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 centred 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:
Timothy van Bremen and Kuldeep S. Meel. 2023. Probabilistic Query Evaluation: The Combined FPRAS Landscape. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3584372.3588677

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 [20], 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^{12} (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:

PODS '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0127-6/23/06.
https://doi.org/10.1145/3584372.3588677

Table 1: Tractability results for PQE

Bounded HW?	Self-Join-Free?	Safe?	Prior Results (Data Complexity)	Our Results (Combined Complexity)
✓	✓	✓	FP [10]	**FPRAS**
✓	✓	✗	#P-hard [10]	**FPRAS**
✗	✓	✓	FP [10]	Open
✓/✗	✗	✓	Depends [11]	Open

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].

> 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 PQEEstimate such that, for all $\epsilon \in (0,1)$:*

$$(1-\epsilon)\Pr_H(Q) \leq \textit{PQEEstimate}(Q,H) \leq (1+\epsilon)\Pr_H(Q)$$

with high probability. Moreover, PQEEstimate has runtime:

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

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 3Path $= \cup_{i\geq 3} Q_i$ 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 3Path 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 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 $(1\pm\epsilon)$-factor with high probability can be performed in time $\mathrm{poly}(|Q|, |H|, \epsilon^{-1})$*

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

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 any atom in 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 new automaton gadget through a construction based on binary comparators, in the process scaling the number of trees accepted proportional to the fact probabilities. We can then apply the aforementioned FPRAS to count the trees accepted by this NFTA, thereby yielding our desired result.

Organization. The rest of the paper is organized as follows: we review some technical background in Section 2. Next, in Section 3, we build some intuition on how probabilistic databases and automata are connected, by proving a simplified theorem pertaining to path queries on graphs that is a special case of our main result. Section 4 is split into two parts: in Section 4.1, we introduce *augmented NFTAs*, which form a syntactic building block we use in the main result as it applies to uniform reliability in Section 4.2. We then follow a similar approach in Section 5, by first introducing *NFTAs with multipliers* in Section 5.1, which we then use in proving the primary theorem of this paper for PQE in Section 5.2. We finally conclude in Section 6, and discuss some possible directions for future work.

2 PRELIMINARIES

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

Probabilistic Databases. A *relational schema* σ 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 σ is a finite set of facts of the form $R_i(c_1, \ldots, c_k)$, where R_i is some relation name in σ 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 $\pi : D \to [0, 1]$, mapping each fact in D to an independent probability label. For the sake of more easily formalizing the representation of π, we assume in this paper that probability labels are rational (that is, $\pi : D \to [0, 1] \cap \mathbb{Q}$). The labelling π induces a probability distribution on the subinstances $D' \subseteq D$ as follows:

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

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 π. 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.

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 $Q = R_1(\overline{x_1}), \ldots, R_n(\overline{x_n})$. The set of variables occurring in Q is denoted by $\mathrm{vars}(Q)$, and the set of atoms by $\mathrm{atoms}(Q)$. Similarly, for any atom $A \in \mathrm{atoms}(Q)$, $\mathrm{vars}(A)$ denotes the set of variables occurring in A. The notation is likewise extended to sets of atoms: for a set of atoms $R \subseteq \mathrm{atoms}(Q)$, we define $\mathrm{vars}(R) = \cup_{A \in R} \mathrm{vars}(A)$. 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:

$$\Pr_H(Q) = \sum_{\substack{D' \subseteq D \\ D' \models Q}} \Pr(D')$$

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 $\mathrm{UR}(Q, D)$ 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 Σ 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 λ. We denote by Σ^* the set of all strings over Σ. A set A of strings over Σ 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 $A^0 = \{\lambda\}$.

A *non-deterministic finite (string) automaton* (NFA) is a tuple $(S, \Sigma, \delta, I, F)$ where S is a finite set of states, Σ is a finite alphabet of input symbols, $\delta : S \times \Sigma \to 2^S$ 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 $\mathcal{M}$, denoted $|\mathcal{M}|$, as the size of the encoding of its transition relation δ over a suitable alphabet. We assume the standard semantics for deciding whether a string lies in the language $\mathcal{L}(\mathcal{M})$ of strings accepted by $\mathcal{M}$. We denote by $\mathcal{L}_n(M)$ the (finite) language of strings of length n accepted by M. By the result in [5], we assume the existence of an FPRAS CountNFA for approximating $|\mathcal{L}_n(\mathcal{M})|$, running in time polynomial in n and $|\mathcal{M}|$.

