Probabilistic Query Evaluation: The Combined FPRAS Landscape

Timothy van Bremen
National University of Singapore
Singapore
tvanbr@comp.nus.edu.sg

Kuldeep S. Meel
National University of Singapore
Singapore
meel@comp.nus.edu.sg

ABSTRACT
We consider the problem of computing the probability of a query over a tuple-independent probabilistic database, known as the probabilistic query evaluation (PQE) problem. The problem is well-known to be #P-hard in data complexity for conjunctive queries in general, as well as for several subclasses of conjunctive queries. Existing approximation approaches for dealing with hard queries have centered on computing the lineage of the query over the database, which can be intractable for all but the smallest of queries due to the exponential dependence of the lineage size on the query length.

In this paper, we take a first step towards bridging this gap, by showing how to construct a fully polynomial-time randomized approximation scheme (FPRAS) for the PQE problem for any class of self-join-free conjunctive queries of bounded hypertree width, that runs in time polynomial in both the query length and database instance size. An interesting consequence of our result is the existence of classes of queries that are #P-hard in data complexity to evaluate exactly, yet easy to approximate both in terms of query length and database size.

CCS CONCEPTS
• Information systems → Database query processing; • Theory of computation → Formal languages and automata theory; • Mathematics of computing → Hypergraphs.

KEYWORDS
probabilistic query evaluation, approximation algorithms, nondeterministic finite tree automata

ACM Reference Format:

1 INTRODUCTION
A wide range of applications exist that require querying answers over structured datasets in the presence of uncertainty or imprecision. Consider, for example, knowledge extracted from text using an imperfect NLP system, or data collected from noisy sensors. Such information is often specified in a relational format suitable for use in a relational database system, but the standard relational model makes no provisions for modeling the uncertainty inherent in the data. Probabilistic databases have been proposed as a simple, principled formalism for filling this need [12, 23]. In this model, each fact appearing in an underlying relational database is annotated with an independent probability, which intuitively represents the probability of that fact’s presence.

A substantial body of research exists studying the probabilistic query evaluation (PQE) problem, the canonical problem in the context of probabilistic databases. Given a query specified in some logical language, and a probabilistic database as described above, the goal is determine the likelihood of the query holding on a randomly sampled database, in which each row is included independently with the probability of its annotated value. Following the framework of data and query complexity introduced in the seminal work of Vardi [24], the problem of PQE has mainly been studied through the lens of data complexity (barring limited exceptions [4]), in which we are interested in time complexity with respect to the size of the database instance for a fixed query. The pioneering work of Dalvi and Suciu [8–10] eventually culminated in the well-known dichotomy theorem [11] for PQE of unions of conjunctive queries (UCQs). The dichotomy theorem states that, given a fixed UCQ, computing its probability over some input probabilistic database is either in FP or #P-hard, depending on the query in question. This dichotomy result was later extended beyond UCQs to the more general setting of homomorphism-closed queries, in the special case of probabilistic graphs (essentially probabilistic databases over schemas limited to binary relations) [2].

While extensive effort has been undertaken to classify queries by their tractability, the body of work studying how queries that are known to be intractable can be dealt with in practice has been more limited. Indeed, the primary approach, exact or otherwise, to dealing with queries that are #P-hard in data complexity has been to take the so-called intensional approach to PQE [26], which involves computing the lineage of the query over the database as a propositional formula, and computing the weighted model count of this formula (either exactly or approximately). Unfortunately, the size of this lineage can be exponential in the length of the query, thus rendering the intensional approach of limited practical utility for all but the smallest of queries. For example, evaluating a conjunctive query of only five atoms over a database with just a few hundred rows can yield a propositional DNF formula with over $10^{10}$ (one trillion!) clauses, out of reach of even the most cutting-edge approximate model counters. Consequently, it is desirable to avoid the above intensional approach, and instead develop algorithms that can avoid exponential dependence on the query length. This raises the question:
Can we design an approximation scheme with guarantees for the PQE problem, whose runtime is polynomial in both the query length and database size?

Note that we likely cannot hope to get a fully polynomial-time approximation scheme (FPRAS) applicable to all conjunctive queries. Recall that Boolean conjunctive query evaluation on deterministic databases is NP-complete in combined complexity [7]. If such an FPRAS for probabilistic query evaluation were possible, we could answer any Boolean conjunctive query on a deterministic database with high probability in polynomial combined complexity (just set all probabilities to 1), hence implying NP ⊆ BPP.

Therefore, we instead answer the question posed above in the affirmative for a large class of queries whose deterministic query evaluation problem is tractable. In particular, we propose an FPRAS for the PQE problem, applicable to any class of conjunctive queries of bounded hypertree width [18], so long as queries in the class do not contain repeated relation symbols (in other words, they are self-join-free). Crucially, our FPRAS runs in time polynomial in both the query length and database instance size, setting it apart from classical intensional approaches described above which suffer from an exponential dependence on the query length. Indeed, although we are not the first to study approximation for the PQE problem [14–16, 22], so far no other techniques have been proposed that have both polynomial runtime in combined complexity for a wide class of queries, as well as rigorous guarantees on the quality of the probability computed.

Finally, it is worth remarking that the study of bounded hypertree width queries is motivated by the observation that conjunctive queries found in real-world benchmarks typically have very low hypertree width in practice (usually no more than 3) [17]. Therefore, we believe that our approach could serve as a useful starting point for the development of practical, scalable algorithms for the probabilistic query evaluation problem.

### 1.1 Technical Contributions

The primary contribution of our work is to establish the following result.

**Theorem 1.** Let $Q$ be a self-join-free conjunctive query of bounded hypertree width, and $H$ a probabilistic database instance. Then there exists an algorithm $\text{PQE}\text{Estimate}$ such that, for all $\epsilon \in (0, 1)$:

$$ (1 - \epsilon) \Pr_H(Q) \leq \text{PQE}\text{Estimate}(Q, H) \leq (1 + \epsilon) \Pr_H(Q) $$

with high probability. Moreover, $\text{PQE}\text{Estimate}$ has runtime:

$$ \text{poly}(|Q|, |H|, \epsilon^{-1}) $$

An interesting consequence of the FPRAS presented here is the existence of classes of queries for which PQE is provably #P-hard even in data complexity alone, yet are tractable to approximate in both database size and query length. For example, consider the class $3\text{PATH} = \cup_{i \geq 3} Q_i$ of self-join-free path queries of length at least three:

$$ Q_i := R_1(x_1, x_2), \ldots, R_i(x_i, x_{i+1}) $$

It is easy to check that every query in the class $3\text{PATH}$ is non-hierarchical [11], which is known to be an equivalent condition to #P-hardness in data complexity for self-join-free conjunctive queries. Attempts to approximate its probability by computing its lineage as a weighted DNF formula are unlikely to succeed: the lineage of $Q_i$ over a database $D$ expressed as a propositional formula can have size $\Theta(|D|^i)$. However, path queries have bounded hypertree width—in fact, since they are acyclic they have hypertree width 1. The FPRAS given here therefore shows that the probability of any query in the class can be tractably approximated in a manner polynomial not only in terms of $|D|$, but also in $i$, thereby eliminating the exponential dependence. We hence obtain the following corollary.

