When Is Approximate Counting for Conjunctive Queries Tractable?

Marcelo Arenas
PUC & IMFD Chile
Santiago, Chile
marenas@ing.puc.cl

Luis Alberto Croquevielle
PUC & IMFD Chile
Santiago, Chile
lacroquevielle@uc.cl

Rajesh Jayaram
Carnegie Mellon University
Pittsburgh, USA
rkjayara@cs.cmu.edu

Cristian Riveros
PUC & IMFD Chile
Santiago, Chile
cristian.riveros@uc.cl

ABSTRACT

Conjunctive queries are one of the most common class of queries used in database systems, and the best studied in the literature. A seminal result of Grohe, Schwentick, and Segoufin (STOC 2001) demonstrates that for every class $\mathcal{G}$ of graphs, the evaluation of all conjunctive queries whose underlying graph is in $\mathcal{G}$ is tractable if, and only if, $\mathcal{G}$ has bounded treewidth. In this work, we extend this characterization to the counting problem for conjunctive queries. Specifically, for every class C of conjunctive queries with bounded treewidth, we introduce the first fully polynomial-time randomized approximation scheme (FPRAS) for counting answers to a query in C, and the first polynomial-time algorithm for sampling answers uniformly from a query in C. As a corollary, it follows that for every class $\mathcal{G}$ of graphs, the counting problem for conjunctive queries whose underlying graph is in $\mathcal{G}$ admits an FPRAS if, and only if, $\mathcal{G}$ has bounded treewidth (unless BPP $\neq$ P). In fact, our FPRAS is more general, and also applies to conjunctive queries with bounded *hypertree width*, as well as unions of such queries.

The key ingredient in our proof is the resolution of a fundamental counting problem from automata theory. Specifically, we demonstrate the first FPRAS and polynomial time sampler for the set of trees of size n accepted by a *tree automaton*, which improves the prior quasi-polynomial time randomized approximation scheme (QPRAS) and sampling algorithm of Gore, Jerrum, Kannan, Sweedyk, and Mahaney '97. We demonstrate how this algorithm can be used to obtain an FPRAS for many open problems, such as counting solutions to constraint satisfaction problems (CSP) with bounded hypertree width, counting the number of error threads in programs with nested call subroutines, and counting valid assignments to structured DNNF circuits.

CCS CONCEPTS

• Theory of computation → Tree languages; **Design and analysis of algorithms**; **• Information systems → Relational database query languages**.

KEYWORDS

Conjunctive queries, fully polynomial-time approximation scheme (FPRAS), approximate counting, tree automata

ACM Reference Format:
Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2021. When Is Approximate Counting for Conjunctive Queries Tractable?. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3406325.3451014

1 INTRODUCTION

Conjunctive queries (CQ) are expressions of the form

$$Q(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_n(\bar{y}_n)$$

where each R_i is a relational symbol, each $\bar{y}_i$ is a tuple of variables, and $\bar{x}$ is a tuple of output variables with $\bar{x} \subseteq \bar{y}_1 \cup \cdots \cup \bar{y}_n$. Conjunctive queries are the most common class of queries used in database systems. They correspond to *select-project-join* queries in relational algebra and *select-from-where* queries in SQL, and are closely related to *constraint satisfaction problems* (CSPs). Therefore, the computational complexity of tasks related to the evaluation of conjunctive queries is a fundamental object of study. Given as input a database instance D and a conjunctive query $Q(\bar{x})$, the *query evaluation problem* is defined as the problem of computing $Q(D) := \{\bar{a} \mid D \models Q(\bar{a})\}$. Namely, $Q(D)$ is the set of answers $\bar{a}$ to Q over D, where $\bar{a}$ is an assignment of the variables $\bar{x}$ which agrees with the relations R_i. The corresponding *query decision problem* is to verify whether or not $Q(D)$ is empty. It is well known that even the query decision problem is NP-complete for conjunctive queries [16]. Thus, a major focus of investigation in the area has been to find tractable special cases [18, 28, 32, 33, 35, 37, 62].

In addition to evaluation, two fundamental problems for conjunctive queries are counting the number of answers to a query and uniformly sampling such answers. The counting problem for CQ is of fundamental importance for query optimization [47, 50]. Specifically, the optimization process of a relational query engine requires, as input, an estimate of the number of answers to a query (without evaluating the query). Furthermore, uniform sampling is used to efficiently generate representative subsets of the data, instead of computing the entire query, which are often sufficient for data mining and statistical tasks [1]. Starting with the work of Chaudhuri, Motwani and Narasayya [17], the study of random sampling from queries has attracted significant attention from the database community [19, 63].

Beginning with the work in [62], a fruitful line of research for finding tractable cases for CQs has been to study the *degree of acyclicity* of a CQ. In particular, the *treewidth* $\mathrm{tw}(Q)$ of Q [18, 37], and more generally the *hypertree width* $\mathrm{hw}(Q)$ of Q [35], are two primary measurements of the degree of acyclicity. It is known that the query decision problem can be solved in polynomial time for every class C of CQs with bounded treewidth [18, 37] or bounded hypertree width [35].[1] A seminal result of Grohe, Schwentick, and Segoufin [37] demonstrates that for every class $\mathcal{G}$ of graphs, the evaluation of all conjunctive queries whose underlying graph is in $\mathcal{G}$ is tractable if, and only if, $\mathcal{G}$ has bounded treewidth. Hence, the property of bounded treewidth provides a characterization of tractability of the query decision problem.

Unfortunately, uniform generation and exact counting are more challenging than query evaluation for CQs. Specifically, given as input a conjunctive query Q and database D, computing $|Q(D)|$ is #P-complete even when $\mathrm{tw}(Q) = 1$ [47] (that is, for so called *acyclic* CQs [62]). Moreover, even approximate counting is intractable for queries with unbounded treewidth, since any multiplicative approximation clearly solves the decision problem. On the other hand, these facts do not preclude the existence of efficient approximation algorithms for classes of CQs with bounded treewidth, as the associated query decision problem is in P. Despite this possibility, to date no efficient approximation algorithms for these classes are known.

In this paper, we fill this gap by demonstrating the existence of a fully polynomial-time randomized approximation scheme (FPRAS) and a fully polynomial-time almost uniform sampler (FPAUS) for every class of CQs with bounded hypertree width. Since $\mathrm{hw}(Q) \leq \mathrm{tw}(Q)$ for every CQ Q [35], our result also includes every class of CQs with bounded treewidth, as well as classes of CQs with bounded hypertree width but unbounded treewidth [35]. Specifically, we show the following.

Theorem 1.1 (Theorem 3.2 informal)**.** *Let C be a class of CQs with bounded hypertree width. Then there exists a fully polynomial-time randomized approximation scheme (FPRAS) that, given $Q \in C$ and a database D, estimates $|Q(D)|$ to multiplicative error $(1 \pm \epsilon)$. Moreover, there is a fully polynomial-time almost uniform sampler (FPAUS) that generates samples from $Q(D)$.*

Our algorithm of Theorem 1.1 in fact holds for a larger class of queries, including *unions* of conjunctive queries with bounded hypertree width (Proposition 3.5). Note that, as defined in [40], an FPAUS samples from a distribution with variational distance δ from uniform (see Section 2 for a formal definition).

An interesting question is whether there exists a larger class of queries C that admits an FPRAS. Since the decision problem for C is in BPP whenever C admits an FPRAS, as a corollary of Theorem 1.1 and the characterization of [37], we obtain the following answer to this question (see Section 3.3 for a precise statement of this result, and for the necessary terminology for this statement):

[1] C has bounded treewidth (hypertree width) if $\mathrm{tw}(Q) \leq k$ ($\mathrm{hw}(Q) \leq k$) for every $Q \in C$, for a fixed constant k.

Corollary 1.2 (Corollary 3.3 informal)**.** *Let $\mathcal{G}$ be a class of (undirected) graphs and C be the class of all CQs whose underlying graph is in $\mathcal{G}$. Then assuming $W[1] \neq FPT$ and $BPP = P$, the following are equivalent: (1) the problems of computing $|Q(D)|$ and of sampling from $Q(D)$, given as input $Q \in C$ and a database D, admit an FPRAS and an FPAUS, respectively; and (2) $\mathcal{G}$ has bounded treewidth.*

Corollary 1.2 shows that the results of [37] can be extended to the approximate counting problem for CQs. Perhaps surprisingly, this demonstrates that the classes of CQs for which the decision problem is tractable, in the sense studied in [37], are precisely *the same* as the classes which admit an FPRAS. Besides, this gives a positive answer to the line of research started in [17], by providing a characterization of the class of queries that admit an almost uniform sampler.

1.1 An FPRAS for Tree Automata

The key to our results is the resolution of a fundamental counting problem from automata theory; namely, the counting problem for *tree automata*. Specifically, we first demonstrate that the solution space $Q(D)$ of a conjunctive query with bounded hypertree width can be efficiently expressed as the language accepted by a tree automaton $\mathcal{T}$. We then demonstrate the first FPRAS for the problem of counting the number of trees accepted by a tree automaton $\mathcal{T}$.

Tree automata are the natural extension of *non-deterministic finite automata* (NFA) from words to trees. This extension is a widely studied topic, since they have a remarkable capacity to model problems, while retaining many of the desirable computational properties of NFAs [54, 57]. Beginning with the strong decidability result established by Rabin [49], many important problems have been shown to be decidable via tree automata. Moreover, the fact that tree automata are equivalent to monadic second order-logic [56] is a basic component of the proof of Courcelle's theorem [21]. Further applications of tree automata, among others, include model checking [27, 60], program analysis [2–4], databases [45, 53], and knowledge representation [10, 15, 55] (see also [20, 57] for a survey).

The counting problem of tree automata. Similarly to how a non-deterministic finite automaton N accepts a language $\mathcal{L}(N)$ of words, a tree automaton $\mathcal{T}$ accepts a language $\mathcal{L}(\mathcal{T})$ of labeled trees. Given $\mathcal{T}$ and an integer n, define the n-slice of $\mathcal{L}(\mathcal{T})$ as:

$$\mathcal{L}_n(\mathcal{T}) = \{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}$$

where $|t|$ is the number of vertices in t. We consider the counting problem #TA, which is the main counting problem studied in this paper regarding tree automata:

Problem:	#TA
Input:	A tree automaton $\mathcal{T}$ and a string 0^n
Output:	$\lvert\mathcal{L}_n(\mathcal{T})\rvert$

While exactly computing the size of the n-slice for deterministic finite automata and deterministic tree automata is tractable [11, 42, 44], this is not the case for their *non-deterministic* counterparts. In fact, given as input an NFA $\mathcal{A}$ and a number n in unary, the problem of computing $|\mathcal{L}_n(\mathcal{A})|$ is #P-hard [5], which implies #P-hardness for tree automata. Naturally, this does not rule out the possibility of efficient approximation algorithms. This observation was first exploited by Kannan, Sweedyk, and Mahaney, who gave a *quasi polynomial-time approximation scheme* (QPRAS) for NFAs [42], which was later extended by the aforementioned authors, along with Gore and Jerrum [31], to the case of tree automata.[2] Specifically, the algorithm of [31] runs in time $\epsilon^{-2}(nm)^{O(\log(n))}$, where $m = |\mathcal{T}|$ is the size of the description of $\mathcal{T}$, and ϵ is the error parameter. Improving the complexity of this algorithm to polynomial time has been an open problem.