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 Σ, we denote by $\mathrm{Trees}_k[\Sigma]$ the language of k-trees in which each node $u \in t$ is labelled with a symbol from Σ. We abuse notation slightly and write $t(u)$ to denote the label of a node $u \in t$. We then define a *(top-down) non-deterministic finite tree automaton* (NFTA) in the standard manner, as a tuple $\mathcal{T} = (S, \Sigma, \Delta, s_{\text{init}})$, where S is a finite set of states, Σ is a finite alphabet of input symbols, $\Delta \subseteq S \times \Sigma \times (\cup_{i=0}^{k} S^i)$ is the transition relation, and $s_{\text{init}} \in S$ is the initial state. Without loss of generality, we also slightly abuse notation by allowing λ-transitions of the form (s, λ, R) for $s \in S$ and $R \in \cup_{i=0}^{k} S^i$, noting that such an automaton can easily be converted to an equivalent one without λ-transitions using standard procedures. We sometimes refer to NFTAs as *ordinary* NFTAs, to emphasize their distinction from *augmented* NFTAs and NFTAs *with multipliers* defined later in the paper. Like for NFAs, the size of an NFTA $\mathcal{T}$, denoted $|\mathcal{T}|$, is defined as the size of the encoding of its transition relation Δ over some suitable alphabet.

A *run* of $\mathcal{T}$ over a labelled tree $t \in \mathrm{Trees}_k[\Sigma]$ is a function $\rho : t \to S$ 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 $\mathcal{T}$ *accepts* t if there exists a run of $\mathcal{T}$ over t, and write $\mathcal{L}(\mathcal{T})$ to denote the language of all labelled trees accepted by $\mathcal{T}$, and $\mathcal{L}_n(\mathcal{T})$ for the (finite) language of labelled trees of size n accepted by $\mathcal{T}$. By the result in [6], we assume the existence of an FPRAS CountNFTA for approximating $|\mathcal{L}_n(\mathcal{T})|$, running in time polynomial in n and $|\mathcal{T}|$.

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 $\langle T, \chi, \xi \rangle$, where $T = (N, E)$ is a tree on vertices N with edges E, χ and ξ are labelling functions mapping each $p \in N$ to sets of variables $\chi(p) \subseteq \mathrm{vars}(Q)$ and atoms $\xi(p) \subseteq \mathrm{atoms}(Q)$. We use $\mathrm{vertices}(T)$ as shorthand to denote N, the set of vertices of T. Moreover, for a set of vertices $P \subseteq \mathrm{vertices}(T)$, we define $\xi(P) = \cup_{p \in P} \xi(p)$.

A *hypertree decomposition* for a conjunctive query Q is a hypertree $\langle T, \chi, \xi \rangle$ that satisfies the following conditions:

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

A hypertree decomposition is said to have *width* k if $\max_{p \in \mathrm{vertices}(T)} |\xi(p)| = k$. 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 $\langle T, \chi, \xi \rangle$ be a hypertree decomposition of Q. For any $A \in \mathrm{atoms}(Q)$, we call a vertex $p \in \mathrm{vertices}(T)$ a *covering vertex for* A if $A \in \xi(p)$ and $\mathrm{vars}(A) \subseteq \chi(p)$.

We say a hypertree decomposition $\langle T, \chi, \xi \rangle$ is *complete* if every atom $A \in \mathrm{atoms}(Q)$ 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 $A \in \mathrm{atoms}(Q)$ that does not have a corresponding covering vertex, create a new vertex p_A with $\chi(P) = \mathrm{vars}(A)$ and $\xi(P) = \{A\}$. Then attach p_A as a child of some vertex p that satisfies $\chi(P) \subseteq \mathrm{vars}(A)$ (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 $\mathrm{ghtw}(Q) \leq \mathrm{htw}(Q) \leq 3 \cdot \mathrm{ghtw}(Q) + 1$, where (g)htw 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* PathEstimate *such that, for all $\epsilon \in (0, 1)$,*

$$(1 - \epsilon)\mathrm{UR}(Q, D) \leq \textsc{PathEstimate}(Q, D) \leq (1 + \epsilon)\mathrm{UR}(Q, D)$$

with high probability. Moreover, PathEstimate *has runtime:*

$$\mathrm{poly}(|Q|, |D|, \epsilon^{-1})$$

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 $\mathcal{M} = (S, \Sigma, \delta, I, F)$ whose accepted strings correspond one-to-one with the subinstances of D that satisfy Q, upon which we can apply the CountNFA FPRAS. Before doing so, we first construct a slightly different NFA $\mathcal{M}' = (S', \Sigma', \delta', I, F)$, as follows: let the alphabet be $\Sigma' = \{R_i(x, y) \mid R_i(x, y) \in D\}$, and add to S' one state $[i, x, y]$ for each fact $R_i(x, y) \in D$, as well as a single auxiliary state s_{end}. Define the transition relation δ' by adding to δ' the tuple $([i, x, y],\ R_i(x, y),\ [i + 1, y, z])$ 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 $([n, x, y],\ R_n(x, y),\ s_{\text{end}})$ 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 $F = \{s_{\text{end}}\}$.

