A Unifying Algorithm for Hierarchical Queries

MAHMOUD ABO KHAMIS, RelationalAI, USA
JESSE COMER, University of Pennsylvania, USA
PHOKION G. KOLAITIS, UC Santa Cruz & IBM Research, USA
SUDEEPA ROY, Duke University, USA
VAL TANNEN, University of Pennsylvania, USA

The class of hierarchical queries is known to define the boundary of the dichotomy between tractability and intractability for the following two extensively studied problems about self-join free Boolean conjunctive queries (SJF-BCQ): (i) evaluating a SJF-BCQ on a tuple-independent probabilistic database; (ii) computing the Shapley value of a fact in a database on which a SJF-BCQ evaluates to true. Here, we establish that hierarchical queries define also the boundary of the dichotomy between tractability and intractability for a different natural algorithmic problem, which we call the *bag-set maximization* problem. The bag-set maximization problem associated with a SJF-BCQ Q asks: given a database $\mathcal{D}$, find the biggest value that Q takes under bag semantics on a database $\mathcal{D}'$ obtained from $\mathcal{D}$ by adding at most θ facts from another given database $\mathcal{D}^r$.

For non-hierarchical queries, we show that the bag-set maximization problem is an NP-complete optimization problem. More significantly, for hierarchical queries, we show that all three aforementioned problems (probabilistic query evaluation, Shapley value computation, and bag-set maximization) admit a single unifying polynomial-time algorithm that operates on an abstract algebraic structure, called a *2-monoid*. Each of the three problems requires a different instantiation of the 2-monoid tailored for the problem at hand.

CCS Concepts: • **Information systems → Relational database model**; • **Theory of computation → Database query processing and optimization (theory)**; **Data provenance**.

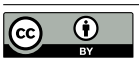
Additional Key Words and Phrases: hierarchical queries; dichotomy; bag-set semantics; database repair; probabilistic databases; Shapley values

ACM Reference Format:
Mahmoud Abo Khamis, Jesse Comer, Phokion G. Kolaitis, Sudeepa Roy, and Val Tannen. 2025. A Unifying Algorithm for Hierarchical Queries. *Proc. ACM Manag. Data* 3, 5 (PODS), Article 274 (November 2025), 22 pages. https://doi.org/10.1145/3767710

1 Introduction

Dichotomy theorems are classification results that pinpoint the computational complexity of every problem in a collection of algorithmic problems of interest by showing that some of these problems are tractable, while all others are complete for some complexity class, such as NP or #P. It is well known that the existence of a dichotomy for a collection of algorithmic problems cannot be taken for granted a priori. Indeed, Ladner [14] showed that if P ≠ NP, then there are problems in NP that are neither in P nor NP-complete. Thus, unless P = NP, no dichotomy theorem for NP exists.

Authors' Contact Information: Mahmoud Abo Khamis, mahmoudabo@gmail.com, RelationalAI, Berkeley, California, USA; Jesse Comer, jacomer@seas.upenn.edu, University of Pennsylvania, Philadelphia, Pennsylvania, USA; Phokion G. Kolaitis, kolaitis@ucsc.edu, UC Santa Cruz & IBM Research, Almaden, California, USA; Sudeepa Roy, sudeepa@cs.duke.edu, Duke University, Durham, North Carolina, USA; Val Tannen, val@seas.upenn.edu, University of Pennsylvania, Philadelphia, Pennsylvania, USA.



ACM 2836-6573/2025/11-ART274
https://doi.org/10.1145/3767710

A Boolean conjunctive query (BCQ) Q is called *self-join-free* if no two atoms in Q share the same relation symbol; we will use the abbreviation SJF-BCQ for the term *self-join free Boolean conjunctive query*. There is extensive earlier work on dichotomy theorems for two different collections of algorithmic problems parameterized by SJF-BCQs: (i) evaluating a SJF-BCQ on a tuple-independent probabilistic database; (ii) computing the Shapley value of a fact in a database on which a SJF-BCQ evaluates to true. We now describe these two problems in more detail. Note that since all problems studied in this paper are parameterized by a SJF-BCQ Q, we always assume that the query Q is fixed, hence the size of the query is not used in measuring the complexity of the problem. Thus, we use *data complexity* as our complexity measure throughout this paper.

Probabilistic Query Evaluation: Every fixed SJF-BCQ Q gives rise to the following problem: Given a *tuple-independent* probabilistic database $\mathcal{D}$ (which means that each fact in $\mathcal{D}$ is viewed as an independent event with a given probability), compute the *marginal probability* of Q on $\mathcal{D}$, i.e., compute the sum of the probabilities of the databases $\mathcal{D}'$ that satisfy Q and are obtained from $\mathcal{D}$ by either keeping or removing each fact of $\mathcal{D}$ independently.

Shapley Value Computation: Every fixed SJF-BCQ Q gives rise to the following problem: Given a database $\mathcal{D}$ on which Q evaluates to true and given a fact f of $\mathcal{D}$, compute the *Shapley value* of f, which, informally, is the "contribution" of f to Q evaluating to true on $\mathcal{D}$. Formally, the Shapley value of f is defined via the following process: Take a permutation of the facts in $\mathcal{D}$ uniformly at random, and insert the facts one by one while re-evaluating Q after each insertion; the Shapley value of f is the probability that the insertion of f flips the answer of Q from false to true.

Consider the collection of all SJF-BCQs. Dalvi and Suciu [5] showed that Probabilistic Query Evaluation exhibits the following dichotomy: for every fixed SJF-BCQ Q, either this problem is solvable in polynomial time or it is #P-complete. More recently, Livshits, Bertossi, Kimelfeld, and Sebag [16] showed that Shapley Value Computation exhibits the following dichotomy: for every fixed SJF-BCQ Q, either this problem is solvable in polynomial time or it is $\mathsf{FP}^{\#\mathsf{P}}$-complete, where $\mathsf{FP}^{\#\mathsf{P}}$ is the class of all problems that can be solved in polynomial time using a #P-oracle. Remarkably, it turned out that in both these results the boundary of the dichotomy is defined by the class of *hierarchical* SJF-BCQs: if Q is a hierarchical SJF-BCQ, then both probabilistic query evaluation and Shapley value computation are solvable in polynomial time, whereas if Q is a non-hierarchical SJF-BCQ, then these problems are intractable. Recall that a SJF-BCQ Q is *hierarchical* if for every two variables X and Y occurring in Q, one of the following three conditions holds: (i) $\mathrm{at}(X) \subseteq \mathrm{at}(Y)$; (ii) $\mathrm{at}(Y) \subseteq \mathrm{at}(X)$; (iii) $\mathrm{at}(X) \cap \mathrm{at}(Y) = \emptyset$, where if Z is a variable occurring in Q, then $\mathrm{at}(Z)$ is the set of atoms of Q in which Z occurs. For example, the query $Q_{\mathrm{h}}() \text{ :- } E(X,Y) \wedge F(Y,Z)$ is hierarchical, while the query $Q_{\mathrm{nh}}() \text{ :- } R(X) \wedge S(X,Y) \wedge T(Y)$ is not.[1]

In this paper, we introduce and investigate the collection of *bag-set maximization* problems, which is a different collection of natural algorithmic problems parameterized by SJF-BCQs. Before describing these problems, we briefly recall the standard notion of *bag-set semantic* [18]. Under bag-set semantics, the input database is always a *set*, i.e. no duplicate facts are allowed, but the output of a query is a *bag*, i.e., duplicate answers are allowed. In the special case of a Boolean conjunctive query Q, the value of Q under bag-set semantics on a *set* database $\mathcal{D}$, denoted $Q(\mathcal{D})$, is the number of distinct satisfying assignments of Q over $\mathcal{D}$.

Bag-Set Maximization: Every fixed SJF-BCQ Q gives rise to the following problem: given two (set) databases $\mathcal{D}$ and $\mathcal{D}^r$, and a positive integer θ, find the maximum value $Q(\mathcal{D}')$ that Q takes under bag-set semantics over all (set) databases $\mathcal{D}'$ obtained from $\mathcal{D}$ by adding at most θ facts from $\mathcal{D}^r$.

[1] We suppress existential quantifiers in our queries.

Intuitively, θ is a "budget" and $\mathcal{D}^r$ is a "repair" database used to augment the set database $\mathcal{D}$ with extra facts so that the value of Q under bag-set semantics is maximized, provided the "budget" is not exceeded. For example, consider the following query:

$$Q() \text{ :- } R(A, B) \wedge S(A, C) \wedge T(A, C, D). \tag{1}$$

Suppose we are given the (set) database instance $\mathcal{D}$ in Figure 1a. Initially, Q has one satisfying assignment over $\mathcal{D}$, namely $(A, B, C, D) = (1, 5, 2, 4)$. Hence, $Q(\mathcal{D}) = 1$ under bag-set semantics. Now suppose we are allowed to amend $\mathcal{D}$ with at most $\theta = 2$ facts from the repair database $\mathcal{D}^r$ in Figure 1b, and we want to maximize the resulting $Q(\mathcal{D})$. We could amend $\mathcal{D}$ with the two facts $R(1, 6)$ and $R(1, 7)$ from $\mathcal{D}^r$, which would bring $Q(\mathcal{D})$ to 3 since Q now has two extra satisfying assignments, namely $(A, B, C, D) = (1, 6, 2, 4)$ and $(1, 7, 2, 4)$. However, a better repair is to amend $\mathcal{D}$ with the two facts $R(1, 6)$ and $T(1, 2, 9)$, since this would bring $Q(\mathcal{D})$ to 4. In this example, this would be an optimal repair, hence the answer to this Bag-Set Maximization instance is 4.

(a) Database $\mathcal{D}$ (b) Database $\mathcal{D}^r$

Fig. 1. Input instance for Bag-Set Maximization on query Q from Eq. (1). The value of θ is 2.

In some sense, the Bag-Set Maximization problem is dual to the thoroughly studied Resilience problem [8, 9, 17], which asks for the minimum number of facts that can be removed from a given database so that a conjunctive query that was true on the original database becomes false in the resulting database. It should be noted, however, that complexity results about the Resilience problem for SJF-BCQs do not yield complexity results about the Bag-Set Maximization problem for SJF-BCQs. The reason is that one has to also "dualize" the query; hence, a SJF-BCQ Q translates to a query Q^* definable by a universal first-order sentence with some additional syntactic constraints.

On the face of it, for every fixed SJF-BCQ Q, Bag-Set Maximization is a *polynomially-bounded* NP-*optimization problem*, which means that: (i) the optimum value is bounded by a polynomial in the size of the input; (ii) the underlying decision problem is in NP. Here, the underlying decision problem asks: given two databases $\mathcal{D}$ and $\mathcal{D}^r$, and two positive integers θ and τ, is there a database $\mathcal{D}'$ obtained from $\mathcal{D}$ by adding at most θ facts from $\mathcal{D}^r$, so that $Q(\mathcal{D}') \geq \tau$? The polynomial boundedness of Bag-Set Maximization implies that, for each fixed SJF-BCQ Q, this problem is also in the class $\mathsf{FP}^{\mathsf{NP}[\log(n)]}$ of function problems solvable in polynomial time using logarithmically-many calls to an NP-oracle, where n is the size of the input instance. As such, the worst-case complexity of Bag-Set Maximization is lower than the worst-case complexity of Probabilistic Query Evaluation and Shapley Value Computation, unless $\mathsf{P} = \mathsf{NP}$.[2]

We show that Bag-Set Maximization exhibits the following dichotomy: if Q is a hierarchical SJF-BCQ, then bag-set maximization is solvable in polynomial time, whereas if Q is a non-hierarchical SJF-BCQ, then bag-set maximization is an NP-complete optimization problem, i.e., its underlying decision problem is NP-complete. The intractability of Bag-Set Maximization for non-hierarchical queries is shown via a reduction from the Balanced Complete Bipartite Subgraph problem [10, Problem GT24] (also known as the Bipartite Clique problem - see [12]), a problem that has also been investigated in the context of parameterized complexity by Lin [15]. Using the results in [15],

[2]In fact, Krentel [13] showed that if $\mathsf{P} \neq \mathsf{NP}$, then $\mathsf{FP}^{\mathsf{NP}[\log(n)]} \neq \mathsf{FP}^{\mathsf{NP}}$ (hence, also $\mathsf{FP}^{\mathsf{NP}[\log(n)]} \neq \mathsf{FP}^{\#\mathsf{P}}$, etc).

we show that the natural parameterized versions of Bag-Set Maximization are W[1]-hard, and hence they are unlikely to be fixed-parameter tractable.

