General Space-Time Tradeoffs via Relational Queries

Author: Please provide author information

Abstract

In this paper, we investigate space-time tradeoffs for data structure problems. The goal is to create a data structure in an initial preprocessing phase and use it for answering (multiple) queries. Previous work has developed data structures that trade off space usage for answering time and has proved conditional space lower bounds for queries of practical interest such as the path and triangle query. However, most of these results cater to only those queries, lack a comprehensive framework, and are not generalizable. The isolated treatment of these queries also fails to utilize the connections with extensive research on related problems within the database community. The key insight in this work is to exploit the formalism of relational algebra by casting the problems as answering join queries over a relational database. Using the notion of *adorned queries* and *access patterns*, we propose a simple unified framework that captures several widely studied algorithmic problems and recovers existing space-time tradeoffs. Our algorithm is based on an application of the *join size bound* to capture the space usage of our data structure. We combine our data structure with *query decomposition* techniques to further improve the tradeoffs and show that it is readily extensible to queries with negation.

2012 ACM Subject Classification

Keywords and phrases space-time tradeoffs, conjunctive queries

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Recent work has made remarkable progress in developing data structures and algorithms for answering set intersection problems [11], reachability oracles and directed reachability [4, 3, 9], histogram indexing [7, 17], and problems related to document retrieval [2, 18]. This class of problems splits an algorithmic task into two phases: the *preprocessing phase*, which computes a space-efficient data structure, and the *answering phase*, which uses the data structure to answer the requests to minimize the answering time. A fundamental algorithmic question related to these problems is the tradeoff between the space S necessary for data structures and the answering time T for requests.

For example, consider the 2-Set Disjointness problem: given a universe of elements U and a collection of m sets $C_1, \ldots, C_m \subseteq U$, we want to create a data structure such that for any pair of integers $1 \leq i, j \leq m$, we can efficiently decide whether $C_i \cap C_j$ is empty or not. Previous work [9, 11] has shown that the space-time tradeoff for 2-Set Disjointness is captured by the equation $S \cdot T^2 = N^2$, where N is the total size of all sets. The data structure obtained is conjectured to be optimal [11], and its optimality was used to develop conditional lower bounds for other problems, such as approximate distance oracles [4, 3]. Similar tradeoffs have been independently established for other data structure problems as well. In the k-Reachability problem [11, 8] we are given as an input a directed graph $G = (V, E)$, an arbitrary pair of vertices u, v, and the goal is to decide whether there exists a path of length k between u and v. In the edge triangle detection problem [11], we are given an input undirected graph $G = (V, E)$, and the goal is to develop a data structure that takes space S and can answer in time T whether a given edge $e \in E$ participates in a triangle or not. Each of these problems has been studied in isolation and, as a result, the algorithmic solutions are not generalizable.

In this paper, we cast many of the above problems into answering *Conjunctive Queries (CQs)* over a relational database. CQs are a powerful class of relational queries with widespread applications in data analytics and graph exploration [28, 27, 10]. For example, by using the relation $R(x, y)$ to encode that element x belongs to set y, 2-Set Disjointness can be captured by the following CQ: $\varphi(y_1, y_2) = R(x, y_1) \wedge R(x, y_2)$. As we will see later, k-Reachability can also be naturally captured by a CQ. The insight of casting data structure problems into CQs over a database allows for a unified treatment for developing algorithms within the same framework. In particular, we can leverage the techniques developed by the data management community through a long line of research on efficient join evaluation [29, 21, 20], including worst-case optimal join algorithms [20] and tree decompositions [12, 24]. Building upon these techniques, we achieve the following:

- We obtain in a simple way general space-time tradeoffs for any Boolean CQ (a Boolean CQ is one that outputs only true or false). As a consequence, we recover state-of-the-art tradeoffs for several existing problems (e.g., 2-Set Disjointness as well as its generalization k-Set Disjointness and k-Reachability) as special cases of the general tradeoff. We can even obtain improved tradeoffs for some specific problems, such as edge triangles detection, thus falsifying existing conjectures. This also gives us a way to construct data structures for any new problem that can be cast as a Boolean CQ (e.g., finding any subgraph pattern in a graph).
- Space-time tradeoffs for enumerating (non-Boolean) query results under static and dynamic settings have been a subject of previous work [1, 13, 10, 22, 16, 15]. The space-time tradeoffs from [10] can be applied to the setting of this paper by stopping the enumeration after the first result is observed. We improve upon this result by (i) showing a much simpler data structure construction and proofs, and (ii) shaving off a polylogarithmic factor from the space-time tradeoff.

We next summarize our three main technical contributions.

1. Our first main result (Theorem 7) is a simple algorithm that builds a data structure to answer any Boolean CQ under a specific access pattern. Importantly, the data structure can be tuned to trade off space for answering time, thus capturing the full continuum between optimal space and answering time. At one extreme, the data structure achieves constant answering time by explicitly storing all possible answers. At the other extreme, the data structure stores nothing, but we execute each request from scratch. We show how to recover existing and new tradeoffs using this general framework. We also refute a lower bound conjecture for the edge triangles detection problem established by [11].
2. The first main result may sometimes lead to suboptimal tradeoffs since it does not take into account the structural properties of the query. Our second main result (Theorem 13) combines tree decompositions of the query structure with access patterns to improve space efficiency. This result allows us to recover the state-of-the-art space/time tradeoffs for k-Reachability and other problems.
3. Our third main contribution applies our framework to CQs with negation. This allows us to construct space-time tradeoffs for tasks such as detecting open triangles in a graph.

Organization. We introduce the basic terminology and problem definition in Section 2 and Section 3. We presents our main results for Boolean adorned queries in Section 4 and Section 5. Section 6 presents the extension of our results to queries containing negations. We discuss related work in Section 7, and finally conclude in Section 8 with promising future research directions and open problems.

2 Preliminaries

In this section, we present the basic notation and terminology.

Data Model. A *schema* is defined as a collection of relation names, where each relation name R is associated with an arity n. Assuming a (countably infinite) domain $\mathbf{dom}$, a tuple t of relation R is an element of $\mathbf{dom}^n$. An instance of relation R with arity n is a finite set of tuples of R; the size of the instance will be denoted as $|R|$. An input database D is a set of relation instances over the schema. The size of the database $|D|$ is the sum of sizes of all its instances.