It is not difficult to see that every string accepted by $\mathcal{M}'$ takes the form $R_1(z_1, z_2)R_2(z_2, z_3) \dots R_n(z_n, z_{n+1})$, 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 $\mathcal{M}'$ already accepts strings with atoms appearing only in the order they appear in the query $R_1 \prec \dots \prec R_n$. 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 $\mathcal{M} = (S, \Sigma, \delta, I, F)$. We first expand Σ 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:

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

Intuitively, the purpose of the state $[i, a, b, x, y]$ is to record the presence or absence of the fact $R_i(a, b)$, 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 δ from scratch, by adding the following tuples for every $i \in [n-1]$ and joining pair of facts $R_i(a_k, b_k), R_{i+1}(b_k, f) \in D$:

$$\begin{aligned}
&([i, a_1, b_1, a_k, b_k],\ R_i(a_1, b_1),\ [i, a_2, b_2, a_k, b_k])\\
&([i, a_1, b_1, a_k, b_k],\ \neg R_i(a_1, b_1),\ [i, a_2, b_2, a_k, b_k])\\
&\qquad\dots\\
&([i, a_k, b_k, a_k, b_k],\ R_i(a_k, b_k),\ [i, a_{k+1}, b_{k+1}, a_k, b_k])\\
&\qquad\dots\\
&([i, a_{c_i}, b_{c_i}, a_k, b_k],\ R_i(a_{c_i}, b_{c_i}),\ [i + 1, p, q, b_k, f])\\
&([i, a_{c_i}, b_{c_i}, a_k, b_k],\ \neg R_i(a_{c_i}, b_{c_i}),\ [i + 1, p, q, b_k, f])
\end{aligned}$$

where $R_i(a_1, b_1) \prec_i \dots \prec_i R_i(a_k, b_k) \prec_i \dots \prec_i R_i(a_{c_i}, b_{c_i})$ 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 $R_n(a_k, b_k) \in D$ add the tuples:

$$\begin{aligned}
&([n, a_1, b_1, a_k, b_k],\ R_i(a_1, b_1),\ [i, a_2, b_2, a_k, b_k])\\
&([n, a_1, b_1, a_k, b_k],\ \neg R_i(a_1, b_1),\ [i, a_2, b_2, a_k, b_k])\\
&\qquad\dots\\
&([n, a_k, b_k, a_k, b_k],\ R_i(a_k, b_k),\ [i, a_{k+1}, b_{k+1}, a_k, b_k])\\
&\qquad\dots\\
&([n, a_{c_n}, b_{c_n}, a_k, b_k],\ R_i(a_{c_n}, b_{c_n}),\ s_{\text{end}})\\
&([n, a_{c_n}, b_{c_n}, a_k, b_k],\ \neg R_i(a_{c_n}, b_{c_n}),\ s_{\text{end}})
\end{aligned}$$

where $R_n(a_1, b_1) \prec_n \dots \prec_n R_n(a_k, b_k) \prec_n \dots \prec_n R_i(a_{c_n}, b_{c_n})$ are the R_n-facts in D. Finally, set $I = \{[1, a, b, x, y] \in S\}$, and $F = \{s_{\text{end}}\}$. One can check that the strings accepted by $\mathcal{M}$ will always list the presence or absence of each fact in a consistent order. Moreover, by construction the strings it accepts (all of which have length $|D|$) correspond precisely to the subsets $D' \subseteq D$ satisfying Q. Lastly, note that $|\mathcal{M}|$ is clearly polynomial in both $|Q|$ and $|D|$.