More significantly, on the tractability side, we show that for hierarchical SJF-BCQs, there is a single unifying polynomial-time[3] algorithm that solves Bag-Set Maximization, Probabilistic Query Evaluation, and Shapley Value Computation. The algorithm operates on an abstract algebraic structure that we call a *2-monoid*. A 2-monoid resembles a commutative semiring, except that it is *not* required to satisfy the distributive property. Specifically, a 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ consists of two commutative monoids $(K, \oplus)$ and $(K, \otimes)$ with neutral elements $\mathbf{0}$ and $\mathbf{1}$ in K for $\oplus$ and $\otimes$, respectively, and it also satisfies the identity $\mathbf{0} \otimes \mathbf{0} = \mathbf{0}$. Each of the three aforementioned problems requires a different instantiation of the 2-monoid tailored for the problem at hand.

Interestingly, each instantiation of the 2-monoid that we consider for each of the three problems is *not* going to be a semiring; specifically, it will violate the distributive property. This is expected since all the three problems are intractable for non-hierarchical queries, including the standard non-hierarchical query $Q_{\text{nh}}() \text{ :- } R(X) \wedge S(X, Y) \wedge T(Y)$, which is acyclic. If the distributive property were to hold, then our generic algorithm would have been able to solve all acyclic queries, including the non-hierarchical query Q_{nh} above, in polynomial time, which would contradict the known hardness results for these queries in all three problems. In some sense, the lack of the distributivity property in the three different 2-monoids considered here limits the power of our unifying algorithm to solving just the hierarchical queries, instead of the entire collection of all acyclic queries.

Paper Organization. The rest of the paper is organized as follows: Section 2 gives an example that illustrates the high-level idea of our unifying algorithm for hierarchical queries. Section 3 contains preliminaries. Section 4 contains the intractability results about the Bag-Set Maximization problem for non-hierarchical queries. Section 5 presents our unifying algorithm for hierarchical queries and its instantiations for each of the three problems considered. Section 6 contains the proof of correctness and the runtime analysis of this unifying algorithm. Finally, Section 7 contains concluding remarks.

2 Example of Our Unifying Algorithm for Hierarchical Queries

In this section, we illustrate our unifying algorithm by applying it to the query Q from Eq. (1) (which is a hierarchical query) and focusing on the problems of Probabilistic Query Evaluation and Bag-Set Maximization. Depending on the problem at hand, we will introduce an appropriate 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ and convert an input for the problem into a $\boldsymbol{K}$*-annotated* database $\mathcal{D}$, i.e., a database in which each fact f is associated with a value from K, called the *annotation* of f. Then, our unifying algorithm will consist of a sequence of operations over $\boldsymbol{K}$-annotated relations.

Suppose first that we are trying to solve the Probabilistic Query Evaluation problem for Q. In this case, the input to the problem is a tuple-independent probabilistic database $\mathcal{D}$ where each fact f is associated with its probability $p_f \in [0, 1]$. We define a 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ where $K = [0, 1]$ is the domain of probabilities, and $\otimes$ and $\oplus$ are defined as follows for all $p_1, p_2 \in [0, 1]$:

$$p_1 \otimes p_2 \stackrel{\text{def}}{=} p_1 \times p_2 \tag{2}$$

$$p_1 \oplus p_2 \stackrel{\text{def}}{=} 1 - (1 - p_1) \times (1 - p_2) = p_1 + p_2 - p_1 \times p_2 \tag{3}$$

Note that $p_1 \otimes p_2$ is the probability of the *conjunction* of two independent events with probabilities p_1 and p_2, whereas $p_1 \oplus p_2$ is the probability of their *disjunction*. Moreover, note that $\otimes$ does *not* distribute over $\oplus$ above, i.e., $p_1 \otimes (p_2 \oplus p_3)$ does not equal $(p_1 \otimes p_2) \oplus (p_1 \otimes p_3)$ in general. The input probabilistic database can be viewed as a $\boldsymbol{K}$-annotated database instance where each fact f is

[3]Polynomial in the input database size. Recall that we use data complexity throughout the paper.

annotated with its probability that f appears when an instance (a possible world) is sampled. The output is a value in K representing the probability of Q evaluating to true on a random instance. Our unified algorithm computes this output using a sequence of operations over $\boldsymbol{K}$-annotated relations that we depict below. For this particular application, our unified algorithm specializes precisely to the algorithm of Dalvi and Suciu [5] for evaluating a SJF-BCQ on a tuple-independent probabilistic database. Consider again the query in Eq. (1). Below, we use capital letters A, B to denote variables and small letters a, b to denote values for the corresponding variables, where the values come from a domain Dom. Given a tuple $(a, c, d) \in \mathsf{Dom}^3$, we use $T(a, c, d)$ to denote the $\boldsymbol{K}$-*annotation* of the tuple (a, c, d) in the $\boldsymbol{K}$-annotated relation T, which, in this case, corresponds to the probability of the tuple (a, c, d) appearing in the relation T:

$$T'(a,c) \leftarrow \bigoplus_{d \in \mathsf{Dom}} T(a,c,d), \qquad \forall (a,c) \in \mathsf{Dom}^2 \tag{4}$$

$$S'(a,c) \leftarrow S(a,c) \otimes T'(a,c), \qquad \forall (a,c) \in \mathsf{Dom}^2 \tag{5}$$

$$S''(a) \leftarrow \bigoplus_{c \in \mathsf{Dom}} S'(a,c), \qquad \forall a \in \mathsf{Dom} \tag{6}$$

$$R'(a) \leftarrow \bigoplus_{b \in \mathsf{Dom}} R(a,b), \qquad \forall a \in \mathsf{Dom} \tag{7}$$

$$R''(a) \leftarrow R'(a) \otimes S''(a), \qquad \forall a \in \mathsf{Dom} \tag{8}$$

$$Q() \leftarrow \bigoplus_{a \in \mathsf{Dom}} R''(a) \tag{9}$$

Consider the first operation in Eq. (4). Given a tuple $(a, c) \in \mathsf{Dom}^2$, this rule computes $T'(a, c)$, which is the probability of a fact of the form $T(a, c, d)$ appearing in an instance for some value of d. Note that the facts $T(a, c, d)$ for different d values are independent events, hence the probability of their disjunction can be correctly computed using the $\oplus$ operator from Eq. (3). In contrast, the second operation in Eq. (5) computes the probability of (a, c) occurring in both S and T', which are two independent events. Hence, the probability of their conjunction can be computed using the $\otimes$ operator from Eq. (2). The last rule in Eq. (9) computes the probability of Q evaluating to true over possible databases.

Next, suppose that we are trying to solve the Bag-Set Maximization problem for Q. In this paper, we establish that we can still use the exact same algorithm above, depicted in Eq. (4)-(9). The only thing that will change is how we define the 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ and how we construct a $\boldsymbol{K}$-annotated database to use as input to the above algorithm. Consider an input instance $(\mathcal{D}, \mathcal{D}^r, \theta)$ to the Bag-Set Maximization problem. For each fact f, we will need to store a (finitely representable) vector $\boldsymbol{x}$ of natural numbers, i.e., $\boldsymbol{x} \in \mathbb{N}^{\mathbb{N}}$, where the i-the entry of this vector represents the maximum multiplicity of the fact f that can be achieved with a repaired budget of i, that is, by adding up to i facts to the database $\mathcal{D}$ we are trying to repair. In particular, the domain K of the 2-monoid will be the set of all vectors $\boldsymbol{x} \in \mathbb{N}^{\mathbb{N}}$. Given two vectors $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{N}^{\mathbb{N}}$, we define the $\oplus$ and $\otimes$ operators as follows:

$$(\boldsymbol{x}_1 \oplus \boldsymbol{x}_2)(i) \stackrel{\text{def}}{=} \max_{i_1, i_2 \in \mathbb{N} : i_1 + i_2 = i} \boldsymbol{x}_1(i_1) + \boldsymbol{x}_2(i_2), \qquad \forall i \in \mathbb{N} \tag{10}$$

$$(\boldsymbol{x}_1 \otimes \boldsymbol{x}_2)(i) \stackrel{\text{def}}{=} \max_{i_1, i_2 \in \mathbb{N} : i_1 + i_2 = i} \boldsymbol{x}_1(i_1) \times \boldsymbol{x}_2(i_2), \qquad \forall i \in \mathbb{N} \tag{11}$$

We will argue later that the algebraic structure $\boldsymbol{K} = (K, \oplus, \otimes)$ defined above is indeed a 2-monoid (but *not* a semiring), and it can be used to solve the Bag-Set Maximization problem using the algorithm from Eq. (4)-(9) in polynomial time. To that end, we also need to construct an input

$\boldsymbol{K}$-annotated database instance, which is done as follows: (i) every fact f that is already present in the database instance $\mathcal{D}$ that we want to repair already has a multiplicity of 1 for free, hence its $\boldsymbol{K}$-annotation is the all-ones vector $\boldsymbol{x} = (1, 1, \ldots)$; (ii) in contrast, facts f that are not in $\mathcal{D}$ but in the repair database $\mathcal{D}^{\mathrm{r}}$ have a multiplicity of 0 for free, but that multiplicity can be raised to one if we are willing to pay a repair cost of one or more. Hence, such facts are annotated with the vector $\boldsymbol{x} = (0, 1, 1, \ldots)$.

The intuition behind the $\oplus$ and $\otimes$ operators from Eq. (10) and (11) is as follows. Consider a disjunction of two formulas $f = f_1 \vee f_2$, and suppose that for each formula f_j, we already have a vector $\boldsymbol{x}_j$, where for any given repair budget i_j, the i_j-th entry of the vector $\boldsymbol{x}_j$ gives us the maximum multiplicity of f_j that can be achieved within a repair budget of i_j. Moreover, suppose that the two formulas f_1 and f_2 are "independent" in the sense that they don't share any input facts, hence they have to be repaired independently. Now we ask: Given a repair budget of i, what is the maximum multiplicity of the disjunction $f = f_1 \vee f_2$ that we can achieve within this repair budget? To that end, we have to break down the repair budget i into i_1 and i_2 such that $i = i_1 + i_2$ where we spend a budget of i_1 into repairing f_1 and i_2 into repairing f_2. The multiplicity of f that we can achieve will be the sum of the two multiplicities we can achieve for f_1 and f_2, and we take the maximum over all possible ways to break i down into i_1 and i_2. This gives us the formula from Eq. (10), which is basically the *convolution* of the two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ over the $(\mathbb{N}, \max, +)$ semiring. A similar reasoning for conjunctions $f = f_1 \wedge f_2$ gives us Eq. (11), which is the convolution of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ over the $(\mathbb{N}, \max, \times)$ semiring.

Our correctness proofs and runtime analysis are not written for each of the three specific problems but rather in a generic way that utilizes the abstract algebraic properties of the 2-monoid. This makes them applicable to all three problems and possibly more. We describe the particular 2-monoid that is used to solve the Shapley Value Computation problem in Section 5.

3 Preliminaries

We use a capital letter X to denote a variable and a small letter x to denote a value for the corresponding variable X, where values come from a countably infinite domain Dom. We use boldface $\boldsymbol{X}$ to denote a set of variables and $\boldsymbol{x}$ to denote a tuple of values for the corresponding set of variables $\boldsymbol{X}$. In particular, a tuple $\boldsymbol{x}$ over a variable set $\boldsymbol{X}$ is a function $\boldsymbol{x} : \boldsymbol{X} \to \mathrm{Dom}$. Given two sets S and T, we use T^S to denote the set of all functions from S to T. As a special case, given a set of variables $\boldsymbol{X}$, we use $\mathrm{Dom}^{\boldsymbol{X}}$ to denote the set of all tuples of values for the variable set $\boldsymbol{X}$. Given a tuple $\boldsymbol{x}$ over a variable set $\boldsymbol{X}$ and a subset $\boldsymbol{Y} \subseteq \boldsymbol{X}$, we use $\boldsymbol{x}(\boldsymbol{Y})$ to denote the restriction of $\boldsymbol{x}$ to $\boldsymbol{Y}$, i.e., the tuple over $\boldsymbol{Y}$ obtained by restricting the domain of $\boldsymbol{x}$ to $\boldsymbol{Y}$. Given two tuples $\boldsymbol{x}$ and $\boldsymbol{y}$ over two disjoint variable sets $\boldsymbol{X}$ and $\boldsymbol{Y}$, we use $(\boldsymbol{x}, \boldsymbol{y})$ to denote the tuple over $\boldsymbol{X} \cup \boldsymbol{Y}$ obtained by taking the union of the two tuples. With some abuse of notation, we blur the line between a value z of a variable Z and a singleton tuple in $\mathrm{Dom}^{\{Z\}}$. For example, given a value z of a variable $Z \notin \boldsymbol{X}$, we use $(\boldsymbol{x}, z)$ to denote the tuple over $\boldsymbol{X} \cup \{Z\}$ obtained by adding z to $\boldsymbol{x}$.