**Corollary 1.** There exists a class of queries $C$ such that:

1. for an arbitrary $q \in C$, PQE for $q$ on an input probabilistic database instance is #P-hard
2. for an arbitrary $q \in C$ and probabilistic database instance $H$, approximating $\Pr_H(Q)$ to a $(1 + \epsilon)$-factor with high probability can be performed in time $\text{poly}(|Q|, |H|, \epsilon^{-1})$

We contextualize our main result in Table 1, by placing our FPRAS among some existing tractability results for PQE.

### Key Ideas.

Our approach follows the spirit of seminal work by Kolaitis and Vardi [21] that connected two fundamental, yet seemingly unrelated problems: conjunctive query containment and constraint satisfaction. While conjunctive query containment and constraint satisfaction are decision problems, the PQE problem is a counting problem, and therefore, we must turn our attention to corresponding counting problems. In this work, we focus on uncovering a fundamental relationship between PQE and counting problems in the context of tree automata. In particular, we demonstrate that PQE for self-join-free conjunctive queries of bounded hypertree width can be reduced to counting the number of trees accepted by a non-deterministic finite tree automaton (NFTA). This NFTA is constructed from both the query and database instance together. We can then leverage a recent breakthrough FPRAS for counting

### Table 1: Tractability results for PQE

<table>
<thead>
<tr>
<th>Bounded HW?</th>
<th>Self-Join-Free?</th>
<th>Safe?</th>
<th>Prior Results (Data Complexity)</th>
<th>Our Results (Combined Complexity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>FP [10]</td>
<td>FPRAS</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>#P-hard [10]</td>
<td>FPRAS</td>
</tr>
<tr>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>FP [10]</td>
<td>Open</td>
</tr>
</tbody>
</table>

*Note: The top two rightmost cells highlighted in bold indicate the contribution of this paper. The “Bounded HW?” and “Self-Join-Free?” columns, respectively, denote bounded hypertree width and self-join-freeness for all queries in the class. The “Safe?” column denotes the syntactic notion of safety for queries in the class, as defined by Dalvi and Suciu [11].*
these trees, that was originally designed to tackle the entirely separate problem of answer counting for (bounded hypertree width) conjunctive queries [6]. Although some aspects of the reduction employed there are similar to the one here, the fact that the same approximation result can also be leveraged for PQE is not at all obvious. In the context of PQE there are new challenges that we must address: namely, dealing with the exponential number of subinstances possible, as well as incorporating individual fact probabilities.

At a high level, our procedure as it applies to counting subinstances of $D$ that satisfy $Q$ (a special case of the PQE problem, known as uniform reliability [3, 19]) is as follows. We take the hypertree decomposition of $Q$, which intuitively gives us an efficient evaluation plan for $Q$ on any database $D$. We then note the first vertex in the hypertree decomposition that is a “covering vertex” for some atom in $Q$—essentially, this means that once we have reached that point in the decomposition, we have fixed our witness for that atom. Since $Q$ is self-join-free, we know that upon fixing that fact we are free to make any selection of the remaining non-witnessing facts for that relation in $D$. Thus, we can design our tree automaton to accept all the possible traversals of the hypertree decomposition with assignments from facts in $D$. A key point to note here is that even though the number of satisfying subinstances of $D$ may be exponentially large, the number of witnesses in $D$ of $Q$ is at most the size of $D$, making such a construction feasible.

In order to extend this approach to the PQE problem in general, we also need a way to incorporate the fact probabilities into the reduction. To this end, we substitute every transition with a probabilistic database instance is defined as the size of its underlying database instance $|D|$, plus the aggregate size of the bit encodings of its fact probabilities.

Conjunctive Queries. We focus on (Boolean) conjunctive queries: existentially quantified constant-free first-order sentences comprising conjunctions of atoms, which we write in the form $Q = R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$. The set of variables occurring in $Q$ is denoted by $vars(Q)$, and the set of atoms by $atoms(Q)$. Similarly, for any $A \in atoms(Q)$, $vars(A)$ denotes the set of variables occurring in $A$. The notation is likewise extended to sets of atoms: for a set of atoms $R \subseteq atoms(Q)$, we define $vars(R) = \cup_{A \in R} vars(A)$. We define the length of a query $|Q|$ as the number of atoms it contains. If $Q$ contains no repeated relation names, then $Q$ is said to be self-join-free. A path query is a conjunctive query $Q$, comprising only binary atoms, of the form:

$$Q = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_n(x_n, x_{n+1})$$

We use the usual semantics to determine if a database instance $D$ satisfies a conjunctive query $Q$, and write $D \models Q$ to indicate this. The probability of a query on a probabilistic database instance $H = (D, \pi)$ is:

$$Pr_H(Q) = \sum_{D' \models Q} Pr_D(D')$$

Computing the probability of a query over a given probabilistic database instance is known as the probabilistic query evaluation (PQE) problem. For a conjunctive query $Q$ and database $D$, we denote by $UR(Q,D)$ the uniform reliability of $Q$ on $D$: the number of subinstances of $D$ that satisfy $Q$. Note that this is equivalent (up to a factor of $2^{|D|}$) to computing $Pr_H(Q)$, in the special case where $H$ is the probabilistic database comprising $D$ equipped with uniform tuple probabilities of 0.5.

Strings and String Automata. A string over an alphabet $\Sigma$ is a sequence $w_1 \ldots w_n$ of symbols, with each $w_i \in \Sigma$. We denote the empty string with no symbols by $\lambda$. We denote by $\Sigma^*$ the set of all strings over $\Sigma$. A set $A$ of strings over $\Sigma$ is said to be prefix-closed, if $u \cdot v \in A$ implies $u \in A$ for any $u, v \in \Sigma^*$, where $u \cdot v$ denotes the concatenation of $u$ and $v$. Further, denote by $A^i$ the set of strings obtainable by concatenation of $i$ strings selected from $A$, and define $A^0 = \{\lambda\}$.  

2 PRELIMINARIES

We begin by reviewing some background on probabilistic databases, conjunctive queries, and automata.