Thus, we can apply the FPRAS CountNFA [5] on $\mathcal{M}$ to approximate the number of strings accepted of length $|D|$, which yields with high probability a $(1 \pm \epsilon)$-approximation of the uniform reliability of Q on D. Hence, we can realize the algorithm PathEstimate as above. □

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 $\gamma_1 \dots \gamma_n$ (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 $\gamma_1 \dots \gamma_n$ is accepted.
(2) (*? Symbols*) Second, we also allow a symbol γ_i appearing in this string to be annotated with a $^?$, as shorthand expressing that either the symbol γ_i or $\neg\gamma_i$ 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 $\mathcal{T}^+ = (S, \Sigma, \Delta, s_{init})$, where S is a finite set of states, Σ is a finite alphabet of input symbols, and $s_{init} \in S$ is the initial state. The transition relation is defined as $\Delta \subseteq S \times \Gamma \times (\cup_{i=0}^{k} S^i)$, where $\Gamma = \{\gamma \mid \gamma \in \zeta^*\} \setminus \lambda$ and $\zeta = \{\alpha,\ \alpha^? \mid \alpha \in \Sigma\}$. In other words, Γ is the set of non-empty strings formed by symbols from Σ, where some of these symbols may be annotated with a $^?$. 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 $t \in Trees_k[\Sigma]$ by an augmented NFTA $\mathcal{T}^+ = (S, \Sigma, \Delta, s_{init})$ is defined by translating $\mathcal{T}^+$ into an ordinary NFTA $\mathcal{T}$, and checking whether $\mathcal{T}$ accepts t. The translation happens in two stages:*

(1) *First, replace every tuple in Δ of the form:*

$$(s, \gamma_1\gamma_2 \dots \gamma_j, s_1 \dots s_k)$$

such that $1 \leq c \leq k$ and $\gamma_1\gamma_2 \dots \gamma_j$ is a string of length $j > 1$ with the series of transitions:

$$(s, \gamma_1, r_1)$$
$$(r_1, \gamma_2, r_2)$$
$$\dots$$
$$(r_{j-1}, \gamma_j, s_1 \dots s_k)$$

where $r_1, \dots, r_{j-1}$ are fresh states not occurring elsewhere in S, obtaining a new transition relation Δ'. Defining $S' = S \cup \{r_1, \dots, r_{j-1}\}$, we therefore obtain a new augmented NFTA $\mathcal{T}'^+ = (S', \Sigma, \Delta', s_{init})$.

(2) *Next, we turn the augmented NFTA $\mathcal{T}'^+$ obtained above into an ordinary NFTA $\mathcal{T}$. We first define $\Sigma' = \{\alpha, \neg\alpha \mid \alpha \in \Sigma\}$. We then replace every transition in Δ' of the form:*

$$(s, \alpha^?, s_1 \dots s_k)$$

where $\alpha \in \Sigma$ with the two transitions:

$$(s, \alpha, s_1 \dots s_k)$$
$$(s, \neg\alpha, s_1 \dots s_k)$$

to get a new transition relation Δ'', thereby obtaining our final NFTA $\mathcal{T} = (S, \Sigma', \Delta'', s_{init})$.

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 $\mathcal{T}^+$ to its corresponding ordinary NFTA $\mathcal{T}$ can be performed in time $O(\text{poly}(|\mathcal{T}^+|))$.*

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 adapations 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 $\mathcal{T}^+$ in time $\text{poly}(|Q|, |D|)$ such that $|\mathcal{L}_{|D|}(\mathcal{T}^+)| = \text{UR}(Q, D)$.*

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 $\mathcal{T}^+ = (S, \Sigma, \Delta, s_{\text{init}})$ satisfying the desired properties. For each relation R_i occurring in D, fix some total ordering $\prec_i$ over the R_i-facts in D. Fix also a total ordering $\prec_{\text{atoms}}$ on $\text{atoms}(Q)$. By the results discussed earlier, we can efficiently construct a complete hypertree decomposition $\langle T, \chi, \xi\rangle$ of Q of constant width in polynomial time. We fix another total ordering $\prec_{\text{vertices}}$, this one over $\text{vertices}(T)$, with the requirement that for any $p, q \in \text{vertices}(T)$, we have $p \prec_{\text{vertices}} q$ if and only if $\text{depth}(p) \leq \text{depth}(q)$ (where $\text{depth}(p)$ denotes the distance of p from the root vertex). We are now ready to begin the construction.