An *atom* is an expression of the form $R(\boldsymbol{X})$ where R is a relation symbol and $\boldsymbol{X}$ is a set of variables. A *schema* $\mathcal{S}$ is a set of atoms. A fact over $\mathcal{S}$ is an expression of the form $R(\boldsymbol{x})$ where $R(\boldsymbol{X}) \in \mathcal{S}$ and $\boldsymbol{x} \in \mathrm{Dom}^{\boldsymbol{X}}$. A *database instance* $\mathcal{D}^{\mathcal{S}}$ over schema $\mathcal{S}$ is a set of facts over $\mathcal{S}$. When the schema $\mathcal{S}$ is clear from the context, we will simply write $\mathcal{D}$ instead of $\mathcal{D}^{\mathcal{S}}$. Given a schema $\mathcal{S} = \{R_1(\boldsymbol{X}_1), \ldots, R_m(\boldsymbol{X}_m)\}$, it will sometimes be convenient to write $\mathcal{D} = \{R_1^{\mathcal{D}}, \ldots, R_m^{\mathcal{D}}\}$, where $R_i^{\mathcal{D}}$ is the collection of R_i facts in $\mathcal{D}$, i.e., facts of the form $R_i(\boldsymbol{x}_i)$. We will typically suppress the superscript, writing R_i instead of $R_i^{\mathcal{D}}$ when no confusion results. The *size* of a database instance $\mathcal{D}$, denoted $|\mathcal{D}|$, is the number of facts in $\mathcal{D}$.

A *Boolean Conjunctive Query (BCQ)*, Q, has the following form, where $\text{at}(Q)$ is the set of atoms in Q:

$$Q() \text{ :- } \bigwedge_{R(\boldsymbol{X}) \in \text{at}(Q)} R(\boldsymbol{X}) \tag{12}$$

We use $\text{vars}(Q)$ to denote the set of all variables in Q, i.e., $\text{vars}(Q) \stackrel{\text{def}}{=} \bigcup_{R(\boldsymbol{X}) \in \text{at}(Q)} \boldsymbol{X}$. Given a variable $Y \in \text{vars}(Q)$, we use $\text{at}(Y)$ to denote the set of all atoms in Q that contain Y, i.e. $\text{at}(Y) \stackrel{\text{def}}{=} \{R(\boldsymbol{X}) \in \text{at}(Q) \mid Y \in \boldsymbol{X}\}$. A BCQ Q is said to be *self-join-free* if $\text{at}(Q)$ have distinct relation symbols, i.e., for every pair of distinct atoms $R_1(\boldsymbol{X}), R_2(\boldsymbol{X}) \in \text{at}(Q)$, we have $R_1 \neq R_2$. Throughout the paper, we only consider *Self-Join-Free BCQs*, abbreviated SJF-BCQs. A database instance $\mathcal{D}$ for Q is a database instance over the schema $\text{at}(Q)$. Throughout the paper, we will refer to a database instance $\mathcal{D}$ in the context of a specific query Q to implicitly mean that $\mathcal{D}$ is over the schema $\text{at}(Q)$.

4 The Bag-Set Maximization Problem

In this section, we define the Bag-Set Maximization problem, and show that it is NP-complete for all non-hierarchical SJF-BCQs.

*Definition 4.1 (*Bag-Set Maximization*).* The Bag-Set Maximization problem is an optimization problem that is parameterized by a self-join-free Boolean conjunctive query Q, which is interpreted under *bag-set semantics* [18]. The input to the problem is a tuple $(\mathcal{D}, \mathcal{D}^{\mathsf{r}}, \theta)$ where:

- $\mathcal{D}$ is a set database instance, which is the input instance that we want to repair.
- $\mathcal{D}^{\mathsf{r}}$ is another set database instance, called the *repair database*, which contains the facts that we are allowed to add to $\mathcal{D}$ in order to repair $\mathcal{D}$.
- θ is a natural number representing an upper bound on the number of facts we are allowed to add to $\mathcal{D}$ in order to repair $\mathcal{D}$.

The target is to find the maximum value of $Q(\mathcal{D}')$ under bag-set semantics over all set database instances $\mathcal{D}'$ that are *valid repairs* of $\mathcal{D}$, i.e., that satisfy:

- $\mathcal{D} \subseteq \mathcal{D}' \subseteq \mathcal{D} \cup \mathcal{D}^{\mathsf{r}}$ and $|\mathcal{D}' \setminus \mathcal{D}| \leq \theta$. In words, a valid repair $\mathcal{D}'$ results from $\mathcal{D}$ by adding at most θ facts from $\mathcal{D}^{\mathsf{r}}$. We call the quantity $|\mathcal{D}' \setminus \mathcal{D}|$ the *repair cost* and denote it $\texttt{cost}(\mathcal{D}, \mathcal{D}')$.

The above optimization problem has a natural decision version, which we define below.

*Definition 4.2 (*Bag-Set Maximization Decision*).* The Bag-Set Maximization Decision problem is the decision version of Bag-Set Maximization. It is parameterized by a self-join-free Boolean conjunctive query Q, and has the same inputs as Bag-Set Maximization and one additional input which is a natural number $\tau \in \mathbb{N}$. The target is to decide whether the maximum value of $Q(\mathcal{D}')$ under bag-set semantics over all valid repairs $\mathcal{D}'$ of $\mathcal{D}$ is at least τ.

The following proposition is straightforward.

Proposition 4.3. *For any SJF-BCQ Q, the* Bag-Set Maximization Decision *problem is in* NP.

To prove the above proposition, we can guess a valid repair $\mathcal{D}'$ of $\mathcal{D}$ and verify that $Q(\mathcal{D}') \geq \tau$. Both the size of $\mathcal{D}'$ and the time needed to evaluate $Q(\mathcal{D}')$ are polynomial in the input size. (Recall that the query Q is a fixed parameter to the problem, hence its size is a constant, which means we are using data complexity.)

We now prove that for any *non-hierarchical* SJF-BCQ Q, the Bag-Set Maximization Decision problem for Q is NP-complete. To prove NP-hardness, we rely on a reduction from the Balanced

Complete Bipartite Subgraph (BCBS) problem, which is a well-known NP-complete problem, that is defined as follows: Given a (undirected and self-loop-free) graph $G = (V, E)$ and a positive integer k, is there a complete bipartite subgraph of G in which each of the two parts has size k? BCBS is known to be NP-complete [10, Problem GT24].

Theorem 4.4. *For any non-hierarchical SJF-BCQ Q, the* Bag-Set Maximization Decision *problem for Q is* NP*-complete.*

Proof. Membership in NP follows from Proposition 4.3. We now prove NP-hardness. Let Q be an arbitrary non-hierarchical SJF-BCQ. Since Q is not hierarchical, Q must have the form:

$$Q() \text{ :- } R(A, \boldsymbol{X}), S(A, B, \boldsymbol{Y}), T(B, \boldsymbol{Z}), P_1(\boldsymbol{D}_1), \ldots, P_n(\boldsymbol{D}_n),$$

where A, B are distinct variables and $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{D}_1, \ldots, \boldsymbol{D}_n$ are sets of variables, $A \notin \boldsymbol{Z}$ and $B \notin \boldsymbol{X}$, and $n \geq 0$. We give a polynomial-time reduction from BCBS to Bag-Set Maximization Decision for Q. Let (G, k) be an input instance to BCBS where $G = (V, E)$ is an undirected graph without self-loops and k is a positive integer. We construct an input instance $(\mathcal{D}, \mathcal{D}^{\mathrm{r}}, \theta, \tau)$ for the Bag-Set Maximization Decision problem for Q as follows: Let Dom be the set of vertices V, and fix an arbitrary vertex $a \in V$. Let $\boldsymbol{W} \stackrel{\text{def}}{=} \mathrm{vars}(Q)$ and Γ be the set of all tuples $\boldsymbol{w} \in \mathrm{Dom}^{\boldsymbol{W}}$ such that $\boldsymbol{w}(X) = a$ for all $X \in \boldsymbol{W} \setminus \{A, B\}$. The database instance $\mathcal{D} = \{R^{\mathcal{D}}, S^{\mathcal{D}}, T^{\mathcal{D}}, P_1^{\mathcal{D}}, \ldots, P_n^{\mathcal{D}}\}$ is defined as follows:

$$\begin{aligned}
R^{\mathcal{D}} &= T^{\mathcal{D}} = \emptyset, \\
S^{\mathcal{D}} &= \{S(\boldsymbol{w}(A), \boldsymbol{w}(B), \boldsymbol{w}(\boldsymbol{Y})) \mid \boldsymbol{w} \in \Gamma \text{ and } \{\boldsymbol{w}(A), \boldsymbol{w}(B)\} \in E\}, \\
P_i^{\mathcal{D}} &= \{P_i(\boldsymbol{w}(\boldsymbol{D}_i)) \mid \boldsymbol{w} \in \Gamma \text{ and } \{\boldsymbol{w}(A), \boldsymbol{w}(B)\} \in E\}, \quad \forall i \in [n].
\end{aligned}$$

In contrast, the repair database $\mathcal{D}^{\mathrm{r}} = \{R^{\mathcal{D}^{\mathrm{r}}}, S^{\mathcal{D}^{\mathrm{r}}}, T^{\mathcal{D}^{\mathrm{r}}}, {P_1}^{\mathcal{D}^{\mathrm{r}}}, \ldots, {P_n}^{\mathcal{D}^{\mathrm{r}}}\}$ is defined as follows:

$$R^{\mathcal{D}^{\mathrm{r}}} = \{R(\boldsymbol{w}(A), \boldsymbol{w}(\boldsymbol{X})) \mid \boldsymbol{w} \in \Gamma\}, \quad T^{\mathcal{D}^{\mathrm{r}}} = \{T(\boldsymbol{w}(B), \boldsymbol{w}(\boldsymbol{Z})) \mid \boldsymbol{w} \in \Gamma\}, \quad S^{\mathcal{D}^{\mathrm{r}}} = P_1^{\mathcal{D}^{\mathrm{r}}} = \ldots = P_n^{\mathcal{D}^{\mathrm{r}}} = \emptyset.$$

In other words, we encode the edge relation into $S^{\mathcal{D}}$, and we permit repairs $\mathcal{D}'$ of $\mathcal{D}$ to be obtained by adding R or T facts where A and B are assigned to arbitrary vertices in V, while all other variables must be assigned to the fixed vertex a. Finally, we set $\theta = 2k$ and $\tau = k^2$, thus specifying the input instance $(\mathcal{D}, \mathcal{D}^{\mathrm{r}}, \theta, \tau)$ for Bag-Set Maximization Decision.

We claim that the following two statements are equivalent:

(1) (G, k) is a "yes" instance of BCBS, i.e., G has a complete bipartite subgraph in which each of the two parts has size k.
(2) There is a database $\mathcal{D}'$ such that $\mathcal{D} \subseteq \mathcal{D}' \subseteq \mathcal{D} \cup \mathcal{D}^{\mathrm{r}}$, $\mathtt{cost}(\mathcal{D}, \mathcal{D}') \leq 2k$, and $Q(\mathcal{D}') \geq k^2$.

Proving (1) $\Longrightarrow$ (2)*:* If G has a complete balanced bipartite subgraph with parts U_1 and U_2, each of size k, then let $\mathcal{D}' = \{R^{\mathcal{D}'}, S^{\mathcal{D}'}, T^{\mathcal{D}'}, P_1^{\mathcal{D}'}, \ldots, P_n^{\mathcal{D}'}\}$ be the set database in which $S^{\mathcal{D}'} = S^{\mathcal{D}}$, $P_i^{\mathcal{D}'} = P_i^{\mathcal{D}}$ for each $i \in [n]$, and

$$\begin{aligned}
R^{\mathcal{D}'} &= \{R(\boldsymbol{w}(A), \boldsymbol{w}(\boldsymbol{X})) \mid \boldsymbol{w} \in \Gamma \text{ and } \boldsymbol{w}(A) \in U_1\}, \\
T^{\mathcal{D}'} &= \{T(\boldsymbol{w}(B), \boldsymbol{w}(\boldsymbol{Z})) \mid \boldsymbol{w} \in \Gamma \text{ and } \boldsymbol{w}(B) \in U_2\}.
\end{aligned}$$

Then, $\mathtt{cost}(\mathcal{D}, \mathcal{D}') = |U_1| + |U_2| = 2k$. Let Γ' denote the set tuples $\boldsymbol{w} \in \Gamma$ such that $\boldsymbol{w}(A) \in U_1$ and $\boldsymbol{w}(B) \in U_2$. Note that Γ' contains exactly k^2 tuples, each of which represents a satisfying assignment for Q in $\mathcal{D}'$. Hence $Q(\mathcal{D}') \geq k^2$.