**Probabilistic Databases.** A relational schema $\sigma$ is a collection of relation names, each with an associated arity. We assume the existence of a countably infinite universe of constants $U$ that serve as values that can appear in our databases. A database instance (or simply database) $D$ over $\sigma$ is a finite set of facts of the form $R_i(c_1, \ldots, c_k)$, where $R_i$ is some relation name in $\sigma$ with arity $k$, and $c_1, \ldots, c_k \in U$. We define the size $|D|$ of a database instance as the number of facts appearing in it. A database instance $D'$ is said to be a subinstance of $D$ if $D' \subseteq D$.

A probabilistic database instance $H = (D, \pi)$ is a database instance $D$ equipped with a probability function $\pi : D \rightarrow [0, 1]$, mapping each fact in $D$ to an independent probability label. For the sake of more easily formalizing the representation of $\pi$, we assume in this paper that probability labels are rational (that is, $\pi : D \rightarrow [0, 1] \cap \mathbb{Q}$). The labelling $\pi$ induces a probability distribution on the subinstances $D' \subseteq D$ as follows:

$$Pr_H(D') = \prod_{f \in D'} \pi(f) \prod_{f \in D \setminus D'} (1 - \pi(f))$$

A probabilistic database instance $H' = (D', \pi)$ is a subinstance of $H = (D, \pi)$ (denoted $H' \subseteq H$) if $D' \subseteq D$ and they have the same labelling function $\pi$. The size of $|H|$ of a probabilistic database instance is defined as the size of its underlying database instance $|D|$, plus the aggregate size of the bit encodings of its fact probabilities.
A non-deterministic finite (string) automaton (NFA) is a tuple \((S, \Sigma, \delta, I, F)\) where \(S\) is a finite set of states, \(\Sigma\) is a finite alphabet of input symbols, \(\delta : S \times \Sigma \rightarrow 2^S\) is a transition function, \(I \subseteq S\) is a set of initial states, and \(F \subseteq S\) is a set of accepting states. We define the size of an automaton \(M\), denoted \(|M|\), as the size of the encoding of its transition relation \(\delta\) over a suitable alphabet. We assume the standard semantics for deciding whether a string lies in the language \(L(M)\) of strings accepted by \(M\). We denote by \(L_n(M)\) the (finite) language of strings of length \(n\) accepted by \(M\). By the result in [5], we assume the existence of an FPRAS \(\text{CountNFTA}\) for approximating \(|L_n(M)|\), running in time polynomial in \(n\) and \(|M|\).

Trees and Tree Automata. For \(k \in \mathbb{N}\), a \(k\)-tree (or simply tree) is a prefix-closed non-empty finite subset \(t \subseteq [k]^*\). A path is a 1-tree. The root of a tree \(t\) is the empty string \(\lambda \in t\), and the maximal elements of \(t\) under prefix order are called leaves. For \(u, v \in t\), \(u\) is said to be a parent of \(v\), and \(v\) a child of \(u\), if \(v = u \cdot i\) for some \(i \in [k]\). The size of \(t\) is simply its cardinality \(|t|\). Given a finite alphabet \(\Sigma\), we denote by \(\text{Trees}_k[\Sigma]\) the language of \(k\)-trees in which each node \(u \in t\) is labelled with a symbol from \(\Sigma\). We abuse notation slightly and write \(t(u)\) to denote the label of a node \(u \in t\). We then define a \((\text{top-down})\) non-deterministic finite tree automaton (NFTA) in the standard manner, as a tuple \(T = (S, \Sigma, \Delta, s_	ext{init})\), where \(S\) is a finite set of states, \(\Sigma\) is a finite alphabet of input symbols, \(\Delta \subseteq S \times \Sigma \times (\mathbb{L}^k_{\text{root}} S)^{\text{root}}\) is the transition relation, and \(s_	ext{init} \in S\) is the initial state. Without loss of generality, we also slightly abuse notation by allowing \(\lambda\)-transitions of the form \((s, \lambda, R)\) for \(s \in S\) and \(R \in \mathbb{L}^k_{\text{root}} S\), noting that such an automaton can easily be converted to an equivalent one without \(\lambda\)-transitions using standard procedures. We sometimes refer to NFAs as ordinary NFAs, to emphasize their distinction from augmented NFAs and NFAs with multipliers defined later in the paper. Like for NFAs, the size of an NFTA \(T\), denoted \(|T|\), is defined as the size of the encoding of its transition relation \(\Delta\) over some suitable alphabet.

A run of \(T\) over a labelled tree \(t \in \text{Trees}_k[\Sigma]\) is a function \(\rho : t \rightarrow S\) such that for every \(u \in t\) with children \(u \cdot 1, \ldots, u \cdot n\), we have \((\rho(u), t(u), \rho(u \cdot 1) \ldots, \rho(u \cdot n)) \in \Delta\). In particular, if \(u\) is a leaf, then we require \((\rho(u), t(u), \lambda) \in \Delta\). We say \(T\) accepts \(t\) if there exists a run of \(T\) over \(t\), and \(L(T)\) to denote the language of all labelled trees accepted by \(T\), and \(L_n(T)\) for the (finite) language of labelled trees of size \(n\) accepted by \(T\). By the result in [6], we assume the existence of an FPRAS \(\text{CountNFTA}\) for approximating \(|L_n(T)|\), running in time polynomial in \(n\) and \(|T|\).

Hypertree Decompositions. We briefly review some background on hypertree decompositions, and refer the reader to the comprehensive paper by Gottlob et al. [18] for more details. A hypertree for a conjunctive query \(Q\) is a tuple \((T, \chi, \xi)\), where \(T = (N, E)\) is a tree on vertices \(N\) with edges \(E\), \(\chi\) and \(\xi\) are labelling functions mapping each \(p \in N\) to sets of variables \(\chi(p) \subseteq \text{vars}(Q)\) and atoms \(\xi(p) \subseteq \text{atoms}(Q)\). We use \(\text{vertices}(T)\) as shorthand to denote \(N\), the set of vertices of \(T\). Moreover, for a set of vertices \(P \subseteq \text{vertices}(T)\), we define \(\xi(P) = \cup_{p \in P} \xi(p)\).

A hypertree decomposition for a conjunctive query \(Q\) is a hypertree \((T, \chi, \xi)\) that satisfies the following conditions:

1. for each atom \(A \in \text{atoms}(Q)\), there exists \(p \in \text{vertices}(T)\) such that \(\text{vars}(A) \subseteq \chi(p)\)
2. for each variable \(x \in \text{vars}(Q)\), the set \(\{p \in \text{vertices}(T) \mid x \in \chi(p)\}\) induces a connected subtree of \(T\)
3. for each vertex \(p \in \text{vertices}(T)\), \(\chi(p) \subseteq \text{vars}(\xi(p))\)
4. for each vertex \(p \in \text{vertices}(T)\), \(\text{vars}(\xi(p)) \cap \chi(T_p) \subseteq \chi(p)\)

Where \(T_p\) is the subtree of \(T\) rooted at \(p\).