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

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

$$\begin{aligned} S(p) = \{[p, \bar{y} \mapsto \bar{a}, \overline{t_1} \mapsto \overline{c_1}, \dots, \overline{t_s} \mapsto \overline{c_s}] \mid \\ R_i(\overline{c_i}) \text{ is a fact in } D \text{ for every } i \in [s], \\ \bar{y} \mapsto \bar{a} \text{ is consistent with } \overline{t_i} \mapsto \overline{c_i} \text{ for every } i \in [s] \\ \overline{t_i} \mapsto \overline{c_i} \text{ is consistent with } \overline{t_j} \mapsto \overline{c_j} \text{ for every } i,\ j \in [s]\} \end{aligned}$$

where $\bar{y} = (y_1, \dots, y_r)$. We then 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 Δ as follows. First consider each non-leaf $p \in \text{vertices}(T)$ such that $\chi(p) = \{y_1, \dots, y_r\}$ and $\xi(p) = \{R_1, \dots, R_s\}$, with child nodes $p_1, \dots, p_l$ ($l \geq 1$). Denote $\chi(p_i) = \{u_{i,1}, \dots, u_{i,r_i}\}$ and $\xi(p_i) = \{R_{i,1}, \dots, R_{i,s}\}$ for $i \in [l]$. Then for each such p, include all tuples of the following form in Δ:

$$\begin{aligned} &([p, \bar{y} \mapsto \bar{a}, \overline{t_1} \mapsto \overline{c_1}, \dots, \overline{t_s} \mapsto \overline{c_s}], L, \\ &[p_1, \overline{u_1} \mapsto \overline{d_1}, \overline{t_{1,1}} \mapsto \overline{f_{1,1}}, \dots, \overline{t_{1,s_1}} \mapsto \overline{f_{1,s_1}}] \\ &\dots \\ &[p_l, \overline{u_l} \mapsto \overline{d_l}, \overline{t_{l,1}} \mapsto \overline{f_{l,1}}, \dots, \overline{t_{l,s_l}} \mapsto \overline{f_{l,s_l}}]) \end{aligned}$$

where $\overline{u_i} = (u_{i,1}, \dots, u_{i,r_i})$, such that the following conditions are satisfied:

(1) $[p, \bar{y} \mapsto \bar{a}, \overline{t_1} \mapsto \overline{c_1}, \dots, \overline{t_s} \mapsto \overline{c_s}] \in S$
(2) $[p_i, \overline{u_i} \mapsto \overline{d_i}, \overline{t_{i,1}} \mapsto \overline{f_{i,1}}, \dots, \overline{t_{i,s_i}} \mapsto \overline{f_{i,s_i}}] \in S$ for every $i \in [l]$
(3) $\overline{t_i} \mapsto \overline{c_i}$ is consistent with $\overline{t_{j_1,j_2}} \mapsto \overline{f_{j_1,j_2}}$ for every $i \in [s]$, $j_1 \in [l]$, and $j_2 \in [s_{j_i}]$

(4) $\overline{t_{i_1,i_2}} \mapsto \overline{f_{i_1,i_2}}$ is consistent with $\overline{t_{j_1,j_2}} \mapsto \overline{f_{j_1,j_2}}$ for every $i_1, j_1 \in [l]$, $i_2 \in [s_{i_1}]$, and $j_2 \in [s_{j_1}]$.
(5) L is the string:

$$R_{m_1}(\overline{b_{1,1}})^? \dots R_{m_1}(\overline{b_{1,h_1}}) \dots R_{m_1}(\overline{b_{1,o_1}})^?$$
$$\dots$$
$$R_{m_n}(\overline{b_{n,1}})^? \dots R_{m_n}(\overline{b_{n,h_n}}) \dots R_{m_n}(\overline{b_{n,o_n}})^?$$

where:
(a) $\{m_1, \dots m_n\}$ is the set of all $m_i \in [s]$ such that p is a $\prec_{\text{vertices}}$-minimal covering vertex of R_{m_i}
(b) $R_{m_1} \prec_{\text{atoms}} \cdots \prec_{\text{atoms}} R_{m_n}$
(c) For each $i \in [n]$, $R_{m_i}(\overline{b_{i,1}}) \prec_{m_i} \cdots \prec_{m_i} R_{m_i}(\overline{b_{i,h_i}}) \prec_{m_i} \cdots \prec_{m_i} R_{m_i}(\overline{b_{i,o_i}})$ 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 $\overline{b_{i,h_i}} = \overline{c_{m_i}}$