Proving (2) $\Longrightarrow$ (1)*:* Assume that $\mathcal{D}' = \{R^{\mathcal{D}'}, S^{\mathcal{D}'}, T^{\mathcal{D}'}, P_1^{\mathcal{D}'}, \ldots, P_n^{\mathcal{D}'}\}$ is a set database such that $\mathcal{D} \subseteq \mathcal{D}' \subseteq \mathcal{D} \cup \mathcal{D}^{\mathrm{r}}$, $\mathtt{cost}(\mathcal{D}, \mathcal{D}') \leq 2k$, and $Q(\mathcal{D}') \geq k^2$. By the construction of $\mathcal{D}$ and $\mathcal{D}^{\mathrm{r}}$, we must have $S^{\mathcal{D}'} = S^{\mathcal{D}}$ and $P_i^{\mathcal{D}'} = P_i^{\mathcal{D}}$ for each $i \in [n]$. By the construction of $\mathcal{D}$, every satisfying assignment to Q in $\mathcal{D}'$ corresponds to a tuple $\boldsymbol{w} \in \Gamma$. Furthermore, every tuple $\boldsymbol{w} \in \Gamma$ is uniquely determined by its A and B values. Therefore, $k^2 \leq Q(\mathcal{D}') \leq \left|R^{\mathcal{D}'}\right| \times \left|T^{\mathcal{D}'}\right|$. Since $\mathtt{cost}(\mathcal{D}, \mathcal{D}') \leq 2k$, we also have that $\left|R^{\mathcal{D}'}\right| + \left|T^{\mathcal{D}'}\right| \leq 2k$. By a straightforward calculus argument, this implies that $\left|R^{\mathcal{D}'}\right| = \left|T^{\mathcal{D}'}\right| = k$. Now let

$$U_1 = \{\boldsymbol{w}(A) \in V \mid \boldsymbol{w} \in \Gamma \text{ and } R(\boldsymbol{w}(A), \boldsymbol{w}(\boldsymbol{X})) \in R^{\mathcal{D}'}\}$$
$$U_2 = \{\boldsymbol{w}(B) \in V \mid \boldsymbol{w} \in \Gamma \text{ and } T(\boldsymbol{w}(B), \boldsymbol{w}(\boldsymbol{Z})) \in T^{\mathcal{D}'}\}.$$

For $Q(\mathcal{D}')$ to be at least k^2, for every $(v_1, v_2) \in U_1 \times U_2$, we must have $S(v_1, v_2, \boldsymbol{w}(\boldsymbol{Y})) \in S^{\mathcal{D}'} = S^{\mathcal{D}}$, which implies that $\{v_1, v_2\} \in E$. Since G does not contain self-loops, it follows that $v_1 \neq v_2$ for all $(v_1, v_2) \in U_1 \times U_2$. Hence, U_1 and U_2 form a complete bipartite subgraph of size $k \times k$ in G. □

We conclude this section with some immediate corollaries of Theorem 4.4 regarding the parameterized complexity and approximation hardness of Bag-Set Maximization.

One natural parameterization of Bag-Set Maximization Decision treats both the repair cost θ and the target τ as parameters. An alternative parameterization is the *standard parameterization* [7], where only the target τ is a parameter. In either case, the polynomial-time reduction in Theorem 4.4 can be extended to an FPT-reduction from the parameterized BCBS problem – parameterized by the size k of each part of the desired balanced complete bipartite subgraph – by observing that θ and τ in the resulting Bag-Set Maximization Decision instance are computable functions of k. Hence by the W[1]-hardness of BCBS [15], we obtain the following.

Corollary 4.5. *For every non-hierarchical SJF-BCQ Q, the parameterized* Bag-Set Maximization Decision *problem for Q (with parameters θ and τ or with just parameter τ) is* W[1]*-hard.*

Additionally, the Bag-Set Maximization problem can be interpreted as an approximation problem by taking the set of feasible solutions for an instance $(\mathcal{D}, \mathcal{D}^{\mathrm{r}}, \theta)$ of the problem to be any repair $\mathcal{D}'$ of repair cost at most θ. The *value* associated to this instance and a feasible repair $\mathcal{D}'$ is $Q(\mathcal{D}')$. While hardness-of-approximation results for BCBS are known [6], it is not obvious how to extend the reduction of Theorem 4.4 to an appropriate approximation-preserving reduction. However, it is known that, if an NP-optimization problem has a *fully polynomial-time approximation scheme* (FPTAS), then its standard parameterization is in FPT [3]. Hence, by the W[1]-hardness of its standard parameterization, we obtain the following.

Corollary 4.6. *For every non-hierarchical SJF-BCQ Q, the* Bag-Set Maximization *problem for Q does not have an* FPTAS *unless* FPT = W[1].

5 The Unifying Algorithm for Hierarchical Queries

In order to describe our general-purpose polynomial-time algorithm for hierarchical SJF-BCQs, we need to develop some technical tools.

5.1 An Elimination Procedure for Hierarchical Queries

As an alternative to the standard definition from the introduction, hierarchical queries can be defined in terms of a specific "elimination procedure", where a query Q is hierarchical if and only if this elimination procedure eliminates all its variables, thus reducing it to a query of the form $Q()$:- $R()$. This elimination procedure will be at the heart of our polynomial-time algorithm where the algorithm's steps mirror the elimination steps. The elimination procedure is also similar in

spirit to the *GYO elimination procedure* for acyclic queries [1], and also runs in polynomial time in the size of the query, just like the GYO procedure does.

Proposition 5.1 (Elimination Procedure for Hierarchical Queries). *A SJF-BCQ Q is hierarchical if and only if the following elimination procedure transforms Q into a query with a single nullary atom, i.e., $Q() \text{ :- } R()$. Repeatedly, apply either one of the following two rules, until neither rule applies:*

- (Rule 1) *Eliminate a "private" variable Y: Find a variable $Y \in \text{vars}(Q)$ that occurs in only one atom $R(\boldsymbol{X})$ in Q, and replace $R(\boldsymbol{X})$ with $R'(\boldsymbol{X} \setminus \{Y\})$ where R' is a new relation symbol.*
- (Rule 2) *Eliminate a "duplicate atom": Find two distinct atoms $R_1(\boldsymbol{X})$ and $R_2(\boldsymbol{X})$ that share the same set of variables $\boldsymbol{X}$, and replace both atoms with a single atom $R'(\boldsymbol{X})$ where R' is a new relation symbol.*[4]

Moreover, the above two rules maintain the hierarchical property. In particular, after each application of Rule 1 or Rule 2, the resulting query is hierarchical if and only if the original query was hierarchical.

Note that the above elimination procedure is very similar in nature to the GYO elimination procedure [1], which is typically used to detect if a query is acyclic. The only difference is that in GYO, Rule 2 is more relaxed: We find two atoms $R_1(\boldsymbol{X})$ and $R_2(\boldsymbol{Y})$ where $\boldsymbol{X} \subseteq \boldsymbol{Y}$, and we replace them with a single atom $R'(\boldsymbol{Y})$. As a result, whenever the above hierarchical elimination procedure succeeds on a query, the GYO elimination procedure is guaranteed to succeed on the same query (but the opposite is not necessarily true). This in turn is consistent with the well-known fact that hierarchical queries are a (strict) subset of acyclic queries. Moreover, just like GYO, there could be multiple ways to apply the elimination procedure to a given query, but they are all guaranteed to lead to the same conclusion, regarding the hierarchical property of the query. Before proving Proposition 5.1, we give a few examples.

Example 5.2. Consider the query Q from Eq. (1). Here is one way to apply the elimination procedure. It shows that the query is hierarchical. Any other way to apply the procedure will lead to the same conclusion.

$$
\begin{aligned}
Q() &\text{ :- } R(A,B) \wedge S(A,C) \wedge T(A,C,D) && \text{(Rule 1)}\\
Q() &\text{ :- } R(A,B) \wedge S(A,C) \wedge T'(A,C) && \text{(Rule 2)}\\
Q() &\text{ :- } R(A,B) \wedge S'(A,C) && \text{(Rule 1)}\\
Q() &\text{ :- } R(A,B) \wedge S''(A) && \text{(Rule 1)}\\
Q() &\text{ :- } R'(A) \wedge S''(A) && \text{(Rule 2)}\\
Q() &\text{ :- } R''(A) && \text{(Rule 1)}\\
Q() &\text{ :- } R'''() && \text{(Done!)}
\end{aligned}
$$

Moreover, each one of the above intermediate queries is also hierarchical.

Example 5.3. Consider the following query:

$$Q() \text{ :- } R(A,B) \wedge S(B,C) \wedge T(C,D)$$

The elimination procedure can proceed as follows:

$$
\begin{aligned}
Q() &\text{ :- } R(A,B) \wedge S(B,C) \wedge T(C,D) && \text{(Rule 1)}\\
Q() &\text{ :- } R(A,B) \wedge S(B,C) \wedge T'(C) && \text{(Rule 1)}
\end{aligned}
$$

[4]Note that we are *not* adding two copies of the new atom $R'(\boldsymbol{X})$ but only a single copy. Hence, the resulting query is still self-join-free.

$$Q() \text{ :- } R'(B) \wedge S(B,C) \wedge T'(C) \qquad \text{(Stuck!)}$$

But now we are left with a query where neither rule applies. This shows that the original query and all of the above intermediate queries are *not* hierarchical.

The following example shows that even if the input hierarchical query was disconnected, the elimination procedure still reduces it to a *single* nullary atom.

Example 5.4. Consider the query:

$$Q() \text{ :- } R(A) \wedge S(B)$$

The elimination procedure proceeds as follows:

$$\begin{aligned} Q() &\text{ :- } R(A) \wedge S(B) && \text{(Rule 1)} \\ Q() &\text{ :- } R(A) \wedge S'() && \text{(Rule 1)} \\ Q() &\text{ :- } R'() \wedge S'() && \text{(Rule 2)} \\ Q() &\text{ :- } R''() && \text{(Done!)} \end{aligned}$$

The query is hierarchical.

In order to prove Proposition 5.1, we need some additional definitions and results. Given a SJF-BCQ Q, two atoms $R_1(\boldsymbol{X}_1), R_2(\boldsymbol{X}_2) \in \mathsf{at}(Q)$ are called *connected* if:

- they share at least one variable, i.e., $\boldsymbol{X}_1 \cap \boldsymbol{X}_2 \neq \emptyset$, or
- they are both connected to a third atom $R_3(\boldsymbol{X}_3)$.

A SJF-BCQ Q is called *connected* if every pair of atoms in Q are connected. Given a SJF-BCQ Q, the *connected components* of Q are the (unique) connected SJF-BCQs $Q_1, Q_2, \ldots, Q_m$ that satisfy $Q = Q_1 \wedge Q_2 \wedge \ldots \wedge Q_m$ and $\mathsf{vars}(Q_i) \cap \mathsf{vars}(Q_j) = \emptyset$ for all $i \neq j$. The following proposition is well-known [4].

Proposition 5.5 ([4]). *A connected SJF-BCQ Q is hierarchical if and only if there exists a rooted tree T whose set of nodes is exactly $\mathsf{vars}(Q)$ that satisfies the following: For every atom $R(\boldsymbol{X}) \in \mathsf{at}(Q)$, there exists a node $Y \in \mathsf{vars}(Q)$ where the set of variables along the path from Y to the root of T (including Y and the root) is exactly $\boldsymbol{X}$.*

Proof of Proposition 5.1. To prove Proposition 5.1, we will prove the following claims:

Claim 5.1. *Given a hierarchical SJF-BCQ Q that does not have the form $Q() \text{ :- } R()$, either Rule 1 or Rule 2 must be applicable to Q, and the resulting query must be hierarchical.*

Claim 5.2. *Given a non-hierarchical SJF-BCQ Q, if either Rule 1 or Rule 2 is applicable to Q, then the resulting query must be non-hierarchical.*

Proof of Claim 5.1. Claim 5.1 follows from Proposition 5.5. In particular, consider a hierarchical SJF-BCQ Q that does not have the form $Q() \text{ :- } R()$. If $\mathsf{vars}(Q) = \emptyset$, then all connected components of Q must have the form $Q_i() \text{ :- } R_i()$, in which case we can apply Rule 2 on them. On the other hand, if $\mathsf{vars}(Q) \neq \emptyset$, let Q_i be a connected component of Q that has at least one variable. Let T_i be the tree that satisfies Proposition 5.5, let $Y \in \mathsf{vars}(Q)$ be any leaf in the tree, and let $\boldsymbol{X}$ be the set of variables from Y to the root of T_i. We recognize two cases:

- If Q_i contains more than one atom whose variables are $\boldsymbol{X}$, then we can apply Rule 2 on them.

- Otherwise, let $R(\boldsymbol{X})$ be the unique atom in Q_i with a variable set $\boldsymbol{X}$. In this case, $R(\boldsymbol{X})$ must be the only atom in Q_i that contains Y, in which case we can apply Rule 1.