A hypertree decomposition is said to have width \(k\) if \(\max_{p \in \text{vertices}(T)} |\chi(p)| = k\). The width of a conjunctive query \(Q\) is the minimal width across all its possible hypertree decompositions. A class of conjunctive queries is said to have bounded hypertree width if all queries in the class have hypertree width at most \(k\), for some constant \(k\). For any input conjunctive query of hypertree width \(k\), one can compute in time polynomial in the query size a hypertree decomposition of width at most \(k\).

Let \(Q\) be a conjunctive query, and \((T, \chi, \xi)\) be a hypertree decomposition of \(Q\). For any \(A \in \text{atoms}(Q)\), we call a vertex \(p \in \text{vertices}(T)\) a covering vertex for \(A\) if \(A \in \xi(p)\) and \(\text{vars}(A) \subseteq \chi(p)\).

We say a hypertree decomposition \((T, \chi, \xi)\) is complete if every atom \(A \in \text{atoms}(Q)\) has a covering vertex in \(T\). Any decomposition of width \(k\) for a conjunctive query \(Q\) can be transformed in logspace to a complete decomposition of equal width by the following process: for any atom \(A \in \text{atoms}(Q)\) that does not have a corresponding covering vertex, create a new vertex \(p_A\) with \(\chi(p_A) = \text{vars}(A)\) and \(\xi(p) = \{A\}\). Then attach \(p_A\) as a child of some vertex \(p\) that satisfies \(\chi(p) \subseteq \text{vars}(A)\) (such a vertex must exist by condition (1) in the definition of a hypertree decomposition).

We finish by remarking that removing condition (4) in the definition of a hypertree decomposition above results in defining the closely related notion of a generalized hypertree decomposition. However, testing whether a query has generalized hypertree width at most \(k\) (for a fixed constant \(k \geq 3\)) is NP-complete. Moreover, for any conjunctive query \(Q\) it is known that \(\text{ghtw}(Q) \leq \text{htw}(Q) + 1\), where \(\text{ghtw}\) denotes (generalized) hypertree width [1]. Hence, we write our results here in terms of hypertree decompositions, bearing in mind that our results apply equally to queries of bounded generalized hypertree width.

3 A WARM-UP: PATH QUERIES ON GRAPHS

In this section, we prove a special case of the main result of this paper. In particular, we show the existence of an FPRAS for computing the uniform reliability of self-join-free path queries on database instances where all relations are binary—in other words, labelled graphs. In doing so, we build some intuition on the proof of the full theorem given later.

Theorem 2. Let \(D\) be a database instance in which all relations are binary, and \(Q\) be a self-join-free path query. Then there exists an algorithm \(\text{PathEstimate}\) such that, for all \(e \in (0, 1)\),

\[(1 - e) \cdot \text{UR}(Q, D) \leq \text{PathEstimate}(Q, D) \leq (1 + e) \cdot \text{UR}(Q, D)\]

with high probability. Moreover, \(\text{PathEstimate}\) has runtime:

\[\text{poly}(|Q|, |D|, e^{-1})\]

Proof intuition. Let \(D\) be our input database instance, and \(Q\) the self-join-free path query, taking the form:

\[Q = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_n(x_n, x_{n+1})\]
with all of the $R_i$ (for $i \in [n]$) distinct.

Our ultimate goal will be to construct an NFA $M = (S, \Sigma, \delta, I, F)$ whose accepted strings correspond one-to-one with the subinstances of $D$ that satisfy $Q$, upon which we can apply the COUNT-NEA FPRAS. Before doing so, we first construct a slightly different NFA $M' = (S', \Sigma', \delta', I', F)$, as follows: let the alphabet be $\Sigma' = (R_i(x, y) \mid R_i(x, y) \in D)$, and add to $S'$ one state $[x, y]$ for each fact $R_i(x, y) \in D$, as well as a single auxiliary state $\epsilon_{\text{end}}$. Define the transition relation $\delta'$ by adding to $\delta'$ the tuple $([x, y], R_i(x, y), [1 + i, y, z])$ for every joining pair of facts $R_i(x, y), R_{i+1}(y, z) \in D$ whose relations occur in sequence in $Q$, as well as the tuples $([n, x, y], R_n(x, y), \epsilon_{\text{end}})$ for every fact of the form $R_n(x, y) \in D$. Finally, define the initial state set $I = \{(1, x, y) \mid R_1(x, y) \in D\}$, and accepting state set $F = \{\epsilon_{\text{end}}\}$. Notice that it is not difficult to see that every string accepted by $M'$ takes the form $R_1([z_1, z_2])R_2([z_2, z_3]) \cdots R_n([z_n, z_{n+1}])$, corresponding one-to-one to the possible sequences of witnessing facts for $Q$ on $D$. Every subinstance $D' \subseteq D$ must therefore contain all of the facts in one of these strings to be a satisfying subinstance for $Q$. Moreover, as long as our selection of facts contains the facts in one of these strings, we can make any choice as to the presence of the remaining facts in $D$. However, we want to do this in a way that avoids representing the same subinstance twice. We can avoid this issue by:

(1) indicating the absence of a fact from our subinstance, rather than only the presence (so all strings accepted should have length $|D|$); and

(2) ensuring a consistent ordering in which the symbols indicating each fact’s presence or absence in $D'$ appears in any accepted string

Notice that $M'$ already accepts strings with atoms appearing only in the order they appear in the query $R_1 \cdots R_n$. It will therefore suffice to fix an arbitrary total ordering $\prec_i$ on the $R_i$-facts for each atom $R_i$. We are now ready to begin the construction of $M = (S, \Sigma, \delta, I, F)$. We first expand $\Sigma$ to allow us to indicate the absence of a fact by defining $\Sigma = \Sigma' \cup \{\neg R_i(x, y) \mid R_i(x, y \in D\}$. We also define an expanded state set:

$$ S = S' \cup \{(i, a, b, x, y) \mid R_i(x, y) \in D\} $$

Intuitively, the purpose of the state $[i, a, b, x, y]$ is to record the presence or absence of the fact $R_i(x, y)$, given that $R_i(x, y)$ has been chosen as our witness for the $R_i$-atom in $Q$. We now rebuild our transition set $\delta$ from scratch, by adding the following rules for every $i \in [n-1]$ and joining pairs of facts $R_i(a_k, b_k), R_{i+1}(b_{k'}, f) \in D$:

$$(i, 1, a_1, b_1, a_k, b_k) \to (i, a_1, b_1, a_k, b_k) \to (i, 1, a_2, b_2, a_k, b_k) \to \cdots \to (i, a_k, b_k, a_k, b_k) \to (i, 1, a_{k+1}, b_{k+1}, a_k, b_k) \to \cdots$$

