


Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries

NEHA MAKHIJA,  Northeastern University, USA

WOLFGANG GATTERBAUER,  Northeastern University, USA

We consider the problem of finding the minimal-size factorization of the provenance of self-join-free conjunctive queries, i.e., we want to find a formula that *minimizes the number of variable repetitions*. This problem is equivalent to solving the fundamental Boolean formula factorization problem for the restricted setting of the provenance formulas of self-join free queries. While general Boolean formula minimization is Σ_2^P -complete, we show that the problem is NP-complete in our case. Additionally, we identify a large class of queries that can be solved in PTIME, expanding beyond the previously known tractable cases of read-once formulas and hierarchical queries.

We describe connections between factorizations, Variable Elimination Orders (VEOs), and minimal query plans. We leverage these insights to create an Integer Linear Program (ILP) that can solve the minimal factorization problem exactly. We also propose a Max-Flow Min-Cut (MFMC) based algorithm that gives an efficient approximate solution. Importantly, we show that both the Linear Programming (LP) relaxation of our ILP, and our MFMC-based algorithm are *always correct for all currently known PTIME cases*. Thus, we present two unified algorithms (ILP and MFMC) that can both recover all known PTIME cases in PTIME, yet also solve NP-complete cases either exactly (ILP) or approximately (MFMC), as desired.

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

Additional Key Words and Phrases: Factorization, Provenance, Boolean Formulas



ACM Reference Format:

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

1 INTRODUCTION

Given the provenance formula for a Boolean query, what is its *minimal size equivalent formula*? And under what conditions can this problem be solved efficiently? This paper investigates the complexity of minFACT, i.e. the problem of finding a minimal factorization for the provenance of self-join-free conjunctive queries (sj-free CQs). While the general Boolean formula minimization is Σ_2^P -complete [7], several important tractable subclasses have been identified, such as read-once formulas [32]. In this paper, we identify additional tractable cases by identifying a large class of queries for which the minimal factorization of any provenance formula can be found in PTIME.

We focus on provenance formulas for two key reasons: 1) Provenance computation and storage is utilized in numerous database applications. The issue of storing provenance naturally raises the question: How can provenance formulas be represented minimally? This problem has previously been investigated in this context [50, 51], where algorithms were described for factorizations with

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



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).

ACM 2836-6573/2024/5-ART104

<https://doi.org/10.1145/3651605>

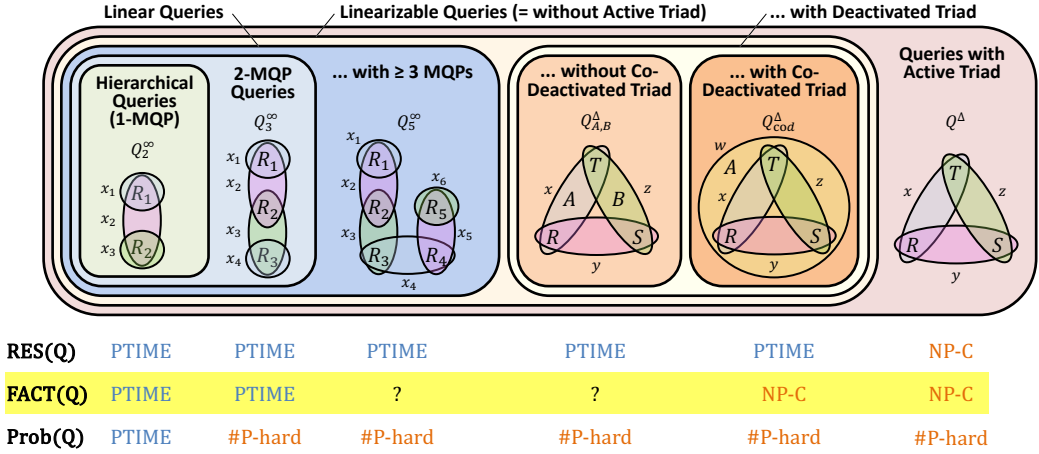


Fig. 1. This paper gives hardness results, identifies PTIME cases, and gives exact and approximate algorithms for self-join-free conjunctive queries. We prove that the tractable queries for minFACT reside firmly between the tractable cases for probabilistic query evaluation (PROB) = the hierarchical queries with one minimal query plan, and those for resilience (RES) = queries without active triads. The open cases are linear queries with ≥ 3 minimal query plans (though we know that Q_4^∞ is in PTIME), and linearizable queries with deactivated triads and without co-deactivated triads (though we know that the triangle unary query Q_U^Δ is in PTIME).

asymptotically optimal sizes, leading to work on factorized databases. However, finding instance-optimal factorizations i.e. factorizations that are guaranteed to be the smallest possible, for any arbitrary input, remains an open challenge, and is the focus of our work.

2) Minimal factorizations of provenance formulas can be used to obtain probabilistic inference bounds. Prior approaches for approximate probabilistic inference are either incomplete i.e. focus on just PTIME cases [17, 54, 56], or do not solve all PTIME cases exactly [17, 28]. As we show, using minimal factorization as a preprocessing step achieves the best of both worlds: It is complete (i.e. it applies to easy and hard cases) while recovering all known PTIME cases exactly.

In this paper, we prove that the minimal factorization problem is NP-complete (NP-C) for provenance formulas, and give two algorithms for all sj-free CQs that are unified algorithms in the sense that they solve all known tractable cases in PTIME, and provide approximations for hard cases. We further place the set of tractable queries firmly between the tractable queries for two other related problems: resilience [24] and probabilistic query evaluation [16] (Fig. 1).

Contributions & Outline. ① The minFACT problem has strong ties to the diverse problems of Boolean factorization, factorized databases, probabilistic inference, and resilience, among others. Section 2 explains these connections before Section 3 formalizes the problem. ② Section 4 describes connections between provenance factorizations, variable elimination orders (VEOs) and query plans. These connections allow us to reformulate minFACT as the problem of assigning each witness to one of several “minimal VEOs.” ③ Section 5 develops an ILP encoding to solve minFACT for any sj-free CQ exactly. We are not aware of any prior ILP formulation for minimal-size encodings of propositional formulas, for restricted cases like for monotone formulas. ④ Section 6 describes our two unified PTIME algorithms that are exact for *all known* PTIME cases, and approximations otherwise. The first one encodes the problem in the form of a “factorization flow graph” s.t. a *minimal cut* of the graph corresponds to a valid factorization of the instance. We refer to this algorithm as the MFMC (Max-Flow Min-Cut) based algorithm. The second is an LP relaxation of our ILP encoding, for which we also prove a guaranteed constant factor approximation for hard queries.

5 **Section 7** proves that both our unified algorithms can solve the minFACT problem *exactly* if the database instance allows a *read-once factorization*. This implies that our algorithms recover and generalize prior approaches [54] that are limited to read-once formulas. 6 **Section 8** provides a large class of queries for which our PTIME algorithms can solve the minFACT problem exactly over any database instance. This class includes hierarchical queries as a strict subset, proving that the tractable queries for minFACT are a strict superset of those for probabilistic query evaluation (PROB) [17]. 7 **Section 9** proves that the decision variant of minFACT is NP-C for a set of queries that form a strict superset of queries that contain “active triads”. This result proves that the intractable queries for minFACT are a strict superset of those that are intractable for resilience (RES) [24], thereby bounding the tractable queries for our problem firmly between those tractable for PROB and those tractable for RES.

Appendix. The appendix contains details about notations and some additional discussion about related work. However, a much longer online appendix [43] contains all proofs, more illustrating examples, and further discussions. It also shows that using minimal factorization can lead to more accurate probabilistic inference [43]. We also perform experiments evaluating the performance and results of the ILP encoding, LP relaxation, or MFMC-based algorithm [43].

2 RELATED WORK

Boolean Factorization. Minimum Equivalent Expression (MEE) is the problem of deciding whether a given Boolean formula φ (note that we use the terms expressions and formulas interchangeably) has a logically equivalent formula φ' that contains $\leq k$ occurrences of literals. It was known to be at least NP-hard for over 40 years [26, Section 7.2] and was shown to be Σ_p^2 -complete only 10 years ago [7]. The problem is more tractable for certain restrictions like Horn formulas [38] as input, or if allowing arbitrary Boolean functions as connectors [39], or if posed as the Minimum Formula Size Problem (MFSP) that takes the uncompressed truth table as input [2, 40]. There is a lot of work on approximate Boolean function factorization [45, 47], however efficient, exact methods are limited to classes such as read-once [32] and read-polarity-once formulas [9] (see [14, Section 10.8] for a detailed historical overview). Our problem restricts the formula to be minimized to the provenance of a sj-free conjunctive query (i.e. a monotone, m -partite DNF that follows join dependencies), with the goal of uncovering important classes that permit a PTIME exact evaluation. An illustration of an overview of known results can be found in the full online appendix. [43, Fig 7] To the best of our knowledge, no prior work has provided a general approach for finding the minimal factorization of monotone k -partite Boolean formulas given as DNFs, and we are unaware of prior work that provides an ILP for the problem, even under restricted settings.

Factorized Databases and Related Work on Factorization. Our problem has been studied before in the context of Factorized Databases (FDBs) [49–52]. Five key differences in focus are: (i) The tight bounds provided through that line of work are on “readability” i.e. the lowest k such that each variable in the factorized formula is repeated at most k times. The work shows that the class of queries with bounded readability is strictly that of hierarchical queries [51]. In contrast, we focus on the minimal number of variable repetitions and show this can be calculated in PTIME for a strict superset of hierarchical queries. (ii) For bounds on the minimal length (as is our focus), FDBs focus only *asymptotic* bounds on the size of query result representations [52] whereas we focus on minimizing the exact number of variables (e.g. whether a provenance is read-once or has a factor 2 bigger size is of no relevance in the asymptotic analysis of FDBs). (iii) variants of FDBs permit the reuse of intermediate results, i.e. they focus on the corresponding *circuit* size, while we focus on *formulas*. (iv) Intuitively, FDBs study the trade-offs between applying one of several factorizations (or query plans or variable elimination orders) to the entire query results at once,

whereas we may *factorize each witness in different ways*. (v) Except for [50], the work on FDBs focuses on factorizations in terms of *domain values* whereas provenance formulas are defined in terms of tuple variables (e.g. a tuple from an arity-3 relation has 3 different domains, but is still represented by a single tuple variable). These discrepancies lead to different technical questions and answers. Also related is the very recently studied problem of finding a factorized representation of all the homomorphisms between two finite relational structures [5]. Similar to FDBs, that work also differs from ours in that it focuses on the asymptotic factorization size (and proves lower bounds and allows a circuit factorization instead of a formula). Our problem is also different from the problem of calculating a “*p*-minimal query” for a given query [3]: The solution to our problem depends on the database instance and factorizes a given provenance formula, whereas the latter problem is posed irrespective of any given database, chooses among alternative polynomials, and becomes trivial for queries without self-joins.