The algorithms of [31] and [42] are based on a recursive form of Karp-Luby sampling [43], which is a type of rejection sampling. This approach has the drawback that the probability a sample is chosen is exponentially small in the depth of the recursion. Recently, using a different sampling scheme, it was shown that an FPRAS and an FPAUS exist for NFAs [8]. However, the techniques in [8] break down fundamentally (discussed in the following) when applied to tree automata. The main technical contribution of this work is to address the failing points of [8] for tree automata, and design an FPRAS for this case.

Theorem 1.3. *Given a tree automaton $\mathcal{T}$ and $n \geq 1$, there is an algorithm which runs in time poly$(|\mathcal{T}|, n, \epsilon^{-1}, \log(\delta^{-1}))$ and with probability $1-\delta$, outputs an estimate $\widetilde{N}$ with:*

$$(1-\epsilon)|\mathcal{L}_n(\mathcal{T})| \leq \widetilde{N} \leq (1+\epsilon)|\mathcal{L}_n(\mathcal{T})|$$

Conditioned on the success of this event, there is a sampling algorithm where each call runs in time poly$(|\mathcal{T}|, n, \log(\delta^{-1}))$, and either outputs a uniformly random tree $t \in \mathcal{L}_n(\mathcal{T})$, or $\perp$. Moreover, it outputs $\perp$ with probability at most $1/2$.

Note that conditioned on the success of the above FPRAS (run once), every subsequent call to the sampler generates a *truly* uniform sample (or $\perp$). Observe that this notion of sampling is stronger than the standard notion of FPAUS (see Section 2). We note that the existence of an FPAUS is in fact a corollary of the existence of an FPRAS for the above [40].

Succinct NFAs. A key step in the proof of Theorem 1.3 is a reduction to counting and sampling from a *succinct* NFA $\mathcal{N}$, which is an NFA with succinctly encoded alphabet and transitions. Formally, a succinct NFA $\mathcal{N}$ is a 5-tuple $(S, \Sigma, \Delta, s_{\text{init}}, s_{\text{final}})$, where S is a set of states, Σ is an alphabet, $s_{\text{init}}, s_{\text{final}} \in S$ are the initial and final states, and $\Delta \subseteq S \times 2^{\Sigma} \times S$ is the transition relation, where each transition is labeled by a set $A \subseteq \Sigma$. We assume that Σ is succinctly encoded via some representation (e.g. a DNF formula), and likewise for each set A such that $e = (s, A, s')$ is a transition in Δ. Therefore, the size of the alphabet Σ and the size of each such set A can be exponentially large in the representation of $\mathcal{N}$. A word $w = w_1 w_2 \dots w_n \in \Sigma^*$ is *accepted* by $\mathcal{N}$ if there is a sequence $s_{\text{init}} = s_0, s_1, \dots, s_n = s_{\text{final}}$ of states such that there exists a transition $(s_{i-1}, A, s_i) \in \Delta$ with $w_i \in A$ for each $i = 1, 2, \dots, n$. Note that the special case where each transition $(s, A, s') \in \Delta$ satisfies $|A| = 1$ is precisely the standard definition of an NFA. To solve the aforementioned problems for succinct NFA, we must assume that the encodings of the label sets satisfy some basic conditions. Specifically, we require that for each transition (s, A, s'), we are given an oracle which can **(1)** test membership in A, **(2)** produce an estimate of the size of $|A|$, and **(3)** generate almost-uniform samples from A.

[2]The QPRAS of [31] holds more generally for *context free grammars* (CFG).

Theorem 1.4. *Let $\mathcal{N} = (S, \Sigma, \Delta, s_{init}, s_{final})$ be a succinct NFA and $n \geq 1$. Suppose that the sets A in each transition $(s, A, s') \in \Delta$ satisfy the properties **(1)**, **(2)** and **(3)** described above. Then there is an FPRAS and an FPAUS for $\mathcal{L}_n(\mathcal{N})$.*

While standard (non-succinct) NFAs are known to admit an FPRAS by the results of [8], Theorem 1.4 is a strong generalization of the main result of [8], and requires many non-trivial additional insights and techniques.

1.2 Additional Applications of the FPRAS

We demonstrate that the FPRAS of Theorem 1.3 results in the first polynomial-time randomized approximation algorithms for many previously open problems in the fields of constraint satisfaction problems, verification of correctness of programs with nested calls to subroutines, and knowledge compilation. We give a brief overview of these results in what follows. Further details are deferred to the full version.

Constraint satisfaction problems. Constraint satisfaction problems (CSPs) offer a general and natural setting to represent a large number of problems where solutions must satisfy some constraints, and which can be found in different areas [12, 22, 39, 51, 52, 61]. The most basic task associated to a CSP is the problem of verifying whether it has a solution, which corresponds to an assignment of values to the variables of the CSP that satisfies all the constraints of the problem. Tightly related with this task is the problem of counting the number of solutions to a CSP. In this work, we consider this counting problem in the usual setting where a projection operator for CSPs is allowed, so that it is possible to indicate the output variables of the problem. We denote this setting as ECSP.

As counting the number of solutions of an ECSP is #P-complete and cannot admit an FPRAS (unless NP = RP), we focus on two well known notions of acyclicity that ensure that solutions can be found in polynomial time [34, 35]. More precisely, we define #AECSP as the problem of counting, given an acyclic ECSP $\mathcal{E}$, the number of solutions to $\mathcal{E}$. Moreover, given a fixed $k \geq 0$, we define #k-HW-ECSP as the problem of counting, given an ECSP $\mathcal{E}$ whose hypertree width is at most k, the number of solutions for $\mathcal{E}$. Although both problems are known to be #P-complete [47], we obtain as a consequence of Theorem 1.3 that both #AECSP and #k-HW-ECSP admit FPRAS.

Software verification. Nested words have been proposed as a model for the formal verification of correctness of structured programs that can contain nested calls to subroutines [2–4]. In particular, the execution of a program is viewed as a linear sequence of states, where a matching relation is used to specify the correspondence between each point during the execution at which a procedure is called with the point when we return from that procedure call. This idea gives rise to the notion of nested word, which is defined as a regular word accompanied by a matching relation. Moreover, properties of programs to be formally verified are specified by using nested word automata (NWA). The emptiness problem for nested word automata ask whether, given a NWA $\mathcal{N}$, there exists a nested word accepted by $\mathcal{N}$. This is a fundamental problem when looking for faulty executions of a program with nested calls to subroutines; if $\mathcal{N}$ is used to encode the complement of a property we expect to be satisfied by a program, then a nested word accepted by $\mathcal{N}$ encodes a bug of this program. In this sense, the following is also a very relevant problem for understanding how faulty a program is. Define #NWA as the problem of counting, given a nested word automaton $\mathcal{N}$ and a string 0^n, the number of nested words of length n accepted by $\mathcal{N}$. As expected, #NWA is a #P-complete problem. Interestingly, from Theorem 1.3 and the results in [4] showing how nested word automata can be represented by using tree automata over binary trees, it is possible to prove that #NWA admits an FPRAS.

Knowledge compilation. Model counting is the problem of counting the number of satisfying assignments given a propositional formula. Although this problem is #P-complete [59], there have been several approaches to tackle it [30]. One of them comes from the field of *knowledge compilation*, a subarea in artificial intelligence [25]. Roughly speaking, this approach consists in dividing the reasoning process in two phases. The first phase is to compile the formula into a target language (e.g. Horn formulae, BDDs, circuits) that has good algorithmic properties. The second phase is to use the new representation to solve the problem efficiently. The main goal then is to find a target language that is expressive enough to encode a rich set of propositional formulae and, at the same time, that allows for efficient algorithms to solve the counting problem.

A target language for knowledge compilation that has attracted a lot of attention is the class of DNNF circuits [24]. DNNF has good algorithmic properties in terms of satisfiability and logical operations. Furthermore, DNNF can be seen as a generalization of DNF formulae, ordered binary decision diagrams (OBDDs) [13] and free binary decision diagrams (FBDDs) [25], in the sense that every expression in these formalisms can be transformed into a DNNF circuit in polynomial time. Moreover, DNNF is exponentially more succinct than DNF, OBDD and FBDD [25], and hence it is a more appealing language for knowledge compilation. Regarding model counting, DNNF circuits can easily encode #P-complete problems (e.g. #DNF) and, therefore, researchers have look into subclasses of DNNF where counting can be done more efficiently. One such a class that has recently received a lot of attention is the class of structured DNNF [48], which has been used for efficient enumeration [6, 7], and has proved to be appropriate to compile propositional CNF formulae with bounded width (e.g. CV-width) [46]. Unfortunately, the problem of computing the number of propositional variable assignments that satisfy a structured DNNF circuit is a #P-complete problem, as these circuits include the class of DNF formulae. However, and in line with the idea that structured DNNF circuits allow for more efficient counting algorithms, we can prove that the counting problem of structured DNNF circuits admits a fully-polynomial time randomized approximation scheme as a consequence of Theorem 1.3.

1.3 Outline of the Paper

In Section 2, we start by defining the notions of FPRAS and FPAUS. Section 3 formalizes the connection between conjunctive queries and tree automata. Section 4 gives a general overview of the partition based approach to get an FPRAS and Section 5 shows the specific partition scheme for tree automata. In Section 6 we give the last ingredient of the algorithm, namely, an FPRAS for succinct NFAs. In Section 7, we discuss some open problems and future work. The complete proofs of all the results in the paper can be found at [9].

2 PRELIMINARIES: FPRAS AND FPAUS

We start by giving the formal definition of fully polynomial time randomized approximation scheme (FPRAS) and fully polynomial time almost uniform sampler (FPAUS) used throughout the paper.

