y^1, y^2\}, \{y^3\}, \{z^1, z^2, z^3\}\}$ *on the canonical databases for three witnesses* $\mathbf{w}_i = \{A(x^i), R(x^i, y^i), S(y^i, z^i), T(z^i, x^i)\}$, $i = 1, 2, 3$, *expressing the shared constants in each subset. Then above database instance results from the following valuation* v *of the quotient set* $\{[x^1], [x^2], [y^1], [y^3], [z^1]\}$ *to constants:* $v : (x^1, y^1, z^1, x^2, y^3) \rightarrow (1, 2, 3, 4, 5)$. *Notice that* $\mathcal{S}$ *and* $\mathcal{T}$ *form a valid endpoint pair because (i)* $\mathcal{S}$ *and* $\mathcal{T}$ *are isomorphic with the mapping* $f = \{1 : 3, 2 : 4\}$ *and (ii) there is no endogenous tuple with constants only from* $\{1, 2, 3, 4\}$. $A(1)$ *and* $A(4)$ *violate the subset requirement, however they are exogenous, so the definition is fulfilled.*

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

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

Proposition 7.2 (Triangle composition). *Assume a join path (JP) with endpoints* $\mathcal{S}$ *and* $\mathcal{T}$. *If 3 isomorphic JPs composed in a triangle with directions as shown in Fig. 2 are non-leaking, then any composition of JPs is non-leaking.*

Proof Intuition (Proposition 7.2). Since JPs can be asymmetric, the composability due to sharing the $\mathcal{S}$ tuples in two isomorphic JPs differs from sharing $\mathcal{S}$ and $\mathcal{T}$. We show that the three JP interactions in Fig. 2 act as sufficient base cases to model all types of interactions. We show via induction that sharing the same end tuples across multiple JPs cannot leak if it does not leak in the base case. □

Definition 7.3 (Independent Join Path). A Join Path D forms an *Independent Join Path (IJP)* if it fulfills two additional conditions:

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

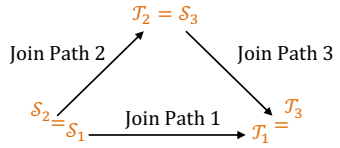


Fig. 2. 3 JPs composed in a triangle with shown edge directions.

Our definition of Independent Join Paths differs from earlier work [31], in that it is a completely *semantic* definition that is based on all the properties that must be captured by an Independent Join Path that does not enforce any structural criteria. We believe such a semantic definition will help show that IJPs are a sufficient criterion for hardness. This definition allows us to find IJPs via an automatic search procedure (Fig. 3).

Example 7 (IJPs). *Consider again the JP from Fig. 1a. The resilience is $c = 2$ as removing $\Gamma = \{S(2,3), T(3,4)\}$ destroys all 3 witnesses. Removing $\mathcal{S} = \{(1,2)\}$ destroys $\mathbf{w}_1$, and it suffices to just remove one tuple $\Gamma' = \{T(3,4)\}$ to destroy the remaining 2 witnesses. Similarly, for removing either $\mathcal{T}$, or both $\mathcal{S}$ and $\mathcal{T}$. This proves the OR-property of this JP. Further, composing 3 JPs in a triangle as shown in Fig. 2 is non-leaking (the resulting database has 9 witnesses), and thus this JP is an IJP.*

We now prove that the ability to create an IJP for a query proves its resilience to be hard. This was left as an open conjecture in [31, Conjecture 49].

Theorem 7.4 (IJPs $\Rightarrow$ NPC). *If there is a database D under set/bag semantics that forms an IJP for a query Q, then* RES(Q) *is* NPC *for the same semantics.*

Proof Intuition. We use a reduction from minimum vertex cover to prove that RES(Q) is NPC for any database that forms an IJP for Q. IJPs allow us to abstract the hardness gadgets (and can be thought of as a "template") that are used to reduce vertex cover to our problems. The problem of minimum vertex cover in graphs is closely related to resilience (resilience can be thought of as minimum vertex cover in the data instance hypergraph). For the reduction, IJPs are used as edge gadgets to compute the Vertex Cover while the endpoint tuples form the nodes. The reduction is based on the idea that a node is in the min vertex cover set iff the tuples are in the corresponding resilience/responsibility set. The IJPs are designed such that they have the OR property (if one endpoint set is not chosen, then the other needs to be chosen in order to get the resilience for that edge). This is just like in Vertex Cover: either one of the nodes is required and sufficient to cover an edge. □

We next prove that the ability to create an IJP for a self-join free CQ is not only a sufficient but also a *necessary criterion* for hardness. We prove Theorem 7.5, which does not add new complexity results over [30], but together with Theorem 7.4 shows that IJPs are strictly more general and thus a strictly more powerful criterion for resilience than the previous notion of triads[30] (a triad always implies an IJP, but not vice versa) : they capture the same hardness for SJ-free queries, but can also prove hardness for queries with self-joins that do not contain a triad.

Theorem 7.5 (IJPs $\Leftrightarrow$ NPC for SJ-free CQs). *The resilience of a SJ-free CQ under set/bag semantics is* NPC *iff it has an IJP under the same semantics.*

Proof Intuition (Theorem 7.5). We generalize all past hardness results [30] for SJ-free queries by showing that the same hardness criteria (*triads*) that was necessary and sufficient for hardness, can always be used to construct an IJP and show this construction. □

We conjecture that the existence of an IJP is a necessary criterion for hardness for all queries. In addition, we conjecture that the size of smallest IJP formed by database under a hard query Q is bounded by a small constant factor of the query size.