Probabilistic Inference, Read-Once Formulas, and Dissociation. Probabilistic query evaluation (PROB) is #P-hard in general [17]. However, if a provenance formula φ can be represented in *read-once form* then its marginal probability $\mathbb{P}[\varphi]$ can be computed in linear time in the number of literals. Olteanu and Huang [48] showed that the previously known tractable queries called hierarchical queries lead to read-once factorizations. A query Q is called hierarchical [17] iff for any two existential variables x, y , one of the following three conditions holds: $\text{at}(x) \subseteq \text{at}(y)$, $\text{at}(x) \supseteq \text{at}(y)$, or $\text{at}(x) \cap \text{at}(y) = \emptyset$, where $\text{at}(x)$ is the set of atoms of Q in which x participates. Roy et al. [54] and Sen et al. [56] independently proposed algorithms for identifying read-once provenance for non-hierarchical queries in PTIME. Notice that finding the read-once form of a formula (if it exists) is just an extreme case of representing a Boolean function by a minimum length (\vee, \wedge) -formula. Our solution is a *natural generalization* that is *guaranteed* to return a read-once factorization in PTIME should there be one. We give an interesting connection by proving that the tractable queries for our problem are a *strict superset* of hierarchical queries and thus the tractable queries for probabilistic query evaluation.

Resilience. The resilience problem [24, 25] is a variant of the deletion propagation problem [8, 19] focusing on Boolean queries: Given $D \models Q$, what is the minimum number of tuples to remove (called a “contingency set”) in order to make the query false? We give an interesting connection by proving that the tractable queries for our problem are a *strict subset* of the tractable queries for resilience. Concretely, we show that the structural hardness criterion for resilience also makes the factorization problem hard. We achieve separation by giving a query that is easy for resilience yet hard to factorize. Additionally, we hypothesize that linear queries. Additionally, very recent concurrent work by us on resilience [44] has made a similar observation that the LP relaxation of a natural ILP formulation for resilience solves all PTIME queries exactly, which suggests a deeper connection of our problems with general reverse data management problems [46].

Linear Optimization. The question of when an Integer Linear Program (ILP) is tractable has many theoretical and practical consequences [13]. Since we model our problem as an ILP, we can leverage some known results for ILPs to evaluate the complexity of our problem. We show that for a certain class of queries, the constraint matrices of our ILP are Totally Unimodular [55] and hence the ILP is guaranteed to be solvable in PTIME. Additionally, we find cases that *do not fit known tractable classes*, such as Total Unimodularity [55], Iterative Rounding [41], or Balanced Matrices [12]. We *nevertheless prove that they are in PTIME and can be solved efficiently by ILP solvers*. We believe that additional optimization theory will be instrumental in completing the dichotomy. From a practical perspective, modeling our problem as an ILP allows us to use highly optimized solvers [35] to obtain exact results even for hard queries.

Relation to Holistic Join Algorithms. Our approach has an interesting conceptual connection to “holistic” join algorithms [1] that rely on not just a single tree decomposition (thus one query

plan) but rather multiple tree decompositions (thus multiple plans) *for different output tuples*. Very similarly, our approach also carefully assigns different witnesses to different query plans.

3 FORMAL SETUP

This section introduces our notation and defines the problem `minFACT` i.e. the problem of finding the *minimal factorization* of the provenance of a query.

3.1 Standard database notations

We write D for the database, i.e. the set of tuples in the relations. A *conjunctive query* (CQ) is a first-order formula $Q(\mathbf{y}) = \exists \mathbf{x} (g_1 \wedge \dots \wedge g_m)$ where the variables $\mathbf{x} = (x_1, \dots, x_\ell)$ are called existential variables, \mathbf{y} are called the head variables and each atom g_i represents a relation $g_i = R_{j_i}(\mathbf{x}_i)$ where $\mathbf{x}_i \subseteq \mathbf{x} \cup \mathbf{y}$.¹ W.l.o.g., we discuss only connected queries.² We write $\text{var}(X)$ for the set of variables occurring in atom/ relation/ query/ formula X and $\text{at}(x)$ for the set of atoms that contain variable x . We write $[\mathbf{w}/\mathbf{x}]$ as a valuation (or substitution) of query variables \mathbf{x} by constants \mathbf{w} . These substitutions may be written explicitly by “domain-annotating” variables with domain constants as subscripts. *Domain-annotated tuples* use such domain-annotated variables as subscripts, e.g. r_{x_1, y_2} represents a tuple of relation $R(x, y)$ with $x = 1$ and $y = 2$. We sometimes informally omit the variables and use the notation $r_{v_1 v_2 \dots v_a}$ where $v_1 v_2 \dots v_a$ are the domain values of $\text{var}(R)$ in the order that they appear in atom R . Thus, r_{12} also represents $R(1, 2)$. A *self-join-free* (sj-free) CQ is one where no relation symbol occurs more than once and thus every atom represents a different relation. Thus, for sj-free CQs, one may refer to atoms and relations interchangeably. We focus on Boolean queries (i.e., where $\mathbf{y} = \emptyset$), since the problem of finding the minimal factorization of the provenance for one particular output tuple of a non-Boolean query immediately reduces to the Boolean query case (see e.g. [57]).³ Unless otherwise stated, a query in this paper denotes a *sj-free Boolean CQ*. [Appendix A](#) defines further notation.

3.2 Boolean and Provenance formulas

The terms provenance and lineage are used in the literature with slightly different meanings. While lineage was originally formalized in [15], we follow the modern treatment of *data provenance* as denoting a proposition formula that corresponds to the *Boolean provenance semiring* of Green et al. [33, 34], which is the commutative semiring of positive Boolean expressions $(\mathbb{B}[X], \vee, \wedge, 0, 1)$. We sometimes write \vee as semiring-plus (\oplus) and \wedge as times (\otimes).

We assign to every tuple $t \in D$ a *provenance token*, i.e. we interpret each tuple as a Boolean variable. Then the provenance formula (equivalently, provenance expression) φ_p of a query $Q := R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ on D is the positive Boolean DNF formula

$$\text{Prov}(Q, D) = \bigvee_{\theta: D \models Q[\theta(\mathbf{x})/\mathbf{x}]} R_1(\theta(\mathbf{x}_1)) \wedge \dots \wedge R_m(\theta(\mathbf{x}_m))$$

where $D \models Q[\theta(\mathbf{x})/\mathbf{x}]$ denotes that $\theta(\mathbf{x})$ is a *valuation* or assignment of \mathbf{x} to constants in the active domain that make the query true over database D . Notice that for sj-free queries, this DNF is always

¹We follow the conventional notation for boolean CQs that omits writing the existential quantification and that replaces \wedge by a comma. W.l.o.g., we assume that \mathbf{x}_i is a tuple of only variables and don't write the constants. Selections can always be directly pushed into the database before executing the query. In other words, for any constant in the query, we can first apply a selection on each relation and then consider the modified query with a column removed.

²Results for disconnected queries follow immediately by factorizing each of the query components *independently*.

³A solution to Boolean queries immediately also provides an answer to a non-Boolean query $Q(\mathbf{y})$: For each output tuple $t \in Q(D)$, solve the problem for a Boolean query Q' that replaces all head variables \mathbf{y} with constants of the output tuple t .

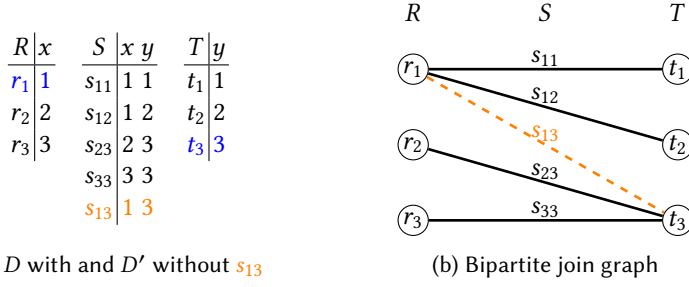


Fig. 2. **Examples 1 and 2:** (a): Database instance with provenance tokens to the left of each tuple, e.g. s_{12} for $S(1, 2)$. (b): $\text{Prov}(Q_2^*, D)$ for $Q_2^* := R(x), S(x, y), T(y)$ represented as bipartite graph. D denotes the database with the orange tuple s_{13} and D' denotes the database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

m -partite as each disjunct contains one tuple from each of the m tables and that the notions of provenance polynomial and provenance formula are interchangeable.

Read-once. For a formula φ , we denote by $\text{var}(\varphi)$ the set of variables that occur in φ , and by $\text{len}(\varphi)$ its length, i.e., the number of its literals.⁴ A provenance is called *read-once* if it can be represented in *read-once* form, i.e. there is an equivalent formula in which each literal appears exactly once [30, 37, 48]. This is possible iff that equivalent formula can be built up recursively from the provenance tokens by disjunction (and conjunction), s.t. whenever $\varphi = \varphi_1 \vee \varphi_2$ (or $\varphi = \varphi_1 \wedge \varphi_2$), then $\text{var}(\varphi_1) \cap \text{var}(\varphi_2) = \emptyset$.

Witnesses. We call a *witness* \mathbf{w} a valuation of all variables \mathbf{x} that is permitted by D and that makes Q true (i.e. $D \models Q[\mathbf{w}/\mathbf{x}]$).⁵ The set of witnesses $\text{witnesses}(Q, D)$ (shorthand W) is then

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

Since every witness implies exactly one set of m tuples from D that make the query true, we will slightly abuse the notation and also refer to this set of tuples as a “witness.” We will also use “witness” to refer to a product term in a DNF of the provenance polynomial.

EXAMPLE 1 (PROVENANCE). Consider the Boolean 2-star query $Q_2^* := R(x), S(x, y), T(y)$ over the database D' in Fig. 2 (ignore the tuple s_{13} for now). Each tuple is annotated with a Boolean variable (or provenance token) r_1, r_2, \dots . The provenance φ_p is the Boolean expression about which tuples need to be present for Q_2^* to be true:

$$\varphi_p = r_1 s_{11} t_1 \vee r_1 s_{12} t_2 \vee r_2 s_{23} t_3 \vee r_3 s_{33} t_3 \quad (1)$$

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

The provenance can be re-factored into a read-once factorization φ' which is a factorized representation of the provenance polynomial in which every variable occurs once, and thus $\text{len}(\varphi') = |\text{var}(\varphi')| = 10$. It can be found in PTIME in the size of the database [31]:

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

⁴Notice that the length of a Boolean expression φ is also at times defined as the total number of symbols (including operators and parentheses, e.g. in [14]). In our formulation, we only care about the number of variable occurrences.

⁵Note 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 [11].

