The Dichotomy of Conjunctive Queries on Probabilistic Structures*

Nilesh Dalvi
University of Washington
Seattle, WA
nilesh@cs.washington.edu

Dan Suciu
University of Washington
Seattle, WA
suciu@cs.washington.edu

ABSTRACT

We show that for every conjunctive query, the complexity of evaluating it on a probabilistic database is either PTIME or #P-complete, and we give an algorithm for deciding whether a given conjunctive query is PTIME or #P-complete. The dichotomy property is a fundamental result on query evaluation on probabilistic databases and it gives a complete classification of the complexity of conjunctive queries.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic**]; G.3 [**Probability and statistics**]; H.2.5 [**Heterogeneous databases**]

General Terms

Algorithms, Management, Theory

Keywords

probabilistic databases, query processing

1. PROBLEM STATEMENT

Fix a relational vocabulary $R_1, \ldots, R_k$, denoted $\mathcal{R}$. A *tuple-independent probabilistic structure* is a pair $(\mathbf{A}, p)$ where $\mathbf{A} = (A, R_1^A, \ldots, R_k^A)$ is first order structure and p is a function that associates to each tuple t in $\mathbf{A}$ a rational number $p(t) \in [0, 1]$. A probabilistic structure $(\mathbf{A}, p)$ induces a probability distribution on the set of substructures $\mathbf{B}$ of $\mathbf{A}$ by:

$$p(\mathbf{B}) = \prod_{i=1}^{k} \Big(\prod_{t \in R_i^B} p(t) \times \prod_{t \in R_i^A - R_i^B} (1 - p(t)) \Big) \quad (1)$$

where $\mathbf{B} \subseteq \mathbf{A}$, more precisely $\mathbf{B} = (A, R_1^B, \ldots, B_k^B)$ is s.t. $R_i^B \subseteq R_i^A$ for $i = 1, k$.

*This research was supported in part by Suciu's NSF CAREER grant IIS-0092955, NSF grants IIS-0415193, IIS-0513877, IIS-0428168, and a gift from Microsoft.

PODS'07, June 11–13, 2007, Beijing, China.

A *conjunctive query*, q, is a sentence of the form

$$\exists \bar{x}.(\varphi_1 \wedge \ldots \wedge \varphi_m)$$

where each φ_i is a positive atomic predicate $R(t)$, called a *subgoal*, and the tuple t consists of variables and/or constants. As usual, we drop the existential quantifiers and the $\wedge$, writing $q = \varphi_1, \varphi_2, \ldots, \varphi_m$. A *conjunctive property* is a property on structures defined by a conjunctive query q, and its probability on a probabilistic structure $(\mathbf{A}, p)$ is defined as:

$$p(q) = \sum_{\mathbf{B} \subseteq \mathbf{A}: \mathbf{B} \models q} p(\mathbf{B}) \quad (2)$$

In this paper we study the data complexity of Boolean conjunctive properties on tuple independent probabilistic structures. (When clear from the context we blur the distinction between queries and properties).

More precisely, for a fixed vocabulary and a Boolean conjunctive query q we study the following problem:

EVALUATION For a given probabilistic structure $(\mathbf{A}, p)$, compute the probability $p(q)$.

The complexity is in the size of $\mathbf{A}$ and in the size of the representations of the rational numbers $p(t)$. This problem is contained in #P, as discussed in [5]. We show here conditions under which it is in PTIME, and conditions where it is #P-hard. The class #P [12] is the counting analogue of the class NP.

THEOREM 1.1. *(Dichotomy Theorem) Given any conjunctive query q, the complexity of* EVALUATION *is either* PTIME *or* #P*-complete.*

Background and motivation Dichotomy theorems are fundamental to our understanding of the structure of conjunctive queries. A widely studied problem, which can be viewed as the dual of our problem, is the *constraint satisfaction problem* (CSP) and is as follows: given a fixed relational structure, what is the complexity of evaluating conjunctive queries over the structure? Shaefer [11] has shown that over binary domains, CSP has a dichotomy into PTIME and NP-complete. Feder and Vardi [6] have conjectured that a similar dichotomy holds for arbitrary (non-binary) domains. Creignou and Hermann [3] showed that the counting version of the CSP problem has a dichotomy into PTIME and #P-complete. The problem we study in this paper is different in nature, yet still interesting.

In addition to the pure theoretical interest we also have a practical motivation. Probabilistic databases are increasingly used to manage a wide range of imprecise data [13, 2]. But general purpose probabilistic database are difficult to build, because query evaluation is difficult: it is both theoretically hard (#P-hard [8, 4]) and

plain difficult to understand. All systems reported in the literature have circumvented the full query evaluation problem by either severely restricting the queries [1], or by using a non-scalable (exponential) evaluation algorithm [7], or by using a weaker semantics based on intervals [9]. In our own system, MystiQ [2], we support arbitrary conjunctive queries as follows. For queries without self-joins, we test if they have a PTIME plan using the techniques in [10]; if not, then we run a Monte Carlo simulation algorithm. The query execution times between the two cases differ by one or two orders of magnitude (seconds v.s. minutes). The desire to improve MystiQ's query performance on arbitrary queries (i.e. with self-joins) has partially motivated this work.

1.1 Overview of Results

We summarize here our main results on the query evaluation problem. Some of this discussion is informal and is intended to introduce the major concepts needed to understand the evaluation of conjunctive queries on probabilistic structures.

Hierarchical queries: For a conjunctive query q, let $Vars(q)$ denote its set of variables, and, for $x \in Vars(q)$, let $sg(x)$ be the set of sub-goals that contain x.

DEFINITION 1.2. *A conjunctive query is* hierarchical *if for any two variables x, y, either $sg(x) \cap sg(y) = \emptyset$, or $sg(x) \subseteq sg(y)$, or $sg(y) \subseteq sg(x)$. We write $x \sqsubseteq y$ whenever $sg(x) \subseteq sg(y)$ and write $x \equiv y$ when $sq(x) = sg(y)$. A conjunctive property is hierarchical if it is defined by some hierarchical conjunctive query.*

It is easy to check that a conjunctive property is hierarchical if the minimal conjunctive query defining it is hierarchical. As an example, the query $q_{\text{hier}} = R(x), S(x, y)$ is hierarchical because $sg(x) = \{R, S\}$, $sg(y) = \{S\}$. On the other hand, the query $q_{\text{non-h}} = R(x), S(x, y), T(y)$ is not hierarchical because $sg(x) = \{R, S\}$ and $sg(y) = \{S, T\}$.

In prior work [4] we have studied the evaluation problem under following restriction: every sub-goal of q refers to a different relation name. We say that q has no self-joins. The main result in [4], restated in the terminology used here, is:

THEOREM 1.3. *[4] Assume q has no self joins. Then: (1) If q is hierarchical, then it is in* PTIME. *(2) If q is not hierarchical then it is #*P*-hard.*

Moreover, the PTIME algorithm for a hierarchical query is the following simple recurrence on query's structure. Call a variable x *maximal* if for all y, $y \sqsupseteq x$ implies $x \sqsupseteq y$. In a hierarchical query, every connected component has a maximal variable. Let $f_0, f_1(x_1), \ldots, f_m(x_m)$ be the connected components of q, with each x_i a maximal variable: f_0 contains all constant sub-goals, and $f_i(x_i)$ consists of all sub-goals containing x_i for $i = 1, m$. Then:

$$p(q) \quad = \quad p(f_0) \cdot \prod_{i=1,m} (1 - \prod_{a \in A} (1 - p(f_i[a/x_i]))) \qquad (3)$$

This formula is a recurrence on the query's structure (since each $f_i[a/x_i]$ is simpler than q) and it is correct because $f_i[a/x_i]$ is independent from $f_j[a'/x_j]$ whenever $i \neq j$ or $a \neq a'$. As an example, for query $f_{\text{hier}} = R(x), S(x, y)$, we have

$$p(q) = 1 - \prod_{a \in A} (1 - p(R(a))(1 - \prod_{b \in A} (1 - p(S(a, b)))))$$

In this paper we study arbitrary conjunctive queries (i.e. allowing self-joins), which turn out to be significantly more complex. The starting point is the following extension of Theorem 1.3 (2)

THEOREM 1.4. *If q is not hierarchical then it is #*P*-hard.*

Thus, from now on we consider only hierarchical conjunctive queries in this paper, unless otherwise stated.

As a first contact with the issues raised by self-joins, let us consider the following query:

$$q \quad = \quad R(x), S(x, y), S(x', y'), T(x')$$

We write it as $q = f_1(x)f_2(x')$, where $f_1(x) = R(x), S(x, y)$ and $f_2(x') = S(x', y'), T(x')$. The query is hierarchical, but it has a self-join because the symbol S occurs twice: as a consequence $f_1[a/x]$ is no longer independent from $f_2[a/x']$ (they share common tuples of the form $S(a, b)$), which prevents us from applying Equation (3) directly. Our approach here is to define a new query by equating $x = x'$,

$$\begin{aligned} f_3(x) \quad &= \quad f_1(x)f_2(x) \\ &= \quad R(x), S(x, y), S(x, y'), T(x) \\ &\equiv \quad R(x), S(x, y), T(x) \end{aligned}$$

We show that the probability $p(q)$ can be expressed recursively over the probabilities of queries of the form $f_1[a/x_1]$, $f_2[a'/x_2]$, $f_3[a''/x_3]$, as a sum of a few formulas[1] in the same style as (3) (see Example 3.10). The correctness is based on the fact that $f_i[a/x_i]$ and $f_j[a'/x_j]$ are independent if $i \neq j$ or $a \neq a'$.