In both cases above, the resulting component Q_i still has a tree T_i' satisfying Proposition 5.5, hence the resulting Q is still hierarchical. This proves Claim 5.1. □

Proof of Claim 5.2. In order to prove Claim 5.2, consider a non-hierarchical SJF-BCQ Q. By definition of hierarchical queries, Q must contain two distinct variables $A, B \in \text{vars}(Q)$ and three distinct atoms $R(\boldsymbol{X}), S(\boldsymbol{Y}), T(\boldsymbol{Z}) \in \text{at}(Q)$ such that:

- A is in $\boldsymbol{X}$ and $\boldsymbol{Y}$ but not in $\boldsymbol{Z}$, and
- B is in $\boldsymbol{Y}$ and $\boldsymbol{Z}$ but not in $\boldsymbol{X}$.

Suppose that Rule 1 is applicable to Q. Note that the private variable that we eliminate in Rule 1 cannot be either A or B because each one of them is shared by at least two atoms. Hence after applying Rule 1, we can still use the same two variables A and B along with the atoms $R(\boldsymbol{X}), S(\boldsymbol{Y}), T(\boldsymbol{Z})$ (one of which might be the one containing the private variable) to show that the resulting query is still non-hierarchical.

Now suppose that Rule 2 is applicable to Q. Note that the two duplicate atoms from Rule 2 that share the same set of variables cannot both belong to the set of atoms $\{R(\boldsymbol{X}), S(\boldsymbol{Y}), T(\boldsymbol{Z})\}$ because none of these three atoms share the same set of variables. Hence, even after applying Rule 2, we will still be left with three distinct atoms with variable sets $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$. We can still use these atoms along with the two variables A and B to show that the resulting query is still non-hierarchical. This proves Claim 5.2. □

We now show how to use Claims 5.1 and 5.2 to prove Proposition 5.1. Note that applying Rule 1 reduces the number of variables in the query by one, while applying Rule 2 reduces the number of atoms by one. Therefore, the two rules can only be applied a finite number of times. Claim 5.1 guarantees that if the query is hierarchical, then the two rules can be repeatedly applied until the query is reduced to the form $Q()$:- $R()$. On the other hand, Claim 5.2 guarantees that if the query is non-hierarchical, then the two rules are going to preserve it as non-hierarchical, hence it will never be reduced to the form $Q()$:- $R()$, since the latter is a hierarchical query. This proves Proposition 5.1. □

5.2 The Underlying Algebraic Structure: 2-monoid

Our algorithm operates on a general algebraic structure called *2-monoid*, which we define below. A specific instantiation of this structure is chosen depending on the specific application domain that the algorithm is used for.

A *2-monoid* is a structure $\boldsymbol{K} = (K, \oplus, \otimes)$ that satisfies the commutative semiring properties, with two exceptions: (a) it doesn't necessarily satisfy the distributive property, and (b) the annihilation-by-zero property is replaced by the weaker property $\mathbf{0} \otimes \mathbf{0} = \mathbf{0}$.

Definition 5.6 (2-monoid). A triple $\boldsymbol{K} = (K, \oplus, \otimes)$ is a *2-monoid* iff it satisfies the following:

- $(K, \oplus)$ is a commutative monoid[5] with identity element $\mathbf{0}$.
- $(K, \otimes)$ is a commutative monoid with identity element $\mathbf{1}$.
- The identity element, $\mathbf{0}$, of $(K, \oplus)$ satisfies $\mathbf{0} \otimes \mathbf{0} = \mathbf{0}$.

[5] A *monoid* $(K, \oplus)$ is a pair where K is a set and $\oplus$ is an associative binary operator over the elements of K that has an *identity element* $\mathbf{0}$, i.e., an element $\mathbf{0} \in K$ where $a \oplus \mathbf{0} = \mathbf{0} \oplus a = a$, for all $a \in K$. A monoid $(K, \oplus)$ is *commutative* if $\oplus$ is commutative.

We illustrate the 2-monoid structure for the three problems we consider in Sections 5.4, 5.5, and 5.6. [6]

5.3 The Algorithm

We now present our general algorithm for hierarchical queries. Depending on the application, the algorithm can be instantiated to compute different values representing the solutions to different problems. The algorithm is instantiated by providing a specific instantiation of the 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ as a parameter. The algorithm is depicted in Algorithm 1. It takes as input a hierarchical SJF-BCQ Q and a $\boldsymbol{K}$-annotated database instance $\mathcal{D}$. The algorithm mirrors the elimination procedure from Prop. 5.1. As an example, note that the specific algorithm from Eq. (4)-(9) mirrors the elimination steps from Example 5.2. The algorithm repeatedly applies either Rule 1 or Rule 2 until the query is reduced to the form $Q() \text{ :- } R()$. We use the $\oplus$-operator in Rule 1 and the $\otimes$-operator in Rule 2 to compute new $\boldsymbol{K}$-annotated relations that replace old ones. When the elimination procedure stops, the remaining query is guaranteed to have the form $Q() \text{ :- } R()$, assuming it was hierarchical to begin with. We return the $\boldsymbol{K}$-annotation of the nullary tuple $()$ in R as the output of the algorithm.

Algorithm 1 General-Purpose Algorithm for Hierarchical Queries

Parameters:
- A 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$.

Input:
- A hierarchical SJF-BCQ Q (Eq. (12)).
- A $\boldsymbol{K}$-annotated database instance $\mathcal{D}$.
  - For each $R(\boldsymbol{X}) \in \text{at}(Q)$, the relation R is $\boldsymbol{K}$-annotated in $\mathcal{D}$.

Output:
- A value $q \in K$. The semantics for q depends on the instantiation of the algorithm.

while Q is not of the form $Q() \text{ :- } R()$ **do** ▷ Apply Rule 1 or 2 from Prop. 5.1
  if $\exists Y \in \text{vars}(Q)$ where Y occurs in only one atom $R(\boldsymbol{X})$ **then** ▷ Rule 1
    Let $\boldsymbol{X}' \stackrel{\text{def}}{=} \boldsymbol{X} \setminus \{Y\}$, hence $R(\boldsymbol{X}) = R(\boldsymbol{X}', Y)$
    Create a new atom $R'(\boldsymbol{X}')$ defined as $R'(\boldsymbol{x}') \stackrel{\text{def}}{=} \bigoplus_{y \in \text{Dom}} R(\boldsymbol{x}', y)$, for all $\boldsymbol{x}' \in \text{Dom}^{X'}$
    Replace $R(\boldsymbol{X})$ with $R'(\boldsymbol{X}')$
  else if $\exists$ two atoms $R_1(\boldsymbol{X})$ and $R_2(\boldsymbol{X})$ with the same set of variables $\boldsymbol{X}$ **then** ▷ Rule 2
    Create a new atom $R'(\boldsymbol{X})$ defined as $R'(\boldsymbol{x}) \stackrel{\text{def}}{=} R_1(\boldsymbol{x}) \otimes R_2(\boldsymbol{x})$, for all $\boldsymbol{x} \in \text{Dom}^{X}$
    Replace both $R_1(\boldsymbol{X})$ and $R_2(\boldsymbol{X})$ with a single atom $R'(\boldsymbol{X})$
  else
    Report Q is not hierarchical! ▷ By Prop. 5.1
return $R()$, i.e., the $\boldsymbol{K}$-annotation of the tuple $()$ in R ▷ Q now has the form $Q() \text{ :- } R()$

5.4 First Instantiation: Probabilistic Query Evaluation

We now instantiate Algorithm 1 to compute the probability of a query Q evaluating to true over a tuple-independent probabilistic database $\mathcal{D}$. The 2-monoid we use is defined as follows:

[6] Note that if $\mathbf{0} = \mathbf{1}$, then the 2-monoid becomes trivial. Nevertheless, our claims and proofs still hold, which is why we don't explicitly exclude this case.

Definition 5.7 (2-monoid for Probabilistic Query Evaluation*).* We define a specific 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ where the domain K is the set of all probabilities, i.e. $K \stackrel{\text{def}}{=} [0, 1]$, and the $\oplus$ and $\otimes$ are defined using Eq. (3) and (2) respectively, for every $p_1, p_2 \in [0, 1]$. The identities for $\oplus$ and $\otimes$ are $\mathbf{0} \stackrel{\text{def}}{=} 0$ and $\mathbf{1} \stackrel{\text{def}}{=} 1$ respectively.

We prove the following theorem in the next section.

Theorem 5.8. *Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be the 2-monoid from Definition 5.7. Let Q be a hierarchical SJF-BCQ, and $\mathcal{D}$ be a tuple-independent probabilistic database, viewed as a $\boldsymbol{K}$-annotated database instance $\mathcal{D}$, where each fact is annotated with its probability. Then using Q and $\mathcal{D}$ as inputs, Algorithm 1 runs in time $O(|\mathcal{D}|)$ and outputs the probability of Q evaluating to true over all possible worlds.*

5.5 Second Instantiation: Bag-Set Maximization

In order to use Algorithm 1 to solve Bag-Set Maximization, we define a suitable 2-monoid.

Definition 5.9 (2-monoid for Bag-Set Maximization*).* We define a specific 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ as follows. The domain K is the set of all vectors of natural numbers $\mathbb{N}^{\mathbb{N}}$ that are *monotonic*[7], i.e., $K \stackrel{\text{def}}{=} \left\{\boldsymbol{x} \in \mathbb{N}^{\mathbb{N}} \mid \forall i < j, \boldsymbol{x}(i) \leq \boldsymbol{x}(j)\right\}$. For every pair of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2 \in K$, the $\oplus$ and $\otimes$ operators are defined using Eq. (10) and (11) respectively. The identities $\mathbf{0}$ and $\mathbf{1}$ for $\oplus$ and $\otimes$ are the all-zeros and all-ones vectors respectively, i.e., $\mathbf{0}(i) \stackrel{\text{def}}{=} 0$ and $\mathbf{1}(i) \stackrel{\text{def}}{=} 1$ for all $i \in \mathbb{N}$.

It is straightforward to verify that the structure $\boldsymbol{K} = (K, \oplus, \otimes)$ from Definition 5.9 satisfies all 2-monoid properties from Definition 5.6.

Given an input instance $(\mathcal{D}, \mathcal{D}^{\mathrm{r}}, \theta)$ to the Bag-Set Maximization problem from Definition 4.1, the following definition helps us construct a $\boldsymbol{K}$-annotated database instance $\psi(\mathcal{D}, \mathcal{D}^{\mathrm{r}})$ that is used as input to Algorithm 1. To that end, we define a vector $\bigstar \in K$ as follows. For any $i \in \mathbb{N}$:

$$\bigstar(i) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{otherwise} \end{cases}$$

Definition 5.10 (Input $\boldsymbol{K}$-annotated database instance for Bag-Set Maximization*).* Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be the 2-monoid from Definition 5.9, Q be a SJF-BCQ, and $(\mathcal{D}, \mathcal{D}^{\mathrm{r}}, \theta)$ be an input instance to the Bag-Set Maximization problem from Definition 4.1. We define a function ψ that maps a fact $R(\boldsymbol{x})$ where $R(\boldsymbol{X}) \in \mathrm{at}(Q)$ and $\boldsymbol{x} \in \mathrm{Dom}^{\boldsymbol{X}}$ to a monotonic vector in K as follows:

$$\psi(R(\boldsymbol{x})) \stackrel{\text{def}}{=} \begin{cases} \mathbf{1} & \text{if } R(\boldsymbol{x}) \in \mathcal{D} \\ \bigstar & \text{if } R(\boldsymbol{x}) \notin \mathcal{D} \text{ and } R(\boldsymbol{x}) \in \mathcal{D}^{\mathrm{r}} \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{13}$$

Moreover, we define $\psi(\mathcal{D}, \mathcal{D}^{\mathrm{r}})$ as a $\boldsymbol{K}$-annotated database instance where for every $R(\boldsymbol{X}) \in \mathrm{at}(Q)$ and $\boldsymbol{x} \in \mathrm{Dom}^{\boldsymbol{X}}$, the fact $R(\boldsymbol{x})$ is annotated with $\psi(R(\boldsymbol{x}))$.

We prove the following theorem in the next section.

Theorem 5.11. *Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be the 2-monoid from Definition 5.9, Q be a hierarchical SJF-BCQ, $(\mathcal{D}, \mathcal{D}^{\mathrm{r}}, \theta)$ be an input instance to* Bag-Set Maximization *from Definition 4.1, and $\psi(\mathcal{D}, \mathcal{D}^{\mathrm{r}})$ be the $\boldsymbol{K}$-annotated database instance from Definition 5.10. Then using Q and $\psi(\mathcal{D}, \mathcal{D}^{\mathrm{r}})$ as inputs, Algorithm 1 runs in time $O((|\mathcal{D}| + |\mathcal{D}^{\mathrm{r}}|) \cdot |\mathcal{D}^{\mathrm{r}}|^2)$ and space $O((|\mathcal{D}| + |\mathcal{D}^{\mathrm{r}}|) \cdot |\mathcal{D}^{\mathrm{r}}|)$ and outputs a vector $q \in K$ that satisfies the following:*

[7]Recall that we use the notation T^S to denote the set of all functions from S to T. Hence, $\mathbb{N}^{\mathbb{N}}$ is the set of all functions from $\mathbb{N}$ to $\mathbb{N}$, or equivalently vectors of natural numbers.