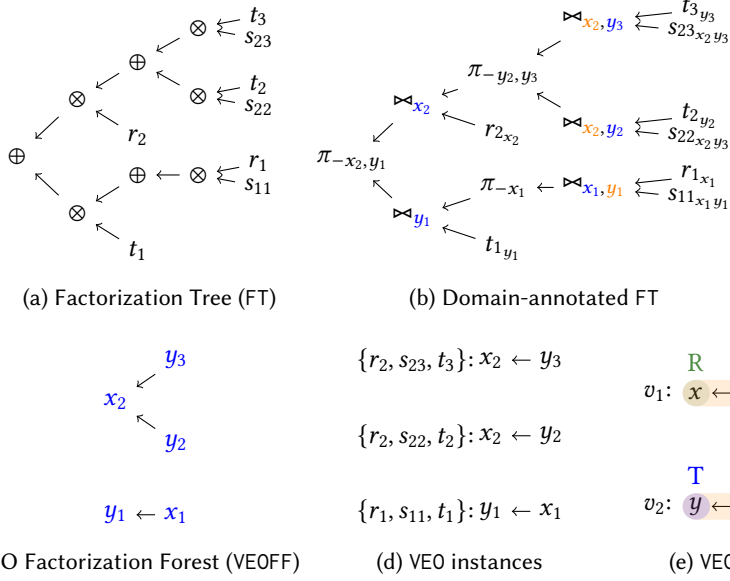


Fig. 3. Representation of a factorization as a mapping of witnesses to VEOs for an example database under query Q_2^* . **Theorem 4.5** shows the correspondence of (a) via (b) to (c). **Theorem 4.6** shows the correspondence of (c) via (d) to (e) for some minimal factorization tree.

3.3 Minimal factorization minFACT

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

Definition 3.1 (FACT). Given a query Q and database D , we say that $(D, k) \in \text{FACT}(Q)$ if there is a formula φ' of length $\text{len}(\varphi') \leq k$ that is equivalent to the expression $\varphi_p = \text{Prov}(Q, D)$.

Our focus is to determine the difficulty of this problem in terms of data complexity [60], i.e., we treat the query size $|Q|$ as a constant. We are interested in the optimization version of this decision problem: given Q and D , find the *minimum* k such that $(D, k) \in \text{FACT}(Q)$. We refer to this optimization variant as the minFACT problem and use $\text{minFACT}(Q, D)$ to refer to the length of the minimal size factorization for the provenance of database D under query Q .

EXAMPLE 2 (FACT). Now consider the provenance of Q_2^* over the modified database D with tuple s_{13} from Fig. 2. It has no read-once form and a minimal size formula is

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

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

4 SEARCH SPACE FOR minFACT

Factorizations. In order to find the minimal factorization of a provenance formula, we first define a search space of all permissible factorizations. Each factorized formula can be represented as a *factorization tree* (or FT), where each literal of the formula corresponds to a leaf node,⁶ and internal

⁶A variable may appear in multiple leaves just as it can in a factorized formula.

nodes denote the \oplus and \otimes operators of the commutative provenance semiring. The length (size) of a FT is the number of leaves. We allow the semiring operations to be k -ary (thus even unary) and use *prefix notation* for the operators when writing FTs in linearized text. Notice that the space of FTs is strictly larger than the space of factorized expressions: E.g., the FT $\otimes(r_1, s_1, t_1)$ is *not* equivalent to $\otimes(r_1, \otimes(s_1, t_1))$, although they represent the same formula $r_1 s_1 t_1$. We consider FTs as equivalent under commutativity i.e. we treat $\otimes(r_1, s_1, t_1)$ as equivalent to $\otimes(s_1, t_1, r_1)$. Furthermore, w.l.o.g., we only consider trees in which the operators \oplus, \otimes *alternate*: E.g., $\otimes(r_1, \otimes(s_1, t_1))$ is not alternating but represents the same formula as the alternating tree $\otimes(r_1, \oplus(\otimes(s_1, t_1)))$ using unary \oplus . Henceforth, we use factorization trees or FTs as a short form for *alternating factorization trees*.

Variable Elimination Order (VEO). FTs describe tuple-level factorizations, however, they fail to take into account the structure (and resulting join dependencies) of the query producing the provenance. For this purpose, we define query-specific *Variable Elimination Orders* (VEOs). They are similar to VEOs in general reasoning algorithms, such as bucket elimination [20] and VEOs defined in FDBs [49] for the case of no caching (i.e. corresponding to formulas, not circuits). However, our formulation allows each node to have a *set of variables* instead of a single variable. This allows VEOs to have a 1-to-1 correspondence to the sequence of variables *projected away* in an “alternating” query plan, in which projections and joins alternate, just as in our FTs (details in [43]). Furthermore, we show VEOs can be “annotated” with a data instance and “merged” to form forests that describe a minimal factorization tree of any provenance formula.

Definition 4.1 (Variable Elimination Order (VEO)). A VEO v of a query Q is a rooted tree whose nodes are labeled with non-empty sets of query variables s.t. (i) each variable of Q is assigned to exactly one node of v , and (ii) all variables x for any atom $R(x)$ in Q must occur in the prefix of some node of v .

Definition 4.2 (VEO instance). Given a VEO v and witness \mathbf{w} , a VEO *instance* $v\langle\mathbf{w}\rangle$ is the rooted tree resulting from annotating the variables x in v with the domain values of \mathbf{w} .

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

EXAMPLE 3 (VEO AND VEO INSTANCE). Consider the 3-chain Query $Q_3^\infty :- R(x, y), S(y, z), T(z, u)$. An example VEO is $v = z \leftarrow (u, y \leftarrow x)$ [43]. To make it a unique serialization, we need to assume an ordering on the children of each parent. Notice that our definition of VEO also allows sets of variables as nodes. As an extreme example, the legal query plan $P' = \pi_{-xyz} \bowtie (R(x, y), S(y, z), T(z, u))$ corresponds to a VEO v' with one single node containing all variables. In our short notation, we denote nodes with multiple variables in brackets without commas between the variables to distinguish them from children: $v' = \{xyz\}$.

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

Definition 4.3 (VEO table prefix). Given an atom R in a query Q and a VEO v , the *table prefix* v^R is the smallest prefix in v that contains all the variables $x \in \text{var}(R)$.

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

EXAMPLE 4 (VEO TABLE PREFIX AND VEO TABLE PREFIX INSTANCE). Consider again the VEO $v = z \leftarrow (u, y \leftarrow x)$ in [Example 3](#). The table prefix of table $S(y, z)$ on v is $v^S = z \leftarrow y$. Assume a set of two witnesses $W = \{(x_1, y_1, z_1, u_1), (x_1, y_1, z_1, u_2)\}$. Then for both witnesses \mathbf{w}_1 and \mathbf{w}_2 , the table prefix instances for S are identical: $v^S\langle\mathbf{w}_1\rangle = v^S\langle\mathbf{w}_2\rangle = z_1 \leftarrow y_1$ [43].

Definition 4.4 (VEO factorization forest (VEOFF)). A VEOFF \mathcal{V} of provenance φ_p of database D over query Q is a forest whose nodes are labeled with non-empty sets of domain-annotated variables, such that: (1) For every $\mathbf{w} \in \text{witnesses}(Q, D)$ there exists exactly one subtree in \mathcal{V} that is a VEO instance of \mathbf{w} and Q ; (2) There is no strict sub-forest of \mathcal{V} that fulfills condition (1).

EXAMPLE 5 (VEO FACTORIZATION FOREST). Continuing with the Q_3^∞ query, and the witnesses $W = \{(x_1, y_1, z_1, u_1), (x_1, y_1, z_1, u_2)\}$ as in [Example 4](#), we illustrate several valid and invalid VEOFFs (with accompanying figures in the online appendix [43]). We represent a forest of VEO instances in-text as a set of trees $\{t_1, t_2, \dots\}$.

The forest $\mathcal{V}_1 = \{x_1 \leftarrow (u_1, u_2, (y_1 \leftarrow z_1))\}$ is a valid VEOFF since (1) for both \mathbf{w}_1 and \mathbf{w}_2 there is exactly one subtree each in \mathcal{V} that is a VEO instance. These subtrees are $x_1 \leftarrow (u_1, (y_1 \leftarrow z_1))$ and $x_1 \leftarrow (u_1, (y_1 \leftarrow z_1))$. This VEOFF also satisfies property (2) removing any variable would lead to a VEOFF that does not satisfy property (1).

The forest $\mathcal{V}_2 = \{x_1 \leftarrow (u_1, (y_1 \leftarrow z_1)), y_1 \leftarrow (z_1, (x_1 \leftarrow u_1))\}$ is also a valid VEOFF since (1) for both \mathbf{w}_1 and \mathbf{w}_2 there is exactly one subtree each in \mathcal{V} that is a VEO instance. These subtrees are $x_1 \leftarrow (u_1, (y_1 \leftarrow z_1))$ and $y_1 \leftarrow (z_1, (x_1 \leftarrow u_2))$. This VEOFF also satisfies property (2) removing any variable would lead to a VEOFF that does not satisfy property (1).

The forest $\mathcal{V}_3 = \{x_1 \leftarrow (u_1, (y_1 \leftarrow z_1)), y_1 \leftarrow (z_1, (x_1 \leftarrow u_1 \leftarrow u_2))\}$ is not a valid VEOFF, although it satisfies property (1) with the same subtrees above. It does not satisfy property (2) since removing u_2 in the second tree would lead to a VEOFF that still satisfies property (1).

THEOREM 4.5 (FACTORIZATIONS AND VEOs). There exist transformations from FTs to VEOFFs and back such that the transformations can recover the original FT for at least one minimal size FT φ' of any provenance formula φ_p .

PROOF INTUITION. We describe a transformation from FTs to VEOFFs via domain-annotated FTs as intermediate step ([Fig. 3](#)). A domain-annotated FT is constructed as follows: We first replace the \otimes operator with a join (\bowtie) and the \oplus operator with a projection (π) and label the leaves with the domain-annotated variables. We then recursively label each join and projection bottom-up as follows: (1) label each \bowtie by the union of variables of its children, and (2) label each π with the subset of variables of its children that are not required for subsequent joins (this can be inferred from the query). To get the VEOFF instance, we remove all variables on joins that appear in ancestor joins. We remove the leaves and absorb all non-join (projection) nodes into their parents (eliminating the root projection node).

We show that if this transformation succeeds then it is a bijection and can be reversed. The only case when this transformation fails is when it results in an empty annotation for a node, i.e. when there is a join after which no variable is projected away (since by design VEOs do not permit empty nodes). In that case, the FT can always be simplified by removing a \oplus node and merging two \otimes nodes.

THEOREM 4.6 (minFACT WITH VEOs). There exists a transformation that constructs FTs of a provenance φ_p from mappings of each witness of φ_p to a VEO of Q , and there exists a mapping that is transformed into a minimal size factorization tree φ' of φ_p under this transformation.

PROOF INTUITION. From [Theorem 4.5](#), we know that for every provenance formula φ_p there exists a minimal size FT that has a reversible transformation to a VEOFF. We show all such VEOFFs can be constructed by assigning a VEO to each witness of φ_p . This is constructed by defining a merge operation on VEOs that greedily merges common prefixes.