Conjunctive Queries. A *Conjunctive Query* (CQ) is an expression of the form

$$\varphi(\mathbf{y}) = R_1(\mathbf{x}_1) \wedge R_2(\mathbf{x}_2) \wedge \ldots \wedge R_n(\mathbf{x}_n).$$

The expressions $\varphi(\mathbf{y}), R_1(\mathbf{x}_1), R_2(\mathbf{x}_2), \ldots, R_n(\mathbf{x}_n)$ are called *atoms*. The atom $\varphi(\mathbf{y})$ is the *head* of the query, while the atoms $R_i(\mathbf{x}_i)$ form the *body*. Here, $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are vectors where each position is a variable (typically denoted as $x, y, z, \ldots$) or a constant from $\mathbf{dom}$ (typically denoted $a, b, c, \ldots$). Each $\mathbf{x}_i$ must match the arity of the relation R_i, and the variables in $\mathbf{y}$ must occur in the body of the query. We use $\mathsf{vars}(\varphi)$ to denote the set of all variables occurring in φ, and $\mathsf{vars}(R_i)$ to denote the set of variables in atom $R_i(\mathbf{x}_i)$. A CQ is *full* if every variable in the body appears also in the head (a.k.a. *quantifier-free*), and *Boolean* if the head contains no variables (a.k.a. *fully-quantified*).

Given variables $x_1, \ldots, x_k$ from $\mathsf{vars}(\varphi)$ and constants $a_1, \ldots, a_k$ from $\mathbf{dom}$, we define $\varphi[a_1/x_1, \ldots, a_k/x_k]$ to be the CQ where every occurrence of a variable x_i, $i = 1, \ldots, k$, is replaced by the constant a_i.

Given an input database D and a CQ φ, we define the query result $\varphi(D)$ as follows. A *valuation* v is a mapping from $\mathbf{dom} \cup \mathsf{vars}(\varphi)$ to $\mathbf{dom}$ such that $v(a) = a$ whenever a is a constant. Then, $\varphi(D)$ is the set of all tuples t such that there exists a valuation v for which $t = v(\mathbf{y})$ and for every atom $R_i(\mathbf{x}_i)$, we have $R_i(v(\mathbf{x}_i)) \in D$.[1]

▶ **Example 1.** Suppose that we have a directed graph G that is represented through a binary relation $R(x, y)$: this means that there exists an edge from node x to node y. We can compute the pairs of nodes that are connected by a directed path of length k using the following CQ, which we call a *path query*: $P_k(x_1, x_{k+1}) = R(x_1, x_2) \wedge R(x_2, x_3) \wedge \cdots \wedge R(x_k, x_{k+1})$.

Output Size Bounds. Let $\varphi(\mathbf{y}) = R_1(\mathbf{x}_1) \wedge R_2(\mathbf{x}_2) \wedge \ldots \wedge R_n(\mathbf{x}_n)$ be a CQ. A weight assignment $\mathbf{u} = (u_i)_{i=1,\ldots,n}$ is called a *fractional edge cover* of $S \subseteq \mathsf{vars}(\varphi)$ if (i) for every atom R_i, $u_i \geq 0$ and (ii) for every $x \in S, \sum_{i:x \in \mathsf{vars}(R_i)} u_i \geq 1$. The *fractional edge cover number* of S, denoted by $\rho^*(S)$ is the minimum of $\sum_{i=1}^{n} u_i$ over all fractional edge covers of S. Whenever $S = \mathsf{vars}(\varphi)$, we call this a fractional edge cover of φ and simply use ρ^*. In a celebrated result, Atserias, Grohe and Marx [5] proved that for every fractional edge cover $\mathbf{u}$ of φ, the size of the output is bounded by the *AGM inequality*:

$$|\varphi(D)| \leq \prod_{i=1}^{n} |R_i|^{u_i}$$

The above bound is constructive [21, 20]: there exists an algorithm that computes the result $\varphi(D)$ in $O(\prod_i |R_i|^{u_i})$ time for every fractional edge cover $\mathbf{u}$.

[1] Here we extend the valuation to mean $v((a_1, \ldots, a_n)) = (v(a_1), \ldots, v(a_n))$.

Tree Decompositions. Let $\varphi(\mathbf{y}) = R_1(\mathbf{x}_1) \wedge R_2(\mathbf{x}_2) \wedge \ldots \wedge R_n(\mathbf{x}_n)$ be a CQ. A *tree decomposition* of φ is a tuple $(\mathcal{T}, (\mathcal{B}_t)_{t \in V(\mathcal{T})})$ where $\mathcal{T}$ is a tree, and every $\mathcal{B}_t$ is a subset of $\mathsf{vars}(\varphi)$, called the *bag* of t, such that

- For every atom R_i, the set $\mathsf{vars}(R_i)$ is contained in some bag; and
- For each variable $x \in \mathsf{vars}(\varphi)$, the set of nodes $\{t \mid x \in \mathcal{B}_t\}$ form a connected subtree of $\mathcal{T}$.

The *fractional hypertree width* of a decomposition is defined as $\max_{t \in V(\mathcal{T})} \rho^*(\mathcal{B}_t)$, where $\rho^*(\mathcal{B}_t)$ is the minimum fractional edge cover of the vertices in $\mathcal{B}_t$. The fractional hypertree width of a query φ, denoted $\mathtt{fhw}(\varphi)$, is the minimum fractional hypertree width among all tree decompositions. We say that a query is *acyclic* if $\mathtt{fhw}(\varphi) = 1$.

Computational Model. To measure the running time of our algorithms, we will use the uniform-cost RAM model [14], where data values and pointers to databases are of constant size. Throughout the paper, all complexity results are with respect to data complexity, where the query is assumed fixed.

3 Framework

In this section, we discuss the concept of adorned queries and present our framework.