Conjecture 7.6 (Necessary hardness condition). *If there exists no database D under set/bag semantics that forms an IJP from some tuples $\mathcal{S}$ to $\mathcal{T}$ under query Q, then* $\mathsf{RES}(Q)$ *is in* PTIME *under the same semantics.*

Conjecture 7.7 (IJP Size Bound). *If there exists a database D under set/bag semantics of domain size that forms an IJP under query Q, then there exists a database under same semantics as D, with domain size $d \leq 7 \cdot |\mathtt{var}(Q)|$, that forms an IJP from some tuples $\mathcal{S}$ to $\mathcal{T}$ under query Q.*

Intuition (Conjecture 7.7). The intuition for bounding the size of the certificate to domain $d = 7 \cdot |\mathtt{var}(Q)|$ comes from the connections between an IJP and the OR property. Each known IJP exhibits a "core" of 3 witnesses that exhibit the OR property (which can be seen simply in the self-join free case as parallel to the three independent relations of the triad as in Fig. 1a). This core could take up to $d = 3 \cdot |\mathtt{var}(Q)|$ size. However, this "core" may (1) not have isomorphic endpoint tuple pairs and (2) not be able to exist "independently" and form additional witnesses under Q due to Join dependencies (this is the intuition behind Definition 7.3 (5)). We hypothesize that the endpoint tuple pairs can each be connected to "legs" of 2 witnesses each, thus resulting in a new endpoint pair that is isomorphic. This would add up to 2 times $2 \cdot |\mathtt{var}(Q)|$ constants, bringing the total size up to $7 \cdot |\mathtt{var}(Q)|$. To resolve (2), we must add the witnesses formed due to join dependencies to the certificate. However, this does not increase the number of constants used and hence we hypothesize $d = 7 \cdot |\mathtt{var}(Q)|$ as an upper bound. We show an additional figure in the full paper [52], in which we highlight the cores and legs of the example IJPs in Fig. 3. □

7.2 Automatic creation of hardness certificates

We introduce a Disjunctive Logic Program DLP[RESIJP] that finds IJPs to prove hardness for RES. Each DLP requires Q, a domain d (which bounds the size of the IJP), and two endpoints $\mathcal{S}, \mathcal{T}$.[11] DLP[RESIJP] programs are generated automatically for a given input, are short (200-300 lines depending on the query) and leverage many key technical insights used to model DLPs.

The goal of DLP[RESIJP] is to find a database that fulfills the conditions of Definition 7.3. The search space is a database with all possible tuples given domain d (thus of size $O(d^a)$ where a is the maximum arity of any relation). Each tuple in the search space must be either "picked" in the target database or not. The constraints of our definition are modeled as disjunctive rules with negation. We solve our DLP with the open-source ASP solver *clingo* [59] which uses an enhancement of the DPLL algorithm [22] (used in SAT solvers) and works far faster in practice than a brute force approach. Here we talk only about the overall structure and intuition, but make examples available in the code [51] and full paper [52].

(1) **Search Space:** For all relations in Q, we initialize all possible tuples permitted in domain d as input facts and provide them with an additional tuple id (*TID*). Thus, each relation R has a corresponding relation in the program with $\mathtt{arity}(R)^d$ facts.
(2) **"Guess" an IJP:** Each tuple either participates in the IJP or not. We follow the Guess-Check methodology [28] and use a relation $indb(\underline{R}, \underline{TID}, I)$ to "guess" for each tuple whether it is in the IJP database or not. Here R stands for a relation and together with *TID* uniquely identifies a tuple. The binary value I is 1 if the tuple is in the IJP, and 0 otherwise.

[11] Since the number of possible endpoint configurations is polynomial in the query size, we can simply run parallel programs for different endpoints as input. Notice that endpoints $e_1 = \{A(1)\}, e_2 = \{A(2)\}$ is exactly the same as $e_1 = \{A(3)\}, e_2 = \{A(4)\}$ since the actual value does not matter. In practice, we used any subset of endogenous tuples from a canonical database that can be shared across two witnesses without creating another witness.

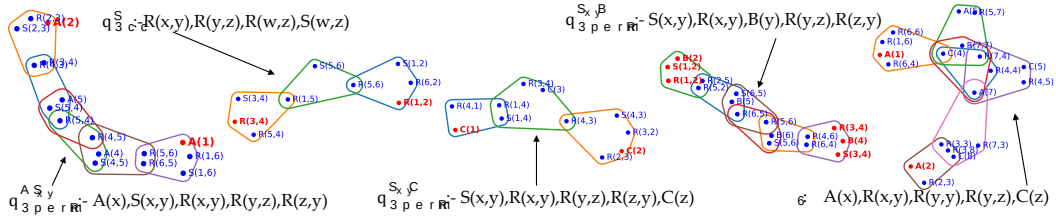


Fig. 3. Automatically generated and visualized IJPs for 5 previously open queries. The nodes corresponding to tuples in $\mathcal{S} \cup \mathcal{T}$ are in red.

(3) **Enforce JP endpoint conditions:** Since the endpoints are considered "input", we do not need to check condition (3*i*) for the JP endpoints (Definition 7.1). However, we need to verify condition (3*ii*) as it depends on the other tuples in the IJP and translate the condition directly into a logic rule.
(4) **Calculate Resilience using "Saturation":** We solve a problem that is NPC (i.e. check that there is a valid contingency set of size c), and a problem that is co-NP-complete (i.e. there is no valid contingency set of size $c-1$). For solving the NP problem we use the guess-check methodology and to solve the co-NP problem, we use the saturation technique.
(5) **Enforce OR-property:** We calculate resilience for 4 databases using the previous step: our original "guess", and the guess with either or both endpoints removed. The removal of endpoints here simply implies defining a new relation that has all tuples of *indb* except the removed endpoint tuples.
(6) **Enforce non-leaking composition:** We define a mapping relation to create 3 isomorphs of the tuples in *indb*. We combine them into one database and check that computing query Q results in exactly 3 times the number of original witnesses.
(7) **(Optional) Minimize the size of the IJP:** To generate smaller certificates that are more human-readable, we simply minimize the number of witnesses in the IJP. We use weak constraints [27] to perform this optimization.

Corollary 7.8 (Sufficient hardness condition). *If there is a domain d and endpoints $\mathcal{S}, \mathcal{T}$ such that* DLP[RESIJP$(Q, d, \mathcal{S}, \mathcal{T})$] *is satisfiable, then* RES(Q) *is* NPC.

Corollary 7.9 (Complexity bound). *It is in Σ_p^2 of d to check if a query Q can form an IJP of domain size d or less.*

The guarantees of our DLP is one-sided: if it finds a certificate, then resilience of the query is guaranteed to be NPC. If it does not provide a certificate, then we have no guarantee. So far we have not found any query that is known to be hard and for which our DLP could not create a certificate for $d = 3 \cdot |\mathtt{var}(Q)|$. This is in line with Conjecture 7.7 that implies that DLP[RESIJP] is not only a sufficient but also complete algorithm for $d = 7 \cdot |\mathtt{var}(Q)|$ (i.e. if the algorithm does not find a certificate for $d = 7 \cdot |\mathtt{var}(Q)|$, then the query is in PTIME).