- *For every $i \in \mathbb{N}$, the i-the entry, $q(i)$, is the maximum multiplicity of Q that can be achieved within a repair cost at most i.*

In particular, the output to the Bag-Set Maximization problem is $q(\theta)$, i.e., the θ-th entry of q.

5.6 Third Instantiation: Shapley Value Computation

We start with a brief overview of the concept of Shapley values of facts in a database, and then explain how to instantiate Algorithm 1 to compute these values.

Definition 5.12 (Shapley value of a fact with respect to a query and a database). Let Q be a Boolean Conjunctive Query and $\mathcal{D}$ be a database instance. The instance $\mathcal{D}$ is divided into two parts:

- *Exogenous part*, denoted $\mathcal{D}^{\mathsf{x}}$. These are facts that are fixed.
- *Endogenous part*, denoted $\mathcal{D}^{\mathsf{n}}$. These are facts that can be added or deleted from $\mathcal{D}$.

Suppose that we pick a permutation of the facts in $\mathcal{D}^{\mathsf{n}}$ uniformly at random, and we begin to insert those facts one by one into $\mathcal{D}^{\mathsf{x}}$. The *Shapley value of a given fact f with respect to Q and $\mathcal{D}$* is the probability of adding the fact f flipping the answer of Q from false to true. Formally, let $\pi(\mathcal{D}^{\mathsf{n}})$ denote the set of all permutations of facts in $\mathcal{D}^{\mathsf{n}}$. Given a permutation $\sigma \in \pi(\mathcal{D}^{\mathsf{n}})$ and a fact f in $\mathcal{D}^{\mathsf{n}}$, let $\sigma_{<f}$ denote the set of facts that appear before f in σ. Given a fact f in $\mathcal{D}^{\mathsf{n}}$, the *Shapley value of f with respect to Q and $\mathcal{D}$* is defined as:

$$\mathsf{Shapley}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(f) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{D}^{\mathsf{n}}|!} \sum_{\sigma \in \pi(\mathcal{D}^{\mathsf{n}})} \Big(Q(\mathcal{D}^{\mathsf{x}} \cup \sigma_{<f} \cup \{f\}) - Q(\mathcal{D}^{\mathsf{x}} \cup \sigma_{<f})\Big) \tag{14}$$

Recent work [16] has shown a reduction from computing $\mathsf{Shapley}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(f)$ to computing the quantity $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$ that is defined below.

Definition 5.13 ($\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$). Let Q be a Boolean Conjunctive Query, and $\mathcal{D}$ be a database instance that is divided into an exogenous part $\mathcal{D}^{\mathsf{x}}$ and an endogenous part $\mathcal{D}^{\mathsf{n}}$. Given a number $k \in \mathbb{N}$, we define $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$ as the number of possible subsets $\mathcal{D}' \subseteq \mathcal{D}^{\mathsf{n}}$ of size $|\mathcal{D}'| = k$ such that $Q(\mathcal{D}^{\mathsf{x}} \cup \mathcal{D}')$ is true.

We summarize the reduction by [16] from $\mathsf{Shapley}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(f)$ to $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$:

$$\begin{aligned}
\mathsf{Shapley}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(f) &= \sum_{\mathcal{D}' \subseteq \mathcal{D}^{\mathsf{n}} \setminus \{f\}} \frac{|\mathcal{D}'|! \cdot (|\mathcal{D}^{\mathsf{n}}| - |\mathcal{D}'| - 1)!}{|\mathcal{D}^{\mathsf{n}}|!} \cdot (Q(\mathcal{D}^{\mathsf{x}} \cup \mathcal{D}' \cup \{f\}) - Q(\mathcal{D}^{\mathsf{x}} \cup \mathcal{D}')) \\
&= \sum_{\mathcal{D}' \subseteq \mathcal{D}^{\mathsf{n}} \setminus \{f\}} \frac{|\mathcal{D}'|! \cdot (|\mathcal{D}^{\mathsf{n}}| - |\mathcal{D}'| - 1)!}{|\mathcal{D}^{\mathsf{n}}|!} \cdot Q(\mathcal{D}^{\mathsf{x}} \cup \mathcal{D}' \cup \{f\}) \\
&\quad - \sum_{\mathcal{D}' \subseteq \mathcal{D}^{\mathsf{n}} \setminus \{f\}} \frac{|\mathcal{D}'|! \cdot (|\mathcal{D}^{\mathsf{n}}| - |\mathcal{D}'| - 1)!}{|\mathcal{D}^{\mathsf{n}}|!} \cdot Q(\mathcal{D}^{\mathsf{x}} \cup \mathcal{D}') \\
&= \sum_{k=0}^{|\mathcal{D}^{\mathsf{n}}|-1} \frac{k! \cdot (|\mathcal{D}^{\mathsf{n}}| - k - 1)!}{|\mathcal{D}^{\mathsf{n}}|!} \cdot \#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}} \cup \{f\},\mathcal{D}^{\mathsf{n}} \setminus \{f\}}(k) \\
&\quad - \sum_{k=0}^{|\mathcal{D}^{\mathsf{n}}|-1} \frac{k! \cdot (|\mathcal{D}^{\mathsf{n}}| - k - 1)!}{|\mathcal{D}^{\mathsf{n}}|!} \cdot \#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}} \setminus \{f\}}(k) \\
&= \sum_{k=0}^{|\mathcal{D}^{\mathsf{n}}|-1} \frac{k! \cdot (|\mathcal{D}^{\mathsf{n}}| - k - 1)!}{|\mathcal{D}^{\mathsf{n}}|!} \cdot \Big(\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}} \cup \{f\},\mathcal{D}^{\mathsf{n}} \setminus \{f\}}(k) - \#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}} \setminus \{f\}}(k)\Big)
\end{aligned}$$

We now explain how to instantiate Algorithm 1 to compute $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$. Expectedly, the instantiation of our algorithm in this case is similar in spirit to the algorithm from [16]: Our Algorithm 1 uses the 2-monoid abstraction to *generalize* the algorithm from [16] to other application domains. The specific 2-monoid we use in this case is defined as follows.

Definition 5.14 (2-monoid for computing $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$). We define a specific 2-monoid $\boldsymbol{K} = (K, \oplus, \otimes)$ as follows. Let $\mathbb{B} \stackrel{\text{def}}{=} \{\text{true}, \text{false}\}$. The domain K is $\mathbb{N}^{\mathbb{N}\times\mathbb{B}}$, i.e., the set of all vectors of natural numbers that are indexed by pairs of natural numbers and Boolean values. The $\oplus$ and $\otimes$ operations are defined as follows for any pair of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2 \in K$:

$$(\boldsymbol{x}_1 \oplus \boldsymbol{x}_2)(i,b) \stackrel{\text{def}}{=} \sum_{i_1,i_2\in\mathbb{N}:i_1+i_2=i} \; \sum_{b_1,b_2\in\mathbb{B}:b_1\vee b_2=b} \boldsymbol{x}_1(i_1,b_1)\times\boldsymbol{x}_2(i_2,b_2), \qquad \forall i\in\mathbb{N}, b\in\mathbb{B} \tag{15}$$

$$(\boldsymbol{x}_1 \otimes \boldsymbol{x}_2)(i,b) \stackrel{\text{def}}{=} \sum_{i_1,i_2\in\mathbb{N}:i_1+i_2=i} \; \sum_{b_1,b_2\in\mathbb{B}:b_1\wedge b_2=b} \boldsymbol{x}_1(i_1,b_1)\times\boldsymbol{x}_2(i_2,b_2), \qquad \forall i\in\mathbb{N}, b\in\mathbb{B} \tag{16}$$

The identities $\mathbf{0}$ and $\mathbf{1}$ for $\oplus$ and $\otimes$ are as follows:

$$\mathbf{0}(i,b) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } i=0 \text{ and } b=\text{false} \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{1}(i,b) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } i=0 \text{ and } b=\text{true} \\ 0 & \text{otherwise} \end{cases}$$

Note that the above 2-monoid does *not* satisfy the annihilation-by-zero property: $a \otimes \mathbf{0} = \mathbf{0}$ for all $a \in K$. It does however satisfy the weaker property $\mathbf{0} \otimes \mathbf{0} = \mathbf{0}$ that is required by Definition 5.6.

Given Q, $\mathcal{D}^{\mathsf{x}}$ and $\mathcal{D}^{\mathsf{n}}$, the following definition constructs a $\boldsymbol{K}$-annotated database instance $\psi(\mathcal{D}^{\mathsf{x}}, \mathcal{D}^{\mathsf{n}})$ that is used as input to Algorithm 1. In addition to the identity vectors $\mathbf{0}$ and $\mathbf{1}$, we also need to use the following vector $\star \in K$:

$$\star(i,b) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } (i=0 \text{ and } b=\text{false}) \text{ or } (i=1 \text{ and } b=\text{true}) \\ 0 & \text{otherwise} \end{cases}$$

Definition 5.15 (Input $\boldsymbol{K}$-annotated database instance for computing $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k)$). Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be the 2-monoid from Definition 5.14, Q be a SJF-BCQ and $\mathcal{D}^{\mathsf{x}}$ and $\mathcal{D}^{\mathsf{n}}$ be two database instances for Q. We define a function ψ that maps a fact $R(\boldsymbol{x})$ where $R(\boldsymbol{X}) \in \text{at}(Q)$ and $\boldsymbol{x} \in \text{Dom}^{\boldsymbol{X}}$ to a vector in K as follows:

$$\psi(R(\boldsymbol{x})) \stackrel{\text{def}}{=} \begin{cases} \mathbf{1} & \text{if } R(\boldsymbol{x}) \in \mathcal{D}^{\mathsf{x}} \\ \star & \text{if } R(\boldsymbol{x}) \notin \mathcal{D}^{\mathsf{x}} \text{ and } R(\boldsymbol{x}) \in \mathcal{D}^{\mathsf{n}} \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{17}$$

Moreover, we define $\psi(\mathcal{D}^{\mathsf{x}}, \mathcal{D}^{\mathsf{n}})$ as a $\boldsymbol{K}$-annotated database instance where for every $R(\boldsymbol{X}) \in \text{at}(Q)$ and $\boldsymbol{x} \in \text{Dom}^{\boldsymbol{X}}$, the fact $R(\boldsymbol{x})$ is annotated with $\psi(R(\boldsymbol{x}))$.

The following theorem will be proved in the next section.

Theorem 5.16. *Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be the 2-monoid from Definition 5.14, Q be a hierarchical SJF-BCQ, $\mathcal{D}^{\mathsf{x}}$ and $\mathcal{D}^{\mathsf{n}}$ be two database instances for Q, and $\psi(\mathcal{D}^{\mathsf{x}}, \mathcal{D}^{\mathsf{n}})$ be the $\boldsymbol{K}$-annotated database instance from Definition 5.15. Then using Q and $\psi(\mathcal{D}^{\mathsf{x}}, \mathcal{D}^{\mathsf{n}})$ as inputs, Algorithm 1 runs in time in $O((|\mathcal{D}^{\mathsf{x}}| + |\mathcal{D}^{\mathsf{n}}|) \cdot |\mathcal{D}^{\mathsf{n}}|^2)$ and space $O((|\mathcal{D}^{\mathsf{x}}| + |\mathcal{D}^{\mathsf{n}}|) \cdot |\mathcal{D}^{\mathsf{n}}|)$ and outputs a vector $q \in K$ that satisfies the following:*

$$q(i, \text{true}) = \#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(i), \qquad \forall i \in \mathbb{N}$$

6 Correctness and Runtime Analysis of the Unifying Algorithm

In this section, we give a correctness proof and a runtime analysis of Algorithm 1, both of which apply regardless of the specific application and the specific 2-monoid that is being used. We then show how to instantiate both the correctness proof and the runtime analysis to each one of the three problems we consider in this paper.

6.1 General Correctness Proof

Definition 6.1 (Provenance Tree). Let Σ be a set of symbols. A *provenance tree* is a rooted tree whose leafs are labeled with symbols from $\Sigma \cup \{\text{true}, \text{false}\}$ and whose internal nodes are labeled with either $\wedge$ or $\vee$. A provenance tree is called *decomposable* if its leaves have distinct labels, including the labels true and false.[8] Given a provenance tree x, the *support* of x, denoted $\text{supp}(x)$, is the set of all symbols that appear in the leaves of x, excluding true and false.

Definition 6.2 (Provenance 2-monoid). Let Σ be a set of symbols. The *provenance 2-monoid* is a 2-monoid $\overline{\boldsymbol{K}} = (\overline{K}, \overline{\oplus}, \overline{\otimes})$ defined as follows.