3.1 Adorned Queries

In order to model different access patterns, we will use the concept of *adorned queries* introduced by [25]. Let $\varphi(x_1, \ldots, x_k)$ be the head of a CQ φ. In an adorned query, each variable in the head is associated with a binding type, which can be either *bound* (b) or *free* (f). We denote this as φ^η, where $\eta \in \{\mathsf{b}, \mathsf{f}\}^k$ is called the *access pattern*. The access pattern tells us for which variables the user must provide a value as input. Concretely, let $x_1, x_2, \ldots, x_\ell$ be the bound variables. An *access request* is sequence of constants $a_1, \ldots, a_\ell$, and it asks to return the result of the query $\varphi^\eta[a_1/x_1, \ldots, a_\ell/x_\ell]$ on the input database. We next demonstrate how to capture several data structure problems in this way.

▶ **Example 2** (Set Disjointness and Set Intersection)**.** In the set disjointness problem, we are given m sets $S_1, \ldots, S_m$ drawn from the same universe U. Let $N = \sum_{i=1}^m |S_i|$ be the total size of input sets. Each access request is a pair of indexes $(i, j), 1 \leq i, j, \leq m$, for which we need to decide whether $S_i \cap S_j$ is empty or not. To cast this problem as an adorned query, we encode the family of sets as a binary relation $R(x, y)$, such that element x belongs to set y. Note that the relation will have size N. Then, the set disjointness problem corresponds to:

$$\varphi^{\mathsf{bb}}(y, z) = R(x, y) \wedge R(x, z).$$

An access request in this case specifies two sets $y = S_i, z = S_j$, and issues the (Boolean) query $\varphi(S_i, S_j) = R(x, S_i) \wedge R(x, S_j)$.

In the related set intersection problem, given a pair of indexes (i, j) for $1 \leq i, j, \leq m$, we instead want to enumerate the elements in the intersection $S_i \cap S_j$, which can be captured by the following adorned query: $\varphi^{\mathsf{bbf}}(y, z, x) = R(x, y) \wedge R(x, z)$.

▶ **Example 3** (k-Set Disjointness)**.** The k-set disjointness problem is a generalization of 2-set disjointness problem, where each request asks whether the intersection between k sets is empty or not. Again, we can cast this problem into the following adorned query:

$$\varphi^{\mathsf{b}\ldots\mathsf{b}}(y_1, \ldots, y_k) = R(x, y_1) \wedge R(x, y_2) \wedge \cdots \wedge R(x, y_k)$$

▶ **Example 4** (k-Reachability)**.** Given a direct graph G , the k-reachability problem asks, given a pair vertices (u, v), to check whether they are connected by a path of length k. Representing the graph as a binary relation $R(x, y)$ (which means that there is an edge from x to y), we can model this problem through the following adorned query:

$$\varphi^{\mathsf{bb}}(x_1, x_{k+1}) = R(x_1, x_2) \wedge R(x_2, x_3) \wedge \cdots \wedge R(x_k, x_{k+1})$$

Observe that we can also check whether there is a path of length at most k by combining the results of k such queries (one for each length $1, \ldots, k$).

▶ **Example 5** (Edge Triangles Detection)**.** Given a graph $G = (V, E)$, this problem asks, given an edge (u, v) as the request, whether (u, v) participates in a triangle or not. This task can be expressed as the following adorned query

$$\varphi^{\mathsf{bb}}_{\triangle}(x, z) = R(x, y) \wedge R(y, z) \wedge R(x, z)$$

In the reporting version, the goal is to enumerate all triangles participated by edge (x, z), which can also be expressed by the following adorned query $\varphi^{\mathsf{bbf}}_{\triangle}(x, z, y) = R(x, y) \wedge R(y, z) \wedge R(x, z)$.

We say that an adorned query is *Boolean* if every head variable is bound. In this case, the answer for every access request is also Boolean, i.e., true or false.

3.2 Problem Statement

Given an adorned query φ^{η} and an input database D, our goal is to construct a data structure, such that we can answer any access request that conforms to the access pattern η as fast as possible. In other words, an algorithm can be split into two phases:

- **Preprocessing phase:** we computes a data structure using space S.
- **Answering phase:** given an access request, we compute the answer using the data structure built in the preprocessing phase, within time T.

In this work, our goal is to study the relationship between the space of the data structure S and the answering time T for a given adorned query φ^{η}. We will focus on Boolean adorned queries, where the output is just true or false.

4 Space-Time Tradeoffs via Worst-case Optimal Algorithms

Let φ^{η} be an adorned query and $\mathcal{V}_{\mathsf{b}}$ denote its bound variables. For any fractional edge cover $\mathbf{u}$, we define the *slack* of $\mathbf{u}$ as:

$$\alpha(\mathbf{u}) := \min_{x \in \mathsf{vars}(\varphi) \setminus \mathcal{V}_{\mathsf{b}}} \left(\sum_{i : x \in \mathsf{vars}(R_i)} u_i \right).$$

In other words, the slack is the maximum factor by which we can scale down the fractional cover $\mathbf{u}$ so that it remains a valid edge cover of the non-bound variables in the query[2]. Hence $\{u_i / \alpha(\mathbf{u})\}_i$ is a fractional edge cover of the nodes in $\mathsf{vars}(\varphi) \setminus \mathcal{V}_{\mathsf{b}}$. We always have $\alpha(\mathbf{u}) \geq 1$.

▶ **Example 6.** Consider $\varphi^{\mathsf{b}\cdots\mathsf{b}}(y_1, \ldots, y_k) = R_1(x, y_1) \wedge R_2(x, y_2) \wedge \ldots R_k(x, y_k)$ with the optimal fractional edge cover $\mathbf{u}$, where $u_i = 1$ for $i \in \{1, \ldots, k\}$. The slack is $\alpha(\mathbf{u}) = k$, since the fractional edge cover $\hat{\mathbf{u}}$, where $\hat{u}_i = u_i / k = 1/k$ covers the only non-bound variable x.

[2] We will omit the parameter $\mathbf{u}$ from the notation of α whenever it is clear from the context.