Example IJPs. Prior work [31] left open the complexity of resilience for 7 binary CQs with three self-join atoms. Our DLP proved 5 of them to be hard (Fig. 3 shows them and their IJPs).

8 COMPLEXITY RESULTS FOR SJ-FREE CQS

This section gives complexity results for both RES and RSP for SJ-free queries, under set and bag semantics (see Table 1). Our results include both prior known results and new results. Importantly, all our hard cases are derived with our unified hardness criterion (IJPs) from Section 7, and all tractable cases follow from our unified algorithms in Sections 4 to 6.

8.1 Necessary notations

Before diving into the proofs, we define a few key concepts stemming from domination (Definition 8.1) that lead up to the three structural criteria (Definition 8.5) which completely describe our dichotomy results. Notice that the notion of *triads* has been previously defined [30]. However, we extend this notion and make it more-fine grained. The previous definition of *triad* now corresponds exactly to the special case of "*active triads*."

Definition 8.1 (Domination [30]). In a query Q with endogenous atoms A and B, we say A *dominates* B iff $\mathtt{var}(A) \subset \mathtt{var}(B)$.

Definition 8.2 (Triad (different from [30])). A *triad* is a set of three atoms, $\mathcal{T} = \{R_1, R_2, R_3\}$ s.t. for every pair $i \neq j$, there is a path from R_i to R_j that uses no variable occurring in the third atom of $\mathcal{T}$.

Definition 8.3 (Solitary variable [30]). In a query Q a variable v in relation A is solitary if, in the query hypergraph it cannot reach any endogenous atom $B \neq A$ without passing through one of the nodes in $\mathtt{var}(A) - v$.

Definition 8.4 (Full domination [30]). An atom A of CQ Q is *fully dominated* iff for all non-solitary variables $y \in \mathtt{var}(A)$ there is another atom B such that $y \in \mathtt{var}(B) \subset \mathtt{var}(A)$.

Definition 8.5 (Active or (fully) deactivated triads). A triad is *deactivated* iff at least one of its three atoms is dominated by another atom of the query. A triad is *fully deactivated* iff at least one of its three atoms is *fully* dominated by another atom of the query. A triad is *active* iff none of its atoms are dominated.

We call queries *linear* if they do not contain triads. Here we depart from prior work that referred to linear queries as queries without what we now call active triads [30]. We instead say that queries without active triads are *linearizable*.[12]

Example 8. *Consider the triad $\{R, S, T\}$ in all 3 queries $Q^{\triangle}$, $Q^{\triangle}_{A}$, and $Q^{\triangle}_{AB}$ from Table 1. The triad is deactivated in $Q^{\triangle}_{A}$ and $Q^{\triangle}_{AB}$ because A dominates both R and T. The triad is fully deactivated in $Q^{\triangle}_{AB}$ because T is fully dominated by A and B. The triad is active in $Q^{\triangle}$ since none of the three tables in the triad are dominated. The chain with ends query Q^{∞}_{2WE} has no triad and is thus linear.*

8.2 Dichotomies for RES under Sets and Bags

This section proves that for all SJ-free CQs, either $\mathtt{LP}[\mathtt{RES}^*]$ solves RES exactly (and the problem is hence easy for any instance), or we can form an IJP (and thus the problem is hard). Our results cover both set and bag semantics (see Table 1).

Theorem 8.6. $\mathtt{LP}[\mathtt{RES}^*(Q, D)] = \mathtt{RES}^*(Q, D)$ *for all database instances D under set or bag semantics if Q is linear.*

Proof Theorem 8.6. Prior approaches show that the witnesses generated by a linear query Q over database instance D can be encoded in a flow graph [55] such that each path of the flow graph represents a witness and each edge with non-infinite weight represents a tuple. The flow graph is such that an edge participates in a path iff the corresponding tuple is part of the corresponding witness. The min-cut of this graph (or the minimum edges to remove to disconnect the source from the target), is equal to $\mathtt{RES}(Q, D)$. We use this prior result to prove that $\mathtt{LP}[\mathtt{RES}^*(Q, D)] = \mathtt{RES}^*(Q, D)$ by showing that the Linear Program solution is a valid cut for the flow graph, and vice versa. Then

[12]The intuition of "linearity" is that the vertices of the dual hypergraph H_d of Q can be mapped onto a line s.t. H_d has the running intersection property [3].

the minimal cut must also be admitted by $\mathtt{LP}[\mathtt{RES}^*(Q,D)]$ and $\mathtt{LP}[\mathtt{RES}^*(Q,D)]$ also cuts the flow graph. Assume we have a fractional LP solution - then for each witness, we still fulfill the constraint that sum of all tuple variables ≥ 1. This implies that the path corresponding to each witness has been cut. Since the number of paths in the flow graph is equal to the number of witnesses, all paths from source to target are cut. By the max-flow Integrality Theorem, there is an equivalent optimal integral solution as well. This integral solution still cuts all paths, and fulfills all conditions of the LP. Thus, for linear queries, $\mathtt{LP}[\mathtt{RES}^*(Q,D)] = \mathtt{ILP}[\mathtt{RES}^*(Q,D)] = RES(Q,D)$. □

Theorem 8.7. $\mathtt{LP}[\mathtt{RES}^*(Q,D)] = \mathtt{RES}^*(Q,D)$ *for all database instances D under set semantics if all triads in Q are deactivated.*

Proof Intuition (Theorem 8.7). Prior work [30] has shown that queries that contain only deactivated triads (previously called dominated triads) can be linearized due to domination (Definition 8.1) We show that this linearization does not change the optimal solution to the LP formulation under set semantics. This is since the dominated table in the deactivated triad can simply be made exogenous, resulting in a linear query. This is equivalent to saying that there is an optimal solution of $\mathtt{LP}[\mathtt{RES}^*]$ where the decision variables of all tuples in dominated table are set to 0. Thus, $\mathtt{LP}[\mathtt{RES}^*]$ models a linear query indirectly, and hence Theorem 8.6 applies to complete the proof. Notice that domination does not work under bag semantics, which leads to a different tractability frontier. □

Theorem 8.8. $\mathtt{RES}(Q)$ *is* NPC *under bag semantics if Q is not linear.*