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

The case for leaf nodes in the hypertree decomposition is essentially identical. For every leaf $p \in \text{vertices}(T)$ such that $\chi(p) = \{y_1, \dots, y_r\}$ and $\xi(p) = \{R_1, \dots, R_s\}$, include all tuples of the following form in Δ:

$$([p, \bar{y} \mapsto \bar{a}, \overline{t_1} \mapsto \overline{c_1}, \dots, \overline{t_s} \mapsto \overline{c_s}], L, \lambda)$$

where $[p, \bar{y} \mapsto \bar{a}, \overline{t_1} \mapsto \overline{c_1}, \dots, \overline{t_s} \mapsto \overline{c_s}] \in S(p)$, and L is the string:

$$R_{m_1}(\overline{b_{1,1}})^? \dots R_{m_1}(\overline{b_{1,h_1}}) \dots R_{m_1}(\overline{b_{1,o_1}})^?$$
$$\dots$$
$$R_{m_n}(\overline{b_{n,1}})^? \dots R_{m_n}(\overline{b_{n,h_n}}) \dots R_{m_n}(\overline{b_{n,o_n}})^?$$

where notation is as above.

It is clear that $\mathcal{T}^+$ can be constructed in time $\text{poly}(|Q|, |D|)$, observing that the number of transitions in Δ is polynomial in $|Q|$ 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 $\mathcal{T}^+$, 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[1], 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, $t \in \mathcal{L}(\mathcal{T}^+)$. Clearly, the mapping just described is injective, as two distinct subinstances will lead to trees with different labellings. It is also surjective, since given any $t \in \mathcal{L}(\mathcal{T}^+)$, one can simply read off the labels on the vertices of t to reconstruct the corresponding subinstance $D' \subseteq D$. This shows the existence of a bijection, as desired. □

[1]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.

Using this construction we can now state our result for self-join-free conjunctive queries of bounded hypertree width, as far as it applies to uniform reliability. Note that in the theorem statement below we have dropped the condition on the database schema that was present in Proposition 1.

Theorem 3. *Let Q be a self-join-free conjunctive query of bounded hypertree width, and D a database instance. Then there exists an algorithm* UREstimate *such that, for all $\epsilon \in (0, 1)$,*

$$(1-\epsilon)\text{UR}(Q, D) \leq \textit{UREstimate}(Q, D) \leq (1+\epsilon)\text{UR}(Q, D)$$

with high probability. Moreover, UREstimate *has runtime:*

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

Proof. We will show how to realize the algorithm UREstimate, given a conjunctive query Q satisfying the specified conditions and database instance D.

Consider the subinstance $D' \subseteq D$ "projected" on the relations in Q, obtained from D by removing all facts over relations that do not occur in Q. By Proposition 1, we can construct an augmented NFTA $\mathcal{T}^+$ such that $|\mathcal{L}_{|D'|}(\mathcal{T}^+)| = \text{UR}(Q, D')$ in time polynomial in $|Q|$ and $|D'|$. Observing that the semantics for this augmented NFTA $\mathcal{T}^+$ is determined by translation to an ordinary NFTA $\mathcal{T}$ accepting the same trees as described in Section 4.1, and that this translation can be performed in polynomial time, we can tractably approximate $\text{UR}(Q, D')$ by applying the NFTA counting algorithm CountNFTA presented in [6] to $\mathcal{T}$. Thus, we have:

$$(1-\epsilon)\text{UR}(Q, D') \leq \text{CountNFTA}(|D'|, \mathcal{T}) \leq (1+\epsilon)\text{UR}(Q, D')$$

Noting that $\text{UR}(Q, D') = 2^{-|D \setminus D'|}\text{UR}(Q, D)$, we get:

$$\frac{(1-\epsilon)}{2^{|D\setminus D'|}}\text{UR}(Q, D) \leq \text{CountNFTA}(|D'|, \mathcal{T}) \leq \frac{(1+\epsilon)}{2^{|D\setminus D'|}}\text{UR}(Q, D)$$

and hence:

$$(1-\epsilon)\text{UR}(Q, D) \leq \text{CountNFTA}(|D'|, \mathcal{T})2^{|D\setminus D'|} \leq (1+\epsilon)\text{UR}(Q, D)$$

thereby showing how to realize UREstimate as desired. □

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 a positive integer n, which we call a "multiplier", indicating the number of extra trees that should be induced upon taking

that transition. Syntactically, NFTAs with multipliers are otherwise identical to NFTAs.