▶ **Theorem 7.** *Let φ^η be a Boolean adorned query. Let $\mathbf{u}$ be any fractional edge cover of φ. Then, for any input database D, we can construct a data structure that answers any access request in time $O(T)$ and takes space*

$$S = O\left(|D| + \prod_{i=1}^{n} |R_i|^{u_i}/T^\alpha\right)$$

Proof. To simplify the presentation, we will assume that φ contains no constants in the body or repeated variables in the same atom, but our approach can be easily extended to cover these cases. Let $\mathcal{V}_\mathsf{b} = \{x_1, \ldots, x_k\}$ be the set of bound variables. Recall that an access request $\mathbf{a} = (a_1, \ldots, a_k)$ corresponds to the query $\varphi[a_1/x_1, \ldots, a_k/x_k]$. For every atom $R_i(\mathbf{x}_i)$ in the query, define $R_i(\mathbf{a}) = \sigma_{x_j = a_j | x_j \in \mathcal{V}_\mathsf{b} \cap \mathsf{vars}(R_i)}(R_i)$. Here, σ_ψ is a selection operator that keeps the tuples from R_i that satisfy the condition ψ. We say that an access request $\mathbf{a}$ is *valid* if for every atom, $R_i(\mathbf{a}) \neq \emptyset$.

If α is the slack for the fractional edge cover $\mathbf{u}$, define $\hat{u}_i = u_i/\alpha$. Note that $\hat{\mathbf{u}} = \{\hat{u}_i\}_i$ is a fractional edge cover for the query $\varphi[a_1/x_1, \ldots, a_k/x_k]$: indeed, it is sufficient to cover only the non-bound variables, since all bound variables are replaced by constants in the query. Hence, using a worst-case optimal join algorithm, we can compute the access request $\varphi[a_1/x_1, \ldots, a_k/x_k]$ with running time

$$T(\mathbf{a}) = \prod_{i=1}^{n} |R_i(\mathbf{a})|^{u_i/\alpha}.$$

The time required is a direct application of the upper bound of worst-case optimal join algorithm running time as described in Section 2. We can now describe the data structure we build. First, for every atom $R_i(\mathbf{x}_i)$ we build a standard hash index that returns in constant time the subset $R_i(\mathbf{a})$ for every access request $\mathbf{a}$. Second, we create a hash index $\mathcal{K}$. Let J be the set of valid access requests such that $T(\mathbf{a}) > T$. For every $\mathbf{a} \in J$, we add to the hash index the key-value entry $(\mathbf{a}, \varphi[a_1/x_1, \ldots, a_k/x_k](D))$. In other words, the value is the (boolean) answer to the access request $\mathbf{a}$.

We claim that the answer time using the above data structure is at most $O(T)$. Indeed, we first check whether $\mathbf{a}$ is valid, which we can do in constant time. If it is not valid, we simply output no. If it is valid, we probe the hash index $\mathcal{K}$. If $\mathbf{a}$ exists in the hash index, we obtain the answer in time $O(1)$ by reading the value of the corresponding entry. Otherwise, we know that $T(\mathbf{a}) < T$ and hence we can compute the answer to the access request in time $O(T)$ using a worst-case optimal join.

It remains to bound the size of the data structure we constructed during the preprocessing phase. Since the size is $O(|D| + |J|)$, we will bound the size of J. Indeed, we have:

$$\begin{aligned}
T \cdot |J| \leq \sum_{\mathbf{a} \in J} T(\mathbf{a}) &= \sum_{\mathbf{a} \in J} \prod_i |R_i(\mathbf{a})|^{u_i/\alpha} \\
&= \sum_{\mathbf{a} \in J} 1^{1-1/\alpha} \cdot \left(\prod_i |R_i(\mathbf{a})|^{u_i}\right)^{1/\alpha} \\
&\leq \left(\sum_{\mathbf{a} \in J} 1\right)^{1-1/\alpha} \cdot \left(\sum_{\mathbf{a} \in J} \prod_i |R_i(\mathbf{a})|^{u_i}\right)^{1/\alpha} \\
&\leq |J|^{1-1/\alpha} \cdot \prod_i |R_i|^{u_i/\alpha}
\end{aligned}$$

The first inequality follows directly from the definition of the set J. The second inequality is Hölders inequality. The third inequality is an application of the query decomposition lemma from [21]. By rearranging the terms, we can now obtain the desired bound. ◀

We should note that Theorem 7 applies even when the relation sizes are different; this gives us sharper upper bounds compared to the case where each relation is bounded by the total size of the input. Indeed, if using $|D|$ as an upper bound on each relation, we obtain a space requirement of $O(|D|^{\rho^*}/T^\alpha)$ for achieving answering time $O(T)$, where ρ^* is the fractional edge cover number. Since $\alpha \geq 1$, this gives us at worst a linear tradeoff between space and time, i.e., $S \cdot T = O(|D|^{\rho^*})$. For cases where $\alpha \geq 1$, we can obtain better tradeoffs.

▶ **Example 8.** Continue the example in this section $\varphi^{\mathsf{b}\cdots\mathsf{b}}(y_1, \ldots, y_k) = R_1(x, y_1) \wedge R_2(x, y_2) \wedge \cdots \wedge R_k(x, y_k)$. We obtain an improved tradeoff: $S \cdot T^k = O(|D|^k)$. Note that this result matches the best-known space-time tradeoff for the k-Set Disjointness problem [11]. (Note that all atoms use the same relation symbol R, so $|R_i| = |D|$ for every $i = 1, \ldots, k$.)

▶ **Example 9** (Edge Triangles Detection)**.** For the Boolean version, it was shown in [11] that – conditioned on the strong set disjointness conjecture – any data structure that achieves answering time T needs space $S = \Omega(|E|^2/T^2)$. A matching upper bound can be constructed by using a fractional edge cover $\mathbf{u} = (1, 1, 0)$ with slack $\alpha = 2$. Thus, Theorem 7 can be applied to achieve answering time T using space $S = O(|E|^2/T^2)$. Careful inspection reveals that a different fractional edge cover $\mathbf{u} = (1/2, 1/2, 1/2)$ with slack $\alpha = 1$, achieves a better tradeoff. Thus, Theorem 7 can be applied to obtain the following corollary.

▶ **Corollary 10.** *For a graph $G = (V, E)$, there exists a data structure of size $S = O(|E|^{3/2}/T)$ that can answer the edge triangles detection problem in $O(T)$ time.*

The data structure implied by Theorem 7 is always better when $T \leq \sqrt{|E|}$ [3], thus refuting the conditional lower bound in [11]. We should note that this does not imply that the strong set disjointness conjecture is false, as we have observed an error in the reduction used by [11].