Proof Intuition (Theorem 8.8). For queries with active triads, the IJPs (Theorem 7.5) imply hardness for bag semantics as well. We prove that *all triads* are hard by showing that including a fixed number of copies of a dominating table is equivalent to making it exogenous. This is equivalent to creating a new IJP where the tuples of the dominating table have c_w copies, where c_w is the number of witnesses in the IJP under set semantics. Now, no minimal contingency set will use tuples of the dominating table, and hence we must consider the tuples from the dominated tables still. Thus, domination does not work under bag semantics, and any triad (even a fully deactivated one) implies hardness. □

The results in this section, along with Theorem 7.5 imply the following dichotomies under both set and bag semantics:

Corollary 8.9. *Under set semantics,* $\mathtt{RES}^*(Q)$ *is in* PTIME *for queries that do not contain active triads, otherwise it is NPC.*

Corollary 8.10. *Under bag semantics,* $\mathtt{RES}^*(Q)$ *is in* PTIME *for queries that do not contain triads, otherwise it is NPC.*

8.3 Dichotomies for RSP under Sets and Bags

This section follows a similar pattern as the previous one to prove that for every SJ-free CQ, either $\mathtt{MILP}[\mathtt{RSP}^*]$ solves RSP exactly (and the problem is hence easy), or we can form an IJP for RSP.

Theorem 8.11. $\mathtt{MILP}[\mathtt{RSP}^*(Q,D,t)] = \mathtt{RSP}^*(Q,D,t)$ *for all database instances D under set or bag semantics if Q is linear.*

Proof Theorem 8.11. Let X_m be an optimal variable assignment generated by solving $\mathtt{MILP}[\mathtt{RSP}^*(Q,D,t)]$ There must be at least one witness $w_p \in D$ such that $t \in w_p$ and $X_m[w_p] = 0$ i.e. the witness is not destroyed (this follows from the fact that the counterfactual clause enforces that all witnesses containing t cannot take value 1). For such a witness, any tuple $t' \in w_p$, must have $X[t'] = 0$ since it satisfies the witness tracking constraints. We also know that since Q is a linear

query, the witnesses can be encoded in a flow graph to find the responsibility [30, 55]. We can map the values of X_m to the flow graph, where $X_m[t]$ now denotes if an edge in the flow graph is cut or not. Consider $X_m[t] = 0$, since it is not modeled in $\texttt{MILP}[\texttt{RSP}^*]$. We see that this disconnects all paths in the graph (since paths that do not contain t are disconnected by virtue of the resilience constraints of $\texttt{MILP}[\texttt{RSP}^*]$). If we set the weight of all tuples in w_p to ∞, the cut value does not change since these tuples were not part of the cut. Prior work [55] has shown that $\texttt{RSP}(Q, D)$ for linear queries can be calculated by taking the minimum of min-cuts of *all* flow graphs such that have 1 of witnesses that contains t, has weight of all other tuples edges set to ∞. Thus, $\texttt{MILP}[\texttt{RSP}^*(Q, D, t)]$ is at least as much as the responsibility computed by a flow graph. In addition to this, the flow graph with the smallest cut also fulfills all the solutions for $\texttt{MILP}[\texttt{RSP}^*]$ (since at least one witness containing t is preserved, and all witnesses not containing t are cut). Thus, the optimal value of $\texttt{RSP}(Q, D, t)$ can be mapped back to a $\texttt{MILP}[\texttt{RSP}^*]$ assignment. □

Theorem 8.12. $\texttt{MILP}[\texttt{RSP}^*(Q, D, t)] = \texttt{RSP}^*(Q, D, t)$ *for any database D under set semantics if all triads in Q are fully deactivated.*

Proof Intuition (Theorem 8.12). This follows directly from the fact that fully deactivated triads can be linearized without changing the optimal solution [30] and Theorem 8.11. □

Theorem 8.13. $\texttt{LP}[\texttt{RSP}^*(Q, D, t)] = \texttt{RSP}^*(Q, D, t)$ *for all database instances D under set semantics if Q does not contain any active triad and t belongs to an atom that dominates some atom in all deactivated triads in Q.*

Proof Intuition (Theorem 8.13). We prove that in every deactivated triad dominated by A, it is always safe to make the dominated table R exogenous since any tuple from R in the responsibility set is either replaceable, or invalid. This linearizes the query, and the rest follows from Theorem 8.11. Notice that prior work [30] identified as tractable cases those without any *active* triad, which a special case of our more general tractable cases. □

Theorem 8.14. $\texttt{RSP}(Q, D, t)$ *is* NPC *if t belongs to an atom that is part of a triad that is not fully deactivated.*

Proof Intuition (Theorem 8.14). The key principle behind this proof is our more fine-grained notion of exogenous tuples. A tuple a such that a has all the same values for the same variables as t and $\texttt{var}(a) \subseteq \texttt{var}(t)$ is necessarily exogenous since it is not possible for t to become counterfactual if a is removed. We construct an IJP possible due to such an *exogenous tuple* from a dominated table. □

Theorem 8.15. *If* $\texttt{RES}(Q)$ *is* NPC *for a query Q under set or bag semantics then so is* $\texttt{RSP}(Q)$.

Proof Intuition (Theorem 8.15). We give a reduction from $\texttt{RES}(Q)$ to $\texttt{RSP}(Q)$ in both set and bag semantics by adding a witness to the given database instance and selecting a tuple whose responsibility is equal the resilience of the original instance. Our approach extends a prior result [30] that applied only to set semantics. □

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

Corollary 8.16. *Under set semantics,* $\texttt{RSP}(Q)$ *is in* PTIME *for queries that contain only fully deactivated triads or deactivated triads that are dominated by the relation of t, otherwise it is NPC.*

Corollary 8.17. *Under bag semantics,* $\texttt{RSP}(Q)$ *is in* PTIME *for queries that do not contain any triads, otherwise it is NPC.*