Inversions: The above approach does not work for all hierarchical queries. Consider:

$$H_0 \quad = \quad R(x), S(x, y), S(x', y'), T(y')$$

This query is hierarchical, but the above approach no longer works. The reason is that the two sub-goals $S(x, y)$ and $S(x', y')$ unify, while $x \sqsupset y$ and $x' \sqsubset y'$: we call this an *inversion* (formal definition is in Sec. 2.2). If we write H_0 as $f_1(x)f_2(y')$ and attempt to apply a recurrence formula, the queries $f_1[a/x]$ and $f_2[a'/y']$ are no longer independent even if $a \neq a'$, because they share the common tuple $S(a, a')$.

Inversions can occur as a result of a chain of unifications:

$$\begin{aligned} &H_k = \\ &R(x), S_0(x, y), \\ &\quad S_0(u_1, v_1), S_1(u_1, v_1) \\ &\quad\quad S_1(u_2, v_2), \ldots \\ &\quad\quad\quad S_{k-1}(u_k, v_k), S_k(u_k, v_k) \\ &\quad\quad\quad\quad S_k(x', y'), T(y') \end{aligned}$$

Here any two consecutive pairs of variables in the sequence $x \sqsupset y$, $u_1 \equiv v_1$, $u_2 \equiv v_2$, ..., $x' \sqsubset y'$ unify, and we also call this an inversion. We prove:

THEOREM 1.5. *For every $k \geq 0$, H_k is #*P*-hard.*

Thus, some hierarchical queries with inversions are #P-hard. We prove, however, that if q has no inversions, then it is in PTIME:

THEOREM 1.6. *If q is hierarchical and has no inversions, then it is in* PTIME.

Erasers The precise boundary between PTIME and #P-hard queries is more subtle than simply testing for inversions: some

[1]This particular example admits an alternative, perhaps simpler PTIME solution, based on a dynamic programming algorithm on the domain A. For other, very simple queries, we are not aware of any algorithm that is simpler than ours (formula (9), Sec. 3.2), for example $R(x, y, y, x), R(x, y, x, z)$, or $R(y, x, y, x, y), R(y, x, y, z, x), R(x, x, y, z, u)$ (both are in PTIME because they have no inversions). To appreciate the difficulties even with such simple queries note, by contrast, $R(y, x, y, x, y), R(y, y, y, z, x), R(x, x, y, z, u)$ is #P-hard.

queries with inversion are #P-hard, while others are in PTIME, as illustrated below:

Example 1.7 Consider the hierarchical query q

$$\begin{aligned} q = & R(r,x), S(r,x,y), U(a,r), U(r,z), V(r,z) \\ & S(r',x',y'), T(r',y'), V(a,r') \\ & R(a,b), S(a,b,c), U(a,a) \end{aligned}$$

Here a, b, c are constants and the rest are variables. This query has an inversion between $x \sqsupset y$ and $x' \sqsubset y'$ (when unifying $S(r,x,y)$ with $S(r',x',y')$). Because of this inversion, one may be tempted to try to prove that it is #P-hard, using a reduction from H_0. Our standard construction starts by equating $r = r'$ to make q "like" H_0: call q' the resulting query (i.e. $q' = q[r/r']$). If one works out the details of the reduction, one gets stuck by the existence of the following homomorphism from $h : q \to q'$ that "avoids the inversion": it maps the variables r, x, y, z, r', x', y' to a, b, c, r, r, x', y' respectively, in particular sending the sub-goals $U(r,z), V(r,z)$ to $U(a,r)$, $V(a,r)$. Thus, h takes advantage of the two sub-goals $U(a,r), V(a,r)$ in q' which did not exists in q, and its image does not contain the sub-goal $S(r,x,y)$, which is part of the inversion. We call such a homomorphism an *eraser* for this inversion: the formal definition is in Sec. 2.3. Because of this eraser, we cannot use the inversion to prove that the query is #P-hard. So far this discussion suggests that erasers are just a technical annoyance that prevent us from proving hardness of some queries with inversions. But, quite remarkably, erasers can also be used in the opposite direction, to derive a PTIME algorithm: they are used to cancel out (hence "erase") the terms in a certain expansion of $p(q)$ that correspond to inversions and that do not have polynomial size closed forms. Thus, our final result is:

THEOREM 1.8 (DICHOTOMY). *Let q be hierarchical.*
(1) If q has an inversion without erasers then q is #P-hard.
(2) If all inversions of q have erasers then q is in PTIME.

As a non-trivial application of (1) we show that each of the following two queries are #P-hard, since each has an inversion between two isomorphic copies of itself:

$$\begin{aligned} q_{\text{2path}} &= R(x,y), R(y,z) \\ q_{\text{marked-ring}} &= R(x), S(x,y), S(y,x) \end{aligned}$$

In general, the hardness proof is by reduction from the query H_k, where k is the length of an inversion without an eraser. The proof is not straightforward. It turns out that not every eraser-free inversion can be used to show hardness. Instead we show that if there is an eraser-free inversion then there is one that admits a reduction from H_k.

The PTIME algorithm in (2) is also not straightforward. It is quite different from the recurrence formula in Theorem 1.6, since we can no longer iterate on the structure of the query: in Example 1.7, the sub-query of q consisting of the first two lines is #P-hard (since without the third line there is no eraser), hence we cannot compute it separately from the third line. Our algorithm here computes $p(q)$ without recurrence, and thus is quite different from the inversion-free PTIME algorithm, but uses the latter as a subroutine.

Organization In Section 2, we introduce the terminology and develop some tools used in the proof of the dichotomy. In Section 3, we prove Theorem 1.6 by giving a PTIME algorithm for evaluating inversion-free queries. In Section 4, we establish the first half of the dichotomy, by proving Theorem 1.8(2). In Section 5, we prove the other half of the dichotomy, Theorem 1.8(1).

2. TERMINOLOGY

We introduce the key terminology and prove an expansion formula for computing the probability of conjunctive queries that will be used to device PTIME algorithms for query evaluation. For the remainder of the paper, all queries are assumed to be hierarchical, as we know that non-hierarchical queries are #P-hard.

2.1 Coverage

We call an *arithmetic predicate* a predicate of the form $u = v$, $u \neq v$, or $u < v$ between a variable and a constant in C, or between two variables[2]. A *restricted arithmetic predicate* is an arithmetic predicate that is either between a variable and a constant, or between two variables u, v that co-occur in some sub-goal (equivalently $u \sqsupseteq v$ or $u \sqsubseteq v$). From now on, we will allow all conjunctive queries to have restricted arithmetic predicates.

DEFINITION 2.1. *A* coverage *for a query q is a set of conjunctive queries $\mathcal{C} = \{qc_1, \ldots, qc_n\}$ such that:*

$$q \equiv qc_1 \vee \ldots \vee qc_n$$

Each query in $\mathcal{C}$ is called a cover. *A* factor *of $\mathcal{C}$ is a connected component of some $qc_i \in C$. We denote the set of all factors in $\mathcal{C}$ by $\mathcal{F} = \{f_1, \ldots, f_k\}$.*

We alternatively represent a coverage by the pair $(\mathcal{F}, C)$, where $\mathcal{F}$ is a set of factors and C is a set of subsets of $\mathcal{F}$. Each element of C determines a cover consisting of the corresponding set of factors from $\mathcal{F}$.

For any query q the set $\mathcal{C} = \{q\}$ is a trivial coverage. We also define $\mathcal{C}^<(q)$, which we call the *canonical coverage*, obtained as follows. Consider all m pairs (u, v) of co-occurring variables u, v in q, or of a variable u and constant v. For each such pair choose one of the following predicates: $u < v$ or $u = v$ or $u > v$, and add it to q. This results in 3^m queries. Remove the unsatisfiable ones, then remove all redundant ones (i.e. remove qc_i if there exists another qc_j s.t. $qc_i \subset qc_j$). The resulting set $\mathcal{C}^<(q) = \{qc_1, \ldots, qc_n\}$ is the *canonical coverage* of q.

Unifiers

Let q, q' be two queries (not necessarily distinct). We rename their variables to ensure that $Vars(q) \cap Vars(q') = \emptyset$, and write qq' for their conjunction. Let g and g' be two sub-goals in q and q' respectively. The *most general unifier*, MGU, of g and g' (or the MGU of q, q' when g, g' are clear from the context) is a substitution θ for qq' s.t. (a) $\theta(g) = \theta(g')$, (b) for any other substitution θ' s.t. $\theta'(g) = \theta'(g')$ there exists ρ s.t. $\rho \circ \theta = \theta'$.

A *1-1 substitution* for queries q, q' is a substitution θ for qq' such that: (a) for any variable x and constant a $\theta(x) \neq a$, and (b) for any two distinct variables x, y in q (or in q'), $\theta(x) \neq \theta(y)$. The *set representation* of a 1-1 substitution θ is the set $\{(x,y) \mid x \in Vars(q), y \in Vars(q'), \theta(x) = \theta(y)\}$.