▶ **Example 11** (Square Detection)**.** Beyond triangles, we consider the edge square detection problem, which checks whether a given edge belongs in a square pattern in a graph $G = (V, E)$, $\varphi^{\mathsf{bb}}_{\square}(x_1, x_2) = R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge R_3(x_3, x_4) \wedge R_4(x_4, x_1)$. Using the fractional edge cover $\mathbf{u} = (1/2, 1/2, 1/2, 1/2)$ with slack $\alpha = 1$, we obtain a tradeoff $S = O(|E|^2/T)$.

5 Space-Time Tradeoffs via Tree Decompositions

Theorem 7 does not always give us the optimal tradeoff. For the k-reachability problem with the adorned query $\varphi^{\mathsf{bb}}(x_1, x_{k+1}) = R_1(x_1, x_2) \wedge \cdots \wedge R_k(x_k, x_{k+1})$, Theorem 7 gives a tradeoff $S \cdot T = |D|^{\lceil (k+1)/2 \rceil}$, by taking the optimal fractional edge covering number $\rho^* = \lceil (k+1)/2 \rceil$ and slack $\alpha = 1$, which is far from efficient. In this section, we will show how to leverage tree decompositions to further improve the space-time tradeoff in Theorem 7.

Again, let φ^η be an adorned query. Given a set of nodes $C \subseteq \mathcal{V}$, a C-connex tree decomposition of φ is a pair $(\mathcal{T}, A)$, where (i) $\mathcal{T}$ is a tree decomposition of φ, and (ii) A is a connected subset of the tree nodes such that the union of their variables is exactly C. For

[3] All answering times $T > \sqrt{|E|}$ are trivial to achieve using linear space by using the data structure for $T' = \sqrt{E}$ and holding the result back until time T has passed. However, in certain practical settings such as transmitting data structure over a network, it is beneficial to construct a sublinear in size data structure. In those settings, $T > \sqrt{|E|}$ is useful.

Figure 1 Two tree decompositions for the length-5 path query: the left is unconstrained, while the right is a C-connex decomposition with $C = \{x_1, x_6\}$. The bound variables are colored red. The nodes in A are colored grey.

our purposes, we choose $C = \mathcal{V}_\mathsf{b}$. Given a $\mathcal{V}_\mathsf{b}$-connex tree decomposition, we orient the tree from some node in A. We then define the bound variables for the bag t, $\mathcal{V}_\mathsf{b}^t$ as the variables in $\mathcal{B}_t$ that also appear in the bag of some ancestor of t. The free variables for the bag t are the remaining variables in the bag, $\mathcal{V}_\mathsf{f}^t = \mathcal{B}_t \setminus \mathcal{V}_\mathsf{b}^t$.

▶ **Example 12.** Consider the 5-path query $\varphi^{\mathsf{bb}}(x_1, x_6) = R_1(x_1, x_2) \wedge \cdots \wedge R_5(x_5, x_6)$. Here, x_1 and x_6 are the bound variables. Figure 1 shows the unconstrained decomposition as well as the C-connex decomposition for $\varphi^{\mathsf{bb}}(x_1, x_6)$, where $C = \{x_1, x_6\}$. The root bag contains the bound variables x_1, x_6. Bag $\mathcal{B}_{t_2}$ contains x_1, x_6 as bound variables and x_2, x_5 as the free variables. Bag $\mathcal{B}_{t_3}$ contains x_2, x_5 as bound variables for $\mathcal{B}_{t_3}$ and x_3, x_4 as free variables.

Next, we define a parameterized notion of width for the $\mathcal{V}_\mathsf{b}$-connex tree decomposition. The width is parameterized by a function δ that maps each node t in the tree to a non-negative number, such that $\delta(t) = 0$ whenever $t \in A$. The intuition here is that we will spend $O(|D|^{\delta(t)})$ in the node t while answering the access request. The parameterized width of a bag $\mathcal{B}_t$ is now defined as: $\rho_t(\delta) = \min_{\mathbf{u}} \left(\sum_F u_F - \delta(t) \cdot \alpha\right)$ where $\mathbf{u}$ is a fractional edge cover of the bag $\mathcal{B}_t$, and α is the slack (on the bound variables of the bag). The δ-width of the decomposition is then defined as $\max_{t \notin A} \rho_t(\delta)$. Finally, we define the δ-height as the maximum-weight path from the root to any leaf, where the weight of a path P is $\sum_{t \in P} \delta(t)$. We now have all the necessary machinery to state our second main theorem.

▶ **Theorem 13.** *Let φ^η be a Boolean adorned query. Consider any $\mathcal{V}_\mathsf{b}$-connex tree decomposition of φ. For some parametrization δ of the decomposition, let f be its δ-width, and h be its δ-height. Then, for any input database D, we can construct a data structure that answers any access request in time $T = O(|D|^h)$ with space $S = O(|D| + |D|^f)$.*

The function δ allows us to trade off between time and space. If we set $\delta(t) = 0$ for every node t in the tree, then the δ-height becomes $O(1)$, while the δ-width equals to the fractional hypetree width of the decomposition. As we increase the values of δ in each bag, the δ-height increases while the δ-width decreases, i.e., the answer time T increases while the space decreases. Additionally, we note that the tradeoff from Theorem 13 is at least as good as the one from Theorem 7. Indeed, we can always construct a tree decomposition where all variables reside in a single node of the tree. In this case, we recover exactly the tradeoff from Theorem 7. Due to a lack of space, we refer the reader to Appendix A for details.

▶ **Example 14.** We continue with the 5-path query. Since $\mathcal{B}_{t_1} = \{x_1, x_6\} \in A$, we assign $\delta(t_1) = 0$. For $\mathcal{B}_{t_2} = \{x_1, x_2, x_5, x_6\}$, the only valid fractional edge cover assigns weight 1

to both R_1, R_5 and has slack 1. Hence, if we assign $\delta(t_2) = \tau$ for some parameter τ, the width is $2 - \tau$. For $\mathcal{B}_{t_3} = \{x_2, x_3, x_4, x_5\}$, the only fractional cover also assigns weight 1 to both R_2, R_4, with slack 1 again. Assigning $\delta(t_3) = \tau$, the width becomes $2 - \tau$ for t_3 as well. Hence, the δ-width of the tree decomposition is $2 - \tau$, while the δ-height is 2τ. Plugging this to Theorem 13, it gives us a tradeoff with answering time $T = O(|E|^{2\tau})$ and space usage $S = O(|E| + |E|^{2-\tau})$, which matches the state-of-the-art result in [11].