Minimal Variable Elimination Orders (mveo). By reducing the problem of finding the minimal factorization to that of assigning a VEO to each witness, we have so far shown that FACT is in NP with respect to data complexity.⁷ However, we can obtain a more practically efficient result by showing that we need not consider all VEOs, but only the *Minimal Variable Elimination Orders* of a query or $\text{mveo}(Q)$. We can define a partial order \preceq on VEOs of a query Q as follows: $v_1 \preceq v_2$ if for every relation $R_i \in Q$ the variables in the R_i table prefix of v_1 are a subset of the variables of the R_i table prefix of v_2 , i.e. $\forall R_i \in Q : \text{var}(v_1^{R_i}) \subseteq \text{var}(v_2^{R_i})$. $\text{mveo}(Q)$ then is the set of all VEOs of Q that are minimal with respect to this partial order \preceq . For Q_2^* , there are only two minimal variable elimination orders $x \leftarrow y$ and $y \leftarrow x$, but not $\{x, y\}$, and for Q_3^* , there are only 6, despite 13 possible VEOs in total. Interestingly, $\text{mveo}(Q)$ corresponds exactly to Minimal Query Plans as defined in work on probabilistic databases [28], and we can use this connection to leverage prior algorithms for computing $\text{mveo}(Q)$.

THEOREM 4.7 (minFACT WITH mveos). *There exists a transformation that constructs FTs of a provenance φ_p from mappings of each witness of φ_p to a VEO $v \in \text{mveo}(Q)$, and there exists a mapping that is transformed into a minimal size factorization tree φ' of φ_p under this transformation.*

5 ILP FORMULATION FOR minFACT

Given a set of witnesses $W = \text{witnesses}(Q, D)$ for a query Q over some database D , we can use the insight of [Theorem 4.7](#) to describe a 0-1 Integer Linear Program (ILP) minFACT_ILP (3) that chooses a $v \in \text{mveo}$ or equivalently a minimal query plan, for each $\mathbf{w} \in W$, s.t. the resulting factorization is of minimal size. The size of the ILP is polynomial in $n = |D|$ and exponential in the query size.

ILP Decision Variables. The ILP is based on two sets of binary variables: Query Plan Variables (QPV) q use a one-hot encoding for the choice of a minimal VEO (or equivalently minimal query plan) for each witness, and Prefix Variables (PV) p encode sub-factorizations that are a consequence of that choice. Intuitively, shared prefixes encode shared computation through factorization.

(1) *Query Plan Variables (QPV):* For each witness $\mathbf{w} \in W$ and each minimal VEO $v \in \text{mveo}(Q)$ we define a binary variable $q[v(\mathbf{w})]$, which is set to 1 iff VEO v is chosen for witness \mathbf{w} .⁸

(2) *Prefix Variables (PV):* All witnesses must be linked to a set of prefix variables, by creating instances of VEO prefixes that are in a query-specific set called the *Prefix Variable Format (PVF)*. This set PVF is composed of all table prefixes of all minimal VEOs $v \in \text{mveo}(Q)$. Notice that prefix variables can be shared by multiple witnesses, which captures the idea of joint factorization. Additionally, we define a *weight* (or cost)⁹ $c(v^R)$ for each table prefix $v^R \in PVF$; this weight is equal to the number of tables that have the same table prefix for a given VEO. From that PVF set, then binary prefix variables $p[v^R(\mathbf{w})]$ are defined for each table prefix $v^R \in PVF$ and $\mathbf{w} \in W$.

ILP Objective. The ILP should minimize the length of factorization len , which can be calculated by counting the number of times each tuple is written. If a tuple is a part of multiple witnesses, it may be repeated in the factorization. However, if the tuple has *the same table prefix instance across different witnesses* (whether as part of the same VEO or not), then those occurrences are factorized

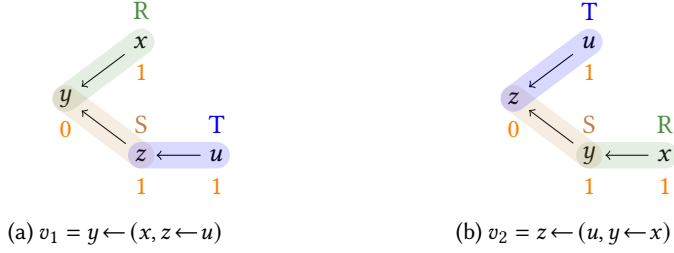
⁷This follows from the fact that the number of witnesses is polynomial in the size of the database, and the number of VEOs only depends on the query size.

⁸Notice that we use indexing in brackets $q[v(\mathbf{w})]$ instead of the more common subscript notation $q_{v(\mathbf{w})}$ since each $v(\mathbf{w})$ can depict a tree. Our bracket notation is more convenient.

⁹We write c for weight (or cost) to avoid confusion with witnesses \mathbf{w} .

$$\begin{aligned}
& \min \sum_{v^R\langle \mathbf{w} \rangle \in PV} c(v^R) \cdot p[v^R\langle \mathbf{w} \rangle] \\
& \text{s.t.} \quad \sum_{v \in \text{mveo}(Q)} q[v\langle \mathbf{w} \rangle] \geq 1, \quad \forall \mathbf{w} \in W \\
& \quad p[v^R\langle \mathbf{w} \rangle] \geq \sum_{v^R\langle \mathbf{w} \rangle \text{ prefix of } v\langle \mathbf{w} \rangle} q[v\langle \mathbf{w} \rangle], \quad \forall p[v^R\langle \mathbf{w} \rangle] \in PV \\
& \quad p[v^R\langle \mathbf{w} \rangle] \in \{0, 1\}, \quad \forall p[v^R\langle \mathbf{w} \rangle] \in PV \\
& \quad q[v\langle \mathbf{w} \rangle] \in \{0, 1\}, \quad \forall q[v\langle \mathbf{w} \rangle] \in QPV
\end{aligned} \tag{3}$$

Fig. 4. ILP Formulation for minFACT

Fig. 5. Example 6: mveo for 3-chain query Q_3^∞ .

together and the tuple is written just once in the factorization. Thus, len is the weighted sum of all selected table prefix instances. The weight accounts for tuples of different tables that have the same table prefix under the same VEO. Since $p[v^R\langle \mathbf{w} \rangle] = 0$ for unselected table prefixes, we can calculate len as:

$$\text{len} = \sum_{v^R\langle \mathbf{w} \rangle \in PV} c(v^R) \cdot p[v^R\langle \mathbf{w} \rangle] \tag{2}$$

ILP Constraints. A valid factorization of W must satisfy three types of constraints:

(1) *Query Plan Constraints:* For every witness $\mathbf{w} \in W$, some $v \in \text{mveo}(Q)$ must be selected.¹⁰ For example, for $\mathbf{w} = (x_1, y_1)$ under Q_2^* , we enforce that: $q[x_1 \leftarrow y_1] + q[y_1 \leftarrow x_1] \geq 1$.

(2) *Prefix Constraints:* For any given table prefix p , it must be selected if any one of the VEOs that has it as a prefix is selected. Since (under a minimization optimization) only one VEO is chosen per witness, we can say that the value of $p[v^R\langle \mathbf{w} \rangle]$ must be at least as much as the sum of all query plan variables $q[v\langle \mathbf{w} \rangle]$ such that $v^R\langle \mathbf{w} \rangle$ is a prefix of $v\langle \mathbf{w} \rangle$. For example, we enforce that $p[x_1]$ must have value at least as much as $q[x_1 \leftarrow y_1 \leftarrow z_1] + q[x_1 \leftarrow z_1 \leftarrow y_1]$. But we cannot enforce $p[x_1] \geq q[x_1 \leftarrow y_1 \leftarrow z_1] + q[x_1 \leftarrow z_1 \leftarrow y_2]$ (as both VEOs do not belong to the same witness.)

(3) *Boolean Integer Constraints:* Since a VEO is either selected or unselected, we set all variables in PV and QPV to 0 or 1.

THEOREM 5.1 (ILP CORRECTNESS). *The objective of minFACT_ILP for a query Q and database D is always equal to minFACT(Q, D).*

COROLLARY 5.2. *FACT, the decision variant of minFACT, is in NP.*

EXAMPLE 6 (ILP FORMULATION FOR 3-CHAIN QUERY). *Consider the 3-Chain query $Q_3^\infty \text{ :- } R(x, y), S(y, z), T(z, u)$ with a set of 2 witnesses $W = \{(x_1, y_1, z_1, u_1), (x_1, y_1, z_1, u_2)\}$ and provenance in DNF of $r_{11}s_{11}t_{11} + r_{11}s_{11}t_{12}$. Using the dissociation based algorithm [28], we see that this query has 2*

¹⁰We wish to have exactly one query plan or minimal VEO per witness, but in a minimization problem, it suffices to say that at least one $v \in \text{mveo}$ is selected - if multiple are selected, either one of them arbitrarily still fulfills all constraints.

minimal query plans corresponding to the two VEOs shown in Fig. 5. We use these VEOs to first build the set QPV (Query Plan Variables) and enforce a query plan constraint for each of the 2 witnesses:

$$\begin{aligned} q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] + q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] &\geq 1 \\ q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] + q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] &\geq 1 \end{aligned}$$

Then, we calculate the elements of the set PVF (Prefix Variable Format) as well as their weights. For the two VEOs from Fig. 5, and the three tables R, S, T, we get 6 distinct table prefixes:

VEO v_1	VEO v_2
$v_1^R = y \leftarrow x$	$v_2^R = z \leftarrow y \leftarrow x$
$v_1^S = y \leftarrow z$	$v_2^S = z \leftarrow y$
$v_1^T = y \leftarrow z \leftarrow u$	$v_2^T = z \leftarrow u$

We add all these table-prefixes to the PVF. Since no table prefix is repeated, they all are assigned weight $c = 1$. Notice that prefixes y for v_1 and z for v_2 are no table prefixes (and thus have weight $c = 0$ and do not participate in the objective).

From the set of table prefixes PVF, we then create the set of prefix variables PV, one for each table prefix and each witness $\mathbf{w} \in W$, and define their prefix constraints. The prefix constraints necessary for witness $\mathbf{w}_1 = (x_1, y_1, z_1, u_1)$ are as follows:

$$\begin{aligned} p[y_1 \leftarrow x_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] & p[y_1 \leftarrow z_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] \\ p[y_1 \leftarrow z_1 \leftarrow u_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_1)] & p[z_1 \leftarrow y_1 \leftarrow x_1] &\geq q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] \\ p[z_1 \leftarrow y_1] &\geq q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] & p[z_1 \leftarrow u_1] &\geq q[z_1 \leftarrow (u_1, y_1 \leftarrow x_1)] \end{aligned}$$

The prefix constraints for witness $\mathbf{w}_2 = (x_1, y_1, z_1, u_2)$ are:

$$\begin{aligned} p[y_1 \leftarrow x_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] & p[y_1 \leftarrow z_1] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] \\ p[y_1 \leftarrow z_1 \leftarrow u_2] &\geq q[y_1 \leftarrow (x_1, z_1 \leftarrow u_2)] & p[z_1 \leftarrow y_1 \leftarrow x_1] &\geq q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] \\ p[z_1 \leftarrow y_1] &\geq q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] & p[z_1 \leftarrow u_2] &\geq q[z_1 \leftarrow (u_2, y_1 \leftarrow x_1)] \end{aligned}$$

Notice that we have 12 constraints (one for each pair of witness and table prefix), yet only 8 distinct prefix variables due to common prefixes across the two witnesses (which intuitively enables shorter factorizations). For this query, for every witness, there are 6 prefix variables in the objective (some of which are used by multiple witnesses), 1 Query Plan constraint, and 6 Prefix constraints.