DEFINITION 2.2. *An MGU θ for two queries q, q' is called* strict *if it is a 1-1 substitution for qq'.*

For a trivial illustration, if $q = R(x,x,y,a,z)$ and $q' = R(u,v,v,w,w)$ and their MGU is θ, then $\theta(x) = \theta(y) = \theta(u) = \theta(v) = x'$, $\theta(w) = \theta(z) = a$, and the effect of the unification is $\theta(qq') = R(x',x',x',a,a)$. This is not strict: e.g. $\theta(x) = \theta(y)$ and also $\theta(z) = a$.

[2]As usual we require every variable to be range restricted, i.e. to occur in at least one sub-goal.

DEFINITION 2.3. *(Strict coverage) Let $\mathcal{C}$ be a coverage and $\mathcal{F}$ be its factors. We say that $\mathcal{C}$ is* strict *if any MGU between any two factors $f, f' \in \mathcal{F}$ is strict.*

Example 2.4 Let $q = T(x), R(x,x,y), R(u,v,v)$. The trivial coverage $\mathcal{C} = \{q\}$ is not strict, as the MGU of the two R subgoals of q equate x with y and u with v. Alternatively, consider the following three queries:

$$\begin{aligned} qc_1 &= T(x), R(x,x,x) \\ qc_2 &= T(x), R(x,x,y), R(u,u,u), x \neq y \\ qc_3 &= T(x), R(x,x,y), R(u,v,v), x \neq y, u \neq v \end{aligned}$$

One can show that $q \equiv qc_1 \vee qc_2 \vee qc_3$, hence $\mathcal{C} = \{qc_1, qc_2, qc_3\}$ is a coverage for q. The set of factors $\mathcal{F}$ consists of the connected components of these queries, which are

$$\begin{aligned} f_1 &= T(x), R(x,x,x) \quad & f_2 &= T(x), R(x,x,y), x \neq y \\ f_3 &= R(u,u,u) & f_4 &= R(u,v,v), u \neq v \end{aligned}$$

and $C = \{\{f_1\}, \{f_2, f_3\}, \{f_2, f_4\}\}$. The coverage is strict, as a unifier cannot equate x with y or u with v in any query because of the inequalities. Similarly, the canonical coverage $\mathcal{C}^<(q)$, which has nine covers containing combinations of $x < y, x = y$, or $x > y$ with $u < v$, $u = v$, $u > v$, is also strict.

LEMMA 2.5. *The canonical coverage $\mathcal{C}^<(q)$ is always strict.*

2.2 Inversions

Fix a strict coverage $\mathcal{C}$ for q, with factors $\mathcal{F}$, and define the following undirected graph G. Its nodes are triples (f, x, y) with $f \in \mathcal{F}$ and $x, y \in Vars(f)$, and edges are pairs $((f,x,y),(f',x',y'))$ s.t. there exists two sub-goals g, g' in f, f' respectively whose MGU θ satisfies $\theta(x) = \theta(x')$ and $\theta(y) = \theta(y')$. We call an edge in G a *unification edge*, and a path a *unification path*. Recall that for a preorder relation $\sqsupseteq$, the notation $x \sqsupset y$ means $x \sqsupseteq y$ and $x \not\sqsubseteq y$.

DEFINITION 2.6. *(Inversion-free Coverage) An inversion in $\mathcal{C}$ is a unification path from a node (f, x, y) with $x \sqsupset y$ to a node (f', x', y') with $x' \sqsubset y'$. An* inversion-free coverage *is a strict coverage that does not have an inversion. We say that q is inversion-free if it has at least one inversion-free coverage. Otherwise, we say that q has inversion.*

PROPOSITION 2.7. *If there exists one coverage of q that does not contain inversion, then the canonical cover $\mathcal{C}^<(q)$ does not contain inversion.*

Example 2.8 We illustrate with two examples:

(a) Consider H_k in Theorem 1.5. The trivial coverage $\mathcal{C} = \{H_k\}$ is strict, and has factors $\mathcal{F} = \{f_0, f_1, \ldots, f_{k+1}\}$ (each line in the definition of H_k is one factor). The following is an inversion: $(f_0, x, y), (f_1, u_1, v_1), \ldots, (f_k, u_k, v_k), (f_{k+1}, x', y')$. This is an inversion because $x \sqsupset y$ and $x' \sqsubset y'$. The canonical coverage $\mathcal{C}^<$ also has an inversion, e.g. along the factors obtained by adding the predicates $x < y, u_1 < v_1, \ldots, u_k < v_k, x' < y'$.

(b) Consider the query $q = R(x), S(x,y), S(y,x)$. The trivial coverage $\mathcal{C} = \{q\}$ is strict, has one factor $\mathcal{F} = \{q\}$, and there is an inversion from (q, x, y) to (q, y, x) because $S(x,y)$ unifies with $S(y,x)$ (recall that we rename the variables before the unification, i.e. the unifier is between $R(x)$, $\underline{S(x,y)}$, $S(y,x)$ and its copy $R(x'), S(x',y'), \underline{S(y',x')}$). In the canonical coverage $\mathcal{C}^<$ there are three factors, corresponding to $x < y$, $x = y$, and $y < x$, and the inversion is between $x < y$ and $y < x$.

2.3 An Expansion Formula for Coverage

Given a conjunctive query q and a probabilistic structure $\mathbf{A} = (A, R_1^A, \ldots, R_k^A)$, we want to compute the probability $p(q)$. Our main tool is a generalized inclusion-exclusion formula that we apply to the coverage of a query.

DEFINITION 2.9. *(Expansion Variables) Let $\mathcal{C} = (\mathcal{F}, C)$ be a strict coverage, where $\mathcal{F} = \{f_1, \cdots, f_k\}$ is a set of factors and C is a set of subsets of $\mathcal{F}$. A set of* expansion variables *is a set $\bar{x} = \{\bar{x}_{f_1}, \cdots, \bar{x}_{f_k}\}$ such that $\bar{x}_{f_i} \subseteq Vars(f_i)$ for $1 \leq i \leq k$.*

We use $(\mathcal{F}, C, \bar{x})$ to denote a coverage where we have chosen the expansion variables.

For $f \in \mathcal{F}$, let $A_f = A^{|\bar{x}_f|}$, and for $\bar{a} \in A_f$, let $f(\bar{a})$ denote the query $f[\bar{a}/\bar{x}_f]$, i.e., the conjunctive query obtained by substituting the variables $\bar{x}_f$ with $\bar{a}$. The following follows simply from the definitions:

$$q = \bigvee_{c \in C} \bigwedge_{f \in c} \bigvee_{\bar{a} \in A_f} f(\bar{a}) \tag{4}$$

Our next step is to apply the inclusion/exclusion formula to (4). We need some notations. We call a subset $\sigma \subseteq \mathcal{F}$ a *signature*. Given $s \subseteq C$, its signature is $sig(s) = \bigcup_{c \in s} c$.

DEFINITION 2.10. *Given a set $\sigma \subseteq \mathcal{F}$, define*

$$N(\mathcal{C}, \sigma) = (-1)^{|\sigma|} \sum_{s \subseteq C : sig(s) = \sigma} (-1)^{|s|}$$

For example, if $C = \{c_1, c_2, c_3\}$ where $c_1 = \{f_1, f_2\}$,$c_2 = \{f_2, f_3\}$ and $c_3 = \{f_1, f_3\}$, then for signature $\sigma = \{f_1, f_2, f_3\}$ we have

$$\begin{aligned} N(\sigma) &= (-1)^{|\{f_1,f_2,f_3\}|}((-1)^{|\{c_1,c_2\}|} + (-1)^{|\{c_1,c_3\}|} + \\ &\quad (-1)^{|\{c_2,c_3\}|} + (-1)^{|\{c_1,c_2,c_3\}|}) \\ &= -2 \end{aligned}$$

Given k sets $\bar{T} = \{T_{f_1}, \ldots, T_{f_k}\}$, where $T_{f_i} \subseteq A_{f_i}$, we denote its signature $sig(\bar{T}) = \{f \mid T_f \neq \emptyset\}$, its cardinality $|\bar{T}| = \sum_i |T_{f_i}|$, and denote $\mathcal{F}(\bar{T})$ the query $\bigwedge_{f \in \mathcal{F}} \bigwedge_{a \in T_f} f(\bar{a})$.

THEOREM 2.11. *(Expansion Theorem) If $\mathcal{C}$ is a coverage for q, then*

$$p(q) = \sum_{\bar{T}} N(\mathcal{C}, sig(\bar{T}))(-1)^{|\bar{T}|} p(\mathcal{F}(\bar{T})) \tag{5}$$

This formula will be the main tool for the PTIME algorithms for queries in the following sections.

3. INVERSION-FREE QUERIES

In this section, we show that all inversion-free queries have a PTIME algorithm. We start by computing simple sums over functions on sets, then use it to give a PTIME algorithm for queries without inversion.