Given an input alphabet Σ, a *randomized approximation scheme* (RAS) for a function $f : \Sigma^* \to [0, \infty)$ is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0, 1) \to [0, \infty)$ such that for every $w \in \Sigma^*$ and $\epsilon \in (0, 1)$:

$$\mathbf{Pr}[|\mathcal{A}(w, \epsilon) - f(w)| \leq \epsilon \cdot f(w)] \quad \geq \quad \frac{3}{4}.$$

Moreover, a randomized algorithm $\mathcal{A} : \Sigma^* \times (0, 1) \to [0, \infty)$ is a *fully polynomial-time randomized approximation scheme* (FPRAS) [40] for f, if it is a randomized approximation scheme for f and, for every $w \in \Sigma^*$ and $\epsilon \in (0, 1)$, $\mathcal{A}(w, \epsilon)$ runs in polynomial time in $|w|$ and ϵ^{-1}. Thus, if $\mathcal{A}$ is an FPRAS for f, then $\mathcal{A}(w, \epsilon)$ approximates the value $f(w)$ with a relative error of $(1 \pm \epsilon)$, and it can be computed in polynomial time in the size of w and the value ϵ^{-1}.

In addition to polynomial time approximation algorithms, we also consider polynomial time (almost) uniform samplers. Given an alphabet Σ and a finite universe Ω, let $g : \Sigma^* \to 2^{\Omega}$. We say that g admits a *fully polynomial-time almost uniform sampler* (FPAUS) [40] if there is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0, 1) \to \Omega \cup \{\bot\}$ such that for every $w \in \Sigma^*$ with $g(w) \neq \emptyset$, and $\delta \in (0, 1)$, $\mathcal{A}(w, \delta)$ outputs a value $x^* \in g(w) \cup \{\bot\}$ with

$$\mathbf{Pr}[x^* = x] = (1 \pm \delta) \frac{1}{|g(w)|} \qquad \text{for all } x \in g(w)$$

and, moreover, $\mathcal{A}(w, \delta)$ runs in polynomial time over $|w|$ and $\log \frac{1}{\delta}$. If $g(w) = \emptyset$, an FPAUS must output the symbol $\bot$ with probability 1. The symbol $\bot$ can be thought of as a "failure" indicator, where the algorithm produces no output. Notice that whenever $g(w)$ admits a deterministic polynomial time membership testing algorithm (i.e. to test if $x \in g(w)$), it is easy to ensure that a sampler only outputs either an element $x \in g(w)$ or $\bot$. Also notice that the conditions imply that if $g(w) \neq \emptyset$, we have $\mathbf{Pr}[x^* = \bot] \leq \delta$. Given a set $S = g(w)$, when the function g and the input w are clear from context, we will say that the set S admits an FPAUS to denote the fact that g admits an FPAUS.

For an example of an FPAUS, w could be the encoding of a non-deterministic finite automata $\mathcal{N}$ and an integer $n \geq 1$ given in unary, and $g(w)$ could be the set of strings of length n accepted by $\mathcal{N}$. A polynomial-time almost uniform sampler must then generate a string in $\mathcal{L}_n(\mathcal{N})$ from a distribution which is pointwise a $(1 \pm \delta)$ approximation of the uniform distribution over $\mathcal{L}_n(\mathcal{N})$, output $\bot$ with probability at most δ, and run in time $\text{poly}(|\mathcal{N}|, n, \log \frac{1}{\delta})$. Notice that an FPAUS must run in time $\text{poly}(\log \frac{1}{\delta})$, whereas an FPRAS may run in time $\text{poly}(\frac{1}{\epsilon})$.

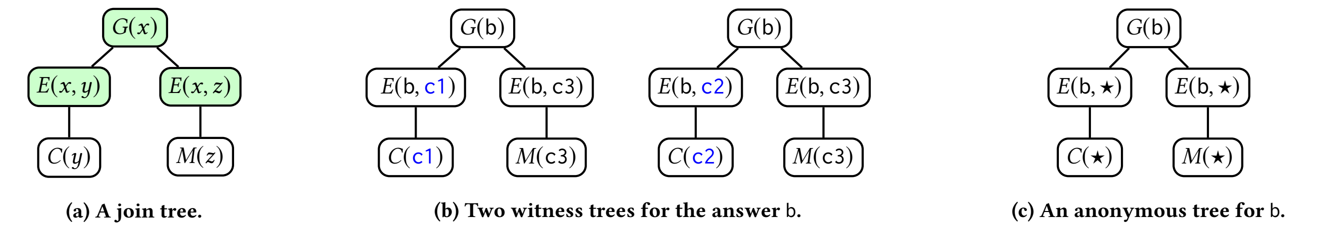


Figure 1: Join, witness and anonymous trees for a CQ.

3 FROM CONJUNCTIVE QUERIES TO TREE AUTOMATA

In this section, we provide the formal link between Conjunctive Queries (CQ) and tree automata, and formalize our results for the former. In particular, we show that it is possible to reduce #ACQ to #TA, where #ACQ is the problem of counting the number of solutions to an acyclic CQ. Hence, the existence of an FPRAS for #ACQ is inferred from the existence of an FPRAS for #TA. In fact, we will prove the more general statement that one can reduce the problem of counting solutions to CQ's with *bounded hypertree width*, and unions of such queries, to the problem of #TA.

We start by formally introducing conjunctive queries. We first fix two disjoint (countably) infinite sets **C** and **V** of constants and variables, respectively. Then a conjunctive query (CQ) is an expression of the form:

$$Q(\bar{x}) \quad \leftarrow \quad R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n), \qquad (1)$$

where for every $i \in \{1, \ldots, n\}$, R_i is a k_i-ary relation symbol ($k_i \geq 1$) and $\bar{u}_i$ is a k_i-ary tuple of variables and constants (that is, elements from **V** and **C**), and $\bar{x} = (x_1, \ldots, x_m)$ is a tuple of variables such that each variable x_i in $\bar{x}$ occurs in some $\bar{u}_i$. The symbol Q is used as the name of the query, and $\text{var}(R_i)$ is used to denote the set of variables in relation symbol R_i. Moreover, $\text{var}(Q)$ denotes the set of all variables appearing in the query (i.e., left- and right-hand sides).

Intuitively, the right-hand side $R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$ of Q is used to specify a pattern over a database, while the tuple $\bar{x}$ is used to store the answer to the query when such a pattern is found. More precisely, a database D is a set of facts of the form $T(\bar{a})$ where $\bar{a}$ is a tuple of constants (elements from C), which indicates that $\bar{a}$ is in the table T in D. Then a homomorphism from Q to D is a function h from the set of variables occurring in Q to the constants in D such that for every $i \in \{1, \ldots, n\}$, it holds that $R_i(h(\bar{u}_i))$ is a fact in D, where $h(\bar{u}_i)$ is obtained by applying h to each component of $\bar{u}_i$ leaving the constants unchanged. Moreover, given such a homomorphism h, the tuple of constants $h(\bar{x})$ is said to be an answer to Q over the database D, and $Q(D)$ is defined as the set of answers of Q over D.

3.1 High-level Overview of the Reduction to #TA

We now give a high-level overview of our reduction to #TA from a simple class of acyclic conjunctive queries. This overview will be sufficient to provide intuition for why tree automata are the correct tool for representing the number of solutions to such conjunctive queries. Consider a CQ:

$$Q_1(x) \leftarrow G(x), E(x,y), E(x,z), C(y), M(z)$$

This query is said to be acyclic, because it can be encoded by a *join tree*, that is, by a tree t where each node is labeled by the relations occurring in the query, and which satisfies the following connectedness property: each variable in the query induces a connected subtree of t [62]. In particular, a join tree for $Q_1(x)$ is depicted in Figure 1a, where the connected subtree induced by variable x is marked in green. An acyclic conjunctive query Q can be efficiently evaluated by using a join tree t encoding it [62]; in fact, a tree *witnessing* the fact that $\bar{a} \in Q(D)$ can be constructed in polynomial time. For example, if $D_1 = \{G(\mathsf{a}), G(\mathsf{b}), E(\mathsf{a}, \mathsf{c1}), E(\mathsf{b}, \mathsf{c1}), E(\mathsf{b}, \mathsf{c2}), E(\mathsf{b}, \mathsf{c3}), C(\mathsf{c1}), C(\mathsf{c2}), M(\mathsf{c3})\}$, then b is an answer to Q_1 over D_1. In fact, two witness trees for this answer are shown in Figure 1b. Notice that the assignments to variable y that distinguish these two trees are marked in blue.

In this work, we consider the following problem:

Problem:	#ACQ
Input:	An acyclic CQ Q and a database D
Output:	$\lvert Q(D) \rvert$

One might think that #ACQ can also be solved in polynomial time given that the number of witness trees can be counted in polynomial time. However, there is no one-to-one correspondence between the answers to an acyclic CQ and their witness trees; as shown in Figure 1b, two trees may witness the same answer. In fact, #ACQ is #P-complete [47].

However, we first observe that in a witness tree t, if only output variables are given actual values and non-output variables are assigned an anonymous symbol $\star$, then there *will* be a one-to-one correspondence between answers to a query and witnesses. Let's us denote such structures as *anonymous* trees, an example of which is given in Figure 1c. But how can we specify when an anonymous tree is valid? For example, if t' is the anonymous tree obtained by replacing b by a in Figure 1c, then t' is not a valid anonymous tree, because a is not an answer to Q_1 over D_1. We demonstrate that tree automata provide the right level of abstraction to specify the validity of such anonymous trees, so that #ACQ can be reduced to a counting problem over tree automata. In fact, in the extended version of this paper at [9], we present this construction for the more general notion of bounded hypertree width, which is introduced in the following section.

3.2 A More General Notion of Acyclicity

In Section 3.1, we consider a CQ as *acyclic* if it can be encoded by a join tree. However, our results apply to a more general notion of acyclicity, known as the *hypertree width* of a CQ. We now formalize this more general notion of acyclicity. Let Q be a CQ of the form $Q(\bar{x}) \leftarrow R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$. A hypertree for Q is a triple $\langle T, \chi, \xi \rangle$ such that $T = (N, E)$ is a rooted tree, and χ and ξ are node-labeling functions such that for every $p \in N$, it holds that $\chi(p) \subseteq \text{var}(Q)$ and $\xi(p) \subseteq \{R_1, \ldots, R_n\}$. Moreover, $\langle T, \chi, \xi \rangle$ is said to be a hypertree decomposition for Q [35] if the following conditions hold:

- for each atom $i \in \{1, \ldots, n\}$, there exists $p \in N$ s.t. $\text{var}(R_i) \subseteq \chi(p)$;
- for each variable $x \in \text{var}(Q)$, the set $\{p \in N \mid x \in \chi(p)\}$ induces a (connected) subtree of T;
- for each $p \in N$, it holds that

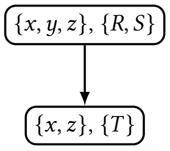
$$\chi(p) \quad \subseteq \quad \bigcup_{R \in \xi(p)} \text{var}(R)$$

- for each $p \in N$, it holds that

$$\left(\bigcup_{R \in \xi(p)} \text{var}(R) \right) \cap \left(\bigcup_{p' \,:\, p' \text{ is a descendant of } p \text{ in } T} \chi(p') \right) \quad \subseteq \quad \chi(p)$$

The width of the hypertree decomposition $\langle T, \chi, \xi \rangle$ is defined as the maximum value of $|\xi(p)|$ over all vertices $p \in N$. Finally, the hypertree width $\text{hw}(Q)$ of CQ Q is defined as the minimum width over all its hypertree decompositions [35].