Finally, we define the objective to minimize the weighted sum of all 8 prefix variables in PV (here all weights are 1):

$$\begin{aligned} \text{len} = & p(y_1 \leftarrow x_1) + p[y_1 \leftarrow z_1] + p[y_1 \leftarrow z_1 \leftarrow u_1] + p[z_1 \leftarrow y_1 \leftarrow x_1] + p[z_1 \leftarrow y_1] + \\ & p[z_1 \leftarrow u_1] + p[y_1 \leftarrow z_1 \leftarrow u_2] + p[z_1 \leftarrow u_2] \end{aligned}$$

In our given database instance, len has an optimal value of 4 when the prefixes $p[z_1 \leftarrow y_1 \leftarrow x_1]$, $p[z_1 \leftarrow y_1]$, $p[z_1 \leftarrow u_1]$ and $p[z_1 \leftarrow u_2]$ are set to 1. This corresponds to the minimal factorization $r_{11}s_{11}(t_{11} + t_{12})$.

6 PTIME ALGORITHMS

We provide two PTIME algorithms: (1) a Max-Flow Min-Cut (MFMC) based approach and (2) an LP relaxation from which we obtain a rounding algorithm that gives a guaranteed $|\text{mveo}|$ -approximation for all instances. Interestingly, Section 8 will later show that both algorithms (while generally just approximations) give *exact answers* for all currently known PTIME cases of minFACT .

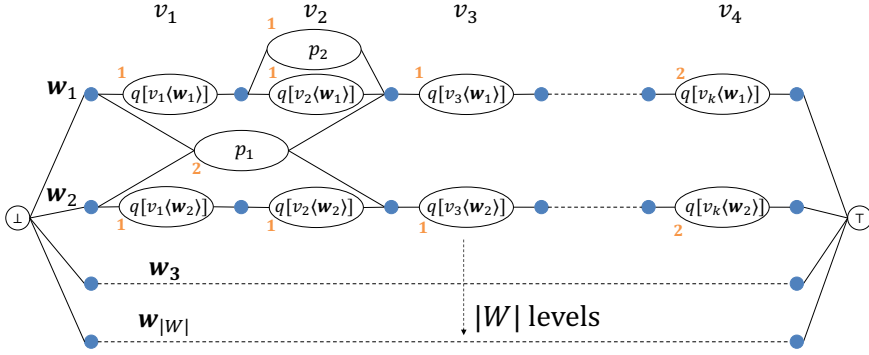


Fig. 6. A flow graph F for minFACT . The goal is to disconnect the source and the target nodes with minimum cuts. White q and p nodes can be cut and have capacities (in orange) equal to the weights of the corresponding variables in the ILP objective. Edges and connector nodes (in blue) have infinite capacity and cannot be cut.

6.1 MFMC-based Algorithm for minFACT

Given witnesses W and $\text{mveo}(Q)$, we describe the construction of a *factorization flow graph* F s.t. any *minimal cut* of F corresponds to a factorization of W . A minimal cut of a flow graph is the smallest set of nodes whose removal disconnects the source (\perp) and target (\top) nodes [62]. Since minimal cuts of flow graphs can be found in PTIME [18], we obtain is a PTIME approximation for minFACT .

6.1.1 Construction of a factorization flow graph We construct a flow graph F s.t. there exists a valid factorization of W of length $\leq c$ if the graph has a cut of size c . F is constructed by transforming decision variables in the ILP into "cut" nodes in the flow graph that may be cut at a penalty equal to their weight. Any valid cut of the graph selects nodes that fulfill all constraints of the ILP. We prove this by describing the construction of F (Fig. 6).

(1) *mveo order*: We use Ω to describe a total order on mveo (i.e. a total order on the set of minimal query plans). In Fig. 6, mveo is ordered by $\Omega = (v_1, v_2, \dots, v_k)$ where $k = |\text{mveo}|$.

(2) *QPV*: For each witness w , connect the query plan variables (QPV) as defined by Ω . Since all paths from source to target must be disconnected, at least one mveo must be cut from this path. Thus, F enforces the *Query Plan Constraints*.

(3) *PV*: For each witness w and prefix variable p , identify the first and last query variable for which p is a prefix and connect the corresponding prefix variable node to connector nodes before and after these query variables. For example, in Fig. 6, for w_2 , p_1 starts at $q[v_1\langle w_2 \rangle]$ and ends at $q[v_2\langle w_2 \rangle]$ implying that p_1 is a prefix for $q[v_1\langle w_2 \rangle]$ and $q[v_2\langle w_2 \rangle]$, but no query plan after that. Now if either of $q[v_1\langle w_2 \rangle]$ or $q[v_2\langle w_2 \rangle]$ are in the minimal cut, the graph is not disconnected until p_1 is added to the cut as well. These nodes guarantee that F enforces *Prefix Constraints*.

(4) *Weights*: Assign each q and p node in F the same weight as in the ILP objective. Recall that this weight is the number of tables with the same table prefix under the same VEO and that it helps calculate the correct factorization length.

Thus, a min-cut of F contains at least one plan for each witness, along with all the prefixes that are necessary for the plan. This guarantees a valid (although not necessarily minimal) factorization.¹¹

¹¹If a min-cut contains more than one plan for a witness, then one can pick either of the plans arbitrarily to obtain a valid factorization.

6.1.2 When is the MFMC-based algorithm optimal? In the previous subsection we saw that a min-cut of F always represents a valid factorization. However, the converse is not true: there can be *factorizations that do not correspond to a cut*. The reason is that *spurious constraints* might arise by the interaction of paths; those additional constraints no longer permit the factorization. There are two types of spurious paths:

(1) Spurious Prefix Constraints. Spurious prefix constraints arise when a prefix node p is in parallel with a query node q of which it is not a prefix. This happens when a q is not prefixed by p , but other query plans before and after are. To avoid this, the ordering Ω must be a *Running-Prefixes (RP) ordering*.¹²

Definition 6.1 (Running-Prefixes (RP) ordering). An ordering $\Omega = (q_1, q_2, \dots, q_k)$ is an RP Ordering and satisfies the RP-Property iff for any p that is a prefix for both q_i and q_j ($i < j$), p is a prefix for all q_k with $i \leq k \leq j$.

EXAMPLE 7 (RP ORDERING). Assume $\text{mveo} = \{(x \leftarrow y \leftarrow z), (x \leftarrow z \leftarrow y), (z \leftarrow y \leftarrow x)\}$. Then $\Omega_1 = ((x \leftarrow y \leftarrow z), (z \leftarrow y \leftarrow x), (x \leftarrow z \leftarrow y))$ is not an RP ordering since the 1st and 3rd VEO share prefix x , however the 2nd starts with z . In contrast, $\Omega_2 = ((x \leftarrow y \leftarrow z), (x \leftarrow z \leftarrow y), (z \leftarrow y \leftarrow x))$ is an RP ordering.

It turns out that for some queries RP-Orderings are impossible (such as Q_{6WE}° [43]). However, we are able to adapt our algorithm for such queries with a simple extension called *nested orderings*. We first define two query plans as *nestable* if each query plan can be “split” into paths from root to leaf such that they have an equal number of resulting paths, and that the resulting paths can be mapped to each other satisfying the property that corresponding paths use the same set of query variables. *Nested orderings* then are partial orders of query plans such that the pair of query plans may be uncomparable iff they are nestable. Finally, we define Nested RP-orderings as those such that all paths in the partial nested order satisfy the RP property. Intuitively, nested orderings add parallel paths for a single witness to model independent decisions. We can now prove that there always is an ordering that avoids spurious prefix constraints.

THEOREM 6.2 (RUNNING PREFIXES (RP) PROPERTY). *For any query, there is a simple or nested ordering Ω that satisfies the RP Property.*

(2) Spurious Query Constraints. Query Plan constraints are enforced by paths from source to target such that at least one node *from each path* must be chosen for a valid factorization. Due to sharing of prefix variables, these paths can interact and can lead to additional *spurious paths* that place additional *spurious constraints* on the query nodes. The existence of spurious query constraints does not necessarily imply that the algorithm is not optimal. In fact, Section 8 shows that Q_U^Δ and Q_4^∞ have spurious query paths, yet the min-cut is guaranteed to correspond to the minimal factorization. However, if the presence of spurious query paths prevents any of the minimal factorizations to be a min-cut for F , then we say that the factorization flow graph F has “leakage” [43]. Since all paths along one witness are cut by construction, a leakage path must contain nodes from at least two different witnesses.

Definition 6.3 (Leakage). Leakage exists in a factorization flow graph if no minimal factorization is a valid cut of the graph. A leakage path is a path from source to target such that a valid minimal factorization is possible without using any node on the path.

Optimality of algorithm. Thus, a solution found by the MFMC-based algorithm is guaranteed to be optimal if it has two properties:

¹²Notice that this concept is reminiscent of the *running intersection* property [4, 6] and the *consecutive ones* property [10].

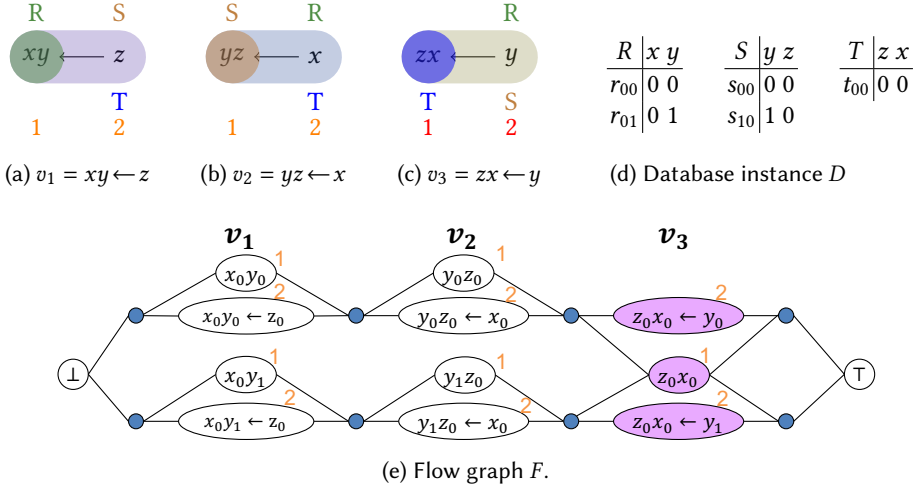


Fig. 7. **Example 8:** Three mveo's for triangle query $Q^\Delta := R(x, y), S(y, z), T(z, x)$ (a)-(c), example database instance D (d), and constructed flow graph for mveo order $\Omega = [v_1, v_2, v_3]$ (e). Notice that several variables may appear in the same node of a mveo (e.g., x and y in $xy \leftarrow z$).