3.1 Simple Sums

Let $A = \{1, \ldots, N\}$, $\bar{g} = (g_1, \ldots, g_k)$ be k functions $g_i : A \to \mathbf{R}$, $i = 1, \ldots, k$, and $\bar{T} = (T_1, \ldots, T_k)$ a k-tuple of subsets of A. Denote $\bar{g}(\bar{T}) = g_1(T_1) \cdots g_k(T_k)$, where $g_i(T_i) = \prod_{\bar{a} \in T_i} g_i(\bar{a})$. $\emptyset \neq \bar{T}$ abbreviates $\emptyset \neq T_1, \ldots, \emptyset \neq T_k$. Let ϕ be a conjunction of statements of the form $T_i \cap T_j = \emptyset$ or $T_i \subseteq T_j$, and define: $S_\phi = \{\sigma \mid \sigma \subseteq [k], \forall i, j \in \sigma, \phi \not\models T_i \cap T_j = \emptyset\} \cap \{\sigma \mid \sigma \subseteq [k], \forall i \in \sigma, j \notin \sigma, \phi \not\models T_i \subseteq T_j\}$.

DEFINITION 3.1. *Denote the following sums:*

$$\bigoplus\nolimits_{\phi} \bar{g} = \sum_{\bar{T} \subseteq A, \phi} \bar{g}(\bar{T})$$
$$\bigoplus\nolimits_{\phi}^{+} \bar{g} = \sum_{\emptyset \neq \bar{T} \subseteq A, \phi} \bar{g}(\bar{T})$$

For $\sigma \subseteq [k]$, denote $\bar{g}_\sigma$ the family of functions $(g_i)_{i \in \sigma}$.

PROPOSITION 3.2. *The following closed forms hold:*

$$\bigoplus\nolimits_{\phi} \bar{g} = \prod_{a \in A} \sum_{\sigma \in S_\phi} \prod_{i \in \sigma} g_i(\bar{a})$$
$$\bigoplus\nolimits_{\phi}^{+} \bar{g} = \sum_{\sigma \subseteq [k]} (-1)^{k-|\sigma|} \bigoplus \bar{g}_\sigma$$

Moreover, the expressions have sizes $O(k2^k N)$ and $O(k2^{2k} N)$ respectively, hence all have an expression size that is linear in N.

Example 3.3 Consider four functions $g_i : A \to R$, $i = 1, 2, 3, 4$, and suppose we want to compute the following sum:

$$\sum_{T_1 \cap T_2 = \emptyset, T_2 \cap T_3 = \emptyset, T_4 \subseteq T_2} g_1(T_1) g_2(T_2) g_3(T_3) g_4(T_4)$$

In our notation, this is $\bigoplus_\phi \bar{g}$, where ϕ is $T_1 \cap T_2 = \emptyset \wedge T_2 \cap T_3 = \emptyset \wedge T_4 \subseteq T_2$. The set S_ϕ is $\{\emptyset, \{1\}, \{2\}, \{2,4\}, \{3\}, \{1,3\}\}$. Thus, the expression for the sum is

$$\prod_{a \in A} (1 + g_1(a) + g_2(a) + g_2(a)g_4(a) + g_3(a) + g_1(a)g_3(a))$$

The size of this expression is $8N$, where N is the size of A.

3.2 Inversion-Free Queries

Let q be an inversion-free query. We give now a PTIME algorithm for computing q on a probabilistic structure.

DEFINITION 3.4. (*Unary Coverage*) A unary coverage is a coverage $\mathcal{C} = (\mathcal{F}, C, \bar{x})$ with the following properties: (1) for each $f \in \mathcal{F}$, the set $\bar{x}_f$ consists of a single variable r_f which is maximal and (2) any MGU between two (not necessarily distinct) factors f, f' maps r_f to $r_{f'}$

THEOREM 3.5. *If q has no inversions then q has a unary coverage.*

Example 3.6 We illustrate Theorem 3.5 on two queries.

$$q_1 = R(x,y), S(x,y), S(x',y'), T(y')$$
$$q_2 = R(x,y), R(y,x)$$

In the trivial coverage $\mathcal{C} = \{q_1\}$ for q_1 the factors are

$$f_1 = R(x,y), S(x,y) \qquad f_2 = S(x',y'), T(y')$$

We see that $r_{f_1} = \{y\}$ and $r_{f_2} = \{y'\}$ satisfy the properties of Theorem 3.5 (there are two maximal variables for f_1, but we have to pick y because it unifies with y'). For q_2, the trivial coverage $\mathcal{C} = \{q_3\}$ does not work since there is a unifier that unifiers x with y, and exactly one of them can be the expansion variable. On the other hand, consider the following coverage:

$$f_3 = R(x_1,y_1), R(y_1,x_1), x_1 > y_1 \qquad f_4 = R(x,x)$$

now we can set $r_{f_3} = x_1$ and $r_{f_4} = x$. □

Now, let q be a query without inversion and $\mathcal{C} = (\mathcal{F}, C, \bar{x})$ be any unary coverage. Theorem 4.9 applied to this unary coverage gives:

$$p(q) = \sum_{\bar{T}} N(\mathcal{C}, sig(\bar{T}))(-1)^{|\bar{T}|} p(\mathcal{F}(\bar{T})) \quad (6)$$

Our first goal is to express $p(\mathcal{F}(\bar{T}))$ as $\prod_f \prod_{\bar{a} \in T_f} p(f(\bar{a}))$, to apply the closed forms for the sums we developed in the previous section. For that we need to ensure that any two queries $f(\bar{a})$, $\bar{a} \in A_f$ and $f'(\bar{a}')$, $\bar{a}' \in A_{f'}$ are independent, and this does not hold in general.

Example 3.7 We revisit the query q_1 in Example 3.6 and consider the trivial coverage $\mathcal{C} = \{q_1\}$, with the two factors

$$f_1 = R(x,y), S(x,y) \qquad f_2 = S(x',y'), T(y')$$

with y and y' as their root variables. Then, we have

$$p(q_1) = \sum_{\bar{T} = T_1, T_2} N(\mathcal{C}, sig(\bar{T}))(-1)^{T_1 + T_2} p(f_1(T_1) f_2(T_2)) \quad (7)$$

However, $f_1(T_1)$ and $f_2(T_2)$ are independent only when $T_1 \cap T_2 = \emptyset$. Therefore, we define a new query f_{12} by equating the root variables of f_1 and f_2, i.e. $f_{12}(r) = f_1(r) f_2(r)$. Define $\bar{T}^* = \{T_1, T_2, T_{12}\}$ and let $ip(\bar{T}^*)$ denote the predicate $T_1 \cap T_2 = T_2 \cap T_{12} = T_{12} \cap T_1 = \emptyset$. Then, $p(q_1)$ equal

$$\sum_{\bar{T}^* | ip(\bar{T}^*)} N(\mathcal{C}, sig(\bar{T}^*))(-1)^{T_1 + T_2} p(f_1(T_1) f_2(T_2) f_{12}(T_{12})) \quad (8)$$

where $sig(\bar{T}^*)$ is obtained in a straightforward way from T_1, T_2, T_{12}. The predicate $ip(\bar{T}^*)$ makes $f_1(T_1)$, $f_2(T_2)$ and $f_{12}(T_{12})$ independent. □

Based on the observation in the above example, we proceed as follows. For each $u \subset \mathcal{F}$, we define a new query f_u which is the conjunction of all queries $f \in u$ with their roots equated, i.e.

$$f_u(r) = \bigwedge_{f \in u} f[r/r_f]$$

Let $\mathcal{F}^* = \{f_u \mid u \subseteq \mathcal{F}\}$. Given a family of sets $\bar{T}^* = (T_u)_{u \subseteq \mathcal{F}}$, define

$$ip(\bar{T}^*) = \bigwedge_{u \neq u'} T_u \cap T_{u'} = \emptyset$$
$$|\bar{T}^*| = \sum_{|u|=1} |T_u|$$

Also, given a set $\sigma^* \subseteq \mathcal{F}^*$, define

$$exp(\sigma^*) = \{f \mid \exists u.\ u \in \sigma^*, f \in u\}$$

Then, we obtain $p(q) =$

$$\sum_{\bar{T}^* | \mathrm{ip}(\bar{T}^*)} N(\mathcal{C}, exp(sig(\bar{T}^*)))(-1)^{|\bar{T}^*|} \prod_{u \in \mathcal{F}^*} \prod_{\bar{a} \in T_u} p(f_u(\bar{a}))$$

Corresponding to each $T_u \in \bar{T}^*$, let $g_u : A \to \mathbf{R}$ denote the function $g_u(\bar{a}) = -p(f_u(\bar{a}))$ if $|u| = 1$ and $g_u(\bar{a}) = p(f_u(\bar{a}))$ if $|u| > 1$. Thus, we have $p(q) =$

$$\sum_{\bar{T}^* | \mathrm{ip}(\bar{T}^*)} N(\mathcal{C}, exp(sig(\bar{T}^*))) \prod_{u \in \mathcal{F}^*} \prod_{\bar{a} \in T_u} g(\bar{a})$$

We then use the closed forms for summations developed in Section 3.1 to obtain the following result.

THEOREM 3.8. *Let q be inversion-free.*

1. *The probability of q is given by*

$$p(q) = \sum_{\sigma \subseteq \mathcal{F}^*} N(\mathcal{C}, \exp(\sigma)) \bigoplus^{+}_{ip} \bar{g}_\sigma \qquad (9)$$

where $\bigoplus^+$ ranges over all sets of the form $\bar{T}^*$.