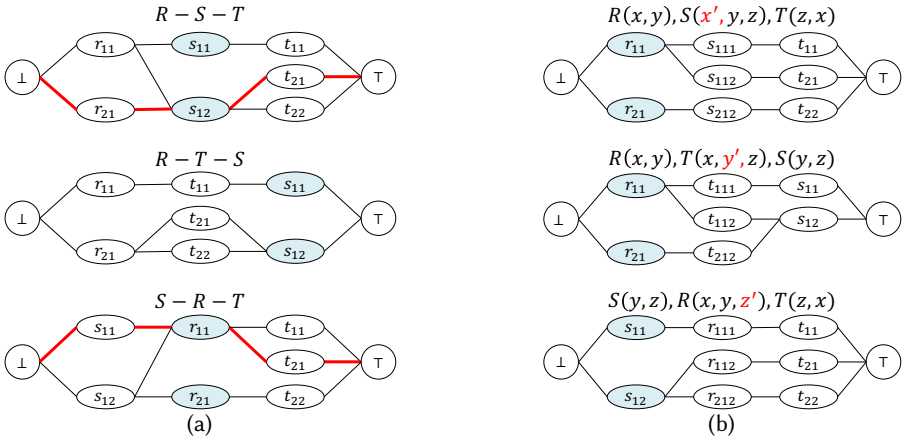


Fig. 4. Flow approximation linearizations for Example 9. We use $\bot$ and $\top$ to represent the source and the target, respectively, of the flow graph to make a connection to an ordering of the atoms of the query.

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

Corollary 8.18. *Under bag semantics, the tractable cases for* $\mathsf{RSP}(Q)$ *are the same as for* $\mathsf{RES}(Q)$.

9 THREE APPROXIMATION ALGORITHMS

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

9.1 LP-based m-factor Approximation

For a given query with m atoms, we use a standard LP rounding technique [70] with the threshold of $1/m$ i.e., we round up variables whose value is $\geq 1/m$ or set them to 0 otherwise.

Theorem 9.1. *The LP Rounding Algorithm is a* PTIME *m-factor approximation for* RES *and* RSP.

Proof Intuition (Theorem 9.1). Verification of PTIME solvability and the m-factor bound is trivial, and correctness follows by showing validity of each constraint for a rounded solution. □

9.2 Flow-based Approximations

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

Constant Tuple Linearization Approximation (`Flow-CT`). We keep the same tuples as the original database in each arrangement. However, since the query is non-linear, these flow graphs may have spurious paths that do not correspond to any original witnesses, thus inadvertently adding witnesses. For a query with m atoms, there are up to $m!/2$ linearizations due to the number of asymmetric ways to order them.

Constant Witness Linearization Approximation (`Flow-CW`). We keep the same witnesses as the original database instance in each linearization, however the query is changed by adding

variables to tables (which is equivalent to dissociating tuples) to make it linear. The number of such linearizations is equal to the number of minimal dissociations [33]. [13]

Example 9. *Consider the $Q^{\triangle}$ query with the following witnesses:*

x	y	z	
1	1	1	$\mathbf{w}_1 = \{r_{11}, s_{11}, t_{11}\}$
1	1	2	$\mathbf{w}_2 = \{r_{11}, s_{12}, t_{21}\}$
2	1	2	$\mathbf{w}_3 = \{r_{21}, s_{12}, t_{22}\}$

Then there are 3 `Flow-CT` *linearizations (Fig. 4a) and* 3 `Flow-CW` *linearizations (Fig. 4b). The approximated resilience corresponds to the minimum of the min-cut over all linearized flow graphs. In this example, we see that both* `Flow-CT` *and* `Flow-CW` *happen to return the optimal value of* 2 *as approximation.*

10 EXPERIMENTS

Our experimental objective is to answer the following questions: (1) How does our ILP scale for PTIME queries, and how does it compare to previously proposed algorithms that use flow-based encodings [55]? (2) Are our LP relaxations (proved to be correct for PTIME queries in Section 8) indeed correct in practice? (3) What is the scalability of ILPs and LPs for settings that are proved NPC? (4) What is the quality of our approximations from Section 9?

Algorithms. `ILP` denotes our ILP formulations for `RES` and `RSP`. `ILP(10)` denotes the solution obtained by stopping the solver after 10 seconds.[14] `LP` denotes LP relaxations for `RES` and `RSP`. `MILP` denotes the MILP formulation for `RSP`. `Flow` denotes an implementation of the prior max-flow min-cut algorithm for `RES` and `RSP` for queries that are in PTIME [30, 55].[15] `LP-UB` denotes our m-factor upper bound obtain by the LP rounding algorithm. `Flow-CW` and `Flow-CT` represent our approximations via Constant Witness Linearization and Constant Tuple Linearizations, respectively.

Data. We use both synthetic and TPC-H data [67]. For any synthetic data experiment, we fix the maximum domain size, and sample randomly from all possible tuples. For testing our methods under bag semantics, each tuple is replicated by a random number that is smaller than a pre-specified max bag size. For TPC-H data, we use the TPC-H data generator at logarithmically increasing scale factors, creating 18 databases ranging from scale factor 0.01 to 1.

Software and Hardware. We implement the algorithms using Python 3.8.5 and solve the respective optimization problems with Gurobi Optimizer 8.1.0 [39]. Experiments are run on an Intel Xeon E5-2680v4 @2.40GHz machine available via the Northeastern Discovery Cluster.

Experimental Protocol. For each plot we run 30 runs of logarithmically and monotonically increasing database instances. We plot all obtained points with a low saturation, and draw a trend line between the median points from logarithmically increasing sized buckets. All plots are log-log, with the x-axis representing the number of witnesses. The y-axis for plots on the left shows the

[13] A detail of implementation here is that for `RSP` it is possible that responsibility tuple t is split into multiple tuples. Then we find responsibility over the set of those tuples, instead of a single tuple. This is a simple extension to make, but differs from the standard definition of responsibility, which allows for just one responsibility tuple.

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

[15] For the min-cut algorithm, we also experimented with both LP and Augmented Path-based algorithms via the NetworkX library [66]. Since the time difference in the methods was not significant, we leave it out and all running times reported in the figures use the same LP library Gurobi [39].