Example 3.1. Consider the CQ $Q(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$. It is easy to see that Q is a non-acyclic query (it cannot be represented by a join tree as defined in Section 3.1), but we can still study its degree of acyclicity using the idea of hypertree width. In particular, the following is a hypertree decomposition for Q, where the values of $\chi(p)$ and $\xi(p)$ are shown on the left- and right-hand sides of the rectangle for node p:



Notice that the width of this hypertree decomposition is 2, as $|\xi(p)| = 2$ for the root. And in fact, no hypertree decomposition of width 1 can be constructed for Q, so that $\text{hw}(Q) = 2$ (otherwise, Q would be acyclic). In some way, we are forced to bundle two of the atoms (R and S) together and in the process increase the width, in order to create a *join tree*-like structure. ■

It was shown in [35] that a CQ Q is acyclic if and only if $\text{hw}(Q) = 1$. Thus, the notion of hypertree width generalizes the notion of acyclicity given before. We are interested in classes of queries with bounded hypertree width, for which it has been shown that the evaluation problem can be solved efficiently [35]. More precisely, for every $k \geq 1$ define the following counting problem.

Problem:	#k-HW
Input:	A CQ Q such that $\text{hw}(Q) \leq k$ and a database D
Output:	$\lvert Q(D) \rvert$

It is important to notice that #ACQ = #1-HW. However, we will keep both languages for historical reasons, as acyclic conjunctive queries were defined two decades earlier, and are widely used in databases. Both #ACQ and #k-HW, for a fixed $k \geq 1$, are known to be #P-complete [47]. On the positive side, based on the relationship with tree automata that we show in the extended version of this paper (see [9]), we can conclude that these problems admit an FPRAS and an FPAUS, as formalized in Section 2.

Theorem 3.2. *#ACQ admits an FPRAS and an FPAUS, and for every constant $k \geq 1$, #k-HW admits an FPRAS and an FPAUS.*

3.3 A Characterization of Classes of Conjunctive Queries Admitting FPRAS

In this section, we describe how Theorem 3.2 and the results in [37] can be combined to obtain a characterization of the classes of conjunctive queries which admit an FPRAS. We consider the slightly different notion of conjunctive query used in [37], which explicitly includes equality atoms and does not contain constants. More precisely, in this section a conjunctive query (CQ) is an expression of the form:

$$Q(\bar{x}) \;\leftarrow\; \alpha_1(\bar{y}_1), \ldots, \alpha_n(\bar{y}_n), \qquad (\ddagger)$$

where for every $i \in \{1, \ldots, n\}$, $\alpha_i(\bar{y}_i)$ is either $R_i(\bar{y}_i)$ with R_i a k_i-ary relation symbol ($k_i \geq 1$) and $\bar{y}_i$ a k_i-ary tuple of variables from $\mathbf{V}$, or $\alpha_i(\bar{y}_i)$ is $y_{i,1} = y_{i,2}$ with $y_{i,1}$, $y_{i,2}$ variables from $\mathbf{V}$. Moreover, $\bar{x} = (x_1, \ldots, x_m)$ is a tuple of variables such that each variable x_i in $\bar{x}$ occurs in some $\bar{y}_i$.

As before, the symbol Q is used as the name of the query, and $\text{var}(Q)$ denotes the set of all variables appearing in Q.

Given a CQ $Q(\bar{x})$ of the form (‡), define a graph G_Q representing Q as follows. The set of vertices of G_Q is $\text{var}(Q)$, and there exists an edge between two distinct variables x and y if, and only if, there exists $i \in \{1, \ldots, n\}$ such that both x and y occur in $\alpha_i(\bar{y}_i)$. We consider here the standard notion of tree-width of a graph G and of a conjunctive query Q [28, 37], which are denoted by $\text{tw}(G)$ and $\text{tw}(Q)$, respectively. Notice that although $\text{tw}(Q)$ is not defined by using G_Q, it holds that $\text{tw}(Q) = \text{tw}(G_Q)$ [28]. Then a class $\mathcal{G}$ of graphs is said to have bounded treewidth if there exists a constant k such that $\text{tw}(G) \leq k$ for every $G \in \mathcal{G}$. Moreover, define $\text{CQ}(\mathcal{G})$ as the class of all conjunctive queries Q of the form (‡) whose representing graph G_Q is in $\mathcal{G}$.

Let $\mathcal{G}$ be a class of graphs, and assume that there exists a constant k such that $\text{tw}(G) \leq k$ for every $G \in \mathcal{G}$. Then for every $Q \in \text{CQ}(\mathcal{G})$, we have that $\text{hw}(Q) \leq \text{tw}(Q) \leq k$. Moreover, given $Q \in \text{CQ}(\mathcal{G})$, if Q' is a query obtained from Q by replacing each equality atom $x = y$ by $EQ(x, y)$, where EQ is a fresh predicate, then we have that $Q(D) = Q'(D')$ with $D' = D \cup \{EQ(a, a) \mid a \text{ is an element of } D\}$, and we also have that $\text{tw}(Q') = \text{tw}(Q)$ and $\text{hw}(Q') = \text{hw}(Q)$. Therefore, by considering Theorem 3.2, we conclude that the problems of computing $|Q(D)|$ and of sampling from $Q(D)$, given as input $Q \in \text{CQ}(\mathcal{G})$ and a database D, admit an FPRAS and an FPAUS.

Given a class $\mathcal{G}$ of graphs, the query decision problem for $\mathrm{CQ}(\mathcal{G})$ is the problem of verifying, given a CQ $Q \in \mathrm{CQ}(\mathcal{G})$ and a database D, whether $Q(D) \neq \emptyset$. By the results of [37], assuming that $\mathrm{W}[1] \neq$ FPT, for every class $\mathcal{G}$ of (undirected) graphs, the query decision problem for $\mathrm{CQ}(\mathcal{G})$ is tractable if, and only if, $\mathcal{G}$ has bounded treewidth. Since the existence of an FPRAS or an FPAUS for the set $Q(D)$ of answers of a conjunctive query results in a BPP algorithm for the query decision problem, it follows that if BPP = P, it is not possible to obtain an FPRAS or an FPAUS for any class of CQs of the form $\mathrm{CQ}(\mathcal{G})$ for a class of graphs $\mathcal{G}$ with unbounded treewidth. As a corollary of this and the discussion in the previous paragraph, we obtain the following characterization of classes of conjunctive queries admitting FPRAS and FPAUS:

Corollary 3.3. *Let $\mathcal{G}$ be a class of (undirected) graphs. Then assuming $W[1] \neq FPT$ and BPP = P, the following are equivalent:*

(1) The problems of computing $|Q(D)|$ and of sampling from $Q(D)$, given as input $Q \in \mathrm{CQ}(\mathcal{G})$ and a database D, admit an FPRAS and an FPAUS, respectively.
(2) $\mathcal{G}$ has bounded treewidth.

It should be mentioned that a refinement of the result of [37] is given in [36], which can be applied over any recursively enumerable class of conjunctive queries of fixed arity. We do not know whether the results of this paper can be extended to this case, which in particular means proving there exists an FPRAS for each class of CQs whose cores [38] have bounded treewidth. We believe this to be an interesting problem for future work.

3.4 Union of Conjunctive Queries

An important and well-studied extension of the class of conjunctive queries is obtained by adding the union operator. A union of conjunctive queries (UCQ) is an expression of the form:

$$Q(\bar{x}) \quad \leftarrow \quad Q_1(\bar{x}) \vee \cdots \vee Q_m(\bar{x}), \qquad (2)$$

where $Q_i(\bar{x})$ is a conjunctive query of the form (1) for each $i \in \{1, \ldots, m\}$, and the same tuple $\bar{x}$ of output variables is used in the CQs $Q_1(\bar{x})$, …, $Q_m(\bar{x})$. As for the case of CQs, the symbol Q is used as the name of the query. A tuple $\bar{a}$ is said to be an answer of UCQ Q in (2) over a database D if and only if $\bar{a}$ is an answer to Q_i over D for some $i \in \{1, \ldots, m\}$. Thus, we have that:

$$Q(D) \quad = \quad \bigcup_{i=1}^{m} Q_i(D)$$

As expected, the problem of verifying, given a UCQ Q, a database D and a tuple of constants $\bar{a}$, whether $\bar{a}$ is an answer to Q over D is an NP-complete problem [16]. Also as expected, the evaluation problem for union of acyclic conjunctive queries can be solved in polynomial time, given that the evaluation problem for acyclic CQs can be solved in polynomial time. Concerning to our investigation, we are interested in the following problem associated to the evaluation problem for union of acyclic conjunctive queries:

Problem:	#UACQ
Input:	A union of acyclic CQ Q and a database D
Output:	$\lvert Q(D)\rvert$

As expected from the result for conjunctive queries, #UACQ is #P-complete [47]. However, #UACQ remains #P-hard even if we focus on the case of UCQs without existentially quantified variables, that is, UCQs of the form (2) where $\bar{x}$ consists of all the variables occurring in CQ $Q_i(\bar{x})$ for each $i \in \{1, \ldots, m\}$. Notice that this is in sharp contrast with the case of CQs, where #ACQ can be solved in polynomial time if we focus on the case of CQs without existentially quantified variables [47]. However, by using Theorem 3.2, we are able to provide a positive result about the possibility of efficiently approximating #UACQ.

Proposition 3.4. #UACQ *admits an FPRAS and an FPAUS.*

As a final fundamental problem, we consider the problem of counting the number of solutions of a union of conjunctive queries of bounded hypertree width.