Definition 2 (NFTA with multipliers)**.** *A (top-down) nondeterministic finite tree automaton with multipliers (NFTA with multipliers, for short) is a tuple $\mathcal{T}^c = (S, \Sigma, \Delta, s_{init})$, where S is a finite set of states, Σ is a finite alphabet of input symbols, $\Delta \subseteq S \times \Sigma \times \mathbb{N} \times (\cup_{i=0}^{k} S^i)$ 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 $t \in Trees_k[\Sigma]$ 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 $\mathcal{T}^c = (S, \Sigma, \Delta, s_{init})$, we now show how to convert it to an ordinary NFTA $\mathcal{T} = (S', \Sigma', \Delta', s_{init})$. We first construct Δ' from Δ as follows. Consider each tuple in Δ of the form:*

$$(s, \alpha, n, s_1 \dots s_v)$$

with $s, s_1, \dots s_v \in S$ ($v \geq 1$), $\alpha \in \Sigma$, and $n \in \mathbb{N}$. If $n = 1$, then simply add to Δ' the tuple:

$$(s, \alpha, s_1 \dots s_v)$$

Otherwise if $n > 1$, take the representation $b_1 \dots b_k$ of $n-1$ as a k-bit binary string (where $k = \lfloor \log_2(n-1) \rfloor + 1$), and add the following transitions to Δ':

(1)

$$(s, \alpha, t_1)$$
$$(t_1, 0, t_2) \quad (t_1, 1, t_{k+1})$$
$$(t_k, 0, s_1 \dots s_v) \quad (t_k, 1, s_1 \dots s_v)$$

(2) *For all $2 \leq i < k$:*

$$(t_i, 0, t_{i+1}) \quad (t_i, 1, t_{i+1})$$

(3) *For all $2 \leq i < k$ such that $b_i = 1$:*

$$(t_{k+i-1}, 0, t_{i+1}) \quad (t_{k+i-1}, 1, t_{k+i})$$

(4) *For all $2 \leq i < k$ such that $b_i = 0$:*

$$(t_{k+i-1}, 0, t_{k+i})$$

(5) *If $b_k = 0$:*

$$(t_{2k-1}, 0, s_1 \dots s_v)$$

(6) *If $b_k = 1$:*

$$(t_{2k-1}, 0, s_1 \dots s_v) \quad (t_{2k-1}, 1, s_1 \dots s_v)$$

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

For every tuple $(s, \alpha, n, s_1 \dots s_v)$ in the NFTA with multipliers, the transitions[2] 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 $k = \lfloor \log_2(n-1) \rfloor + 1$ (for $n > 1$), starting from:

$$\underbrace{0 \dots 0}_{k \text{ times}}$$

up to $b_1 \dots b_k$. 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 $\mathcal{T}^c$ to its corresponding ordinary NFTA $\mathcal{T}$ can be performed in time $O(\text{poly}(|\mathcal{T}^c|))$.*

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 PQEEstimate such that, for all $\epsilon \in (0, 1)$:*

$$(1 - \epsilon) \Pr_H(Q) \leq \textit{PQEEstimate}(Q, H) \leq (1 + \epsilon) \Pr_H(Q)$$

with high probability. Moreover, 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 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 $\pi(f_i) = w_i/d_i$. 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:

$$\begin{aligned}\Pr_\pi(D') &= \prod_{f_i \in D'} \pi(f_i) \prod_{f_i \in D \setminus D'} (1 - \pi(f_i)) \\ &= \prod_{f_i \in D'} \frac{w_i}{d_i} \prod_{f_i \in D \setminus D'} \left(\frac{d_i - w_i}{d_i} \right) \\ &= d^{-1} \prod_{f_i \in D'} w_i \prod_{f_i \in D \setminus D'} (d_i - w_i)\end{aligned}$$

and thus:

$$\begin{aligned}\Pr_H(Q) = \sum_{\substack{D' \subseteq D \\ D' \models Q}} \Pr_\pi(D') &= \sum_{\substack{D' \subseteq D \\ D' \models Q}} d^{-1} \prod_{f_i \in D'} w_i \prod_{f_i \in D \setminus D'} (d - w_i) \\ &= d^{-1} \sum_{\substack{D' \subseteq D \\ D' \models Q}} \prod_{f_i \in D'} w_i \prod_{f_i \in D \setminus D'} (d - w_i)\end{aligned}$$

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

[2] 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.

The transition set Δ' is obtained from Δ as follows. It is clear that every transition in Δ must either take the form[3]:

$$(q,\ R_j(\bar{x}),\ r)$$

or

$$(q,\ \neg R_j(\bar{x}),\ r)$$

We add to Δ' the tuple $(q,\ R_j(\bar{x}),\ w_i,\ r)$ for every such tuple of the former type, where w_i is the numerator of the weight of the fact $R_j(\bar{x})$ appearing in D. Similarly, we add to Δ' the tuple $(q,\ \neg R_j(\bar{x}),\ d_i - w_i,\ r)$ 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 $\mathcal{T}$, it follows that:

$$|\mathcal{L}_k(\mathcal{T}^c)| = \sum_{\substack{D' \subseteq D \\ D' \not\models Q}} \prod_{f_i \in D'} w_i \prod_{f_i \in D \setminus D'} (d_i - w_i)$$

where $k = |D| + \sum_{f_i \in D} u(w_i)$, and:

$$u(w_i) = \begin{cases} 0 & \text{if } w_i = 1 \\ \lfloor \log_2(w_i - 1) \rfloor + 1 & \text{otherwise} \end{cases}$$

Thus, we have:

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

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

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

as desired. □

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 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.