- (1) The ordering Ω is a *Running-Prefixes ordering* or a *nested Running-Prefixes ordering* (always possible).
- (2) There is no *leakage* in the flow graph (not always possible).

We use these properties in [Section 7](#) and [Section 8](#) to prove a number of queries to be in PTIME. In fact, all currently known PTIME cases can be solved exactly with the MFMC-based algorithm via a query-dependent ordering of the mveos.

EXAMPLE 8 (FLOW GRAPH CONSTRUCTION FOR TRIANGLE QUERY). Consider the triangle query $Q^\Delta := R(x, y), S(y, z), T(z, x)$. The query has 3 minimal Query Plans corresponding to mveos shown in [Figs. 7a to 7c](#). The provenance of Q^Δ over the database shown in [Fig. 7d](#), has 2 witnesses: $W = \{r_{00}s_{00}t_{00}, r_{01}s_{10}t_{00}\}$. We build a flow graph to find a factorization. (1) We choose $\Omega = (v_1, v_2, v_3)$ as linear order for the mveo. (2) For each witness, we connect their three query plan variables $q[v(\mathbf{w})]$ in this order serially from source to target. (3) In Q^Δ , each mveo has a single prefix. We attach these variables in parallel to their corresponding query variables. Notice that the prefix z_0x_0 is shared by both \mathbf{w}_1 and \mathbf{w}_2 , and therefore is attached in parallel to both corresponding query variables. (4) Finally, we add weights corresponding to the number of tables having each prefix (see [Figs. 7a to 7c](#)).

The resulting flow graph is shown in [Fig. 7e](#). The min-cut (highlighted in purple) consists of the nodes $\{z_0x_0 \leftarrow y_0, z_0x_0, z_0x_0 \leftarrow y_1\}$. The corresponding factorization using the selected query plans is $t_{00}(r_{00}s_{00} \vee r_{01}s_{10})$. The weighted cut-value (5) is equal to the length of the factorization. This factorization is minimal.

6.2 LP relaxation for minFACT and an LP relaxation-based approximation

Linear Programming relaxation and rounding is a commonly-used technique to find PTIME approximations for NP-C problems [61]. The LP relaxation for minFACT simply removes the integrality constraints on all the problem variables. The LP relaxation may pick multiple query plans for a given witness, each with fractional values. We present a rounding scheme for minFACT and show it to be a $|\text{mveo}|$ -factor approximation of the optimal solution. The rounding algorithm simply picks

the maximum fractional value of query plan variables for each witness, breaking ties arbitrarily. Finally, it counts only the prefix variables necessitated by the chosen query plans.

THEOREM 6.4. *The described rounding scheme gives a PTIME, $|mveo|$ -factor approximation for minFACT.*

6.2.1 When is the LP relaxation optimal? Experimentally, we observed that the LP solution of many queries are equal to the integral ILP solution. This is surprising since the ILP does not satisfy any of the known requirements for tractable ILPs such as Total Unimodularity, Balanced Matrices, or even Total Dual Integrality [55]. Interestingly, we next prove that the LP relaxation has the same objective value as the original ILP whenever the MFMC-based algorithm is optimal.

LEMMA 6.5. *If all database instances can be solved exactly by the MFMC-based algorithm for a given query Q (i.e. for each database instance there exists an ordering that generates a leakage-free graph), then the LP relaxation of minFACT always has the same objective as the original ILP.*

This result is important as it exposes cases for which the optimal objective value of minFACT_ILP is identical to the optimal objective value of a simpler LP relaxation. The only example we know of where this has been shown using flow graphs is in recent work on resilience [44]. In such cases, standard ILP solvers return the optimal solution to the original ILP in PTIME. This is due to ILP solvers using an LP-based branch and bound approach which starts by computing the LP relaxation bound and then exploring 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 is done [36]. We use this knowledge in Section 8 to show that ILP solvers can solve all known PTIME cases in PTIME.

7 RECOVERING READ-ONCE INSTANCES

It is known that the read-once factorization of a read-once instance can be found in PTIME with specialized algorithms [31, 54, 56]. We prove that our more general MFMC based algorithm and LP relaxation are *always guaranteed to find read-once formulas* when they exist, even though they are not specifically designed to do so.

THEOREM 7.1 (READ-ONCE). *minFACT can be found in PTIME by (a) the MFMC based algorithm, and (b) the LP relaxation, for any query and database instance that permits a read-once factorization.*

8 TRACTABLE QUERIES FOR minFACT

We now go beyond cases when PROB is in PTIME (i.e. read-once instances). We first prove that minFACT(Q) is PTIME for the large class of queries with 2 minimal query plans. We then show examples of queries with 3 and 5 minimal query plans that are PTIME as well. All these newly recovered PTIME cases, along with the previously known read-once cases, can be solved *exactly* with both our PTIME algorithms from Section 6. Finally, we hypothesize that minFACT is in PTIME for any linear query.

8.1 All queries with ≤ 2 minimal query plans

We prove that our MFMC-based algorithm has no leakage and thus always finds the minimal factorization for queries with at most 2 minimal VEOs (2-MQP) queries such as Q_2^* and Q_3^∞ . We also give an alternative proof that shows that any ILP generated by such a query is guaranteed to have a *Totally Unimodular (TU)* constraint matrix, and thus is PTIME solvable [55].

THEOREM 8.1 (2-MQP QUERIES). *minFACT can be found in PTIME for any query with max 2 minimal VEOs by (a) the MFMC based algorithm, and (b) the LP relaxation.*

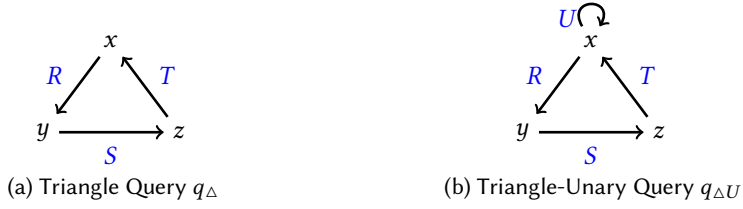


Fig. 8. We show that Q^{Δ} is hard in Section 9 because it contains an “active triad.” Surprisingly, for Q_U^{Δ} , a query that differs by a single unary relation, the minimal factorization can always be found in PTIME by either using our MFMC based algorithm from Section 6.1 or our LP relaxation from Section 6.2.

The theorem recovers the hierarchical queries which are equivalent to 1-MQP queries since they have one “safe plan” [17]. The PTIME nature of 1-MQP queries also follows from Theorem 7.1, as all hierarchical queries have read-once formulations.

COROLLARY 8.1. *minFACT for Hierarchical Queries is in PTIME.*

COROLLARY 8.2. *The classes of queries for which minFACT is in PTIME is a strict super-class of those for which probabilistic query evaluation is in PTIME (if $P \neq NP$).*

8.2 Two queries with ≥ 3 minimal query plans

Triangle-unary Q_U^{Δ} . Q_U^{Δ} is structurally similar to Q^{Δ} (Fig. 8) and both have $|\text{mveo}| = 3$. However, while Q^{Δ} contains an “active triad” [44] and is hard, we show that Q_U^{Δ} is in PTIME by proving that the factorization flow graph has no leakage. Interestingly, Q_U^{Δ} ’s ILP is *not guaranteed to have a TU* constraint matrix, yet the MFMC algorithm is optimal, and the LP relaxation recovers the minimal ILP objective, showing that PTIME cases extend beyond Total Unimodularity of the ILP constraint matrix.

THEOREM 8.2 (Q_U^{Δ} IS EASY). *minFACT(Q_U^{Δ}, D) can be found in PTIME for any database D by (a) the MFMC based algorithm, and (b) the LP relaxation.*

4-chain Q_4^{∞} . This is arguably the most involved proof in the paper. Q_4^{∞} has $|\text{mveo}| = 5$. Yet in a similar proof to Q^{Δ} , we can show that the MFMC-based algorithm and the LP are both optimal. This surprising result leads to the conjecture that minFACT for longer chains, and all linear queries are in PTIME.

THEOREM 8.3 (Q_4^{∞} IS EASY). *minFACT(Q_4^{∞}, D) can be found in PTIME for any database D by (a) the MFMC based algorithm, and (b) the LP relaxation.*

8.3 Conjecture for Linear Queries

A query is acyclic if it has a join tree, i.e. it permits a placement of its atoms into a tree s.t. for any two atoms, the intersection of variables is contained in the union of the variables of the atoms on the unique path between them.¹³ A query is *linear* if it permits a join path.¹⁴ We have spent a lot of time trying to prove the hardness of such queries without success. Based on our intuition we hypothesize that *all linear queries are in PTIME*. Our intuition is strengthened by the fact that over many experimental evaluations, the LP relaxation of the minFACT_ILP was always integral and optimal, thus being able to solve the problem in PTIME.

¹³The concept is alternatively called coherence, the running intersection property, connected subgraph property [4, 6, 58], and is used in the definition of the junction tree algorithm [42] and tree decompositions [21, 53].

¹⁴This definition, introduced in [44], is more restrictive than linear queries defined in the original work on resilience [24] as it does not allow *linearizable queries* (those that can be made linear by “making *dominated* atoms exogenous”).

HYPOTHESIS 1 (PTIME CONJECTURE). *If Q is a linear query, then $\text{minFACT}(Q, D)$ can be found in PTIME for any database D .*

We think that additional insights from optimization theory are needed to explain the integrality of the solution to the LP relaxation and to thus prove this conjecture. We leave open the structural criterion that separates the easy and hard cases.

9 HARD QUERIES FOR minFACT

In this section, we first prove that all queries that contain a structure called “an active triad” (e.g. Q_3^\star and Q^Δ) are NP-C. We then prove another query to be NP-C that does not contain an active triad, but a “co-deactivated triad.” We thus show that while active triads are sufficient for FACT of a query to be NP-C, they are not necessary, and minFACT is a strictly harder problem than RES.

Queries with Active triads. We repeat here the necessary definitions introduced in the context of resilience under bag semantics [44]. A *triad* is a set of three atoms, $\mathcal{T} = \{R_1, R_2, R_3\}$ s.t. for every pair $i \neq j$, there is a path from R_i to R_j that uses no variable occurring in the third atom of \mathcal{T} . Here a *path* is an alternating sequence of relations and variables $R_1 - \mathbf{x}_1 - R_2 - \dots - \mathbf{x}_{p-1} - R_p$ s.t. all adjacent relations R_i, R_{i+1} share variables \mathbf{x}_i . In a query Q with atoms R and S , we say R *dominates* S iff $\text{var}(R) \subset \text{var}(S)$. We call an atom g in a query *independent* iff there is no other atom in the query that contains a strict subset of its variables (and hence it is not dominated). A triad is *active* iff none of its atoms are dominated.