Problem:	#k-UHW
Input:	A database D and a union of CQ $Q(\bar{x}) \leftarrow Q_1(\bar{x}) \vee \cdots \vee Q_m(\bar{x})$ such that $\mathrm{hw}(Q_i) \leq k$ for all $i \in \{1, \ldots, m\}$
Output:	$\lvert Q(D)\rvert$

By using the same ideas as in the proof of Proposition 3.4, we obtain from Theorem 3.2 that:

Proposition 3.5. *For every $k \geq 1$, it holds that* #k-UHW *admits an FPRAS and an FPAUS.*

3.5 Some Related Work on Conjunctive Queries

Several works have looked into the counting problem for CQs (and the related problems we listed above, like CSPs). In order to clarify the discussion, we will give a rough characterization of the research in this area. This will better illustrate how our results relate to previous work. So as a first idea, when counting solutions to CQs, an important source of difficulty is the presence of existentially quantified variables. Consider the query we used in Section 3.1:

$$Q_1(x) \quad \leftarrow \quad G(x), E(x,y), E(x,z), C(y), M(z).$$

Notice that there are three variables x, y and z in the right-hand side, while only x is present in the left-hand side. Thus, x is an output variable, while y and z are existentially quantified variables. An alternative notation for CQs makes the quantification even more explicit:

$$Q_1(x) \quad \leftarrow \quad \exists y \exists z \, (G(x) \wedge E(x,y) \wedge E(x,z) \wedge C(y) \wedge M(z)).$$

As we mention later in Section 3, when variables are existentially quantified, there is no one-to-one correspondence between the answers to a CQ and their witness trees. This introduces a level of ambiguity (i.e. potentially several witness trees for each answer) into the counting problem, which makes it more difficult, even though it does not make the evaluation problem any harder. In fact, it is proved in Theorem 4 in [47] that the counting problem is #P-complete for acyclic CQs over graphs (i.e. bounded arity), even if queries are allowed a single existentially quantified variable (and an arbitrary number of output variables). In contrast, it is known (e.g. [23]) that for each class of CQs with bounded treewidth and without existentially quantified variables, the counting problem can be solved exactly in polynomial time.

It was open what happens with the counting problem when CQs are considered with all their features, that is, when output and existentially quantified variables are combined. In particular, it was open whether the counting problem admits an approximation in that case. Our paper aims to study precisely that case, in contrast with previous work that does not consider such output variables combined with existentially quantified variables [14, 23].

As a second idea, approaches to make the counting or evaluation problem for CQs tractable usually revolve around imposing some structural constraint on the query, in order to restrict its degree of cyclicity. Most well-known is the result in [62], which proves that the evaluation problem is tractable for acyclic queries. In generalizations of this result (e.g. [37]), the acyclicity is usually measured as the width of some query decomposition. Specific to the counting problem, this type of notion is used in [26] to characterize tractable cases. Notice, however, that they rely not only on the width of different query decompositions, but also on a measure of how free variables are spread in the query, which they call *quantified star size*. In contrast, we rely only on the structural width of the hypertree decomposition.

4 TREE AUTOMATA AND A PARTITION BASED APPROACH FOR FINDING AN FPRAS

Given the results of Section 3, we will now focus on the problem of designing an FPRAS for #TA, which has as input a tree automata $\mathcal{T}$ and an integer $n \geq 1$ (given in unary), and asks to output $|\mathcal{L}_n(\mathcal{T})|$. In this section, we give an overview of the main components of our algorithm, their relation to prior techniques, and the technical challenges involved in designing such an FPRAS.

4.1 Binary Tree Automata

To capture the essence of the problem, in the following discussion we consider a simplified version of tree automata. Specifically, we restrict the discussion to *unlabeled* binary ordered trees, which are sufficient to present the main ideas of the algorithm. A binary ordered tree t (or just tree) is a rooted binary tree where the children of each node are ordered; namely, one can distinguish between the left and right child of each non-leaf node. For a non-leaf node u of t, we write $u1$ and $u2$ to denote the left and right children of u, respectively, and we denote the root of any tree t by λ, which representes the empty string. We will write $u \in t$ to denote that u is a node of t, and $|t|$ to denote the number of nodes of t. For example, Figure 2 depicts a binary ordered tree t_1 with $|t_1| = 9$, and another tree t_2 with $|t_2| = 13$, where the children are ordered from left to right.

A tree automaton $\mathcal{T}$ over a binary ordered tree is defined as a tuple $(S, \Delta, s_{\text{init}})$ where S is a finite set of states, $\Delta \subseteq (S \times S \times S) \cup S$ is the transition relation, and $s_{\text{init}} \in S$ is the initial state. A *run* ρ of $\mathcal{T}$ over a tree t is a function $\rho : t \to S$ mapping nodes to states that respects the transition relation. Namely, for every node u of t we have $\rho(u) \in \Delta$ whenever u is a leaf, and $(\rho(u), \rho(u1), \rho(u2)) \in \Delta$, otherwise. We say that $\mathcal{T}$ accepts t if there exists a run ρ of $\mathcal{T}$ over t such that $\rho(\lambda) = s_{\text{init}}$, and such a run ρ is called an accepting run of $\mathcal{T}$ over t. The set of all trees accepted by $\mathcal{T}$ is denoted by $\mathcal{L}(\mathcal{T})$, and the n-slice of $\mathcal{L}(\mathcal{T})$, denoted by $\mathcal{L}_n(\mathcal{T})$, is the set of trees $t \in \mathcal{L}(\mathcal{T})$ with size n (that is, $|t| = n$). For the sake of presentation, in the following we write $s \to qr$ to represent the transition $(s, q, r) \in \Delta$ and $s \to \cdot$ to represent $s \in \Delta$. Note that transitions of the form $s \to \cdot$ correspond to leaves that have no children, and can be thought as "final states" of a run.

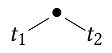
Figure 2 gives an example of a tree automaton $\mathcal{T}$ with states $\{s, r, q\}$. The right-hand side of Figure 2 shows an example of an accepting run of $\mathcal{T}$ over t_2. One can easily check from the transitions of $\mathcal{T}$ in this example that a tree t is in $\mathcal{L}(\mathcal{T})$ if, and only if, there exists a node $v \in t$ such that both children of v are internal (non-leaf) nodes. For example, t_2 satisfies this property and $t_2 \in \mathcal{L}(\mathcal{T})$. On the other hand, all nodes $v \in t_1$ have at least one child that is a leaf, and thus there is no accepting run of $\mathcal{T}$ over t_1, so $t_1 \notin \mathcal{L}(\mathcal{T})$.

4.2 Unrolling the Automaton

Fix $n \geq 1$ and a tree automaton $\mathcal{T} = (S, \Delta, s_{init})$ as defined above. Our first step will be to *unroll* the automaton, so that each state is restricted to only producing trees of a fixed size. Specifically, we construct an automaton $\overline{\mathcal{T}} = (\overline{S}, \overline{\Delta}, s^n_{\text{init}})$, where each state $s \in S$ is duplicated n times into $s^1, s^2, \ldots, s^n \in \overline{S}$, and where s^i is only allowed to derive trees of size i. To enforce this, each transition $s \to rq$ in Δ is replaced with $s^i \to r^j q^k \in \overline{\Delta}$ for all $j, k > 0$ such that $i = j + k + 1$, and each transition $s \to \cdot$ in Δ is replaced with $s^1 \to \cdot \in \overline{\Delta}$. Now for every $s \in S$, let $T(s^i)$ be set of trees that can be derived beginning from the state s^i (all of which have size i). When $i > 1$, we can define $T(s^i)$ via the relation

$$T(s^i) = \bigcup_{(s^i \to r^j q^k) \in \overline{\Delta}} \Big(T(r^j) \otimes T(q^k) \Big) \tag{3}$$

where $T(r^j) \otimes T(q^k)$ is a shorthand to denote the set of all trees that can be created by taking every $t_1 \in T(r^j)$ and $t_2 \in T(q^k)$ and forming the tree:



This fact allows us to define each set $T(s^i)$ recursively as a union of "products" of other such sets. Our goal is then to estimate $|T(s^n_{\text{init}})|$ and sample from $T(s^n_{\text{init}})$.

It should be mentioned that for so-called "bottom-up deterministic" automata $\mathcal{T}$ [20], the sets $T(r^j) \otimes T(q^k)$ in the union in Equation (3) are disjoint, so

$$|T(s^i)| \quad = \sum_{(s^i \to r^j q^k) \in \overline{\Delta}} |T(r^j)| \cdot |T(q^k)|$$

and one can then compute the values $|T(s^i)|$ exactly via dynamic programming. Thus, the core challenge is the *ambiguity* of the problem: namely, the fact that trees $t \in T(s^n_{\text{init}})$ may admit exponentially many runs in the automaton. For example, the tree automaton $\mathcal{T}$ from Figure 2 can accept t_2 by two different runs. In what follows, we will focus on the problem of uniform sampling from such a set $T(s^i)$, since given a uniform sampler the problem of size estimation is routine.

4.3 A QPRAS via Karp-Luby Sampling

To handle the problem of sampling with ambiguous derivations Gore et al. [31] used a technique known as *Karp-Luby* sampling. This technique is a form of rejection sampling, where given sets

$$\mathcal{T} := \begin{array}{ccc} s & \rightarrow & sq \\ s & \rightarrow & qs \\ s & \rightarrow & rr \\ r & \rightarrow & qq \\ q & \rightarrow & qq \\ q & \rightarrow & \cdot \\ s_{\text{init}} & = & s \end{array}$$

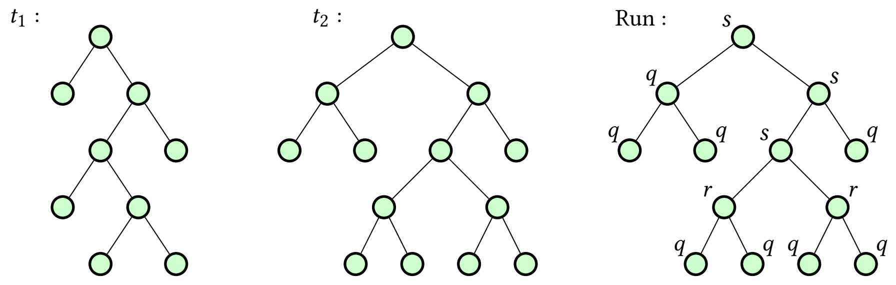


Figure 2: A tree automata $\mathcal{T}$, binary ordered trees t_1 and t_2, and a run of $\mathcal{T}$ over t_2.

$T_1, \ldots, T_k$ and $T = \cup_i T_i$, one can sample from T via the following procedure:

(1) Sample a set T_i with probability proportional to $|T_i|$.
(2) Sample an element t uniformly from T_i.
(3) Accept t with probability $1/m(t)$, where $m(t)$ is the total number of sets T_j which contain t.

The QPRAS of [31] applied this procedure recursively, using approximations $\widetilde{N}(T_i)$ in the place of $|T_i|$, where the union $T = \cup_i T_i$ in question is just the union in Equation (3), and each T_i is a product of smaller sets $T_i = T_{i,1} \otimes T_{i,2}$ which are themselves unions of sets at a lower depth. So to carry out **(2)**, one must recursively sample from $T_{i,1}$ and $T_{i,2}$. The overall probability of acceptance in **(3)** is now exponentially small in the sampling depth. Using a classic depth reduction technique [58], they can reduce the depth to $\log(n)$, but since $m(t)$ can still be as large as $\Omega(n|\mathcal{T}|)$ at each step, the resulting acceptance probability is quasi-polynomially small.