[3]We assume here that λ-transitions in $\mathcal{T}$ have already been eliminated using standard procedures.

ACKNOWLEDGMENTS

The authors 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, and to Antoine Amarilli, Mikaël Monet, and Pierre Senellart for helpful discussions. This work was supported in part by the National Research Foundation Singapore under its NRF Fellowship programme [NRF-NRFFAI1-2019-0004] 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-T2EP20121-0011.

REFERENCES

[1] Isolde Adler, Georg Gottlob, and Martin Grohe. 2007. Hypertree width and related hypergraph invariants. *Eur. J. Comb.* 28, 8 (2007), 2167–2181.
[2] Antoine Amarilli and İsmail İlkan Ceylan. 2022. The Dichotomy of Evaluating Homomorphism-Closed Queries on Probabilistic Graphs. *Log. Methods Comput. Sci.* 18, 1 (2022).
[3] Antoine Amarilli and Benny Kimelfeld. 2021. Uniform Reliability of Self-Join-Free Conjunctive Queries. In *ICDT (LIPIcs, Vol. 186)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 17:1–17:17.
[4] Antoine Amarilli, Mikaël Monet, and Pierre Senellart. 2017. Conjunctive Queries on Probabilistic Graphs: Combined Complexity. In *PODS*. ACM, 217–232.
[5] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2021. #NFA Admits an FPRAS: Efficient Enumeration, Counting, and Uniform Generation for Logspace Classes. *J. ACM* 68, 6 (2021), 48:1–48:40.
[6] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2021. When is approximate counting for conjunctive queries tractable?. In *STOC*. ACM, 1015–1027.
[7] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*. ACM, 77–90.
[8] Nilesh N. Dalvi, Karl Schnaitter, and Dan Suciu. 2010. Computing query probability with incidence algebras. In *PODS*. ACM, 203–214.
[9] Nilesh N. Dalvi and Dan Suciu. 2004. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*. Morgan Kaufmann, 864–875.
[10] Nilesh N. Dalvi and Dan Suciu. 2007. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*. ACM, 293–302.
[11] Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30:1–30:87.
[12] Guy Van den Broeck and Dan Suciu. 2017. Query Processing on Probabilistic Data: A Survey. *Found. Trends Databases* 7, 3-4 (2017), 197–341.
[13] M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. 2019. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration (Invited Paper). In *IPEC (LIPIcs, Vol. 148)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 25:1–25:23.
[14] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. *VLDB J.* 22, 6 (2013), 823–848.
[15] Robert Fink and Dan Olteanu. 2011. On the optimal approximation of queries using tractable propositional languages. In *ICDT*. ACM, 174–185.
[16] Wolfgang Gatterbauer, Abhay Kumar Jha, and Dan Suciu. 2010. Dissociation and Propagation for Efficient Query Evaluation over Probabilistic Databases. In *MUD (CTIT Workshop Proceedings Series, Vol. WP10-04)*. Centre for Telematics and Information Technology (CTIT), University of Twente, The Netherlands, 83–97.
[17] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *PODS*. ACM, 57–74.
[18] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.* 64, 3 (2002), 579–627.
[19] Erich Grädel, Yuri Gurevich, and Colin Hirsch. 1998. The Complexity of Query Reliability. In *PODS*. ACM Press, 227–234.
[20] Abhay Kumar Jha, Dan Olteanu, and Dan Suciu. 2010. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT (ACM International Conference Proceeding Series, Vol. 426)*. ACM, 323–334.
[21] Phokion G. Kolaitis and Moshe Y. Vardi. 1998. Conjunctive-Query Containment and Constraint Satisfaction. In *PODS*. ACM Press, 205–213.
[22] Dan Olteanu, Jiewen Huang, and Christoph Koch. 2010. Approximate confidence computation in probabilistic databases. In *ICDE*. IEEE Computer Society, 145–156.
[23] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool Publishers.
[24] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*. ACM, 137–146.