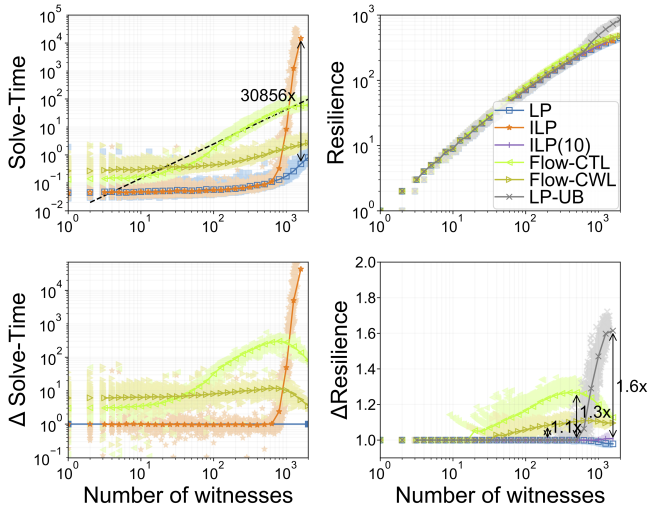
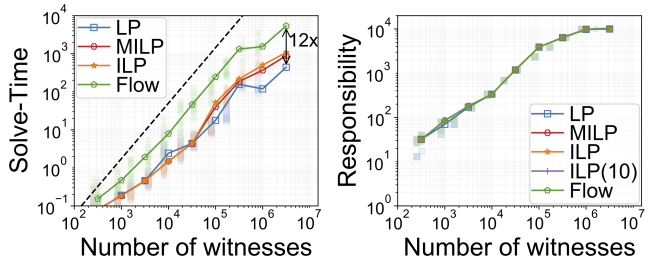
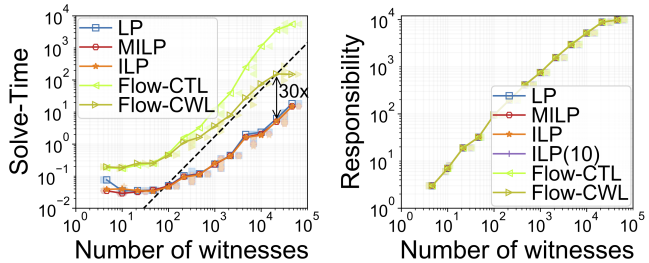


Fig. 5. Setting 1: Hard 3-star query $Q_3^\star$.



(a) 5 Chain Query Q_5^∞ (an easy query)



(b) 5 Cycle Query Q_5° (a hard query)

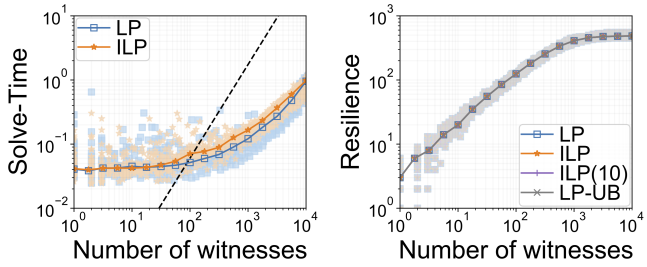
Fig. 6. Setting 2: TPC-H data with FDs.

solve-time (in seconds) taken by the solver to solve a RES, RSP or min-cut problem.[16] We include a dashed line to show linear scalability as reference in the log-log plot.

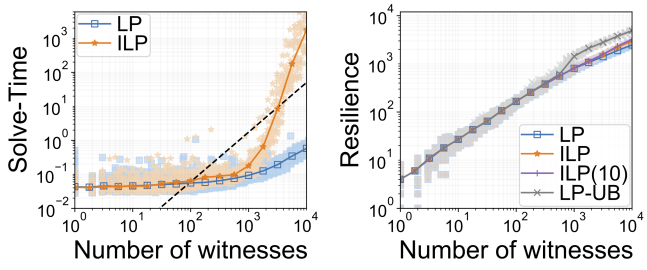
10.1 Experimental Settings

Setting 1: Resilience Under Set Semantics. We consider the 3-star query $Q_3^\star :- R(x), S(y), T(z), W(x, y, z)$ which contains an active triad and is hard (Fig. 5). The top plots show the growth of solve-time and resilience for increasing instances, while the bottom plots show the growths as a

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



(a) SJ-Conf query (an easy query)



(b) SJ-Chain query (a hard query)

Fig. 7. Setting 3: Queries with self-joins.

fraction of the optimal.[17] We see that the solve-time of ILP[RES*] quickly shoots up, while LP[RES*] and the approximations remain PTIME. The bottom plots show a more zoomed-in look, and we see even in the worst case instances, the approximations are only between 1.1x to 1.6x off.

Setting 2: Responsibility With TPCH Data. Fig. 6 shows results for the 5-chain query Q_5^∞ :– Customer(custname, custkey), Orders(custkey, orderkey), Lineitem(orderkey, psid), Partsupplier(id, suppkey) and 5-cycle query Q_5° :– Customer(custname, custkey), Orders(custkey, orderkey), Lineitem(orderkey, psid), Partsupplier(id, suppkey), Supplier(suppkey, suppname) over TPC-H data. While in general Q_5° is NPC, a careful reader may notice that all joins have a primary-foreign key dependencies. We do not inform our algorithms about these dependencies nor make any changes to accommodate them. Yet the solver is able to leverage the dependencies *from the data* and ILP[RES*] scales in PTIME. We see that the ILP is faster than the both dedicated flow algorithm and flow approximation. In both cases, all algorithms (exact and approximate) return the correct responsibility.

Setting 3: Queries with Self-Joins under Bag Semantics. Fig. 7 compares two queries with self-joins: SJ-conf :– $R(x, y), R(x, z), A(x), C(z)$ is easy and SJ-chain :– $R(x, y), R(y, z)$ is hard. The stark difference in the solve-time growth clearly indicates their theoretical complexity. While LP-UB increases as the SJ-chain instance grows, it is still far from the theorized 4-factor worst case bound. We see that ILP-10 is a good indicator for the objective value, even when the ILP takes far longer.

Our full paper provides more experimental settings, such as comparing set and bag semantics [52].

10.2 Key Takeaways from Experiments

We summarize the key takeaways from our experiments:

Result 1. (Scalability of ILP for PTIME Cases) *For easy cases, solving our ILP encoding is in* PTIME *and at times even faster than a previously proposed dedicated flow algorithm.*

[17] The optimal solve-time is LP[RES*] and the optimal resilience is from ILP[RES*].

We see the scalability of ILP for PTIME cases in Figs. 6a and 7a. As expected, solving the ILP formulation takes similar time as LP. We see that Gurobi can solve responsibility around 12 times faster for a PTIME query (Fig. 6a) than the previously proposed flow encoding.