THEOREM 9.1 (ACTIVE TRIADS ARE HARD). *FACT(Q) for a query Q with an active triad is NP-C.*

Separation between RES and minFACT . A triad is *deactivated* if any of the three atoms is dominated. A triad is *co-deactivated* if all three atoms are dominated only by the same (non-empty) set of atoms. The co-deactivated triangle query $Q_{\text{cod}}^\Delta := A(w), R(w, x, y), S(w, y, z), T(w, z, x)$ contains no active triads: notice that the tables R, S and T are not independent and have no independent paths to each other. Thus, $\text{RES}(Q_{\text{cod}}^\Delta)$ is PTIME. However, Q_{cod}^Δ contains a co-deactivated triad since R, S and T are all dominated only by atom A . We next prove that $\text{FACT}(Q_{\text{cod}}^\Delta)$ is NP-C, thus showing a strict separation in the complexities of the two problems.

THEOREM 9.2 (CO-DEACTIVATED TRIADS ARE HARD). *FACT(Q) for a query Q with a co-deactivated triad is NP-C.*

10 CONCLUSION

We propose an ILP framework for minimizing the size of provenance polynomials for sj-free CQs. We show that our problem is NP-C and thus in a lower complexity class than the general Minimum Equivalent Expression (MEE) problem. Key to our formulation is a way to systematically constrain a space of possible minimum factorizations thus allowing us to build an ILP, and connecting minimal variable elimination orders to minimal query plans developed in the context of probabilistic databases. We complement our hardness results with two unified PTIME algorithms that can recover *exact solutions* to a strict superset of *all prior known tractable cases*.

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] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2017. What Do Shannon-Type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?. In *PODS*. 429–444. <https://doi.org/10.1145/3034786.3056105>

- [2] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. 2008. Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table. *SIAM J. Comput.* 38, 1 (2008), 63–84. <https://doi.org/10.1137/060664537>
- [3] Yael Amsterdamer, Daniel Deutch, Tova Milo, and Val Tannen. 2012. On Provenance Minimization. *ACM Trans. Database Syst.* 37, 4, Article 30 (dec 2012), 36 pages. <https://doi.org/10.1145/2389241.2389249>
- [4] 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>
- [5] Christoph Berkholz and Harry Vinnal-Smeeth. 2023. A Dichotomy for Succinct Representations of Homomorphisms. In *ICALP (LIPIcs, Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). 113:1–113:19. <https://doi.org/10.4230/LIPIcs.ICALP.2023.113>
- [6] Philip A. Bernstein and Nathan Goodman. 1981. Power of Natural Semijoins. *SIAM J. Comput.* 10, 4 (1981), 751–771. <https://doi.org/10.1137/0210059>
- [7] David Buchfuhrer and Christopher Umans. 2011. The complexity of Boolean formula minimization. *J. Comput. System Sci.* 77, 1 (2011), 142–153. <https://doi.org/10.1016/j.jcss.2010.06.011>
- [8] 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>
- [9] Vinicius Callegaro, Mayler GA Martins, Renato P Ribas, and André I Reis. 2013. Read-polarity-once Boolean functions. In *2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 1–6. <https://doi.org/10.1109/SBCCI.2013.6644862>
- [10] Zixuan Chen, Subhdeep Mitra, R Ravi, and Wolfgang Gatterbauer. 2024. HITSNDIFFS: From Truth Discovery to Ability Discovery by Recovering Matrices with the Consecutive Ones Property. In *ICDE*. <https://arxiv.org/pdf/2401.00013>
- [11] 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>
- [12] 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>
- [13] 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](https://doi.org/10.1016/S0166-218X(01)00344-4)
- [14] Yves Crama and Peter L. Hammer. 2011. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press. <https://doi.org/10.1017/cbo9780511852008.003>
- [15] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. 2000. Tracing the lineage of view data in a warehousing environment. *ACM TODS* 25, 2 (2000), 179–227. <https://doi.org/10.1145/357775.357777>
- [16] Nilesh N. Dalvi and Dan Suciu. 2004. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*. 864–875. <https://doi.org/10.1016/b978-012088469-8.50076-0>
- [17] 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>
- [18] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. 2008. *Algorithms*. McGraw-Hill Higher Education, Boston. <http://www.loc.gov/catdir/enhancements/fy0665/2006049014-t.html>
- [19] 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>
- [20] Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113, 1 (1999), 41–85. [https://doi.org/10.1016/S0004-3702\(99\)00059-4](https://doi.org/10.1016/S0004-3702(99)00059-4)
- [21] Rina Dechter. 2003. *Constraint Processing*. Morgan Kaufmann. <https://doi.org/10.1016/b978-1-55860-890-0.x5000-2>
- [22] Maarten Van den Heuvel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerts, and Martin Theobald. 2019. Anytime Approximation in Probabilistic Databases via Scaled Dissociations. In *SIGMOD*. 1295–1312. <https://doi.org/10.1145/3299869.3319900>
- [23] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. *VLDB J.* 22, 6 (2013), 823–848. <https://doi.org/10.1007/s00778-013-0310-5>
- [24] 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. <https://doi.org/10.14778/2850583.2850592>
- [25] 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>
- [26] Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. W. H. Freeman & Co. <https://dl.acm.org/doi/10.5555/578533>
- [27] Wolfgang Gatterbauer and Dan Suciu. 2014. Oblivious bounds on the probability of Boolean functions. *TODS* 39, 1 (2014), 1–34. <https://doi.org/10.1145/2532641>

- [28] 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>
- [29] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. 2008. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Information and Computation* 206, 6 (2008), 760–775. <https://doi.org/10.1016/j.ic.2008.03.002>
- [30] Martin Charles Golumbic and Vladimir Gurvich. 2010. *Read-once functions*. Cambridge University Press, Chapter 10. <https://doi.org/10.1017/cbo9780511852008.011>
- [31] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. 2006. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Applied Mathematics* 154, 10 (2006), 1465–1477. <https://doi.org/10.1016/j.dam.2005.09.016>
- [32] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. 2008. An improvement on the complexity of factoring read-once Boolean functions. *Discrete Applied Mathematics* 156, 10 (2008), 1633–1636. <https://doi.org/10.1016/j.dam.2008.02.011>
- [33] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *PODS*. 31–40. <https://doi.org/10.1145/1265530.1265535>
- [34] Todd J. Green and Val Tannen. 2017. The Semiring Framework for Database Provenance. In *PODS*. 93–99. <https://doi.org/10.1145/3034786.3056125>
- [35] LLC Gurobi Optimization. 2021. Gurobi Optimizer Reference Manual. <http://www.gurobi.com>
- [36] LLC Gurobi Optimization. 2021. Mixed-Integer Programming (MIP) – A Primer on the Basics. <https://www.gurobi.com/resource/mip-basics/>
- [37] V.A. Gurvich. 1977. Repetition-free Boolean functions. *Uspekhi Mat. Nauk* 32 (1977), 183–184. <http://mi.mathnet.ru/umn3055> (in Russian).
- [38] Peter L Hammer and Alexander Kogan. 1993. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence* 64, 1 (1993), 131–145. [https://doi.org/10.1016/0004-3702\(93\)90062-G](https://doi.org/10.1016/0004-3702(93)90062-G)
- [39] Edith Hemaspaandra and Henning Schnoor. 2011. Minimization for generalized boolean formulas. In *21st International Joint Conference on Artificial Intelligence (IJCAI)*. 566–571. <https://doi.org/10.1109/sfcs.1997.646147>
- [40] Rahul Ilango. 2022. The Minimum Formula Size Problem is (ETH) Hard. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 427–432. <https://doi.org/10.1109/FOCS52979.2021.00050>
- [41] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. 2011. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press. <https://doi.org/10.1017/cbo9780511977152.002>
- [42] S. L. Lauritzen and D. J. Spiegelhalter. 1988. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society. Series B (Methodological)* 50, 2 (1988), 157–224. <http://www.jstor.org/stable/2345762>
- [43] Neha Makhija and Wolfgang Gatterbauer. 2021. Minimally Factorizing the Provenance of Self-join Free Conjunctive Queries. (2021). <https://arxiv.org/abs/2105.14307> (Full technical report with Online Appendix).
- [44] Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *PACMOD* 1, 4 (dec 2023), 228:1–228:27. <https://doi.org/10.1145/3626715>
- [45] Mayler GA Martins, Leomar Rosa, Anders B Rasmussen, Renato P Ribas, and Andre I Reis. 2010. Boolean factoring with multi-objective goals. In *International Conference on Computer Design (ICCD)*. IEEE, 229–234. <https://doi.org/10.1109/ICCD.2010.5647772>
- [46] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. 2011. Reverse Data Management. *PVLDB* 4, 12 (2011), 1490–1493. <https://doi.org/10.14778/3402755.3402803>
- [47] Aviad Mintz and Martin Charles Golumbic. 2005. Factoring Boolean functions using graph partitioning. *Discrete Applied Mathematics* 149, 1-3 (2005), 131–153. <https://doi.org/10.1016/j.dam.2005.02.007>
- [48] Dan Olteanu and Jiewen Huang. 2008. Using OBDDs for Efficient Query Evaluation on Probabilistic Databases. In *SUM*. 326–340. https://doi.org/10.1007/978-3-540-87993-0_26
- [49] Dan Olteanu and Maximilian Schleich. 2016. Factorized Databases. *SIGMOD Rec.* 45, 2 (2016), 5–16. <https://doi.org/10.1145/3003665.3003667>
- [50] Dan Olteanu and Jakub Závodný. 2011. On Factorisation of Provenance Polynomials. In *3rd Workshop on the Theory and Practice of Provenance (TaPP’11)*. USENIX. <https://www.usenix.org/conference/tapp11/factorisation-provenance-polynomials>
- [51] Dan Olteanu and Jakub Závodný. 2012. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*. 285–298. <https://doi.org/10.1145/2274576.2274607>
- [52] Dan Olteanu and Jakub Závodný. 2015. Size Bounds for Factorised Representations of Query Results. *ACM Trans. Database Syst.* 40, 1 (2015), 2:1–2:44. <https://doi.org/10.1145/2656335>
- [53] Neil Robertson and Paul D. Seymour. 1986. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms* 7, 3 (1986), 309–322. [https://doi.org/10.1016/0196-6774\(86\)90023-4](https://doi.org/10.1016/0196-6774(86)90023-4)