$$(i, a_k, b_k, a_k, b_k) \to (i, a_{k+1}, b_{k+1}, a_k, b_k) \to (i, 1, a_{k+1}, b_{k+1}, a_k, b_k) \to \cdots \to (i, 1, a_{k+1}, b_{k+1}, a_k, b_k) \to (i, 1, a_{k+1}, b_{k+1}, a_k, b_k)$$

where $R_i(a_k, b_k) \prec_i \cdots \prec_i R_i(a_k, b_k) \prec_i \cdots \prec_i R_i(a_k, b_k)$ are all the $R_i$-facts in $D$, and $R_{i+1}(p, q)$ is the $\prec_{i+1}$-minimal fact in $D$. Next, for every $R_0(a_k, b_k) \in D$ add the tuples:

$$(\{n, a_1, b_1, a_k, b_k\}, R_0(a_1, b_1), [i, a_2, b_2, a_k, b_k]$$

$$(\{n, a_1, b_1, a_k, b_k\}, \neg R_0(a_1, b_1), [i, a_2, b_2, a_k, b_k])$$

$$\cdots$$

$$(\{n, a_k, b_k, a_k, b_k\}, R_i(a_k, b_k), [i, a_{k+1}, b_{k+1}, a_k, b_k])$$

$$\cdots$$

$$(\{n, a_c, b_c, a_k, b_k\}, R_i(a_c, b_c), \epsilon_{\text{end}})$$

$$(\{n, a_c, b_c, a_k, b_k\}, \neg R_i(a_c, b_c), \epsilon_{\text{end}})$$

where $R_0(a_1, b_1) \prec \cdots \prec a_i R_0(a_k, b_k) \prec \cdots \prec a_i R_0(a_k, b_k)$ are the strings accepted by $M$.

\[\Box\]

4 UNIFORM RELIABILITY

In this section, we build on the intuition presented in the previous section, generalizing the approach there so that it applies to computing uniform reliability of self-join-free queries of bounded hypertree width over arbitrary instances. To do so, we move from the setting of string automata to tree automata for the remainder of this paper.

4.1 Augmented NFTAs

We first introduce augmented NFTAs, which augment NFTAs with two additional constructs:

(1) (String Annotations) First, we extend the definition of an NFTA to allow a transition to be annotated with a string of symbols $\gamma_1 \cdots \gamma_n$ (rather than a single symbol), with the implicit meaning that an additional $n - 1$ fresh intermediate states are inserted between the start and end states of the transition so that the string $\gamma_1 \cdots \gamma_n$ is accepted.

(2) (? Symbols) Second, we also allow a symbol $? \gamma_i$ appearing in this string to be annotated with a $\lambda$, as shorthand expressing that either the symbol $\gamma_i$ or $\gamma_i \gamma_j$ should be accepted (note that this adds no additional states).

We distil the two ideas above into the definition of an augmented NFTA below, and then define its semantics in terms of a translation into an ordinary NFTA.

Definition 1 (Augmented NFTA). An augmented (top-down) nondeterministic finite tree automaton (augmented NFTA, for short) is a tuple $T^+ = (S, \Sigma, \Lambda, \Delta, \text{init})$, where $S$ is a finite set of states, $\Sigma$ is a finite alphabet of input symbols, and $\text{init} \in S$ is the initial state. The transition relation is defined as $\Delta \subseteq S \times \Gamma \times (\cup_{\alpha \in \Sigma} \Lambda^* \cup \{\epsilon\})$, where $\Gamma = \{\gamma \mid \gamma \in \Sigma^* \} \setminus \{\lambda\}$ and $\Delta = \{a \gamma \mid a \in \Sigma\}$. In other words, $\Delta$ is the set of non-empty strings formed by symbols from $\Sigma$, where some of these symbols may be annotated with a $\lambda$. Analogously to ordinary
NFTAs, we define the size of an augmented NFTA as the size of the encoding of its transition relation.

**Semantics (Augmented NFTA).** The semantics for deciding the acceptance of a labelled tree $t \in \text{Tree}_k[\Sigma]$ by an augmented NFTA $T^+ = (S, \Sigma, \Delta, s_{\text{init}})$ is defined by translating $T^+$ into an ordinary NFTA $T$, and checking whether $T$ accepts $t$. The translation happens in two stages:

1. First, replace every tuple in $\Delta$ of the form:
   $$(s, y_1y_2 \ldots y_j, s_1 \ldots s_k)$$
   such that $1 \leq c \leq k$ and $y_1y_2 \ldots y_j$ is a string of length $j > 1$
   with the series of transitions:
   $$\begin{align*}
   (s, y_1, r_1) \\
   (r_1, y_2, r_2) \\
   \cdots \\
   (r_{j-1}, y_j, s_1 \ldots s_k)
   \end{align*}$$
   where $r_1, \ldots, r_{j-1}$ are fresh states not occurring elsewhere in $S$, obtaining a new transition relation $\Delta'$. Defining $S' = S \cup \{r_1, \ldots, r_{j-1}\}$, we therefore obtain a new augmented NFTA $T'^+ = (S', \Sigma, \Delta', s_{\text{init}})$.

2. Next, we turn the augmented NFTA $T'^+$ obtained above into an ordinary NFTA $T$. We first define $\Sigma' = \{\sigma, \neg \sigma \mid \sigma \in \Sigma\}$. We then replace every transition in $\Delta'$ of the form:
   $$(s, \sigma, s_1 \ldots s_k)$$
   where $\sigma \in \Sigma$ with the two transitions:
   $$\begin{align*}
   (s, \sigma, s_1 \ldots s_k) \\
   (s, \neg \sigma, s_1 \ldots s_k)
   \end{align*}$$
   to get a new transition relation $\Delta''$, thereby obtaining our final NFTA $T = (S, \Sigma', \Delta'', s_{\text{init}})$.

Finally, we remark that the translation outlined above does not lead to any material blow-up in size of the translated NFTA.

**Remark 1.** The translation defined above from an augmented NFTA $T^+$ to its corresponding ordinary NFTA $T$ can be performed in time $O(\text{poly}(\mid T^+\mid))$.

### 4.2 Result

In this section, we show our approximation scheme as far as it applies to computing uniform reliability, which hinges on the construction of an augmented NFTA from the query/database pair, such that there is a bijection between the trees accepted by the NFTA with the subinstances of the database satisfying the query. To do so, we traverse the hypertree decomposition in a manner analogous to [6, Theorem 3.2], but take into account a number of adaptations necessary for the uniform reliability problem that are motivated in the previous section.

**Proposition 1.** Let $Q$ be a self-join-free conjunctive query of bounded hypertree width, and $D$ a database instance defined only over relations occurring in $Q$. Then we can construct an augmented NFTA $T^+$ in time $\text{poly}(\mid Q \mid, \mid D \mid)$ such that $|L_{\mid D\mid}(T^+)\mid = \text{UR}(Q, D)$.