- The domain $\overline{K}$ is the set of all provenance trees over $\Sigma \cup \{\text{true}, \text{false}\}$.
- The $\overline{\oplus}$ operator takes two provenance trees x and y and returns a new provenance tree z that has a root labeled with $\vee$ and whose children are the roots of x and y.
- The $\overline{\otimes}$ operator takes two provenance trees x and y and returns a new provenance tree z that has a root labeled with $\wedge$ and whose children are the roots of x and y.

In order to ensure commutativity of $\overline{\oplus}$ and $\overline{\otimes}$, we treat the children of any node as an unordered set. In order to ensure the associativity of $\overline{\oplus}$ and $\overline{\otimes}$, whenever we have a node that has the same label as its parent, we merge them into a single node. The identity for $\overline{\oplus}$ is the provenance tree with a single leaf labeled with false, and the identity for $\overline{\otimes}$ is the provenance tree with a single leaf labeled with true.

Lemma 6.3 (Algorithm 1 produces *decomposable* provenance trees). *Given a hierarchical SJF-BCQ Q and a database instance $\overline{\mathcal{D}}$, suppose that we annotate every fact in $\overline{\mathcal{D}}$ with a* unique *symbol from a set of symbols Σ. Consider the provenance 2-monoid $\overline{\boldsymbol{K}} = (\overline{K}, \overline{\oplus}, \overline{\otimes})$ from Definition 6.2 where $\overline{K}$ is the set of provenance trees over $\Sigma \cup \{\text{true}, \text{false}\}$. The output of Algorithm 1 on Q and $\overline{\mathcal{D}}$ is a provenance tree that is* decomposable *(Definition 6.1). Moreover, every $\overline{\boldsymbol{K}}$-annotation that is computed throughout the algorithm is also a provenance tree that is* decomposable.

Proof. In order to prove the above lemma, we inductively prove that the algorithm maintains the following invariant. Recall the notion of *support* of a provenance tree from Definition 6.1.

Claim 6.1 (Invariant of Algorithm 1). *Every fact is annotated with a provenance tree that is decomposable. Moreover, all facts have provenance trees whose supports are pairwise disjoint.*

Initially, the above claim holds because every fact is annotated with a provenance tree consisting of a single leaf labeled with a unique symbol from Σ. We now prove that applying both Rule 1 and Rule 2 maintains the above invariant.

For Rule 1 (line 2 of Algorithm 1), every fact $R'(\boldsymbol{x}')$ for $\boldsymbol{x}' \in \text{Dom}^{X'}$ is annotated with a provenance tree which is the $\overline{\oplus}$ of provenance trees of all facts $R(\boldsymbol{x}', y)$ for $y \in \text{Dom}$. If all these provenance trees are decomposable and have disjoint supports, then the resulting provenance

[8] The labels true and false do not need to appear in any trees except for the two trivial trees: the tree that consists of a single node which is true, and the tree that consists of a single false node. This is because whenever true appears under a conjunction, it can be dropped, and whenever it appears under a disjunction, the entire disjunction becomes true. Similar simplifications apply for false, and these simplifications can always eliminate true and false from any node in a given tree, except for the root node.

tree is also decomposable and has a support that is the union of the supports of all the children. Moreover, notice that the provenance tree of any fact $R(\boldsymbol{x}', y)$ is only used in the provenance tree of a single fact $R'(\boldsymbol{x}')$. Therefore, after we replace the atom $R(\boldsymbol{X})$ with $R'(\boldsymbol{X}')$ (thus replacing all facts $R(\boldsymbol{x}', y)$ with new facts $R'(\boldsymbol{x}')$), the resulting facts will still have annotations whose supports are pairwise disjoint.

For Rule 2 (line 6 of Algorithm 1), every fact $R'(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathrm{Dom}^X$ is annotated with a provenance tree which is the $\overline{\otimes}$ of provenance trees of the facts $R_1(\boldsymbol{x})$ and $R_2(\boldsymbol{x})$. If these two provenance trees are decomposable and have disjoint supports, then the resulting provenance tree is also decomposable and has a support that is the union of their supports. Moreover, every fact $R_1(\boldsymbol{x})$ or $R_2(\boldsymbol{x})$ is only used in the provenance tree of a single fact $R'(\boldsymbol{x})$. Therefore, after we replace the two atoms $R_1(\boldsymbol{X})$ and $R_2(\boldsymbol{X})$ with $R'(\boldsymbol{X})$ (thus replacing all facts $R_1(\boldsymbol{x})$ and $R_2(\boldsymbol{x})$ with new facts $R'(\boldsymbol{x})$), the resulting facts will still have annotations whose supports are pairwise disjoint. □

The following theorem shows how to use the provenance 2-monoid from Definition 6.2 as a *universal* 2-monoid to analyze the behavior of Algorithm 1 on any 2-monoid, and prove its correctness.

Theorem 6.4 (Algorithm 1 is correct). *Let Q be a hierarchical SJF-BCQ and $\overline{\mathcal{D}}$ be a database instance where every fact is annotated with a unique symbol from a set of symbols Σ. Let $\overline{\boldsymbol{K}} = (\overline{K}, \overline{\oplus}, \overline{\otimes})$ be the provenance 2-monoid from Definition 6.2 where $\overline{K}$ is the set of provenance trees over $\Sigma \cup \{\mathsf{true}, \mathsf{false}\}$, and let $\overline{\mathbf{0}}$ and $\overline{\mathbf{1}}$ be the identities. Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be another 2-monoid with identities $\mathbf{0}$ and $\mathbf{1}$. Suppose we are given a function $\phi : \overline{K} \to K$ that maps provenance trees to values in K and satisfies the following properties, for all x and y in $\overline{K}$ that are decomposable and have disjoint supports:*

$$\phi(x \overline{\oplus} y) = \phi(x) \oplus \phi(y) \tag{18}$$

$$\phi(x \overline{\otimes} y) = \phi(x) \otimes \phi(y) \tag{19}$$

$$\phi(\overline{\mathbf{0}}) = \mathbf{0} \tag{20}$$

Let $\mathcal{D}$ be the $\boldsymbol{K}$-annotated database instance obtained by replacing every $\overline{\boldsymbol{K}}$-annotation in $\overline{\mathcal{D}}$ with its ϕ-mapping, i.e. its corresponding value in K using ϕ. Then, the output of Algorithm 1 on Q and $\mathcal{D}$ is the ϕ-mapping of the output of Algorithm 1 on Q and $\overline{\mathcal{D}}$.

Note that we do *not* require Eq. (18) and Eq. (19) to hold for *all* provenance trees in $\overline{K}$, but only for those that are decomposable and have disjoint supports. This is going to be crucial because in all applications we consider, Eq. (18) and Eq. (19) are *only* going to hold for decomposable provenance trees with disjoint supports.

Proof. To prove the above theorem, we consider two simultaneous runs of Algorithm 1. The first run is on Q and $\mathcal{D}$, while the second is on Q and $\overline{\mathcal{D}}$. We show that they maintain the invariant:

Claim 6.2. *Every annotation of a fact that is produced by the first run is the ϕ-mapping of the annotation of the same fact that is produced by the second run.*

The above claim initially holds by the construction of $\mathcal{D}$. Thanks to Lemma 6.3, all provenance trees that are produced by the second run are decomposable. Given two provenance trees x and y, if $x \overline{\oplus} y$ (or $x \overline{\otimes}$ y) is decomposable, then x and y are also decomposable and have disjoint supports. Hence, both Eq. (18) and Eq. (19) hold, thus they inductively prove the invariant. □

6.2 General Runtime Analysis

Definition 6.5 (Size of a $\boldsymbol{K}$-annotated database). Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be a 2-monoid where $\oplus$ and $\otimes$ have identities $\mathbf{0}$ and $\mathbf{1}$ respectively. Given a $\boldsymbol{K}$-annotated relation $R(\boldsymbol{X})$, the *support* of $R(\boldsymbol{X})$, denoted $\text{supp}(R(\boldsymbol{X}))$, is the set of all facts $R(\boldsymbol{x})$ in $R(\boldsymbol{X})$ whose $\boldsymbol{K}$-annotation is not $\mathbf{0}$. The *size* of $R(\boldsymbol{X})$ is the size of its support, i.e., $|\text{supp}(R(\boldsymbol{X}))|$. The support of a $\boldsymbol{K}$-annotated database instance $\mathcal{D}$, denoted $\text{supp}(\mathcal{D})$, is the union of the supports of its relations. The *size* of a $\boldsymbol{K}$-annotated database instance $\mathcal{D}$, denoted $|\mathcal{D}|$, is the sum of the sizes of its relations.

In order to analyze the runtime of Algorithm 1, we use the following lemma.

Lemma 6.6. *Throughout Algorithm 1, the size of the $\boldsymbol{K}$-annotated database instance $\mathcal{D}$ never increases.*

Proof. Algorithm 1 updates the $\boldsymbol{K}$-annotated database instance $\mathcal{D}$ in two places, namely lines 5 and 8. We show below that neither update increases the size of $\mathcal{D}$.

- In line 5, we replace the atom $R(\boldsymbol{X}', Y)$ with $R'(\boldsymbol{X}')$. We argue that the following holds: $\text{supp}(R'(\boldsymbol{X}')) \subseteq \pi_{\boldsymbol{X}'}\text{supp}(R(\boldsymbol{X}', Y))$. This is because for any given $\boldsymbol{x}' \in \text{Dom}^{\boldsymbol{X}'}$, we have $R'(\boldsymbol{x}') \stackrel{\text{def}}{=} \bigoplus_{y \in \text{Dom}} R(\boldsymbol{x}', y)$ and $\mathbf{0}$ is the identity for $\oplus$.
- In line 8, we replace the two atoms $R_1(\boldsymbol{X})$ and $R_2(\boldsymbol{X})$ with a single atom $R'(\boldsymbol{X})$. We argue that $\text{supp}(R'(\boldsymbol{X})) \subseteq \text{supp}(R_1(\boldsymbol{X})) \cup \text{supp}(R_2(\boldsymbol{X}))$, which implies $|\text{supp}(R'(\boldsymbol{X}))| \leq |\text{supp}(R_1(\boldsymbol{X}))| + |\text{supp}(R_2(\boldsymbol{X}))|$. This holds because for any given $\boldsymbol{x} \in \text{Dom}^{\boldsymbol{X}}$, we have $R'(\boldsymbol{x}) = R_1(\boldsymbol{x}) \otimes R_2(\boldsymbol{X})$, and a 2-monoid must satisfy $\mathbf{0} \otimes \mathbf{0} = \mathbf{0}$ (Definition 5.6).

□

The following theorem is immediate.

Theorem 6.7 (Algorithm 1 used a linear number of $\oplus$ and $\otimes$ operations). *Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be a 2-monoid, Q be a SJF-BCQ, and $\mathcal{D}$ be a $\boldsymbol{K}$-annotated database instance of size $|\mathcal{D}|$. On inputs Q and $\mathcal{D}$, the total number of $\oplus$ and $\otimes$ operations performed by Algorithm 1 is $O(|\mathcal{D}|)$.*

We are now ready to instantiate the correctness proof and the runtime analysis to each of the three problems considered in this paper. In particular, we will show how to apply Theorem 6.4 and Theorem 6.7 to prove Theorems 5.8, 5.11, and 5.16.

6.3 First Instantiation: Probabilistic Query Evaluation

Proof of Theorem 5.8. First, we prove correctness using Theorem 6.4. We define the function $\phi : \overline{K} \to K$ from Theorem 6.4 as follows: Given a provenance tree x, $\phi(x)$ is the probability of the Boolean formula F_x corresponding to the provenance tree x evaluating to true.

Given two decomposable provenance trees x and y with disjoint supports, consider the Boolean formulas F_x and F_y corresponding to x and y respectively. Because the supports are disjoint, the two formulas evaluating to true are independent events. Let p_1 and p_2 be the probabilities of F_x and F_y, respectively, evaluating to true. By independence, the probability of $F_x \vee F_y$ is given by $p_1 \oplus p_2$ from Eq. (3), while the probability of $F_x \wedge F_y$ is given by $p_1 \otimes p_2$ from Eq. (2). Therefore, our definition of ϕ satisfies Eq. (18) and Eq. (19), and Theorem 6.4 applies. By the theorem, the output of Algorithm 1 on Q and $\mathcal{D}$ is the probability of the Boolean formula corresponding to the provenance tree of Q evaluating to true, as desired.

Finally, the linear runtime follows from Theorem 6.7 and the fact that the $\oplus$ and $\otimes$ operators from Eq. (3) and Eq. (2) take constant time. □

6.4 Second Instantiation: Bag-Set Maximization

Proof of Theorem 5.11. First, we prove correctness using Theorem 6.4. We define the function ϕ from Theorem 6.4 as follows: Given a provenance tree x, let F_x be the corresponding Boolean formula. We define $\phi(x)$ as a monotonic vector in $\mathbb{N}^{\mathbb{N}}$ where for every $i \in \mathbb{N}$, the i-th entry is the maximum multiplicity of F_x that can be achieved with a repair cost at most i, i.e., by adding no more than i facts from the repair database $\mathcal{D}^{\mathsf{r}}$.