2. *For each $u \in \mathcal{F}^*$, $f_u(\bar{a})$ is an inversion-free query.*

We use Proposition 3.2 to write a closed-form expression for Equation (9) in terms of the probabilities $g_f(\bar{a}) = p(f(\bar{a}))$ for $f \in \mathcal{F}^*$. Since each of these queries is inversion-free, we recursively apply Equation (9) to compute their probabilities. For any query q, let $V(q)$ denote the maximum number of distinct variables in any single sub-goal of q. Clearly, for any factor f, we have

$$V(f(\bar{a})) < V(f) \leq V(q)$$

since $\bar{a}$ substitutes a variable in every sub-goal. Thus, the depth of the recursion is bounded by $V(q)$.

COROLLARY 3.9. *If q is an inversion-free query, then $p(q)$ can be expressed as a formula of size $O(N^{V(q)})$, where N is the size of the domain. In particular q is in* PTIME.

Example 3.10 We continue our running example from Example 3.7. For this query, $N(\mathcal{C}, \sigma) = 1$ if $\sigma = \{f_1, f_2\}$ and 0 otherwise. Let $\bar{T}^* = (T_1, T_2, T_{12})$. Let $g_1(a) = -p(f_1(a))$, $g_2(a) = -p(f_2(a))$ and $g_{12}(a) = p(f_{12}(a))$. Let

$$\phi \equiv (T_1 \cap T_2 = \emptyset) \wedge (T_1 \cap T_{12} = \emptyset) \wedge (T_2 \cap T_{12} = \emptyset)$$

The subsets σ^* of $\mathcal{F}^*$ for which $\exp(\sigma^*) = \{f_1, f_2\}$ are $\{f_1, f_2\}$, $\{f_{12}\}$, $\{f_1, f_{12}\}$, $\{f_2, f_{12}\}$ and $\{f_1, f_2, f_{12}\}$. Thus,

$$p(q) = \bigoplus^{+}_{\phi}(g_1, g_2) + \bigoplus^{+}_{\phi}(g_{12}) + \bigoplus^{+}_{\phi}(g_1, g_{12}) + \bigoplus^{+}_{\phi}(g_2, g_{12}) + \bigoplus^{+}_{\phi}(g_1, g_2, g_{12})$$

Now apply Prop. 3.2 to each expression. Each sum in turn has a closed form. Furthermore, each $f_i(a)$ is a query with a single variable (y or y'), hence each $g_i(a) = p(f_i(a))$ can be computed inductively.

3.3 Queries with Negated Subgoals

The PTIME algorithm in this section can be extended to queries with negated sub-goals.

DEFINITION 3.11. *A* conjunctive query with negations *is a query $q = \exists \bar{x}.(\varphi_1 \wedge \ldots \wedge \varphi_k)$, where each φ_i is either a positive sub-goal $R(t)$, or a negative sub-goal $\texttt{not}(R(t))$, or an arithmetic predicate. The query q is said to be inversion-free if the conjunctive query obtained by replacing each $\texttt{not}(R(t))$ sub-goal with $R(t)$ sub-goal is inversion-free.*

DEFINITION 3.12. *(Inversion-free property) A property ϕ is called* inversion-free property *if it can be expressed as a Boolean combination of queries $\{q_1, \cdots, q_m\}$ such that each q_i is a conjunctive query with negation and the query $q_1 q_2 \cdots q_m$ is inversion-free.*

THEOREM 3.13. *If ϕ is any inversion-free property, computing $p(\phi)$ is in* PTIME.

PROOF. (Sketch) Consider a single inversion-free conjunctive query with negation. The same recurrence formula in Theorem 3.8 applies, the only difference is during recursion we will reach negated constant sub goals: $p(\texttt{not}(R(a, b, c)))$ is simply $1 - p(R(a, b, c))$. For any general ϕ, use inclusion/exclusion formula to reduce it to conjunctive queries with negations, each of which is inversion-free. □

3.4 Complex Sums

In Section 3.2, we used simple sums to give a PTIME algorithm for inversion-free queries. Here, we show that the PTIME algorithm can be used to compute closed formulas for complex sums. We call this the *bootstrapping technique*.

Bootstrapping: Let $\bar{g} = (g_1, \ldots, g_k)$ be a family of functions, $g_i : A^{r_i} \rightarrow \mathbf{R}$, where the arity of g_i is r_i. We want to compute sums of the form $\texttt{sum} = \sum_{\bar{S}|\phi} \bar{g}(\bar{S})$, where ϕ is a complex predicate. We cannot use the summations of Section 3.1, which only apply when g_i are unary. Instead, we use a bootstrapping technique to reduce this problem back to evaluating an inversion-free query on a probabilistic database, and use the PTIME algorithm of Section 3.2. The basic principle is that we can reduce the problem to the evaluation of ϕ over a probabilistic database. Create an probabilistic instance of $\mathcal{S}$, where, assuming $k = 1$ for simplicity, for each tuple $\bar{a} \in S$, set its probability to $p(\bar{a}) = g(\bar{a})/(1 + g(\bar{a}))$. Then, the probability of ϕ over this instance is

$$\begin{aligned} p(\phi) &= \sum_S \prod_{\bar{a} \in S} p(\bar{a}) \prod_{\bar{a} \notin S} (1 - p(\bar{a})) \\ &= \prod_{\bar{a}} 1/(1 + g(\bar{a})) \sum_{S|\phi} g(S) \\ &= \prod_{\bar{a}} 1/(1 + g(\bar{a})) \texttt{sum} \end{aligned}$$

Thus, we can compute sum in PTIME if we can evaluate the query ϕ in PTIME.

THEOREM 3.14. *Let ϕ be an inversion-free property. Then, the sum $\sum_{\bar{S}|\phi} \bar{g}(\bar{S})$ has a closed form polynomial in domain size.*

4. THE GENERAL PTIME ALGORITHM

In this section, we formally define the notion of an *eraser* and prove that if all the inversions of a query have erasers, then the query is in PTIME. This establishes one half of the dichotomy.

In the previous section, we used a unary coverage to develop a PTIME algorithm for inversion-free queries. When queries have inversions, a unary coverage does not always exist. Instead, we define the following:

DEFINITION 4.1. (*Complete Coverage*) A complete coverage is a coverage $\mathcal{C} = (\mathcal{F}, C, \bar{x})$ such that for each $f \in F$, $\bar{x}_f = Vars(f)$.

Thus, a complete coverage expands the query on all its variables. We extend the theory of unary coverages for inversion-free queries to complete coverages for the general PTIME algorithm. Informally, the algorithm works as follows:

1. We start with a query q where all inversions have erasers.
2. We use the Expansion Theorem on a complete coverage of q to obtain a summation formula for $p(q)$.
3. We add independence predicates to the summation formula. For unary coverages, these independence predicates were simply predicates of the form $T_u \cap T_{u'} = \emptyset$, but now they are much more involved.
4. We express the summation as a complex sum with a closed form, as developed in Section 3.4. Recall that for unary coverages, we had expressed the summation as a simple sum in the style of Section 3.1.

5. We show that the complex predicate ϕ corresponding to the complex sum is an inversion-free property. Thus, we use the bootstrapping technique of Section 3.4 to obtain a PTIME algorithm.

The rest of this section formalizes this algorithm.

4.1 Some Basic Results

4.1.1 Independence Predicates

Our goal in this section is to generalize the notion of independence predicates. For unary coverages, an independence predicate is simply a set of statements of the form $T_i \cap T_j \neq \emptyset$, but the general case requires more formalism. We first introduce a new relational vocabulary, $\mathcal{T}$ consisting of the relation symbols $T_{f_1}, \cdots, T_{f_k}$ of arities $|x_{f_1}|, \ldots, |x_{f_k}|$ respectively. A structure over this vocabulary is a k-tuple of sets $\bar{T}$; given a conjunctive query ϕ over the vocabulary $\mathcal{T}$, $\bar{T} \models \phi$ means that ϕ is true on $\bar{T}$. For a trivial illustration, assume T_{f_1}, T_{f_2} to be of arity 1, and $\phi = T_{f_1}(x), T_{f_2}(x)$. Then ϕ states that $T_{f_1} \cap T_{f_2} \neq \emptyset$.

Suppose we have two factors f_i and f_j and θ is any 1-1 substitution on f_i, f_j, given in set representation. Define

$$\begin{aligned}\theta^R(f_i, f_j) &= f_i, f_j, \bigwedge_{(x_i,x_j)\in\theta} x_i = x_j \\ \theta^T(f_i, f_j) &= T_{f_i}(\bar{x}_{f_i}), T_{f_j}(\bar{x}_{f_j}), \bigwedge_{(x_i,x_j)\in\theta} x_i = x_j\end{aligned}$$

Note that $\theta^R(f_i, f_j)$ is over the vocabulary $\mathcal{R}$ (same as the original query q), while $\theta^T(f_i, f_j)$ is over the vocabulary $\mathcal{T}$. We call them the *join query* and the *join predicate* respectively. We call the negation of join predicate, $\texttt{not}(\theta^T(f_i, f_j))$, an *independence predicate*.