For the k-reachability problem, a general tradeoff $S \times T^{2/(k-1)} = O(|D|^2)$ was also shown by [11] using a careful recursive argument. The data structure generated using Theorem 13 is able to recover the tradeoff. In particular, we obtain the answering time as $T = O(|E|^{(k-1)\tau/2})$ using space $S = O(|E| + |E|^{2-\tau})$.

▶ **Example 15.** Consider a variant of the square detection problem: given two vertices, the goal is to decide whether they occur in two opposites corners of a square, which can be captured by the following adorned query:

$$\varphi^{\mathsf{bb}}(x_1, x_3) = R_1(x_1, x_2) \wedge R_1(x_2, x_3) \wedge R_3(x_3, x_4) \wedge R_4(x_4, x_1).$$

Theorem 7 gives a tradeoff with answering time $O(T)$ and space $O(|E|^2/T)$. But we can obtain a better tradeoff using Theorem 13. Indeed, consider the tree decomposition where we have a root bag t_1 with $\mathcal{B}_{t_1} = \{x_1, x_3\}$, and two children of t_1 with Boolean $\mathcal{B}_{t_2} = \{x_1, x_2, x_3\}$ and $\mathcal{B}_{t_3} = \{x_1, x_3, x_4\}$. For $\mathcal{B}_{t_2}$, we can see that if assigning a weight of 1 to both hyperedges, we get a slack of 2. Hence, if $\delta(t_2) = \tau$, the δ-width is $2 - 2\tau$. Similarly for t_3, we assign $\delta(t_3) = \tau$, for a δ-width with $2 - 2\tau$. Applying Theorem 13, we obtain a tradeoff with time $T = O(\tau)$ (since both root-leaf paths have only one node), and space $S = O(|E| + |E|^{2-2\tau})$. So the space usage can be improved from $O(|E|^2/T)$ to $O(|E|^2/T^2)$.

6 CQs with Negation

In this section, we present a simple but powerful extension of our result to adorned Boolean CQs with negation. A CQ with negation, denoted as $CQ^\neg$, is a CQ where some of the atoms can be negative, i.e., $\neg R_i(\mathbf{x}_i)$ is allowed. For $\varphi \in CQ^\neg$, we denote by φ^+ the conjunction of the positive atoms in φ and φ^- the conjunction of all negated atoms. A $CQ^\neg$ is said to be *safe* if every variable appears in at least some positive atom. In this paper, we restrict our scope to class of safe $CQ^\neg$, a standard assumption [26, 19] ensuring that query results are well-defined and do not depend on domains.

Given a query $\varphi \in CQ^\neg$, we build the data structure from Theorem 13 for φ^+ but impose two constraints on the decomposition: (i) no leaf node(s) contains any free variables, (ii) for every negated atom R^-, all variables of R^- must appear together as bound variables in some leaf node(s). In other words, there exists a leaf node such that $\mathsf{vars}(R^-)$ is present in it. It is easy to see that such a decomposition always exists. Indeed, we can fix the root bag to be $C = \mathcal{V}_{\mathsf{b}}$, its child bag with free variables as $\mathsf{vars}(\varphi^+) \setminus C$ and bound variables as C, and the leaf bag, which is connected to the child of the root, with bound variables as $\mathsf{vars}(\varphi^-)$ without free variables. Observe that the bag containing $\mathsf{vars}(\varphi^+)$ free variables can be covered by only using the positive atoms since φ is safe. The intuition is the following: during the query answering phase, we wish to find the join result over all variables $\mathcal{V}_{\mathsf{f}}$ before reaching the leaf nodes; and then, we can check whether there the tuples satisfy the negated atoms or not, in $O(1)$ time. The next example shows the application of the algorithm to adorned path queries containing negation.

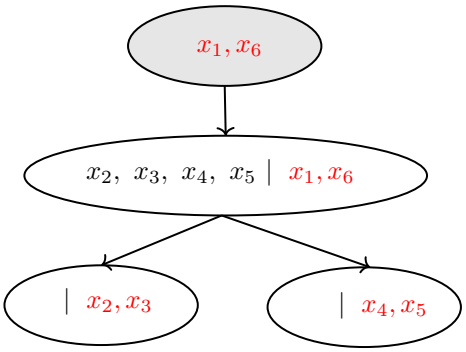

Figure 2 C-connex decomposition for Example 16.

▶ **Example 16.** Consider the query $Q^{\mathsf{bb}}(x_1, x_6) = R(x_1, x_2) \wedge \neg S(x_2, x_3) \wedge T(x_3, x_4) \wedge \neg U(x_4, x_5) \wedge V(x_5, x_6)$. Using the decomposition in Figure 2, we can now apply Theorem 13 to obtain the tradeoff $S = O(|D|^3/\tau)$ and $T = O(\tau)$. Both leaf nodes only require linear space since a single atom covers the variables. Given an access request, we check whether the answer for this request has been materialized or not. If not, we proceed to the query answering phase and find at most $O(\tau)$ answers after evaluating the join in the middle bag. For each of these answers, we can now check in constant time whether the tuples formed by values for x_2, x_3 and x_4, x_5 are not present in relations S and U respectively.

For adorned queries where $\mathcal{V}_{\mathsf{b}} \subseteq \mathsf{vars}(\varphi^-)$, we can further simplify the algorithm. In this case, we no longer need to create a constrained decomposition since the check to see if the negated relations are satisfied or not can be done in constant time at the root bag itself. Thus, we can directly build the data structure from Theorem 13 using the query φ^+.