- [54] Sudeepa Roy, Vittorio Perduca, and Val Tannen. 2011. Faster query answering in probabilistic databases using read-once functions. In *ICDT*. 232–243. <https://doi.org/10.1145/1938551.1938582>
- [55] Alexander Schrijver. 1998. *Theory of linear and integer programming*. John Wiley & Sons. <https://doi.org/10.1137/1030065>
- [56] Prithviraj Sen, Amol Deshpande, and Lise Getoor. 2010. Read-Once Functions and Query Evaluation in Probabilistic Databases. *PVLDB* 3, 1 (2010), 1068–1079. <https://doi.org/10.14778/1920841.1920975>
- [57] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool. <https://doi.org/10.2200/s00362ed1v01y201105dtm016>
- [58] Robert Endre Tarjan and Mihalis Yannakakis. 1984. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM J. Comput.* 13, 3 (1984), 566–579. <https://doi.org/10.1137/0213035>
- [59] Christopher Umans. 2001. The minimum equivalent DNF problem and shortest implicants. *J. Comput. System Sci.* 63, 4 (2001), 597–611. <https://doi.org/10.1109/sfcs.1998.743506>
- [60] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*. 137–146. <https://doi.org/10.1145/800070.802186>
- [61] Vijay V Vazirani. 2001. *Approximation algorithms*. Vol. 1. Springer. <https://dl.acm.org/doi/10.5555/500776>
- [62] David P Williamson. 2019. *Network flow algorithms*. Cambridge University Press. <https://doi.org/10.1017/9781316888568>

A NOMENCLATURE AND CONVENTIONS

Symbol	Definition
Q	a self-join free Boolean CQ
R, S, T, U	relational tables
r_i, s_i, t_i, u_i	tuple identifiers
x, y, z	query variables
m	number of atoms in a query
N	size of database $ D $
φ, ψ	propositional formulas / expressions
$\text{var}(X)$	the set of variables in atom / relation / formula X
W	set of witnesses $W = \text{witnesses}(Q, D)$
\mathbf{w}	witness
$\text{VEO}(Q)$	set of all legal VEOs for Q
$\text{mveo}(Q)$	set of minimal VEOs for Q
$k = \text{mveo}(Q) $	number of minimal VEOs
$v(\mathbf{w})$	a VEO instance of VEO v over witness \mathbf{w}
$\text{var}(g_i)$	set of variables of a query q or atom g_i
P	query plan
\mathcal{P}	set of plans
F	flow graph
$\bowtie(\dots)$	provenance join operator in prefix notation
$\pi_{\mathbf{x}}, \pi_{\neg \mathbf{y}}$	provenance project operators: onto \mathbf{x} , or project \mathbf{y} away
\mathbf{x}	unordered set or ordered tuple
\mathbf{a}/\mathbf{x}	substitute values \mathbf{a} for variables \mathbf{x}
$Q[\mathbf{x}]$	indicates that \mathbf{x} represents the set of all existentially quantified variables for Boolean query Q
len	Length of a Factorization
QPV	Query Plan Variables of an ILP
PV	Prefix Variables of an ILP
$q[\dots]$	a ILP decision query plan variable
$p[\dots]$	a ILP decision prefix variable
c	weight (or cost) of variables in the ILP / nodes in the Factorization Flow Graph
Ω	An Ordering of mveo chosen for MFMC based algorithm
(v_1, v_2, \dots, v_k)	An ordered list of VEOs, VEOFFs or any other set of objects

Query	Definition
Q_2^∞	2-chain query $R(x, y), S(y, z)$
Q_3^∞	3-chain query $R(x, y), S(y, z), T(z, u)$
Q_4^∞	4-chain query $P(u, x), R(x, y), S(y, z), T(z, v)$
Q_5^∞	5-chain query $L(a, u), P(u, x), R(x, y), S(y, z), T(z, v)$
Q_2^\star	2-star query $R(x)S(y), W(x, y)$
Q_3^\star	3-star query $R(x)S(y), T(z)W(x, y, z)$
Q^Δ	Triangle query $R(x, y)S(y, z), T(z, x)$
Q_U^Δ	Triangle-unary query $U(x)R(x, y)S(y, z), T(z, x)$
Q_{6WE}^∞	6-cycle query with end points $A(x), R(x, y), B(y), S(y, z), C(z), T(z, u) D(u), U(u, v), E(v), V(v, w), F(w), W(w, x)$
Q_{cod}^Δ	Co-dominated triangle query $A(w), R(w, x, y), S(w, y, z), T(w, z, x)$

We write $[k]$ as short notation for the set $\{1, \dots, k\}$ and use boldface to denote tuples or ordered sets, (e.g., $\mathbf{x} = (x_1, \dots, x_\ell)$). We fix a relational vocabulary $\mathbf{R} = (R_1, \dots, R_m)$, and denote with

$\text{arity}(R_i)$ the number of attributes of a relation R_i . For notational convenience, we assume w.l.o.g. that there are no two atoms R_i and R_j with $\text{var}(R_i) = \text{var}(R_j)$. A database instance over \mathbf{R} is $D = (R_1^D, \dots, R_m^D)$, where each R_i^D is a finite relation. We call the elements of R_i^D tuples and write R_i instead of R_i^D when D is clear from the context. With some abuse of notation we also denote D as the set of all tuples, i.e. $D = \bigcup_i R_i$. The active domain $\text{dom}(D)$ is the set of all constants occurring in D . W.l.o.g., we commonly use $\text{dom}(D) \subset \mathbb{N} \cup \{a, b, \dots, z\}$. The size of the database instance is $n = |D|$, i.e. the number of tuples in the database.¹⁵

B ADDITIONAL DETAILS ON Section 2: RELATED WORK

B.1 Boolean Factorization

Fig. 9 illustrates the landscape of known results for the problem of Minimum Equivalent Expressions (MEE) applied to formulas.

The general problem of MEE has been long known to be NP-hard [26]. However, only relatively recently it has been proved to be Σ_p^2 -complete [7]. Various important classes of this problem have been studied, a fundamental one being the factorization of DNF expressions. The MinDNF problem [59], deals with finding the minimum equivalent DNF expression of an input DNF formula, and is also known to be Σ_p^2 -complete. However, if the input to the MinDNF is the truth table (or set of all true assignments of the formula) then the problem is NP-C [2]. If we take away the restriction that the factorized formula must be a DNF, then the problem of finding the minimum factorization of an input table is known as the Minimum Formula Size Problem (MFSP) and is shown to be in NP and (ETH)-hard [40].

Another important class of restrictions is over monotone formulas (thus we do not allow negatives in input or output formulas). Surprisingly, we do not know of any work that proves the complexity of the general monotone boolean factorization problem. However, there are many interesting and important restrictions for which complexity results are known. One such important sub-class is that of read-once formulas, which can be factorized in PTIME [32]. For Monotone formulas with DNF input and output restrictions, the problem can be solved in logspace by eliminating monomials [29]. Interestingly the problem monotone formula factorization of an arbitrary formula with a DNF restriction on the output only has differing complexity based on the input encoding of the length of the factorization. Checking if the minimum size of a DNF for a monotone formula is at most k is PP-complete, but for k in unary, the complexity of the problem drops to coNP [29]. The intuition is that in this problem, (which can be seen as “dual” of minFACT since it has a DNF output restriction instead of a DNF input restriction), the optimal output (a DNF) can be exponentially larger than the input (any monotone formula).

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

B.2 Probabilistic Inference and Dissociation

Given a provenance that is not read-once, one can still upper and lower bound its probability efficiently via dissociation [27]: Let φ and φ' be two Boolean formulas with variables \mathbf{x} and \mathbf{x}' , respectively. Then φ' is a *dissociation* of φ if there exists a substitution $\theta : \mathbf{x}' \rightarrow \mathbf{x}$ s.t. $\varphi'[\theta] = \varphi$. If $\theta^{-1}(x) = \{x'_1, \dots, x'_d\}$, then variable x dissociates into d variables x'_1, \dots, x'_d . Every provenance expression has a unique read-once dissociation up to renaming of variables. One application of

¹⁵Notice that other work sometimes uses $|\text{dom}(D)|$ as the size of the database. Our different definition has no implication on our complexity results but simplifies the discussions of our reductions.

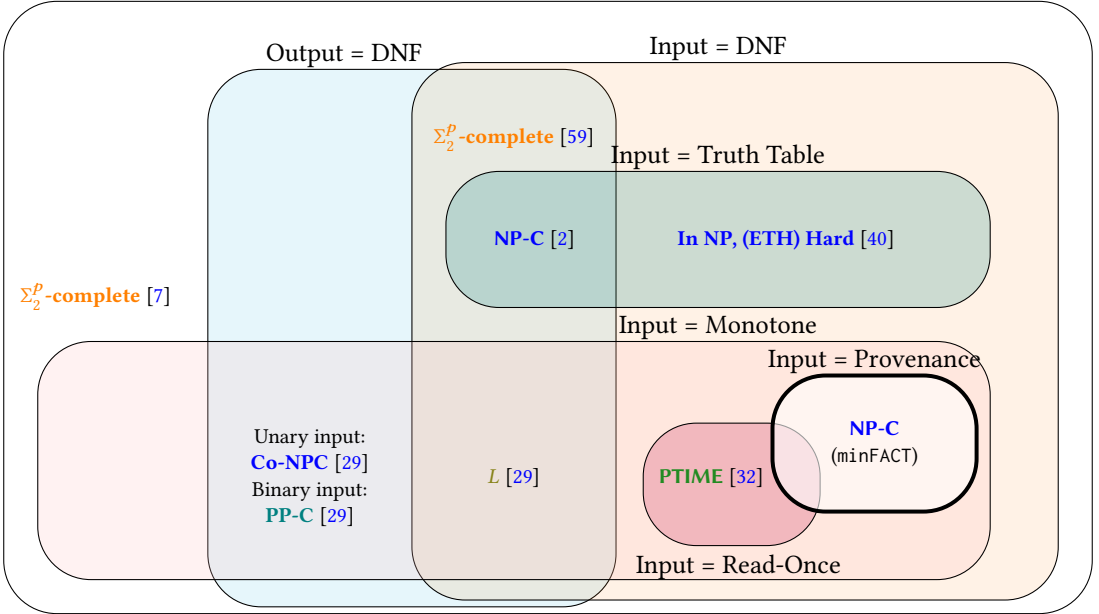


Fig. 9. An overview of related work on the Exact, Minimal Equivalent Expression (MEE) problem applied to formulas.

compiling provenance polynomials into their smallest representation is motivated by the following known results on “oblivious bounds” [27]: (i) lower and upper bounds for intractable expressions can be found very efficiently; and (ii) those bounds work better the fewer times variables are repeated. Similarly, anytime approximation schemes based on branch-and-bound provenance decomposition methods [22, 23] give tighter bounds if Shannon expansions need to be run on fewer variables.

B.3 Resilience

The resilience of a Boolean query measures the minimum number of tuples in database D , the removal of which makes the query false. The optimization version of this decision problem is then: given Q and D , find the *minimum* k so that $(D, k) \in \text{RES}(Q)$. A larger k implies that the query is more “resilient” and requires the deletion of more tuples to change the query output. We know from [24] that all hard queries must have a “triad” which is a set of three non-dominated 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 other atom of \mathcal{T} . In return, the tractable queries for RES are exactly those that are triad-free.

EXAMPLE 9 (RESILIENCE). *The resilience for our example from Fig. 2 is 2 because removing the set $\Gamma = \{r_1, t_3\}$ removes one tuple from each witness and there is no smaller set that achieves that.*

Received December 2023; revised February 2024; accepted March 2024