4.4 A Partition Based Approach

The difficult with Karp-Luby sampling is that it relies on a rejection step to compensate for the fact that some elements can be sampled in multiple ways. Instead, our approach will be to *partition* the sets in question, so that no element can be sampled in more than one way. Simply put, to sample from T, we will first partition T into disjoint subsets $T'_1, \ldots, T'_\ell$. Next, we sample a set T'_i with probability (approximately) proportional to $|T'_i|$, and lastly we set $T \leftarrow T'_i$ and now recursively sample from the new T. The recursion ends when the current set T has just one element. Clearly no rejection procedure is needed now for the sample to be approximately uniform. To implement this template, however, there are two main implementation issues which we must address. Firstly, how to partition the set T, and secondly, how to efficiently estimate the size of each part T'_i. In the remainder, we will consider these two issues in detail.

Our High-Level Sampling Template
Input: Arbitrary set T.

(1) If $|T| = 1$, return T. Otherwise, find some partition
$$T = \cup_{i=1}^{\ell} T'_i$$
(2) Call subroutine to obtain estimates $\widetilde{N}(T'_i) \approx |T'_i|$
(3) Set $T \leftarrow T'_i$ with probability $\widetilde{N}(T'_i)/\sum_j \widetilde{N}(T'_j)$, and recursively sample from T.

5 A PARTITION SCHEME FOR TREE AUTOMATA

For the rest of the paper, fix some state s^i. It will suffice to show how to generate a uniform sample from the set $T(s^i)$. To implement the above template, we will rely on having inductively pre-computed estimates of $|T(r^j)|$ for every $r \in S$ and $j < i$. Specifically, our algorithm proceeds in rounds, where on the j-th round we compute an approximation $\widetilde{N}(r^j) \approx |T(r^j)|$ for each state $r \in S$. In addition to these estimates, a key component of our algorithm is that, on the i-th round, we also store *sketches* $\widetilde{T}(r^j)$ of each set $T(r^j)$ for $j < i$, which consist of polynomially many uniform samples from $T(r^j)$. One can uses these sketches $\widetilde{T}(r^j)$ to aid in the generation of uniform samples for the larger sets $T(s^i)$ on the i-th round. For instance, given a set of trees $T = \cup_{j=1}^{k} T_j$ for some sets $T_1, \ldots, T_k$ where we have estimates $\widetilde{N}(T_j) \approx |T_j|$ and sketches $\widetilde{T}_j \subseteq T_j$, one could estimate $|T|$ by the value

$$\sum_{j=1}^{k} \widetilde{N}(T_j) \left(\frac{\left| \widetilde{T}_j \setminus \bigcup_{j' < j} T_{j'} \right|}{\left| \widetilde{T}_j \right|} \right) \tag{4}$$

Here, the term in parenthesis in 4 estimates the fraction of the set $\widetilde{T}_j$ which is not already contained in the earlier sets $T_{j'}$.

5.1 The Partition Scheme for NFA

The above insight of sketching the intermediate subproblems $T(r^j)$ of the dynamic program and applying 4 was made by [8] in their FPRAS for non-deterministic finite automata (NFA). Given an NFA $\mathcal{N}$ with states S, $\Sigma = \{0, 1\}$, and any state $s \in S$ of $\mathcal{N}$, one can similarly define the intermediate subproblem $W(s^i)$[3] as the set of *words* of length i that can be derived starting at the state s. The FPRAS of [8] similarly pre-computes sketches for these sets in a bottom-up fashion. To sample a string $w = w_1 \cdots w_i \in W(s^i)$, they sampled the symbols in w bit by bit, effectively "growing" a prefix of w. First, $W(s^i)$ is partitioned into $W(s^i, 0) \cup W(s^i, 1)$, where $W(s^i, b) \subseteq W(s^i)$ is the subset of strings $x = x_1 \cdots x_i \in W(s^i)$ with first bit x_1 equal to b. If for any prefix w', we define $R_{w'} \subseteq S$ to be the set of states r such that there is a path of transitions from s to r labeled by w', then observe that $W(s^i, b) = \{b\} \cdot \cup_{r \in R_b} W(r^{i-1})$, where $\cdot$ is the concatenation operation for sets of words. Thus $|W(s^i, b)|$ can be estimated directly by Equation 4 in polynomial

[3]We use W to denote sets of words, and T for sets of trees.

time. After the first bit $w_1 = b$ is sampled, they move on to sample the second bit w_2 conditioned on the prefix $w_1 = b$. By partitioning the strings again into those with prefix equal to either $b0$ or $b1$, each of which is described compactly as $\{bb'\} \cdot \cup_{r \in R_{bb'}} W(r^{i-2})$ for $b' \in \{0, 1\}$, one can use Equation 4 again to sample w_2 from the correct distribution, and so on.

The key "victory" in the above approach is that for NFAs, one can compactly condition on a prefix w' of a word $w \in W(s^i)$ as a union $\cup_{r \in R_{w'}} W(r^{i-|w'|})$ taken over some easy to compute subset of states $R_{w'} \subseteq S$. In other words, to condition on a partial derivation of a word, one need only remember a subset of states. This is possible because, for NFAs, the overall configuration of the automata at any given time is specified only by a single current state of the automata. However, this fact breaks down fundamentally for tree automata. Namely, at any intermediate point in the derivation of a tree, the configuration of a tree automata is described not by a single state, but rather by the combination of states $(r_{t_1}^{j_1}, \ldots, r_{t_k}^{j_k})$ assigned to the (possibly many) leaves of the partially derived tree. So the number of possible configurations is exponential in the number of leaves of the partial tree. Consequentially, the number of sets in the union of Equation 4 is exponentially large.[4] Handling this lack of a compact representation is the main challenge for tree automata, and will require a substantially different approach to sampling.

5.2 The Partition Scheme for Tree Languages

Similarly at a high level to the word case, our approach to sampling will be to "grow" a tree t from the root down. However, unlike in the word case, there is no longer any obvious method to partition the ways to grow a tree (for words, one just partitions by the next bit in the prefix). Our solution to this first challenge is to partition based on the *sizes* of the subtrees of all the leaves of t. Namely, at each step we expand one of the leaves ℓ of t, and choose what the final sizes of the left and right subtrees of ℓ will be. By irrevocably conditioning on the final sizes of the left and right subtrees of a leaf ℓ, we partition the set of possibles trees which t can grow into based on the sizes that we choose. Importantly, we do **not** condition on the states which will be assigned to any of the vertices in t, since doing so would no longer result in a partition of $T(s^i)$.

More formally, we grow a *partial tree* τ, which is an ordered tree with the additional property that some of its leaves are labeled with positive integers, and these leaves are referred to as *holes*. For an example, see the leftmost tree in Figure 3. A partial tree τ is called complete if it has no holes. For a hole H of τ, we denote its integral label by $\tau(H) \geq 1$, and call $\tau(H)$ the *final size* of H, since $\tau(H)$ will indeed be the final size of the subtree rooted at H once τ is complete. Intuitively, to complete τ we must replace each hole H of τ with a subtree of size exactly $\tau(H)$. Because no states are involved in this definition, a partial tree τ is by itself totally independent of the automata.

We can now define the set $T(s^i, \tau) \subseteq T(s^i)$ of *completions* of τ as the set of trees $t \in T(s^i)$ such that τ is a subtree of t sharing the same root, and such that for every hole $H \in \tau$ the subtree rooted at the corresponding node $H \in t$ has size $\tau(H)$. Equivalently, t can be obtained from τ by replacing each hole $H \in \tau$ with a subtree t_H of size $\tau(H)$. If $\textcircled{i}$ is the partial tree consisting of a single hole with final size i, then we have $T(s^i, \textcircled{i}) = T(s^i)$. So at each step in the construction of τ, beginning with $\tau = \textcircled{i}$, we will attempt to sample a tree t uniformly from $T(s^i, \tau)$. To do so, we can pick any hole $H \in \tau$, and expand it by adding left and right children and fixing the final sizes of the subtrees rooted at those children. There are $\tau(H)$ ways of doing this: namely, we can fix the final size of the left and right subtrees to be j and $\tau(H) - j - 1$ respectively, for each $j \in \{0, 1, \ldots, \tau(H) - 1\}$. So let τ_j be the partial tree resulting from fixing these final sizes to be j and $\tau(H) - j - 1$, and notice that $T(s^i, \tau_0), \ldots, T(s^i, \tau_{\tau(H)-1})$ partitions the set $T(s^i, \tau)$. Thus it will now suffice to efficiently estimate the sizes $|T(s^i, \tau_j)|$ of each piece in the partition.

[4]By being slightly clever about the order in which one derives the tree, one can reduce the number of "active" leafs to $O(\log n)$, which would result in a quasi-polynomial $|S|^{O(\log n)}$ time algorithm following the approach of [8], which in fact is a slight improvement on the $(|S|n)^{O(\log n)}$ obtained from [31].

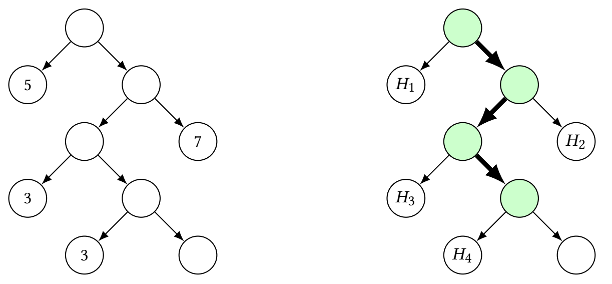


Figure 3: Two examples of partial trees. The left-hand side tree shows the label of each hole written inside the node. The right-hand side tree illustrates the main path, where non-white (green) nodes and thick arcs are used to highlight the vertices and edges on the main path.

5.3 Estimating the Number of Completions via the Main Path

The remaining challenge can now be rephrased in following way: given any partial tree τ, design a subroutine to estimate the number of completions $|T(s^i, \tau)|$. The key tool in our approach to this is a reduction which allows us to represent the set $T(s^i, \tau)$ as the language generated by a succinct NFA, whose transitions are labeled by large sets which are succinctly encoded (see earlier definition Theorem 1.4). In our reduction, on round $i \leq n$, the alphabet Σ of the succinct NFA will be the set of all ordered trees of size at most i. Note that this results in Σ and the label sets A being exponentially large in n, preventing one from applying the algorithm of [8].