Example 4.2 Consider factors f_1 and f_2 in Example 3.7, and let $\theta = \{(y, y')\}$. Then,

$$\begin{aligned}\theta^R(f_1, f_2) &= R(x,y), S(x,y), S(x',y), T(y) \\ &\equiv R(x,y), S(x,y), T(y) \\ \theta^T(f_i, f_j) &= T_1(y), T_2(y)\end{aligned}$$

and the independence predicate $\texttt{not}(\theta^T(f_i, f_j))$ says that T_1 and T_2 are disjoint.

We want to add enough independence predicates so that the following holds: If T_i, T_j satisfy all independence predicates between f_i and f_j, then for all $\bar{a} \in T_i$ and $\bar{a}' \in T_j$, $f_i(\bar{a})$ and $f_j(\bar{a}')$ are independent.

4.1.2 Hierarchical Closure

Recall from Example 3.7 that, in order to introduce an independence predicate between two sets T_1, T_2 we needed to use the join query of their factors, $f_{12}(r) = f_1(r), f_2(r)$. We started from the set of factors $\mathcal{F}$ and added some join queries to obtain a new set of factors. We will proceed similarly here. Starting from a coverage $\mathcal{C}$ we will add join queries repeatedly until we obtain its *hierarchical closure*, denoted $\mathcal{C}^*$, then we will introduce independence predicates. Computing $\mathcal{C}^*$ is straightforward when $\mathcal{C}$ is an inversion-free coverage (which is the case for our first PTIME algorithm), but when $\mathcal{C}$ has inversions then some join queries are non-hierarchical and we cannot add them to $\mathcal{C}^*$. We define next $\mathcal{C}^*$ in the general case. Let $\mathcal{C} = (\mathcal{F}, C, \bar{x})$ be any complete coverage.

DEFINITION 4.3. Given two factors f_1 and f_2, and a MGU given by the set representation θ, the *hierarchical unifier* θ_u is the maximal subset of θ such that:

1. If $(x, y) \in \theta_u$ and (x', y') is such that $x \sqsubseteq x'$ or $y \sqsubseteq y'$ and $(x', y') \in \theta$, then $(x', y') \in \theta_u$.

2. The query $\theta_u^R(f_1, f_2)$ is hierarchical.

It can be shown that θ_u is uniquely determined. If θ_u is non-empty, we say that f_1 and f_2 can be *hierarchical joined* using θ and call the query $\theta_u^R(f_1, f_2)$ the *hierarchical join* of f_1 and f_2, and $\theta_u^T(f_1, f_2)$ the *hierarchical join predicate*.

Example 4.4 Let

$$\begin{aligned}f_1 = \quad & R(r,x), \quad S(r,x,y), U(a,r), U(r,z), V(r,z) \\ f_2 = \quad & \qquad S(r',x',y'), T(r',y'), V(a,r')\end{aligned}$$

and $\theta = \{(r, r'), (x, x'), (y, y')\}$ be the MGU of the two S subgoals. Then, the hierarchical unifier is $\theta_u = \{(r, r')\}$. If we include any of (x, x') or (y, y'), we will have to include the other because $x \sqsubset y$ and $x' \sqsupset y'$, and then the join will not be hierarchical. The hierarchical join for this unifier is

$$\begin{aligned}\theta_u^R(f_1, f_2) = \quad & R(r,x), \quad S(r,x,y), U(a,r), U(r,z), V(r,z) \\ & \qquad S(r,x',y'), T(r,y'), V(a,r)\end{aligned}$$

□.

Starting from the factors $\mathcal{F}$, we construct a set $\mathcal{H}$, a function *Factors* from $\mathcal{H}$ to subsets of $\mathcal{F}$, and a set of expansion variables $\bar{x}_h$ for $h \in \mathcal{H}$. This is done inductively as follows:

1. For each $f \in \mathcal{F}$, add f to $\mathcal{H}$ and let $\mathit{Factors}(f) = \{f\}$.

2. For any two queries h_1, h_2 in $\mathcal{H}$, and any MGU θ between h_1 and h_2, let $h = \theta_u^R(h_1, h_2)$ be their hierarchical join. Then add h to $\mathcal{H}$, define $\mathit{Factors}(h) = \mathit{Factors}(h_1) \cup \mathit{Factors}(h_2)$; define $\bar{x}_h = \theta_u(\bar{x}_{h_1} \cup \bar{x}_{h_2})$.

We need to show that $\mathcal{H}$ is finite. This follows from:

LEMMA 4.5. *Given a fixed relational vocabulary $\mathcal{R}$ and a fixed set of constants C, the number of distinct hierarchical queries over $\mathcal{R}$ and C is finite.*

Define $\mathcal{F}^*$ to be the subset of $\mathcal{H}$ containing queries that are either inversion-free or in $\mathcal{F}$.

DEFINITION 4.6. *(Hierarchical Closure) Given a coverage $\mathcal{C} = (\mathcal{F}, C, \bar{x})$, its* hierarchical closure *is $\mathcal{C}^* = (\mathcal{F}^*, C^*, \bar{x}^*)$ where $\mathcal{F}^*$, $\bar{x}^*$ are defined above and:*

$$C^* = \{c \mid c \subseteq \mathcal{F}^*, \bigcup_{f\in c} \mathit{Factors}(f) \in C\}$$

Note that $\mathcal{C}^*$ is indeed a coverage since the set $\mathcal{F}^*$ contains the set $\mathcal{F}$, the set C^* contains the set C, and the expansion variables satisfy the conditions in Def. 2.9. Let $\texttt{ip}(\mathcal{C}^*)$ be the conjunction of $\texttt{not}(jp)$, where jp ranges over all possible hierarchical join predicates in $\mathcal{F}^*$.

LEMMA 4.7. *If $T \models \texttt{ip}(\mathcal{C}^*)$, then*

$$p(\mathcal{F}(q)) = \prod_{f\in\mathcal{F}^*} \prod_{a\in T_f} p(f(\bar{a}))$$

4.1.3 Adding Independence Predicates

Finally, we look at conditions under which we can add the predicate $\texttt{ip}(\mathcal{C}^*)$ over $\bar{T}$. We divide the join predicates into two disjoint sets, *trivial* and *non-trivial*. A join predicate between factors h_i and h_j is called trivial if the join query is equivalent to either h_i or h_j, and is called non-trivial otherwise. We write $\texttt{ip}(\mathcal{C}^*)$ as $\texttt{ip}^n(\mathcal{C}^*) \wedge \texttt{ip}^t(\mathcal{C}^*)$, where $\texttt{ip}^n(\mathcal{C}^*)$ is the conjunction of $\texttt{not}(jp)$ over all non-trivial join predicates jp, and $\texttt{ip}^t(\mathcal{C}^*)$ is the conjunction over all trivial join predicates.

DEFINITION 4.8. *(Eraser) Given a hierarchical join query $jq = \theta_u^R(f_i, f_j)$, an* eraser *for jp is a set of factors $E \subseteq \mathcal{F}$ s.t.:*

1. *$\forall q \in E$, there is a homomorphism from q to jq.*
2. *$\forall \sigma \subseteq \mathcal{F}$, $N(\mathcal{C}, \sigma \cup \{f_i, f_j\}) = N(\mathcal{C}, \sigma \cup \{f_i, f_j\} \cup E)$.*

THEOREM 4.9. *Let q be a query such that every hierarchical join query $jq = \theta_u^R(f_i, f_j)$ between two factors in $\mathcal{F}^*$ has an eraser. Then,*

$$p(q) = \sum_{\bar{T} | \texttt{ip}^n(\mathcal{C}^*)} N(\mathcal{C}^*, sig(\bar{T}))(-1)^{|\bar{T}|} p(\mathcal{F}(\bar{T}))$$

The theorem allows us to add all possible non-trivial independence predicates over the summation. If the hierarchical join query jp is inversion-free, then it belongs to $\mathcal{F}^*$ and it is its own eraser (i.e. $E = \{jp\}$ satisfies both conditions above). We can use it to separate T_i from T_j. In particular if q is inversion-free, then any hierarchical join query has an eraser, and all sets can be separated. But if jp has an inversion, then jp does not belong to $\mathcal{F}^*$ and we must find some different query (queries) in $\mathcal{F}^*$ that can be used to separate T_i from T_j.

Example 4.10 Consider the query q in Example 1.7 Although q has an inversion (between the two S Subgoals) we have argued in Sec. 1.1 that it is in PTIME. Importantly, the third line of constants sub goals plays a critical role: if we removed it, the query becomes #P-hard.
Consider the coverage $\mathcal{C} = (\mathcal{F}, C, \bar{x})$, where $\mathcal{F}$ is[3]:

$$\begin{aligned} f_1 &= R(r,x), S(r,x,y), U(a,r), U(r,z), V(r,z), r \neq a \\ f_2 &= S(r',x',y'), T(r',y'), V(a,r'), r' \neq a \\ f_3 &= U(a,z'), V(a,z') \\ f_4 &= R(a), S(a,b,c), U(a,a) \end{aligned}$$

and $C = \{\{f_1, f_2, f_4\}, \{f_2, f_3, f_4\}\}$. We cannot simply take the root variables r, r', and z' as expansion variables and proceed with the recurrence formula in Th. 3.8, because the query $f_{12} = f_1(r)f_2(r)$ is #P-hard. We must keep all variables as expansion variables to avoid the inversion. Thus, the root unifiers $\mathcal{H}$ are (recall Example 4.4):

$$\begin{aligned} f_{12} &= f_1, f_2, r = r' \\ f_{23} &= f_2, f_3, r' = z' \\ f_{13} &= f_1, f_3, r = z' \\ f_{123} &= f_1, f_2, f_3, r = r' = z' \end{aligned}$$

Out of these, f_{12} and f_{123} have inversions, thus

$$\mathcal{F}^*(q) = \{f_1, f_2, f_3, f_4, f_{23}, f_{13}\}$$

[3]Strictly speaking each constant sub-goal $R(a), S(a,b,c), U(a,a)$ should be a distinct factor.