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

Figs. 6a and 7a corroborate the correctness of the LP relaxation for PTIME queries, as expected due to the theorems proved in Section 8.

Result 3. *(**Scalability of ILP and its Relaxations for Hard Cases**) For hard queries, we observe that the time taken by the LP and MILP relaxations grows polynomially, while the time taken by the ILP solution grows exponentially. However, in practice (and in the absence of "hardness-creating interactions" in data) the ILP can often be solved efficiently.*

Figs. 5, 6b and 7b show hard cases. The difference in solve-time is best seen in Figs. 5 and 7, where the ILP overtakes linear scalability. However, interestingly some hard queries don't show exponential time complexity, and for more complicated queries it actually quite difficult to even synthetically create random data for which solving the ILP shows exponential growth.

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

Figs. 5 and 7b show that the results from approximation algorithms are well within theorized bounds and run in PTIME. All approximations are very close to the exact answer, and we need the Δ plots in Fig. 5 to see any difference between exact and approximate results. We observe that in this case `Flow-CW` performs better than `Flow-CT` and is faster as well. `LP-UB` is faster than the flow-based approximations but can be worse. We also see that the LP approximation is worst when the ILP takes much longer than the LP.

11 CONCLUSION AND FUTURE WORK

This paper presented a novel way of determining the complexity of resilience. We give a universal encoding as ILP and then investigate when an LP approximation is guaranteed to give an integral solution, thereby proving that modern solvers can return the answer in guaranteed PTIME. While this approach is known in the optimization literature [65], it has so far not been applied as *proof method* to establish dichotomy results in reverse data management. Since the resulting theory is somewhat simpler and naturally captures all prior known PTIME cases, we believe that this approach will also help in related open problems for reverse data management, in particular a so far elusive complete dichotomy for resilience of queries with self-joins [31].

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation (NSF) under award numbers 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. https://doi.org/10.1016/s0927-0507(05)x1200-2

[2] Albert Atserias and Phokion G Kolaitis. 2022. Structure and complexity of bag consistency. *ACM SIGMOD Record* 51, 1 (2022), 78–85. https://doi.org/10.1145/3542700.3542719

[3] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. 1983. On the Desirability of Acyclic Database Schemes. *J. ACM* 30, 3 (July 1983), 479–513. https://doi.org/10.1145/2402.322389

[4] Leopoldo Bertossi. 2021. Specifying and computing causes for query answers in databases via database repairs and repair-programs. *Knowledge and Information Systems* 63, 1 (2021), 199–231. https://doi.org/10.1007/s10115-020-01516-6

[5] Manuel Bodirsky, Žaneta Semanišinová, and Carsten Lutz. 2023. The Complexity of Resilience Problems via Valued Constraint Satisfaction Problems. (2023). arXiv:2309.15654 [math.LO] https://arxiv.org/abs/2309.15654

[6] Béla Bollobás. 1998. *Modern graph theory*. Vol. 184. Springer Science & Business Media. https://doi.org/10.1007/978-1-4612-0619-4

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

[8] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. 2001. Why and Where: A Characterization of Data Provenance. In *ICDT*. 316–330. https://doi.org/10.1007/3-540-44503-x_20

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

[10] Peter Buneman and Wang-Chiew Tan. 2007. Provenance in Databases. In *SIGMOD*. 1171–1173. https://doi.org/10.1145/1247480.1247646

[11] Florent Capelli, Nicolas Crosetti, Joachim Niehren, and Jan Ramon. 2022. Linear programs with conjunctive queries. (2022). https://doi.org/10.4230/LIPIcs.ICDT.2022.5

[12] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*. 77–90. https://doi.org/10.1145/800105.803397

[13] Surajit Chaudhuri and Moshe Y Vardi. 1993. Optimization of real conjunctive queries. In *PODS*. 59–70. https://doi.org/10.1145/153850.153856

[14] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474. https://doi.org/10.1561/9781601982339

[15] Hana Chockler and Joseph Y. Halpern. 2004. Responsibility and Blame: A Structural-Model Approach. *J. Artif. Intell. Res. (JAIR)* 22 (2004), 93–115. https://doi.org/10.1613/jair.1391

[16] 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. https://doi.org/10.1145/3424305

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

[18] Gérard Cornuéjols and Bertrand Guenin. 2002. Ideal clutters. *Discrete Applied Mathematics* 123, 1-3 (2002), 303–338. https://doi.org/10.1016/S0166-218X(01)00344-4

[19] Nilesh N. Dalvi and Dan Suciu. 2007. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4 (2007), 523–544. https://doi.org/10.1007/s00778-006-0004-3

[20] Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30. https://doi.org/10.1145/2395116.2395119

[21] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. 2001. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.* 33, 3 (2001), 374–425. https://doi.org/10.1145/502807.502810

[22] Martin Davis, George Logemann, and Donald Loveland. 1962. A Machine Program for Theorem-Proving. *Commun. ACM* 5, 7 (jul 1962), 394–397. https://doi.org/10.1145/368273.368557

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

[24] Thomas Eiter and Georg Gottlob. 1993. Propositional circumscription and extended closed-world reasoning are Π_2^P-complete. *Theoretical Computer Science* 114, 2 (1993), 231–245. https://doi.org/10.1016/0304-3975(93)90073-3

[25] Thomas Eiter and Georg Gottlob. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence* 15 (1995), 289–323. https://doi.org/10.1007/bf01536399

[26] Thomas Eiter, Georg Gottlob, and Heikki Mannila. 1997. Disjunctive Datalog. *ACM Trans. Database Syst.* 22, 3 (1997), 364–418. https://doi.org/10.1145/261124.261126

[27] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. 2009. *Answer set programming: A primer*. Springer. https://doi.org/10.1007/978-3-642-03754-2_2

[28] Thomas Eiter and Axel Polleres. 2006. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *Theory and Practice of Logic Programming* 6, 1-2 (2006), 23–60.