**Proof.** Let $Q$ be a self-join-free query of bounded hypertree width, and $D$ be a database instance defined only over relations occurring in $Q$. We will construct an augmented NFTA $T^+ = (S, \Sigma, \Delta, s_{\text{init}})$ satisfying the desired properties. For each relation $R_i$ occurring in $D$, fix some total ordering $\preceq_i$ over the $R_i$-facts in $D$. Fix also a total ordering $\preceq_{\text{atoms}}$ on atoms($Q$). By the results discussed earlier, we can efficiently construct a complete hypertree decomposition $(\mathcal{T}, \chi, \xi)$ of $Q$ of constant width in polynomial time. We fix another total ordering $\preceq_{\text{vertices}}$, this one over vertices($T$), with the requirement that for any $p, q \in$ vertices($T$), we have $p \preceq_{\text{vertices}} q$ if and only if depth($p) \leq$ depth($q$) (where depth($p)$ denotes the distance of $p$ from the root vertex). We are now ready to begin the construction.

We start by fixing the alphabet $\Sigma = \{R_i(\overline{x}) \mid R_i(\overline{x}) \in D\}$, that is, one symbol per fact in $D$. Given a tuple of variables $\overline{x} = (x_1, \ldots, x_l)$ and constants $\overline{a} = (a_1, \ldots, a_r)$, we use the notation $\overline{\tau} \mapsto \overline{a}$ to denote the assignment of variables in $\overline{x}$ to the constants in $\overline{a}$. We assume any assignment $\overline{\tau} \mapsto \overline{a}$ is always well-behaved, in the sense that if a variable $x$ appears more than once in $\overline{x}$, it always gets assigned to the same value. We say two assignments $\overline{\tau} \mapsto \overline{a}$ and $\overline{\tau} \mapsto \overline{b}$ are consistent if every variable $x$ that occurs in both $\overline{x}$ and $\overline{y}$ gets assigned to the same value in both $\overline{a}$ and $\overline{b}$. Finally, we use the notation $t_i$ to denote the variable tuple $(x_1, \ldots, x_a)$ for every atom $R_i(x_1, \ldots, x_a)$ appearing in $Q$.

For every $p \in$ vertices($T$) such that $\chi(p) = (y_1, \ldots, y_r)$ and $\xi(p) = (R_1, \ldots, R_s)$, define:

$$S(p) = \{|p, \overline{y} \mapsto \overline{a}, \overline{\tau}_1 \mapsto \overline{c}_1, \ldots, \overline{\tau}_s \mapsto \overline{c}_s| R_i(\overline{\tau}_i) \text{ is a fact in } D \text{ for every } i \in [s]\}$$

where $\overline{\tau}_i \mapsto \overline{a}$ is consistent with $\overline{\tau}_i \mapsto \overline{c}_i$ for every $i \in [s]$.

Then we define the state set as:

$$S = \cup_{p \in \text{vertices} (T)} S(p)$$

and $s_{\text{init}} = S(p_0)$, where $p_0 \in \text{vertices} (T)$ is the root of the hypertree decomposition.

We define the transition relation $\Delta$ as follows. First consider each non-leaf $p \in \text{vertices} (T)$ such that $\chi(p) = (y_1, \ldots, y_r)$ and $\xi(p) = (R_1, \ldots, R_s)$, with child nodes $p_1, \ldots, p_l$. Denote $\chi(p_i) = (u_{i,1}, \ldots, u_{i,r})$ and $\xi(p_i) = (R_{i,1}, \ldots, R_{i,s})$ for every $i \in [l]$. Then for each such $p_i$, include all tuples of the following form in $\Delta$:

$$\begin{align*}
(p, \overline{y} \mapsto \overline{a}, \overline{\tau}_1 \mapsto \overline{c}_1, \ldots, \overline{\tau}_s \mapsto \overline{c}_s) & \rightarrow (p_1, \overline{\tau}_1 \mapsto \overline{d}_1, \overline{\tau}_2 \mapsto \overline{d}_2, \ldots, \overline{\tau}_s \mapsto \overline{d}_s) \\
(p, \overline{y} \mapsto \overline{a}, \overline{\tau}_1 \mapsto \overline{d}_1, \overline{\tau}_2 \mapsto \overline{d}_2, \ldots, \overline{\tau}_s \mapsto \overline{d}_s) & \rightarrow (p_2, \overline{\tau}_1 \mapsto \overline{d}_1, \overline{\tau}_2 \mapsto \overline{d}_2, \ldots, \overline{\tau}_s \mapsto \overline{d}_s)
\end{align*}$$

where $\overline{d_i} = (u_{i,1}, \ldots, u_{i,r})$, such that the following conditions are satisfied:

1. $|p, \overline{y} \mapsto \overline{a}, \overline{\tau}_1 \mapsto \overline{c}_1, \ldots, \overline{\tau}_s \mapsto \overline{c}_s \in S$

2. $(p_i, \overline{\tau}_1 \mapsto \overline{d}_1, \overline{\tau}_2 \mapsto \overline{d}_2, \ldots, \overline{\tau}_s \mapsto \overline{d}_s) \in S$ for every $i \in [l]$

3. $\overline{\tau}_i \mapsto \overline{d}_i$ is consistent with $\overline{\tau}_{j_i} \mapsto \overline{d}_{j_i}$ for every $i \in [s]$, $j_i \in [l]$, and $j_2 \in [s_{j_i}]$. 

Finally, we are ready to begin the construction.
where \( \Longleftrightarrow \) is consistent with \( \Longleftrightarrow \) for every \( i_1, j_1 \in [s], i_2, j_2 \in [s], \) and \( j_2 \in [s_j]. \)

(5) \( L \) is the string:

\[
R_{m_1}(b_{1,1}) \cdots R_{m_1}(b_{1,h_1}) \cdots R_{m_1}(b_{1,o_1}) \\cdots \\
\cdots \\
R_{m_n}(b_{n,1}) \cdots R_{m_n}(b_{n,h_n}) \cdots R_{m_n}(b_{n,o_n})
\]

where:

(a) \( \{m_1, \ldots, m_n\} \) is the set of all \( m_i \in [s] \) such that \( p \) is a \( \prec_{\text{vertices}} \)-minimal covering vertex of \( R_m \),
(b) \( R_m < \text{atoms} \cdots < \text{atoms} R_{m_n} \),
(c) For each \( i \in [n], R_{m_i}(b_{i,1}) <_{m_i} \cdots <_{m_i} R_{m_i}(b_{i,h_i}) <_{m_i} \cdots <_{m_i} R_{m_i}(b_{i,o_i}) \) is the ordered sequence of all \( R_{m_i} \)-facts in \( D \),
(d) For each \( i \in [n], h_i \in [o_i] \) is the unique index such that \( b_{i,h_i} = c_{m_i} \).