In the expansion $Exp(\mathcal{C}^*)$, there are sets $T_1, T_2, T_3, T_4, T_{23}, T_{13}$ but note that they are not unary, e.g. T_1 has arity 4 as $\bar{x}_{f_1} = \{r, x, y, z\}$. The critical question is how to separate now T_1 from T_2, since we don't have the factor f_{12}. Here we use the fact that there exists a homomorphism $f_3 \rightarrow f_{12}$, thus f_3 is an eraser between f_1 and f_2 and will use f_3 to separate T_1, T_2. The definition of an eraser (Def. 4.8) requires us to check $\forall \sigma$, $N(\mathcal{C}, \sigma \cup \{f_1, f_2\}) = N(\mathcal{C}, \sigma \cup \{f_1, f_2, f_3\})$. The only σ that makes both N's non-zero is $\{f_4\}$ (and supersets), and indeed the two numbers are equal to $+1$. It is interesting to note that, if we delete the last line from q, then we have the same set of factors but a new coverage $C' = \{\{f_1, f_2\}, \{f_2, f_3, f_4\}\}$: then f_3 is no longer an eraser because for $\sigma = \emptyset$ we have $N(\{f_1, f_2\}) = 1$ and $N(\{f_1, f_2, f_3\}) = 0$. Continuing the example, we conclude that, with aid from the eraser, we can now insert all non-trivial independence predicates, e.g. between f_1 and f_2, f_2 and f_3, f_1 and f_{23} etc. We still haven't added the trivial independence predicates, e.g. between f_1 and f_{12}, f_{23} and f_{123} etc. Also, observe that the non-trivial independence predicates are no longer simple disjointness conditions e.g. the predicate between T_1 and T_2 is the negation of the query $T_1(r,x,y,z), T_2(r,x',y')$. □

4.2 The General PTIME Algorithm

Let q be a conjunctive query and let $\mathcal{C} = (\mathcal{F}, C)$ be a strict coverage for q and let $\mathcal{H}$ be the set of hierarchical unifiers, as defined in Section 4.1.2. Suppose the following holds: for every hierarchical join predicate $jp = \theta^T(h_i, h_j)$ between two factors in $\mathcal{H}$, the join query $jq = \theta^R(f_i, f_j)$ has an eraser. We will show here that q is in PTIME, thus proving Theorem 1.8(2).

We set the expansion variables $\bar{x}$ to include all variables to obtain a complete coverage $\mathcal{C} = (\mathcal{F}, C, \bar{x})$. Let $\mathcal{C}^* = (\mathcal{F}^*, C^*, \bar{x}^*)$ be the hierarchical closure of $\mathcal{C}$. By Theorem 4.9, we have

$$\begin{aligned} p(q) &= \sum_{\bar{T} | \texttt{ip}^n(\mathcal{C}^*)} N(\mathcal{C}^*, sig(\bar{T}))(-1)^{|\bar{T}|} p(\mathcal{F}^*(\bar{T})) \\ &= \sum_{\sigma} N(\mathcal{C}^*, \sigma) \sum_{\bar{T} | \texttt{ip}^n(\mathcal{C}^*), sig(\bar{T}) = \sigma} (-1)^{|\bar{T}|} p(\mathcal{F}^*(\bar{T})) \end{aligned} \quad (10)$$

We now focus on each of the inner sums in Equation (10). which is of the form

$$\sum_{\bar{T} | \texttt{ip}^n(\mathcal{C}^*), sig(\bar{T}) = \sigma} (-1)^{|\bar{T}|} p(\mathcal{F}^*(\bar{T})) \quad (11)$$

We want to reduce it to evaluation of an inversion-free property, but there are two problems. First, the predicate $\texttt{ip}^n(\mathcal{C}^*)$ over $\bar{T}$ is not an inversion-free property. Second, we still need to add the predicates $\texttt{ip}^t(\mathcal{C}^*)$ to make $p(\mathcal{F}^*(\bar{T}))$ multiplicative. To solve these problems, we apply a preprocessing step on Equation 10, which we call the *change of basis*. In this step, we group $\bar{T}$ that generate the same $\mathcal{F}^*(\bar{T})$ and sum over these groups.

Example 4.11 Consider a factor $f = R_1(x,y), R_2(y,z)$. We look at the set $T(x,y,z)$ corresponding to this factor, which is a ternary set since $\bar{x}_f = \{x,y,z\}$. For every T, define $S^0 = \pi_y(T)$, $S^1 = \pi_{xy}(T)$ and $S^2 = \pi_y(T)$, hence $T = S^0 \bowtie S^1 \bowtie S^2$ (natural join). Clearly, S^0, S^1, S^2 satisfy the predicate $S^0 = \pi_y(S^1) = \pi_y(S^2)$. Consider the sum

$$\sum_{\bar{T}} (-1)^{|\bar{T}|} p(f(\bar{T})) \quad (12)$$

We group all T that generate the same S^0, S^1, S^2 and show that

the summation in Eq. 12 is equivalent to the following:

$$\sum_{\substack{S^1, S^2, S^0 \mid \\ S^0 = \pi_y(S^1) = \pi_y(S^2)}} (-1)^{|S^1|+|S^2|+|S^0|} p(R_1(S^1)R_2(S^2))$$

Thus, we have changed the summation basis from T to S^0, S^1, S^2. □

The change of basis introduces some new predicates between sets, which we call the *link predicates*, e.g. predicates of the form $S^0 = \pi_y(S^1)$. But at the same time, as we shall see, the change of basis simplifies the independence predicates $\mathtt{ip}(\mathcal{C}^*)$, making them inversion-free, so that the computation of Equation (10) can be reduced to evaluation of inversion-free queries. We now formally define the change of basis. This consists of the following steps: (1) we change the summation basis from $\bar{T}$ to $\bar{S}$. (2) we translate the $\mathtt{ip}^n(\mathcal{C}^*)$ predicates from $\bar{T}$ to $\bar{S}$. (3) we introduce a new set of predicates, called the link predicates, on $\bar{S}$. (4) We add the remaining independence predicates, $\mathtt{ip}^t(\mathcal{C}^*)$, translated from $\bar{T}$ to $\bar{S}$.

Consider a factor $f \in \mathcal{F}^*$. It is a connected hierarchical query with the hierarchy relation $\sqsubseteq$ on $Vars(f)$. Given $x \in Vars(f)$, let $[x]$ denotes its equivalence class under $\sqsubseteq$ and let $\lceil x \rceil$ denote $\{y \mid y \sqsupseteq x\}$. Define a *hierarchy tree* for f as the tree where nodes are equivalence classes of variables, and edges are such that their transitive closure is $\sqsubseteq$. For instance, in Example 4.11, the hierarchy tree of f has nodes $\{x\}, \{y\}, \{z\}$ with $\{x\}$ as root and $\{y\}, \{z\}$ its children.

Define a new vocabulary, consisting of a relation $S_f^{[x]}$ for each $f \in \mathcal{F}^*$ and each node $[x]$ in the hierarchy tree of f, with arity equal to the size of $\lceil x \rceil$. Let $\bar{S}$ denote instances of this vocabulary. The intuition is that $S_f^{[x]}$ denotes $\pi_{\lceil x \rceil}(T_f)$ in the change of basis from $\bar{T}$ to $\bar{S}$. This completes step 1.

Step 2 is simple. Let $\mathtt{ip}^n$ denote the set of independence predicates on $\bar{S}$, translated in a straightforward manner from the independence predicates $\mathtt{ip}^n(\mathcal{C}^*)$ on $\bar{T}$.

Define a *link predicate* $S_f^{[x]} = \pi_{\lceil x \rceil}(S_f^{[y]})$ for every edge $([x], [y])$ in the hierarchy tree of f. Let $\mathtt{lp}$ be the set of all link predicates. This is step 3.

Finally, we add the trivial independence predicates $\mathtt{ip}^t$. For this, we expand the basis of summation from $\bar{S}$ to $\bar{S}'$ by adding the following sets. We add a new set $S^{i,j}_{x_i,x_j}$ corresponding to each pair $S_{f_i}^{[x_i]}$, $S_{f_j}^{[x_j]}$ such that (i) f_i and f_j have a hierarchical join query which is equivalent to f_j and (ii) there are sub-goals g_i in h_i and g_j in h_j referring to the same relation such that $Vars(g_i) = \lceil x_i \rceil$ and $Vars(g_j) = \lceil x_j \rceil$. For each such $S^{i,j}_{x_i,x_j}$, $\mathtt{ip}^t$ contains the following conjuncts: $S_{f_i}^{[x_i]} \cap S_{f_j}^{[x_j]} = \emptyset$, $S^{i,j}_{x_i,x_j} \subseteq S_{f_j}^{[x_j]}$. This describes the step 4.