https://doi.org/10.1017/s1471068405002577
[29] Lester Randolph Ford and Delbert R Fulkerson. 1956. Maximal flow through a network. *Canadian journal of Mathematics* 8 (1956), 399–404. https://doi.org/10.4153/cjm-1956-045-5
[30] 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
[31] 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. https://doi.org/10.1145/3375395.3387647
[32] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*. 498–510. https://doi.org/10.1145/3106237.3106277
[33] Wolfgang Gatterbauer and Dan Suciu. 2017. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *VLDB J.* 26, 1 (2017), 5–30. https://doi.org/10.1007/s00778-016-0434-5
[34] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. 2011. Potassco: The Potsdam answer set solving collection. *Ai Communications* 24, 2 (2011), 107–124. https://doi.org/10.3233/aic-2011-0491
[35] Michael Gelfond and Yulia Kahl. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press. https://doi.org/10.1017/cbo9781139342124
[36] 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). https://doi.org/10.1561/9781680838817
[37] 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. https://doi.org/10.1007/978-3-642-78240-4_4
[38] LLC Gurobi Optimization. 2021. Mixed-Integer Programming (MIP) – A Primer on the Basics. https://www.gurobi.com/resource/mip-basics/
[39] LLC Gurobi Optimization. 2022. Gurobi Optimizer Reference Manual. http://www.gurobi.com
[40] Joseph Y. Halpern and Judea Pearl. 2005. Causes and Explanations: A structural-model Approach. Part I: Causes. *Brit. J. Phil. Sci.* 56 (2005), 843–887. https://doi.org/10.1093/bjps/axi147
[41] Joseph Y. Halpern and Judea Pearl. 2005. Causes and Explanations: A structural-model Approach. Part II: Explanations. *Brit. J. Phil. Sci.* 56 (2005), 889–911.
[42] Melanie Herschel, Mauricio A. Hernández, and Wang Chiew Tan. 2009. Artemis: A System for Analyzing Missing Answers. *PVLDB* 2, 2 (2009), 1550–1553. https://doi.org/10.14778/1687553.1687588
[43] 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. https://doi.org/10.14778/3425879.3425892
[44] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2008. On the provenance of non-answers to queries over extracted data. *PVLDB* 1, 1 (2008), 736–747. https://doi.org/10.14778/1453856.1453936
[45] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9
[46] Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. 2021. Bag Query Containment and Information Theory. *ACM TODS* 46, 3 (2021). https://doi.org/10.1145/3472391
[47] Vladimir Kolmogorov, Andrei Krokhin, and Michal Rolínek. 2017. The complexity of general-valued CSPs. *SIAM J. Comput.* 46, 3 (2017), 1087–1110. https://doi.org/10.1109/focs.2015.80
[48] George Konstantinidis and Fabio Mogavero. 2019. Attacking Diophantus: Solving a Special Case of Bag Containment. In *PODS*. 399–413. https://doi.org/10.1145/3294052.3319689
[49] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. 2011. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press. https://doi.org/10.1017/cbo9780511977152
[50] Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. 2009. *Why* and *why not* explanations improve the intelligibility of context-aware intelligent systems. In *CHI*. 2119–2128. http://doi.acm.org/10.1145/1518701.1519023
[51] Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility: Code and Experiments. https://github.com/northeastern-datalab/resilience-responsibility-ilp/
[52] Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. (2023). arXiv:2212.08898 [cs.DB] https://arxiv.org/abs/2212.08898
[53] Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y. Halpern, Christoph Koch, Katherine F. Moore, and Dan Suciu. 2010. Causality in Databases. *IEEE Data Eng. Bull.* 33, 3 (2010), 59–67. http://sites.computer.org/debull/A10sept/suciu.pdf
[54] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2009. 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

[55] 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
[56] 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
[57] Alexandra Meliou and Dan Suciu. 2012. Tiresias: the database oracle for how-to queries. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 337–348. https://doi.org/doi/10.1145/2213836.2213875
[58] 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
[59] the Potsdam Answer Set Solving Collection Potassco. 2022. clingo. https://potassco.org/clingo/
[60] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. 2022. Interpretable data-based explanations for fairness debugging. In *SIGMOD*. 247–261. https://doi.org/10.1145/3514221.3517886
[61] Teodor C. Przymusinski. 1991. Stable Semantics for Disjunctive Programs. *New Generation Computing* 9, 3–4 (1991), 401–424. https://doi.org/10.1007/BF03037171
[62] Sudeepa Roy and Dan Suciu. 2014. A Formal Approach to Finding Explanations for Database Queries. In *SIGMOD*. 1579–1590. https://doi.org/10.1145/2588555.2588578
[63] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional fairness: Causal database repair for algorithmic fairness. In *SIGMOD*. 793–810. https://doi.org/10.1145/3299869.3319901
[64] Alexander Schrijver. 1998. *Theory of linear and integer programming*. John Wiley & Sons. https://doi.org/10.1137/1030065
[65] Alexander Schrijver. 2003. *Combinatorial optimization: polyhedra and efficiency*. Algorithms and Combinatorics, Vol. 24. Springer. https://doi.org/book/9783540443896
[66] Daniel A Schult and P Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in science conferences (SciPy 2008)*, Vol. 2008. Pasadena, CA, 11–16. https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-08-05495
[67] TPC-H. 2022. TPC-H Homepage. https://www.tpc.org/tpch/
[68] Jeffrey D. Ullman. 1990. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. W. H. Freeman & Co., New York, NY, USA.
[69] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*. 137–146. https://doi.org/10.1145/800070.802186
[70] Vijay V Vazirani. 2001. *Approximation algorithms*. Vol. 1. Springer. https://doi.org/10.1007/978-3-662-04565-7
[71] Xiaolan Wang, Mary Feng, Yue Wang, Xin Luna Dong, and Alexandra Meliou. 2015. Error diagnosis and data profiling with data x-ray. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1984–1987. https://doi.org/10.14778/2824032.2824117
[72] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. 2017. QFix: Diagnosing errors through query histories. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1369–1384. https://doi.org/10.1145/3035918.3035925
[73] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6, 8 (2013), 553–564. https://doi.org/10.14778/2536354.2536356
[74] Mihalis Yannakakis. 2022. Technical Perspective: Structure and Complexity of Bag Consistency. *ACM SIGMOD Record* 51, 1 (2022), 77–77. https://doi.org/10.1145/3542700.3542718
[75] Brit Youngmann, Michael Cafarella, Yuval Moskovitch, and Babak Salimi. 2022. On Explaining Confounding Bias. *arXiv preprint arXiv:2210.02943* (2022). https://arxiv.org/abs/2210.02943

Received April 2023; revised July 2023; accepted August 2023