In particular, \( L \) is the empty string \( \lambda \) when no relation index \( m_i \) satisfies condition (a) above exists.

The case for leaf nodes in the hypertree decomposition is essentially identical. For every leaf \( \rho \in \text{vertices}(T) \) such that \( \pi(p) = \{y_1, \ldots, y_\rho\} \) and \( \xi(p) = \{R_1, \ldots, R_{\rho}\} \), include all tuples of the following form in \( \Delta \): 

\[
([p, \bar{y} \mapsto \bar{a}], \bar{r}_1 \mapsto \bar{c}_1, \ldots, \bar{r}_\rho \mapsto \bar{c}_\rho, L, \lambda)
\]

where \( \{p, \bar{y} \mapsto \bar{a}, \bar{r}_1 \mapsto \bar{c}_1, \ldots, \bar{r}_\rho \mapsto \bar{c}_\rho\} \subseteq \mathcal{S}(\rho), \) and \( L \) is the string:

\[
R_{m_1}(b_{1,1}) \cdots R_{m_1}(b_{1,h_1}) \cdots R_{m_1}(b_{1,o_1}) \\cdots \\
\cdots \\
R_{m_n}(b_{n,1}) \cdots R_{m_n}(b_{n,h_n}) \cdots R_{m_n}(b_{n,o_n})
\]

where notation is as above.

It is clear that \( T^+ \) can be constructed in time \( \text{poly}(|\mathcal{Q}|, |D|) \), observing that the number of transitions in \( \Delta \) is polynomial in \( |\mathcal{Q}| \) and \( |D| \) (assuming that the hypertree width is constant), and that each transition can be encoded in \( O(|D|) \) symbols. Thus, the only thing remaining to be proved is the existence of a bijection between the labelled trees of size \( |D| \) accepted by \( T^+ \), and the subinstances of \( D \) satisfying \( Q \). Consider some \( D' \subseteq D \) such that \( D' \models Q \). Construct a labelled tree \( t \) from \( D' \) as follows. Start with a tree with the same structure as \( T \), but perform the following procedure: contract any vertex that is not a \( \prec_{\text{vertices}} \)-minimal covering vertex for some atom in \( Q \) by deleting the vertex, and connecting its children (if any) to its parent \( t \), repeating this process until every vertex in the tree remaining is a \( \prec_{\text{vertices}} \)-minimal covering vertex. Now expand each vertex \( p \) in this tree by replacing it with a path in which each vertex is labelled with an \( R_i \)-fact or its negation (depending on its presence in \( D' \)) for every atom \( R_i \) covered by \( p \), ensuring that the orderings on the facts in \( D \) and on \( \text{atoms}(Q) \) is respected. One can check that, by construction, \( t \in \mathcal{L}(T^+) \). Clearly, the mapping just described is injective, as two distinct subinstances will lead to trees with different labelings. It is also surjective, since given any \( t \in \mathcal{L}(T^+) \), one can simply read off the labels on the vertices of \( t \).

5 PROBABILISTIC QUERY EVALUATION

The approximation scheme presented in the previous section is applicable only for computing the uniform reliability of a (bounded hypertree width, self-join-free) conjunctive query. We now consider how to extend the approach above, to allow for arbitrary rational probability values on the individual facts of the database instance.

We do this by attaching a series of extra states and transitions to the states in the NFTA constructed in Proposition 1, in order to multiply the number of trees accepted proportionally to each subinstance’s weight.

5.1 NFTAs with multipliers

Since the addition of these extra states is rather notation-heavy, we again use some syntactic sugar. We define a notion of NFTAs with multipliers below, which allow the annotation of transitions with positive integer \( n \), which we call a “multiplier”, indicating the number of extra trees that should be induced upon taking

\[\text{such a parent must always exist, since the root node must be a covering vertex by definition of a hypertree decomposition, and the root node is clearly \( \prec_{\text{vertices}} \)-minimal.}\]
that transition. Syntactically, NFTAs with multipliers are otherwise identical to NFTAs.

**Definition 2 (NFTA with multipliers).** A (top-down) non-deterministic finite tree automaton with multipliers (NFTA with multipliers, for short) is a tuple \( T^c = (\Sigma, \Delta, s_{init}) \), where \( \Sigma \) is a finite set of states, \( \Delta \) is a finite alphabet of input symbols, \( \Delta \subseteq S \times X \times N \times (\bigcup_{k=0}^{\infty} S') \) is a set of transition tuples, and \( s_{init} \in S \) is the initial state. Again like for ordinary NFTAs, we define the size of an NFTA with multipliers as the size of the encoding of its transition relation.

**Semantics (NFTA with multipliers).** The semantics of NFTAs with multipliers is similar to that of augmented NFTAs, in that acceptance of a tree \( t \in \text{Trees}_k[\Sigma] \) is determined by a polynomial-time transformation from the NFTA with multipliers to an ordinary NFTA, and then testing acceptance of \( t \) on the ordinary NFTA. Given an NFTA with multipliers \( T^c = (\Sigma, \Delta, s_{init}) \), we now show how to convert it to an ordinary NFTA \( T = (\Sigma', \Delta', s_{init}) \). We first construct \( \Delta' \) from \( \Delta \) as follows. Consider each tuple in \( \Delta \) of the form:

\[
(s, \alpha, n, s_1 \ldots s_0)
\]

with \( s, s_1, \ldots, s_0 \in S \) (\( n \geq 1 \)), \( \alpha \in \Sigma \), and \( n \in \mathbb{N} \). If \( n = 1 \), then simply add to \( \Delta' \) the tuple:

\[
(s, \alpha, t_1)
\]

Otherwise if \( n > 1 \), take the representation \( b_1 \ldots b_k \) of \( n - 1 \) as a \( k \)-bit binary string (where \( k = \lceil \log_2(n-1) \rceil + 1 \)), and add the following transitions to \( \Delta' \):

\[
1. \quad (s, \alpha, t_1)

2. \quad (t_1, 0, t_{i+1}) \quad (t_1, 1, t_{i+1})

3. \quad (t_{k+i}, 0, t_{k+i+1}) \quad (t_{k+i}, 1, t_{k+i+1})

4. \quad (t_{2k+i}, 0, s_1 \ldots s_0)

5. \quad (t_{2k+i+1}, 0, s_1 \ldots s_0)

6. \quad (t_{2k+i+2}, 0, s_1 \ldots s_0)

Set \( \Sigma' = \Sigma \cup \{t_1, t_2 \} \cup \{t_3, \ldots, t_{2k+i+2} \} \), renaming the new states as necessary to ensure they are fresh, and repeat the process above for all tuples in \( \Delta \). Finally, set \( \Sigma' = \Sigma' \cup \{0, 1\} \), where we assume \( \Sigma' \cap \{0, 1\} = \emptyset \) without loss of generality.