Finally, we put it all together. We define a function $G(\bar{S}')$ on $\bar{S}'$ as follows. Consider a relation $S_f^{[x]}$, and let p be the number of children of $[x]$ in the hierarchy tree. For a tuple t in $S_f^{[x]}$, let

$$G(t) = (-1)^{p+1} \prod_{g \in sg(f) \mid Vars(g) = \lceil x \rceil} p(g(t))$$

Define $G(\bar{S}') = \prod_{t \in \bar{S}'} G(t)$.

Denote $sig(\bar{S}')$ the set $\{f \mid S_f^{[r_f]} \neq \emptyset\}$, where $[r_f]$ denotes the root of the hierarchy tree of f.

THEOREM 4.12. *With $\mathtt{ip}^t$, $\mathtt{ip}^n$, $\mathtt{lp}$, sig and G as defined above,*

$$\sum_{\bar{T} \mid \mathtt{ip}(\mathcal{C}^*), sig(\bar{T}) = \sigma} (-1)^{|\bar{T}|} p(\mathcal{F}^*(\bar{T})) = \sum_{\bar{S}' \mid \mathtt{ip}^n, \mathtt{ip}^t, \mathtt{lp}, sig(\bar{S}') = \sigma} G(\bar{S}')$$

Finally, we use the bootstrapping principle to reduce the problem of computing the summation to the evaluation of the query

$$\phi = (\mathtt{ip}^n \wedge \mathtt{ip}^t \wedge \mathtt{lp} \wedge sig(\bar{S}') = \sigma)$$

LEMMA 4.13. *The query ϕ defined above is an inversion-free property.*

By using Theorem 3.14, we get the following:

THEOREM 4.14. *Suppose for every hierarchical join predicate $jp = \theta^T(h_i, h_j)$ between two factors in $\mathcal{H}$, the join query $jq = \theta^R(f_i, f_j)$ has an eraser. Then, q is* PTIME.

5. #P-HARD QUERIES

Here we show the other half of Theorem 1.8, i.e., if q has an inversion without an eraser, then q is #P-hard.

Let $\mathcal{C} = (\mathcal{F}, C, \bar{x})$ be any strict coverage for q, $\mathcal{C}^* = (\mathcal{F}^*, C^*, \bar{x}^*)$ its closure and $\mathcal{H}$ the set of hierarchical join queries over $\mathcal{F}$. Suppose there are factors $h, h' \in \mathcal{H}$ such that the join query $hj = \theta^T(h, h')$ has an inversion, but not an eraser. Among all such hj, we will pick a specific one and use it to show that q is #P-hard. Note that if there is no such hj, then the query is in PTIME by Theorem 4.14.

Let the inversion in hj consist of a unification path of length k from (f, x, y) with $x \sqsubset y$ to (f', x', y') with $x' \sqsupset y'$. Then, we will prove the #P-hardness of q using a reduction from the chain query H_k, which is #P-hard by Theorem 1.5.

Given an instance of H_k, we create an instance of q. The basic idea is as follows: take the unification path in hj that has the inversion and completely unify it. We get a non-hierarchical query (due to the inversion) with two distinguished variables x and y (the inversion variables), $k + 2$ distinguished sub-goals (that participated in the inversion), plus other sub-goals in the factor. Use the structure of this query and the contents of the $k + 2$ relations in the instance of H_k to create an instance for q. We skip the formal description of the reduction, but instead illustrate it on examples.

Example 5.1 Consider $q = U(x), V(x, y), V(y, x)$ and the coverage $\mathcal{C} = (\mathcal{F}, C)$ where $\mathcal{F} = \{f\}$ with $f = U(x), V(x, y), V(y, x), x \neq y$ and $C = \{\{f\}\}$. The coverage has a single factor and a single cover. The first V sub-goal of factor f unifies with the second sub-goal of another copy of f to give an inversion between $x \sqsupset y$ and their copy $y' \sqsubset x'$. If we unify the two sub-goals in two copies of f, we get the query:

$$q_u = \underline{U}(x), \underline{V}(x, y), V(y, x), \underline{U}(y)$$

We have underlined the sub-goals taking part in the inversion. Now we give a reduction from the query

$$H_0 = R(x), S(x, y), S(x', y'), T(y')$$

Given any instance of R, S, T for H_0 construct an instance of U, V as follows. We map the R, S, T relations in H_0 to the U, V, U underlined sub goals of q_u as follows: for each tuple $R(a)$, create a tuple $U(a)$ with same probability. For each $S(a, b)$, create $V(a, b)$ with the same probability. For each $T(a)$, create $U(a)$ with same probability. Also, for each $S(a, b)$, create $V(b, a)$ with probability 1 (this corresponds to the non-underlined sub-goal).

There is a natural 1-1 correspondence between the substructures of U, V and the substructures of R, S, T with the same probability. It can be shown that q is true on a substructure iff the query $R(x), S(x,y) \vee S(x',y'), T(y')$ is true on the corresponding substructure. Thus, we can compute the probability of the query $R(x), S(x,y) \vee S(x',y'), T(y')$, and hence, the probability of H_0, by inclusion-exclusion.

Next, we show why a hardness reduction fails if the inversion has an eraser.

Example 5.2 We revisit the query q in Example 4.10. There is an inversion between $x \sqsubset y$ in f_1 and $x' \sqsupset y'$ in f_2. However, their hierarchical join, f_{12} have an eraser. The unified query consists of

$$\begin{aligned} q_u \quad = \quad & \underline{R}(r,x), \underline{S}(r,x,y), U(a,r), U(r,z), V(r,z), V(a,r), \\ & \underline{T}(r,x), R(a), S(a,b,c), U(a,a) \end{aligned}$$

We construct an instance $RSTUV$ for q from an instance $R'S'T'$ for H_0 as in previous example. However, there is a bad mapping from q to q_u, corresponding to the eraser, which is $\{r \to a, x \to b, y \to c, x' \to x, y' \to y, z \to r\}$, which avoids the $\underline{R}$ sub-goal. The effect is that q is true on a world iff the query $S'(x',y')T'(y')$ (rather that H_0) is true on the corresponding world. So the reduction from H_0 fails. In fact, we know that this query q is in PTIME.

The final example shows that if there are multiple inversions without erasers, we need to pick one carefully, which makes the hardness reduction challenging.

Example 5.3 Consider the following variation of the query in previous example:

$$\begin{aligned} q = & R(x), S(x,y), U(x,y,a,b), U(z_1,z_2,x,y), V(z_1,z_2,x,y) \\ & S(x',y'), T(y'), V(x',y',a,b) \\ & R(a), S(a,b), U(a,b,a,b) \end{aligned}$$

Let f_1 and f_2 denote the factors corresponding to the first two lines of q. There is an inversion from $x \sqsupset y$ in f_1 to $x' \sqsubset y'$ in f_2 via the two S sub-goals, and it does not have an eraser. But if we unify the two S sub-goals to obtain S, there is a "bad mapping" from q to q_u that maps x, y to a, b and z_1, z_2 to x, y. However, as it turns out, there is another inversion in q that we can use for hardness. The inversion is from $x \sqsupset y$ to $z_1 \equiv z_2$ to x', y' through the following unification path: $U(\underline{x}, \underline{y}, x, y)$ unifies with (a copy of) $U(\underline{z}_1, \underline{z}_2, x, y)$ and $V(\underline{z}_1, \underline{z}_2, x, y)$ unifies with $V(\underline{x}', \underline{y}', a, b)$. We can show that this inversion works for the hardness reduction.

By formalizing these ideas, we prove:

THEOREM 5.4. *Suppose there are $h, h' \in \mathcal{H}^*(q)$ such that their hierarchical join hj has an inversion without an eraser. Then, q is #P-complete.*

6. CONCLUSIONS

We show that every conjunctive query has either PTIME or #P-complete complexity on a probabilistic structure. As part of the analysis required to establish this result we have introduced new notions such as hierarchical queries, inversions, and erasers. Future work may include several research directions: a study whether the hardness results can be sharpened to counting the number of substructures (i.e. when all probabilities are 1/2); an analysis of the query complexity; extensions to richer probabilistic models (e.g. to probabilistic databases with disjoint and independent tuples [10]); and, finally, studies for making our PTIME algorithm practical for probabilistic database systems.

7. REFERENCES

[1] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[2] Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD*, pages 891–893, 2005.

[3] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.

[4] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[5] Nilesh Dalvi and Dan Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, 2007.

[6] Tomas Feder and Moshe Y. Vardi. Monotone monadic snp and constraint satisfaction. In *STOC*, pages 612–622, 1993.

[7] Norbert Fuhr and Thomas Rolleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[8] Erich Gradel, Yuri Gurevich, and Colin Hirch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[9] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[10] Christopher Re, Nilesh Dalvi, and Dan Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.

[11] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[12] L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.

[13] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.