Our first observation is that by always choosing the hole H at the lowest depth to expand in the partitioning scheme, the resulting holes $H_1, \ldots, H_k \in \tau$ will be *nested* within each other. Namely, for each $i > 1$, H_i will be contained in the subtree rooted at the sibling of H_{i-1}. Using this fact, we can define a distinguished path P between the parent of H_1 and the parent of H_k. Observe that each hole H_j must be a child of some node in P. We call P the *main path* of τ (see Figure 3). For simplicity, assume that each vertex $v \in P$

has exactly one child that is a hole of τ,[5] and label the vertices of the path $P = \{v_1, v_2, \dots, v_k\}$, so that H_j is the child of v_j. Notice by the above nestedness property, the holes $H_j, H_{j+1}, \dots, H_k$ are all contained in the subtree rooted at v_j.

Observe that any completed tree $t \in T(s^i, \tau)$ can be uniquely represented by the trees $(t_1, \dots, t_k)$, such that t is obtained from τ by replacing each hole $H_i \in \tau$ by the tree t_i. Thinking of each tree t_i as a symbol in the alphabet Σ of all ordered trees, we can thus specify the tree t by a *word* $t_1 t_2 \cdots t_k \in \Sigma^*$. So our goal is to show that the set of words $T(s^i, \tau) = \{t_1 \cdots t_k \in \Sigma^* \mid t_1 \cdots t_k \in T(s^i, \tau)\}$ is the language accepted by an succinct NFA $\mathcal{N}$ over the alphabet of ordered trees Σ with polynomially many states and set-labeled transitions.

Now NFAs can only express labeled paths (i.e., words) and not trees. However, the key observation is that if we restrict ourselves to the main path P, then the sequences of states from $\mathcal{T}$ which can occur along P can indeed be expressed by an NFA. Informally, for every vertex $v_j \in P$ with (wlog) left child H_j, and for every transition $s \longrightarrow rs'$ in the tree automata $\mathcal{T}$ which could occur at v_j, we create a unique transition $s \longrightarrow s'$ in the succinct NFA. Here, the two states s, s' are assigned to the vertices v_j, v_{j+1} on the main path P, and the state r is placed inside of the hole H_j. Now the set of trees t_j which could be placed in H_j by this transition *only* depends on the state r. Specifically, this set of trees is exactly $T(r^{\tau(H_j)})$. Thus, if we label this transition $s \longrightarrow s'$ in the succinct NFA by the set $T(r^{\tau(H_j)})$, the language accepted by the NFA will be precisely $T(s^i, \tau)$. The full details can be found in the full version.

The crucial fact about this construction is that the transition labels of the succinct NFA are all sets of the form $T(r^j)$ for some $r \in S$ and $j < i$. Since $j < i$, our algorithm has already pre-computed the sketches $\widetilde{T}(r^j)$ and estimates $\widetilde{N}(T^j)$ of the label sets $T(r^j)$ at this point. In the next section, we will use these sketches and estimate to satisfy the "oracle" assumptions of Theorem 1.4.

6 AN FPRAS FOR SUCCINCT NFAS

Now that we have constructed the succinct NFA $\mathcal{N}$ which recognizes the language $T(s^i, \tau)$ as its k-slice, we must devise a subroutine to approximate the size of the k-slice of $\mathcal{N}$. Let S', Δ' be the states and transitions of $\mathcal{N}$. In order to estimate $|\mathcal{L}_k(\mathcal{N})|$, we mimic the inductive, dynamic programming approach of our "outside" algorithm.[6] Namely, we define partial states of a dynamic program on $\mathcal{N}$, by setting $W(x^\ell)$ to be the set of words of length ℓ accepted by $\mathcal{N}$ starting from the state $x \in S'$. We then similarly divide the computation of our algorithm into rounds, where on round ℓ of the subroutine, we inductively pre-compute new NFA sketches $\widetilde{W}(x^\ell)$ of $W(x^\ell)$ and estimates $\widetilde{N}(x^\ell)$ of $|W(x^\ell)|$ for each state $x \in S'$. Given these estimates and sketches, our procedure for obtaining the size estimates $\widetilde{N}(x^\ell)$ is straightforward. Thus, similar to the outside algorithm, the central challenge is to design a polynomial time algorithm to sample from the set $W(x^\ell)$, allowing us to construct the sketch $\widetilde{W}(x^\ell)$.

For a string $u \in \Sigma^*$, define $W(x^\ell, u)$ to be the set of strings $w \in W(x^\ell)$ with prefix equal to u. Recall the approach of [8] to this problem for standard NFAs began by partitioning $W(x^\ell)$ into $\bigcup_{\alpha \in \Sigma} W(x^\ell, \alpha)$ and estimating the size $|W(x^\ell, \alpha)|$ for each $\alpha \in \Sigma$. Then one chooses α with probability (approximately)

$$\mathbf{Pr}[\alpha] = \frac{|W(x^\ell, \alpha)|}{\sum_{\beta \in \Sigma} |W(x^\ell, \beta)|}$$

and recurses into the set $W(x^\ell, \alpha)$. Clearly we can no longer follow this strategy, as $|\Sigma|$ is of exponential size with respect to $\mathcal{N}$. Specifically, we cannot estimate $|W(x^\ell, \alpha)|$ for each $\alpha \in \Sigma$. Instead, our approach is to approximate the behavior of the "idealistic" algorithm which *does* estimate all these sizes, by sampling from Σ without explicitly estimating the sampling probabilities $\mathbf{Pr}[\alpha]$. Namely, for a prefix u we must sample a string $v \sim W(x^\ell, u)$, by first sampling the next symbol $\alpha \sim \Sigma$ from a distribution $\widetilde{\mathcal{D}}(u)$ which is close to the true distribution $\mathcal{D}(u)$ over Σ given by $\mathbf{Pr}[\alpha] = |W(x^\ell, u \cdot \alpha)| / |W(x^\ell, u)|$ for each $\alpha \in \Sigma$.

To do this, first note that we can write

$$W\left(x^\ell, u\right) = \{u\} \cdot \bigcup_{y \in R(x,u)} W\left(y^{\ell - |u|}\right)$$

where $R(x, u) \subseteq S'$ is the set of states y such that there is a a path of transitions from x to y labeled by sets $A_1 \dots A_{|u|}$ with $u_j \in A_j$ for each $j \in \{1, \dots, |u|\}$. Thus the set of possible symbols α that we can append to u is captured by the sets of labels of the transitions out of some state $y \in R(x, u)$. Now consider the set of transitions $\{(y, A, z) \in \Delta' \mid y \in R(x, u)\}$, namely, all transitions out of some state in $R(x, u)$. Furthermore, suppose for the moment that we were given an oracle which generates uniform samples from each label set A of a transition (y, A, z), and also provided estimates $\widetilde{N}(A)$ of the size of that set $|A|$. Given such an oracle, we design a multi-step rejection procedure to sample a symbol α approximately from $\mathcal{D}(u)$, based on drawing samples from the external oracle and then rejecting them based on intersection ratios of our pre-computed internal NFA sketches $\widetilde{W}(y^{\ell - |u|})$.

Since α is generated by a transition out of $R(x, u)$, we first sample such a transitions with probability proportional to the number of remaining suffixes which could be derived by taking that transition. More specifically, the number of suffixes that can be produced by following a transition (y, A, z) is given by $|A| \cdot |W(z^{\ell - |u| - 1})|$, which can be approximated by $\widetilde{N}(A) \cdot \widetilde{N}(z^{\ell - |u| - 1})$ using the oracle and our internal estimates. Then if Z is the sum of the estimates $\widetilde{N}(A) \cdot \widetilde{N}(z^{\ell - |u| - 1})$ taken over all transitions $\{(y, A, z) \in \Delta' \mid y \in R(x, u)\}$, we choose a transition (y, A, z) with probability $\widetilde{N}(A) \cdot \widetilde{N}(z^{\ell - |u| - 1}) / Z$ and then call the oracle to obtain a sample $\alpha \sim A$. The sample α now defines a piece $W(x^\ell, u \cdot \alpha)$ of the partition of $W(x^\ell, u)$ which the idealistic algorithm would have estimated and potentially chosen. However, at this point α is not drawn approximately from the correct distribution $\mathcal{D}(u)$, since the sample from the oracle does not taken into account any information about the other transitions which could also produce α. To remedy this, we show that it suffices to accept the symbol α with probability:

$$\frac{\left|\widetilde{W}(z^{\ell - |u| - 1}) \setminus \bigcup_{\zeta \in \mathcal{B}(\alpha) : \zeta \prec z} W(\zeta^{\ell - |u| - 1})\right|}{\left|\widetilde{W}(z^{\ell - |u| - 1})\right|} \qquad (\dagger)$$

where $\prec$ is an ordering over S' and $\mathcal{B}(\alpha)$ is the set of all states that can be reached from $R(x, u)$ by reading α, namely, all states

[5] Extra care should be taken when this is not the case.

[6] We think of this subroutine to estimate $|T(s^i, \tau)|$ as being the "inner loop" of the FPRAS.

ζ such that there exists a transition $(\eta, B, \zeta) \in \Delta'$ with $\eta \in R(x, u)$ and $\alpha \in B$. Otherwise, we reject α. Intuitively, probability (†) is small when the sets of suffixes which could be derived following transitions $\mathcal{B}(\alpha)$ that could also produce α intersect heavily. If this is the case, we have "overcounted" the contribution of the set $W(x^{\ell}, u \cdot \alpha)$ in the partition, and so the purpose of the probability (†) is to compensate for this fact. We show that this procedure results in samples α drawn from a distribution $\widetilde{\mathcal{D}}(u)$ which is close in statistical distance to the exact distribution $\mathcal{D}(u)$. Furthermore, one can bound the acceptance probability by (†) $\geq 1/\text{poly}(n)$ in expectation over the choice of α, so after repeating the oracle call polynomially many times, we will accept a sample α. Once α is accepted, we condition on it and move to the next symbol, avoiding any recursive rejection sampling.

We now return to the assumption of having a oracle to sample from and approximate the size of the label sets A. By construction, A is a set of trees $T(s^j)$ for which we have pre-computed sketches and estimates $\widetilde{T}(s^j)$, $\widetilde{N}(s^j)$ from the external algorithm. To simulate this oracle, we reuse the samples within the sketches $\widetilde{T}(s^j)$ for each call to the succinct NFA sub-routine, pretending that they are being generated fresh and on the fly. However, since the same sketches must be reused on each call to the subroutine, we lose independence between the samples generated within subsequent calls. Ultimately, though, all that matters is that the estimate of $|T(s^i, \tau)|$ produced by the subroutine is correct. So to handle this, we show that one can condition on a deterministic property of the sketches $\{\widetilde{T}(s^j)\}_{s \in S, j < i}$, so that *every* possible run of the succinct NFA subroutine will yield a good approximation, allowing us to ignore these dependencies.