For every tuple \((s, \alpha, n, s_1 \ldots s_0)\) in the NFTA with multipliers, the transitions\(^7\) added by the translation above cause the number of trees accepted to be multiplied by \( n \). The additional trees are obtained by gluing on paths corresponding to binary strings of length \( k = [\log_2(n-1)] + 1 \) for \( n > 1 \), starting from:

\[
0 \ldots 0
\]

up to \( b_1 \ldots b_k \). Moreover, the number of new states added to the translated automaton for each multiplier value is logarithmic in \( n \).

**Remark 2.** The translation defined above from an NFTA with multipliers \( T^c \) to its corresponding ordinary NFTA \( T \) can be performed in time \( O(\text{poly}(|T^c|)) \).

### 5.2 Result

We can now prove the theorem from the introduction to the paper.

**Theorem 1.** Let \( Q \) be a self-join-free conjunctive query of bounded hypertree width, and \( H \) a probabilistic database instance. Then there exists an algorithm \( \text{PQEEstimate} \) such that, for all \( \epsilon \in (0, 1) \):

\[
(1 - \epsilon) \Pr_Q(H) \leq \text{PQEEstimate}(Q, H) \leq (1 + \epsilon) \Pr_Q(H)
\]

with high probability. Moreover, \( \text{PQEEstimate} \) has runtime:

\[
\text{poly}(|Q|, |H|, \epsilon^{-1})
\]

Proof. Like we did for Theorem 3, we will again show how to realize the algorithm \( \text{PQEEstimate} \), given a conjunctive query \( Q \) satisfying the specified conditions and probabilistic database instance \( H = (D, \pi) \). Recall that probability labels are rational numbers taking the form \( \pi(f_1) = w_i / d_i \). Without loss of generality, we can assume that \( D \) is defined only on relations occurring in \( Q \), since the probabilities of the additional subinstances marginalize to 1.

Let \( d \) denote the product of the denominators of all fact labels in \( D \), that is, \( d = \prod_{f_i \in D} d_i \). For any \( (D', \pi) = H' \subseteq H \), we have:

\[
\Pr_D^{\pi}(D') = \prod_{f_i \in D'} \pi(f_i) \prod_{f_i \in D \setminus D'} (1 - \pi(f_i))
\]

\[
= \prod_{f_i \in D'} w_i \prod_{f_i \in D \setminus D'} \left( \frac{d_i - w_i}{d_i} \right)
\]

\[
= d^{-1} \prod_{f_i \in D'} w_i \prod_{f_i \in D \setminus D'} (d_i - w_i)
\]

and thus:

\[
\Pr_H^Q(Q) = \sum_{D' \subseteq D, Q \subseteq D'} \prod_{f_i \in D'} d^{-1} \prod_{f_i \in D \setminus D'} (d_i - w_i)
\]

Noting that \( d \) is a known constant, it will suffice to approximate the sum term in the last equation, which we do below.

Take the augmented NFTA \( T^c \) constructed in Proposition 1 from \( Q \) and \( D \), and consider the corresponding ordinary NFTA obtained from its translation \( T = (\Sigma, \Delta, s_{init}) \). We will construct an NFTA with multipliers \( T^{c'} = (\Sigma, \Delta', s_{init}) \) from \( T \) as follows. The alphabet, state set, and initial state are identical to that of \( T \).

---

\(^7\)In fact, since the new transitions we add in the translation correspond to a degenerate NFTA accepting only paths, they can be seen as forming a non-deterministic finite string automaton. We nevertheless present the transitions in terms of tree automata so that they can be integrated with the result in Proposition 1.
We add to $\Delta'$ the tuple $(q, R_j(\overline{x}), \overline{w}_j, r)$ for every such tuple of the former type, where $\overline{w}_j$ is the numerator of the weight of the fact $R_j(\overline{x})$ appearing in $D$. Similarly, we add to $\Delta'$ the tuple $(q, \neg R_j(\overline{x}), d_i - \overline{w}_j, r)$ for every tuple of the latter type.

Since every fact in the database appears exactly once either in its positive or negated form in each of the trees accepted by $T$, it follows that:

$$|L_k(T^c)| = \sum_{D' \subseteq D} \prod_{f_i \in D'} w_i \prod_{f_i \in D', D'} (d_i - w_j)$$

where $k = |D| + \sum f_i \in D (w_i)$, and:

$$u(w_i) = \begin{cases} 0 & \text{if } w_i = 1 \\ \left\lceil \log_2(w_i - 1) \right\rceil + 1 & \text{otherwise} \end{cases}$$

Thus, we have:

$$\Pr_H(Q) = d^{-1} |L_k(T^c)|$$

and so, applying the NFTA counting procedure $\text{CountNFTA}$ [6] on the translated tree automaton $T'$ derived from $T^c$ we get:

$$(1 - \epsilon) \Pr_H(Q) \leq d^{-1} \text{CountNFTA}(k, T') \leq (1 + \epsilon) \Pr_H(Q)$$

as desired.

\[\Box\]

6 CONCLUSIONS AND FUTURE WORK

We showed how to construct a combined FPRAS for the PQE problem, by taking advantage of recent results in counting trees of a fixed size accepted by an NFTA. There are two key lines of future work we would like to explore.

First, we would like to expand our results to a wider class of queries, for example by relaxing the self-join-free condition in Theorem 1. Initial results we have obtained in this direction suggest that this may come at the cost of necessitating some mild structural constraints on the input database. The second avenue we wish to explore is integration of the proposed FPRAS procedure into practical systems for probabilistic databases. Such an integration would require both a tool for computing hypertree decompositions, as well as a practical implementation of the $\text{CountNFTA}$ algorithm.

The former task has been relatively well-studied, with ready-to-use tools available: see, for example, the 2019 PACE challenge [13], however, practically effective approximation methods for counting fixed-size trees accepted by an NFTA remain limited, given the recency of the corresponding theoretical result. Nevertheless, we are optimistic that future work will bring the constants in this algorithm down, paving the way for a practical implementation of the approach presented here.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers of this paper, whose feedback helped significantly improve the final version. The authors are also deeply grateful to Moshe Vardi, for his suggestion to study combined complexity in the context of probabilistic databases. This work was supported in part by the National Research Foundation Singapore under its NRF Fellowship programme [NRF-NRFFA11-2019-0004] and Campus for Research Excellence and Technological Enterprise (CREATE) programme, as well as the Ministry of Education Singapore Tier 1 and 2 grants R-252-000-B59-114 and MOE-TZEPE20121-0011.

REFERENCES