▶ **Example 17** (Open Triangle Detection)**.** Consider the query $\varphi^{\mathsf{bb}}(x_2, x_3) = R_1(x_1, x_2) \wedge \neg R_2(x_2, x_3) \wedge R_3(x_1, x_3)$, where φ^- is $\neg R_2(x_2, x_3)$ and φ^+ is $R_1(x_1, x_2) \wedge R_3(x_1, x_3)$ with the adorned view as $\varphi^{+\mathsf{bb}}(x_2, x_3) = R_1(x_1, x_2) \wedge R_3(x_1, x_3)$. Observe that $\{x_2, x_3\} \subseteq \mathsf{vars}(\varphi^-)$. We apply Theorem 13 to obtain the tradeoff $S = O(|E|^2/\tau^2)$ and $T = O(\tau)$ with root bag $C = \{x_2, x_3\}$, its child bag with $\mathcal{V}_{\mathsf{b}} = C$ and $\mathcal{V}_{\mathsf{f}} = \{x_1\}$, and the leaf bag to be $\mathcal{V}_{\mathsf{b}} = C$ and $\mathcal{V}_{\mathsf{f}} = \emptyset$. Given an access request (a, b), we check whether the answer for this request has been materialized or not. If not, we traverse the decomposition and evaluating the join to find if there exists a connecting value for x_1. For the last bag, we simply check whether (a, b) exists in R_2 or not in $O(1)$ time.

7 Related Work

The study of fine-grained space/time tradeoffs for query answering is a relatively recent effort in the algorithmic community. The study of distance oracles over graphs was first initiated by [23] where lower bounds are shown on the size of a distance oracle for sparse graphs based on a conjecture about the best possible data structure for a set intersection problem. [9] also considered the problem of set intersection and presented a data structure that can answer boolean set intersection queries which is conditionally optimal [11]. There also exist another line of work that looks at the problem of approximate distance oracles. Agarwal et al. [4, 3] showed that for stretch-2 and stretch-3 oracles, we can achieve $S \times T = O(|D|^2)$ and $S \times T^2 = O(|D|^2)$. They also showed that for any integer k, a stretch-$(1+1/k)$ oracle exhibits $S \times T^{1/k} = O(|D|^2)$ tradeoff. Unfortunately, no lower bounds are known for non-constant

query time. The authors in [11] conjectured that the tradeoff $S \times T^{2/(k-1)} = O(|D|^2)$ for k-reachability is optimal which would also imply that stretch-$(1 + 1/k)$ oracle tradeoff is also optimal. A different line of work has considered the problem of enumerating query results of a non-boolean query. [9] presented a data structure to enumerate the intersection of two sets with guarantees on the total answering time. This result was generalized to incorporate *full* adorned views over CQs [10]. Our work extends the results to the setting where the join variables are projected away from the query result (i.e. the adorned views are *non-full*) and makes the connection between several different algorithmic problems that have been studied independently. Further, we also consider boolean CQs that may contain negations. In the non-static setting, [6] initiated the study of answering conjunctive query results under updates. More recently, [15] presented an algorithm for counting the number of triangles under updates. There have also been some exciting developments in the space of enumerating query results with delay for a proper subset of CQs known as *hierarchical queries*. [16] presented a tradeoff between preprocessing time and delay for enumerating the results of any (not necessarily full) hierarchical queries under static and dynamic settings. It remains an interesting problem to find improved algorithms for more restricted set of CQs such as hierarchical queries.

8 Conclusion

In this paper, we investigate the tradeoffs between answering time and space required by a data structure to answer boolean queries. Our main contribution is a unified algorithm that recovers the best known results for several boolean queries of practical interest. There are several questions that remain open. We describe two problems that are particularly engaging.

Unconditional lower bounds. It remains an open problem to prove unconditional lower bounds on the space requirement for answering boolean star and path queries in the RAM model. For instance, 2-Set Disjointness can be answered in constant time by materializing all answers using $\Theta(|D|^2)$ space but there is no lower bound to rule out if this can be achieved using sub-quadratic space.

Improved approximate distance oracles. It would be interesting to investigate whether our ideas can be applied to existing algorithms for constructing distance oracles to improve their space requirement. [11] conjectured that the k-reachability tradeoff is optimal and used it to prove the conditional optimality of distance oracles. We believe our framework can be used to improve upon the bounds for k-reachability in conjunction with other techniques used to prove bounds for join query processing in the database theory community.

References

1. Mahmoud Abo Khamis, Phokion G Kolaitis, Hung Q Ngo, and Dan Suciu. Decision problems in information theory. In *ICALP*, 2020.
2. Peyman Afshani and Jesper Asbjørn Sindahl Nielsen. Data structure lower bounds for document indexing problems. In *ICALP*, 2016.
3. Rachit Agarwal. The space-stretch-time tradeoff in distance oracles. In *ESA*, pages 49–60. Springer, 2014.
4. Rachit Agarwal, P Brighten Godfrey, and Sariel Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM*, pages 1754–1762. IEEE, 2011.
5. Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

6 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *PODS*, pages 303–318. ACM, 2017.

7 Timothy M Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *STOC*, pages 31–40, 2015.

8 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theoretical Computer Science*, 411(40-42):3795–3800, 2010.

9 Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. *arXiv preprint arXiv:1006.1117*, 2010.

10 Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *PODS*, pages 307–322. ACM, 2018.

11 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *WADS*, pages 421–436. Springer, 2017.

12 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Treewidth and hypertree width. *Tractability: Practical Approaches to Hard Problems*, 1, 2014.

13 Gianluigi Greco and Francesco Scarcello. Structural tractability of enumerating csp solutions. *Constraints*, 18(1):38–74, 2013.

14 John E Hopcroft, Jeffrey D Ullman, and AV Aho. The design and analysis of computer algorithms, 1975.

15 Ahmet Kara, Hung Q Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting triangles under updates in worst-case optimal time. In *ICDT*, 2019.

16 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *PODS*, pages 375–392, 2020.

17 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. In *ESA*, pages 625–636. Springer, 2013.

18 Kasper Green Larsen, J Ian Munro, Jesper Sindahl Nielsen, and Sharma V Thankachan. On hardness of several string indexing problems. *Theoretical Computer Science*, 582:74–82, 2015.

19 Alan Nash and Bertram Ludäscher. Processing unions of conjunctive queries with negation under limited access patterns. In *EDBT*, pages 422–440. Springer, 2004.

20 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *PODS*, pages 37–48. ACM, 2012.

21 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2013. URL: http://doi.acm.org/10.1145/2590989.2590991, doi:10.1145/2590989.2590991.

22 Dan Olteanu and Maximilian Schleich. Factorized databases. *ACM SIGMOD Record*, 45(2):5–16, 2016.

23 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *FOCS*, pages 815–823. IEEE, 2010.