Lastly, we handle the propagation of error resulting from the statistical distance between $\widetilde{\mathcal{D}}(u)$ and $\mathcal{D}(u)$. This statistical error feeds into the error for the estimates $\widetilde{N}(x^{\ell+1})$ on the next step, both of which feed back into the statistical error when sampling from $W(x^{\ell+1})$, doubling the error at each step. We handle this by introducing an approximate rejection sampling step, inspired by an exact rejection sampling technique due to [41] (the exact version was also used in [8]). This approximately corrects the distribution of each sample w, causing the error to increases linearly in the rounds instead of geometrically, which will be acceptable for our purposes.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we prove the existence of a fully polynomial-time randomized approximation scheme (FPRAS) for counting solutions to a large class of conjunctive queries, as well as the existence of a fully polynomial-time almost uniform sampler (FPAUS) for these solutions. In particular, our algorithm applies to all conjunctive queries with bounded hypertree width (as defined in Section 3.2).

In Section 3.3, and based on Theorem 3.2 and the results in [37], we provide a characterization of the classes of conjunctive queries that admit an FPRAS, when such classes are of the form $\text{CQ}(\mathcal{G})$ for a family $\mathcal{G}$ of graphs (see Corollary 3.3). It remains as an open problem how to extend this characterization to any class of conjunctive queries and, in particular, to identify the right restriction on conjunctive queries that is equivalent to the existence of an FPRAS. It should be mentioned that recently, in a follow-up work by Focke, Goldberg, Roth and Živný [29], it was proved that our results can be extended to the case of conjunctive queries with bounded *fractional hypertree width*.

An important component of this work, in addition to the earlier paper [8], is the study of approximate counting problems in automata theory. As demonstrating in this paper, finite automata have strong expressive potential to model the solution space of many problems in computer science. Thus, resolving the counting problem for a class of automata can result in an FPRAS for many additional problems. So far, we know that the counting problem for non-deterministic finite automata (NFA), and now tree automata, admit polynomial time randomized approximations. However, there are still many classes of automata for which it is unknown if an FPRAS exists.

An additional natural case to consider is the class of *context free grammars* (CFG), which generate the class of context free languages (CFL). Context free grammars are more expressive than tree-automata, although they are closely related (for instance, the set of all derivation trees of a CFG can be expressed by a tree automata). There are several barriers to generalizing the approach in this paper to the case of context free grammars. Mainly, in order to carry out our high-level sampling template described in Section 4, we need to partition the current set T into a collection of disjoint subsets $T'_1, \ldots, T'_{\ell}$. We accomplish this for the case of tree automata by splitting based on the final sizes of the left and right subtrees. However, such a partition does not work for the case of CFG's, since a single word in a CFL can have multiple derivations whose subtrees have different sizes, thus this word would be contained in multiple such sets T'_i causing the scheme to fail to be a partition. Thus, the key barrier to extending our algorithmic framework to the case of CFG's is the design of an efficient partition scheme which avoids this issue.

ACKNOWLEDGMENTS

This work was funded by ANID - Millennium Science Initiative Program - Code ICN17_002.

REFERENCES

[1] Serge Abiteboul and Gilles Dowek. 2020. *The Age of Algorithms.* Cambridge University Press.

[2] Rajeev Alur, Kousha Etessami, and P. Madhusudan. 2004. A Temporal Logic of Nested Calls and Returns. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Proceedings.* 467–481.

[3] Rajeev Alur and P. Madhusudan. 2004. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004.* 202–211.

[4] Rajeev Alur and P. Madhusudan. 2009. Adding nesting structure to words. *J. ACM* 56, 3 (2009), 16:1–16:43.

[5] Carme Àlvarez and Birgit Jenner. 1993. A Very Hard log-Space Counting Class. *Theor. Comput. Sci.* 107, 1 (1993), 3–30.

[6] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. 2017. A Circuit-Based Approach to Efficient Enumeration. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland.* 111:1–111:15.

[7] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. 2019. Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019.* 89–103.

[8] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2019. Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems.* ACM, 59–73.

[9] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2020. When is Approximate Counting for Conjunctive Queries Tractable? *arXiv preprint arXiv:2005.10029* (2020). https://arxiv.org/abs/2005.10029

[10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press.

[11] Alberto Bertoni, Massimiliano Goldwurm, and Massimo Santini. 2000. Random generation and approximate counting of ambiguously described combinatorial structures. In *Annual Symposium on Theoretical Aspects of Computer Science.* Springer, 567–580.

[12] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). 2009. *Handbook of Satisfiability.* Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press.

[13] Randal E. Bryant. 1992. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.* 24, 3 (1992), 293–318.

[14] Andrei A. Bulatov and Stanislav Živný. 2020. Approximate Counting CSP Seen from the Other Side. *ACM Trans. Comput. Theory* 12, 2, Article 11 (May 2020), 19 pages. https://doi.org/10.1145/3389390

[15] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 1999. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *IJCAI*, Vol. 99. 84–89.

[16] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77).* 77–90.

[17] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. 1999. On Random Sampling over Joins. In *SIGMOD.* 263–274.

[18] Chandra Chekuri and Anand Rajaraman. 1997. Conjunctive Query Containment Revisited. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings.* 56–70.

[19] Yu Chen and Ke Yi. 2020. Random Sampling and Size Estimation Over Cyclic Joins. In *ICDT.* 7:1–7:18.

[20] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree Automata Techniques and Applications. Available on: http://tata.gforge.inria.fr/. release October, 12th 2007.

[21] Bruno Courcelle. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation* 85, 1 (1990), 12–75.

[22] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. 2001. *Complexity classifications of Boolean constraint satisfaction problems.* SIAM monographs on discrete mathematics and applications, Vol. 7. SIAM.

[23] Víctor Dalmau and Peter Jonsson. 2004. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science* 329, 1 (2004), 315 – 323. https://doi.org/10.1016/j.tcs.2004.08.008

[24] Adnan Darwiche. 2001. Decomposable negation normal form. *Journal of the ACM (JACM)* 48, 4 (2001), 608–647.

[25] Adnan Darwiche and Pierre Marquis. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17 (2002), 229–264.

[26] Arnaud Durand and Stefan Mengel. 2015. Structural Tractability of Counting of Solutions to Conjunctive Queries. *Theory Comput. Syst.* 57, 4 (2015), 1202–1249. https://doi.org/10.1007/s00224-014-9543-y

[27] E Allen Emerson and Charanjit S Jutla. 1991. Tree automata, mu-calculus and determinacy. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science.* IEEE, 368–377.

[28] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory.* Springer.

[29] Jacob Focke, Leslie Ann Goldberg, Marc Roth, and Stanislav Živný. 2021. Approximately Counting Answers to Conjunctive Queries with Disequalities and Negations. *arXiv preprint arXiv:2103.12468* (2021).

[30] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2009. Model Counting. In *Handbook of Satisfiability.* 633–654.

[31] Vivek Gore, Mark Jerrum, Sampath Kannan, Z Sweedyk, and Steve Mahaney. 1997. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation* 134, 1 (1997), 59–74.

[32] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016.* 57–74.

[33] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 1998. The Complexity of Acyclic Conjunctive Queries. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98, November 8-11, 1998, Palo Alto, California, USA.* 706–715.

[34] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2000. A comparison of structural CSP decomposition methods. *Artif. Intell.* 124, 2 (2000), 243–282.

[35] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.* 64, 3 (2002), 579–627.

[36] Martin Grohe. 2007. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM* 54, 1 (2007), 1:1–1:24.

[37] Martin Grohe, Thomas Schwentick, and Luc Segoufin. 2001. When is the evaluation of conjunctive queries tractable?. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece (STOC'01).* 657–666.

[38] Pavol Hell and Jaroslav Nesetril. 1992. The core of a graph. *Discret. Math.* 109, 1-3 (1992), 117–126.

[39] Pavol Hell and Jaroslav Nesetril. 2004. *Graphs and homomorphisms.* Oxford lecture series in mathematics and its applications, Vol. 28. Oxford University Press.

[40] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. 1986. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theor. Comput. Sci.* 43 (1986), 169–188.

[41] Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. 1986. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science* 43 (1986), 169–188.

[42] Sampath Kannan, Z Sweedyk, and Steve Mahaney. 1995. Counting and random generation of strings in regular languages. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, 551–557.

[43] Richard M Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo approximation algorithms for enumeration problems. *Journal of algorithms* 10, 3 (1989), 429–448.

[44] Harry G Mairson. 1994. Generating words in a context-free language uniformly at random. *Inform. Process. Lett.* 49, 2 (1994), 95–99.

[45] Frank Neven. 2002. Automata Theory for XML Researchers. *SIGMOD Record* 31, 3 (2002), 39–46.

[46] Umut Oztok and Adnan Darwiche. 2014. CV-width: A New Complexity Parameter for CNFs.. In *ECAI.* 675–680.

[47] Reinhard Pichler and Sebastian Skritek. 2013. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.* 79, 6 (2013), 984–1001.

[48] Knot Pipatsrisawat and Adnan Darwiche. 2008. New Compilation Languages Based on Structured Decomposability. In *AAAI*, Vol. 8. 517–522.

[49] Michael O Rabin. 1969. Decidability of second-order theories and automata on infinite trees. *Transactions of the american Mathematical Society* 141 (1969), 1–35.

[50] Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. 2003. *Database management systems.* Vol. 3. McGraw-Hill New York.

[51] Francesca Rossi, Peter Van Beek, and Toby Walsh. 2006. *Handbook of constraint programming.* Elsevier.

[52] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.

[53] Thomas Schwentick. 2007. Automata for XML - A survey. *J. Comput. System Sci.* 73, 3 (2007), 289–315.

[54] Helmut Seidl. 1990. Deciding Equivalence of Finite Tree Automata. *SIAM J. Comput.* 19, 3 (1990), 424–437.

[55] Eugenia Ternovskaia. 1999. Automata Theory for Reasoning About Actions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages.* 153–159.

[56] James W. Thatcher and Jesse B. Wright. 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory* 2, 1 (1968), 57–81.

[57] Wolfgang Thomas. 1997. Languages, automata, and logic. In *Handbook of formal languages.* Springer, 389–455.

[58] LG Valiant, S Skyum, S Berkowitz, and C Rackoff. 1983. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM J. Comput.* 12, 4 (1983), 641–644.

[59] Leslie G Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

[60] Moshe Y Vardi. 1995. Alternating automata and program verification. In *Computer Science Today.* Springer, 471–485.

[61] Moshe Y. Vardi. 2000. Constraint Satisfaction and Database Theory: a Tutorial. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA.* 76–85.

[62] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings.* 82–94.

[63] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *SIGMOD.* 1525–1539.