Given two decomposable provenance trees x and y with disjoint supports, consider the Boolean formulas F_x and F_y corresponding to x and y respectively. Because the supports are disjoints, these two formulas have to be repaired independently. In particular, to repair the formula $F_x \vee F_y$ within a repair cost of i, we need to find the best way to repair F_x within a repair cost of i_1 and F_y within a repair cost of i_2 where $i_1 + i_2 = i$. Therefore, $\phi(x \overline{\oplus} y) = \phi(x) \oplus \phi(y)$ where $\oplus$ is given by Eq. (10). The same reasoning goes for repairing the formula $F_x \wedge F_y$, leading to $\phi(x \overline{\otimes} y) = \phi(x) \otimes \phi(y)$ where $\otimes$ is given by Eq. (11). Therefore, our definition of ϕ satisfies Eq. (18) and Eq. (19), and Theorem 6.4 applies. The theorem implies that the output of Algorithm 1 on Q and $\mathcal{D}$ is a monotonic vector in $\mathbb{N}^{\mathbb{N}}$ whose i-th entry is the maximum multiplicity for Q that can be achieved with a repair cost at most i, as desired.

Finally, we use Theorem 6.7 to prove that the time complexity is $O((|\mathcal{D}| + |\mathcal{D}^{\mathsf{r}}|) \cdot |\mathcal{D}^{\mathsf{r}}|^2)$ and the space complexity is $O((|\mathcal{D}| + |\mathcal{D}^{\mathsf{r}}|) \cdot |\mathcal{D}^{\mathsf{r}}|)$. Given an input instance $(\mathcal{D}, \mathcal{D}^{\mathsf{r}}, \theta)$ to the Bag-Set Maximization problem from Definition 4.1, note that the maximum repair cost θ is always upper bounded by the size of the repair database $\mathcal{D}^{\mathsf{r}}$. Let $\boldsymbol{K} = (K, \oplus, \otimes)$ be the 2-monoid from Definition 5.9, and $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be two vectors in K. Note that we only need to maintain the first $\theta + 1$ entries of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Hence, the sizes of these vectors can be assumed to be linear in θ, and by extension, in $|\mathcal{D}^{\mathsf{r}}|$. As a result, the $\oplus$ and $\otimes$ operators from Eq. (10) and Eq. (11) take time $O(|\mathcal{D}^{\mathsf{r}}|^2)$ and space $O(|\mathcal{D}^{\mathsf{r}}|)$. By Theorem 6.7, the total number of $\oplus$ and $\otimes$ operations performed by Algorithm 1 is $O(|\psi(\mathcal{D}, \mathcal{D}^{\mathsf{r}})|) = O(|\mathcal{D}| + |\mathcal{D}^{\mathsf{r}}|)$, where $\psi(\mathcal{D}, \mathcal{D}^{\mathsf{r}})$ is the $\boldsymbol{K}$-annotated database instance from Definition 5.10. This gives the desired complexity. □

6.5 Third Instantiation: Shapley Value Computation

Proof of Theorem 5.16. First, we prove correctness. In order to apply Theorem 6.4, we define the function ϕ as follows. For convenience, we extend Definition 5.13 to define $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k,b)$ (with an extra Boolean parameter $b \in \mathbb{B}$) as the number of possible subsets $\mathcal{D}' \subseteq \mathcal{D}^{\mathsf{n}}$ of size $|\mathcal{D}'| = k$ such that $Q(\mathcal{D}^{\mathsf{x}} \cup \mathcal{D}')$ is b. This way, we can think of $\#\mathsf{Sat}_{Q,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}}(k,b)$ as a vector in $\mathbb{N}^{\mathbb{N}\times\mathbb{B}}$ that is indexed by $(k,b) \in \mathbb{N} \times \mathbb{B}$. In addition, we also extend Definition 5.13 from Boolean conjunctive queries Q to arbitrary Boolean formulas F. Given a provenance tree x, let F_x be the corresponding Boolean formula, and $\mathcal{D}^{\mathsf{n}}[F_x]$ be the set of facts in $\mathcal{D}^{\mathsf{n}}$ that appear in F_x. We define $\phi(x)$ as follows:

$$\phi(x) \stackrel{\text{def}}{=} \#\mathsf{Sat}_{F_x,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_x]} \tag{21}$$

Let x and y be two decomposable provenance trees with disjoint supports, F_x and F_y be the corresponding Boolean formulas, and $\mathcal{D}^{\mathsf{n}}[F_x]$ and $\mathcal{D}^{\mathsf{n}}[F_y]$ be the sets of facts in $\mathcal{D}^{\mathsf{n}}$ that appear in F_x and F_y respectively. Note that $\mathcal{D}^{\mathsf{n}}[F_x]$ and $\mathcal{D}^{\mathsf{n}}[F_y]$ are disjoint, and $\mathcal{D}^{\mathsf{n}}[F_x \vee F_y]$ is their union. As a result we have:

$$\#\mathsf{Sat}_{F_x\vee F_y,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_x\vee F_y]}(k,b) = \sum_{k_1,k_2\in\mathbb{N}:k_1+k_2=k} \; \sum_{b_1,b_2\in\mathbb{B}:b_1\vee b_2=b} \#\mathsf{Sat}_{F_x,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_x]}(k_1,b_1) \times \#\mathsf{Sat}_{F_y,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_y]}(k_2,b_2)$$

$$\#\mathsf{Sat}_{F_x\vee F_y,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_x\vee F_y]} = \#\mathsf{Sat}_{F_x,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_x]} \oplus \#\mathsf{Sat}_{F_y,\mathcal{D}^{\mathsf{x}},\mathcal{D}^{\mathsf{n}}[F_y]}$$

$$\phi(x \overline{\oplus} y) \quad = \quad \phi(x) \quad \oplus \quad \phi(y)$$

where the $\oplus$ operator above is given by Eq. (15). Using a similar reasoning for $F_x \wedge F_y$, we have:

$$\#\mathrm{Sat}_{F_x \wedge F_y, \mathcal{D}^x, \mathcal{D}^n[F_x \wedge F_y]}(k, b) = \sum_{k_1, k_2 \in \mathbb{N}: k_1 + k_2 = k} \; \sum_{b_1, b_2 \in \mathbb{B}: b_1 \wedge b_2 = b} \#\mathrm{Sat}_{F_x, \mathcal{D}^x, \mathcal{D}^n[F_x]}(k_1, b_1) \times \#\mathrm{Sat}_{F_y, \mathcal{D}^x, \mathcal{D}^n[F_y]}(k_2, b_2)$$

$$\#\mathrm{Sat}_{F_x \wedge F_y, \mathcal{D}^x, \mathcal{D}^n[F_x \wedge F_y]} \quad = \quad \#\mathrm{Sat}_{F_x, \mathcal{D}^x, \mathcal{D}^n[F_x]} \quad \otimes \quad \#\mathrm{Sat}_{F_y, \mathcal{D}^x, \mathcal{D}^n[F_y]}$$

$$\phi(x \overline{\otimes} y) \quad = \quad \phi(x) \quad \otimes \quad \phi(y)$$

Hence, Theorem 6.4 applies.

Finally, we prove that the time complexity is $O((|\mathcal{D}^x| + |\mathcal{D}^n|) \cdot |\mathcal{D}^n|^2)$ and the space complexity is $O((|\mathcal{D}^x| + |\mathcal{D}^n|) \cdot |\mathcal{D}^n|)$. Note that by the definition of $\#\mathrm{Sat}_{Q, \mathcal{D}^x, \mathcal{D}^n}(k, b)$, the value of k cannot exceed $|\mathcal{D}^n|$. Hence, the $\oplus$ and $\otimes$ operators from Eq. (15) and Eq. (16) run in time $O(|\mathcal{D}^n|^2)$ and space $O(|\mathcal{D}^n|)$. By Theorem 6.7, the total number of $\oplus$ and $\otimes$ operations performed is $O(|\psi(\mathcal{D}^x, \mathcal{D}^n)|) = O(|\mathcal{D}^x| + |\mathcal{D}^n|)$, where $\psi(\mathcal{D}^x, \mathcal{D}^n)$ is the $\boldsymbol{K}$-annotated database instance from Definition 5.15. □

7 Concluding Remarks

In this paper, we established a dichotomy theorem for the computational complexity of the Bag-Set Maximization problem, parameterized by the collection of self-join free Boolean conjunctive queries. Furthermore, we showed that the boundary of this dichotomy is defined by the collection of hierarchical queries. More importantly, perhaps, we discovered a single unifying polynomial-time algorithm for hierarchical queries that applies not only to Bag-Set Maximization but also to Probabilistic Query Evaluation and to Shapley Value Computation.

The work presented here motivates several different questions, including the following two.

Question 1: Is there a dichotomy for Bag-Set Maximization for the collection of *all* conjunctive queries? If so, what is the boundary of this dichotomy?

Note that in the case of Probabilistic Query Evaluation, Dalvi and Suciu [5] established a dichotomy for the collection of all conjunctive queries. In the case of Shapley Value Computation, however, no dichotomy theorem for the collection of all conjunctive queries is currently known.

Question 2: Are there other natural algorithmic problems in database theory that are tractable for hierarchical queries, and this tractability can be obtained via the unifying polynomial-time algorithm presented here?

Among the candidates are the problems of constant-delay enumeration of conjunctive query answers under updates, as well as counting answers under updates [2]. In both problems, hierarchical queries once again show up in the dichotomy between tractable and intractable cases. It is not clear, however, how to view this tractability result as an instantiation of the unifying algorithm presented in this paper or what 2-monoid to use. Recent work [11] generalizes this dichotomy for conjunctive queries under updates along two dimensions: (i) queries with *free access patterns*, and (ii) queries over *probabilistic databases*. It would be interesting to see if these generalizations can also be captured by our framework.

Acknowledgments

This work was initiated while Abo Khamis, Kolaitis, Roy, and Tannen participated in the Fall 2023 program on *Logic and Algorithms in Databases and AI* at the Simons Institute for the Theory of Computing. Part of this work was done while Roy was a visiting scientist at RelationalAI.

References

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

[2] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries under Updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) *(PODS '17)*. Association for Computing Machinery, New York, NY, USA, 303–318. doi:10.1145/3034786.3034789

[3] Liming Cai and Jianer Chen. 1997. On Fixed-Parameter Tractability and Approximability of NP Optimization Problems. *J. Comput. System Sci.* 54, 3 (1997), 465–474. doi:10.1006/jcss.1997.1490

[4] Nilesh Dalvi and Dan Suciu. 2013. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6, Article 30 (Jan. 2013), 87 pages. doi:10.1145/2395116.2395119

[5] Nilesh N. Dalvi and Dan Suciu. 2007. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4 (2007), 523–544.

[6] Uriel Feige and Shimon Kogan. 2004. Hardness of approximation of the balanced complete bipartite subgraph problem. *Dept. Comput. Sci. Appl. Math., Weizmann Inst. Sci., Rehovot, Israel, Tech. Rep. MCS04-04* (2004).

[7] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory*. Springer. doi:10.1007/3-540-29953-X

[8] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2015. The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries. *Proc. VLDB Endow.* 9, 3 (2015), 180–191. doi:10.14778/2850583.2850592

[9] 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 *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, Dan Suciu, Yufei Tao, and Zhewei Wei (Eds.). ACM, 271–284. doi:10.1145/3375395.3387647

[10] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[11] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2025. Conjunctive Queries with Free Access Patterns under Updates. *Logical Methods in Computer Science* Volume 21, Issue 2, Article 23 (Jun 2025). doi:10.46298/lmcs-21(2:23)2025

[12] Subhash Khot. 2006. Ruling Out PTAS for Graph Min-Bisection, Dense k-Subgraph, and Bipartite Clique. *SIAM J. Comput.* 36, 4 (2006), 1025–1071. doi:10.1137/S0097539705447037

[13] Mark W. Krentel. 1988. The Complexity of Optimization Problems. *J. Comput. Syst. Sci.* 36, 3 (1988), 490–509. doi:10.1016/0022-0000(88)90039-6

[14] Richard E. Ladner. 1975. On the Structure of Polynomial Time Reducibility. *J. ACM* 22, 1 (1975), 155–171. doi:10.1145/321864.321877

[15] Bingkai Lin. 2018. The Parameterized Complexity of the k-Biclique Problem. *J. ACM* 65, 5 (2018), 34:1–34:23. doi:10.1145/3212622

[16] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2021. The Shapley Value of Tuples in Query Answering. *Log. Methods Comput. Sci.* 17, 3 (2021). doi:10.46298/LMCS-17(3:22)2021

[17] 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 (2023), 228:1–228:27. doi:10.1145/3626715

[18] Jeffrey D. Ullman and Jennifer Widom. 1997. *A first course in database systems*. Prentice-Hall, Inc., USA.

Received June 2025; accepted August 2025