24 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

25 Jeffrey D Ullman. An approach to processing queries in a logic-based query language. In *On knowledge base management systems*, pages 147–164. Springer, 1986.

26 Fang Wei and Georg Lausen. Containment of conjunctive queries with safe negeuration. In *ICDT*, pages 346–360. Springer, 2003.

27 Konstantinos Xirogiannopoulos and Amol Deshpande. Extracting and analyzing hidden graphs from relational databases. In *SIGMOD*, pages 897–912. ACM, 2017.

28 Konstantinos Xirogiannopoulos, Udayan Khurana, and Amol Deshpande. Graphgen: Exploring interesting graphs in relational data. *Proceedings of the VLDB Endowment*, 8(12):2032–2035, 2015.

29 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

A Missing Proofs

Proof of Theorem 13. We introduce some new notation. Let $\mathcal{T} = (\mathcal{T}, A)$ denote the $\mathcal{V}_\mathsf{b}$-connex tree decomposition with f as its δ-width, and h as its δ-height. For each node $t \in \mathcal{T} \setminus A$, we denote by $\mathsf{anc}(t)$ the union of all the bags for the nodes that are the ancestors of t, $\mathcal{V}_\mathsf{b}^t = \mathcal{B}_t \cap \mathsf{anc}(t)$ and $\mathcal{V}_\mathsf{f}^t = \mathcal{B}_t \setminus \mathcal{V}_\mathsf{b}^t$. Intuitively, $\mathcal{V}_\mathsf{b}^t(\mathcal{V}_\mathsf{f}^t)$ are the bound (resp. free) variables for the bag t as we traverse the tree top-down. For example, in the decomposition on the right hand side of Figure 1, $A = \mathcal{B}_{t_1}$, $\mathsf{anc}(t_2) = \{x_1, x_6\}$, $\mathcal{V}_\mathsf{b}^t = \{x_1, x_6\}$ and $\mathcal{V}_\mathsf{f}^t = \{x_1, x_5\}$. For node t_3, $\mathsf{anc}(t_3) = \{x_2, x_5, x_1, x_6\}$, $\mathcal{V}_\mathsf{b}^t = \{x_2, x_5\}$ and $\mathcal{V}_\mathsf{f}^t = \{x_3, x_4\}$

Data Structure Construction

We apply Theorem 7 to each bag (except the root bag) in $\mathcal{T}$ with the following parameters: (i) fractional edge cover $\mathbf{u}$ corresponding to bag $\mathcal{B}_t$; (ii) adorned query φ_t^η corresponding to bag t where the query is the join of all atoms that contain at least one variable present in the bag; and (iii) bound variables in φ_t^η are $\mathcal{V}_\mathsf{b}^t = \mathsf{anc}(t) \cap \mathcal{B}_t$. The space requirement for the data structure corresponding to bag $\mathcal{B}_t$ is $S = O(|D| + |D|^{\rho_t(\delta)}) \leq O(|D| + |D|^f)$. This follows directly from the definition of $\delta(t)$ width of bag t which is assumed to be at most f. Recall that the data structure stores a list of all access requests $\mathbf{a}$ defined over the schema $\mathcal{V}_\mathsf{b}^t$ such that answering time $T > O(|D|^f)$. Let us call this stored list as $\mathcal{L}(t)$. For each entry in this list, we will store a yes or no answer to the question whether there exists a valuation for all variables in the subtree rooted at $\mathcal{B}_t$ for a given valuation of bound variables $\mathcal{V}_\mathsf{b}^t$.

Query Answering

We now describe the query answering algorithm. Let $C = \{x_1, \ldots, x_k\}$ and access request $\mathbf{a} = (a_1, \ldots, a_k)$. We first need to check whether $\mathbf{a}$ is valid. If the request is not valid, we can simply output no. This can be done in constant time after creating hash indexes of size $O(|D|)$ during the preprocessing phase. If the access request is valid, the second step is to check whether $Q(\mathbf{a})$ is true or false. Let $\mathcal{P}$ denote the set of bags that are children of root bag. Then, for each bag $\mathcal{B}_t \in \mathcal{P}$, we check whether $\pi_{\mathcal{V}_\mathsf{b}^t}(\mathbf{a}) \in \mathcal{L}(t)$. If it is stored, it means that that running time of $\pi_{\mathcal{V}_\mathsf{b}^t}(\mathbf{a})$ is greater than $O(|D|^{\delta(t)})$. If the entry for $\pi_{\mathcal{V}_\mathsf{b}^t}(\mathbf{a})$ is false in the data structure, we can return false[4] since we know that no output tuple can be formed by the subtree rooted at bag $\mathcal{B}_t$. Similarly, if the entry for $\pi_{\mathcal{V}_\mathsf{b}^t}(\mathbf{a})$ is true, we can immediately return true since we know that there is at least one valid tuple that can be formed by the subtree rooted at bag $\mathcal{B}_t$.

If there is no entry for $\pi_{\mathcal{V}_\mathsf{b}^t}(\mathbf{a})$ in $\mathcal{L}(t)$, this means that answering time of evaluating the join at node t is $T \leq O(|D|^{\delta(t)})$. Thus, we can evaluate the join for the bag by fixing $\mathcal{V}_\mathsf{b}^t$ as $\pi_{\mathcal{V}_\mathsf{b}^t}(\mathbf{a})$ using any worst-case optimal join algorithm, which guarantees that the total running time is at most $O(|D|^{\delta(t)})$. If no output is generated, the algorithm outputs false since no output tuple can be formed by subtree rooted at $\mathcal{B}_t$. If there is output generated, then there can be at most $O(|D|^{\delta(t)})$ tuples at bag $\mathcal{B}_t$. For each of these tuples, we recursively proceed to the children of bag $\mathcal{B}_t$ and repeat the algorithm. Each fixing of variables at bag t acts as the bound variables for the children bag. In the worst case, all bags in $\mathcal{T}$ may require join processing. Since the query size is a constant, it implies that the number of root to leaf paths are also constant. Thus, the answering time is dominated by the

[4] we return to the caller of the function since the query answering algorithm is recursive.

root to leaf path with the largest δ_t sum, i.e the δ-height of the decomposition. Thus, $T = O(|D|^{\sum_{t \in P} \delta(t)}) = O(|D|^h)$. ◀