Abstract

In this paper, we investigate the space-time tradeoffs for answering boolean queries. The task is to create a data structure in an initial preprocessing phase which is then used to solve certain algorithmic tasks. Previous work has proved conditional space lower bounds on data structures for queries of practical interests such as triangle query, star query and path query. This hardness is shown as a tradeoff between the space consumed by the data structure and the time needed to answer queries. Our main contribution is three fold: (i) we propose a unified algorithm that recovers smooth space-time tradeoffs for multiple queries. The algorithm is based on a novel application of Holder’s inequality to bound the space usage of our data structure. (ii) we show that conditional space lower bounds for path queries is false by proposing an algorithm that answers queries faster using the same amount space. (iii) we prove conditional space lower bounds for star queries.

1 Introduction

Recently, there is an intensive research work aimed at understanding the complexity within the class P (decision problems that are solved by polynomial time algorithms). A key question of great interest to the algorithmic community is the study of tight space-time tradeoffs for certain algorithmic tasks and conditional lower bounds proving the optimality of the tradeoffs. Recent groundbreaking work in the community has made remarkable progress in developing data structures and algorithms for answering set disjointness and intersection problems [7], reachability oracles and directed reachability [3, 2, 6], histogram indexing [5, 11] and problem related to document retrieval [1, 12]. A popular framework to study the tradeoffs is to split querying task into two phases: the preprocessing phase, which computes a space-efficient intermediate data structure, and the answering phase, which uses the data structure to answer the query results as fast as possible, with the goal of minimizing the answering time. This distinction is beneficial for several reasons. For instance, in many scenarios, the user wants to see one (or a few) results of the query as fast as possible: in this case, we want to minimize the time of the preprocessing phase, such that we can output the first results quickly. On the other hand, the algorithmic primitives are used in a data processing pipeline in practice that usually involves repeated querying of the same task multiple times: in this case, it is better to spend more time during the preprocessing phase, to guarantee faster answering time.

A class of queries known as boolean queries are of special interest because of their widespread applicability in several practical data analytics tasks and graph exploration tasks [18, 17]. Boolean queries output either true/false as the query answer. For example, 2-Set Disjointness is a popular boolean query where given a collection of m sets $S_1, \ldots, S_m \subset U$
and a pair of integers $1 \leq i, j \leq m$, the goal is to answer whether $S_i \cap S_j$ is empty or not. Previous work [6, 7] has showed that the tradeoff between space used by the data structure and the answering time for any query. The data structure obtained is conjectured to be optimal [7] and its optimality was used to develop conditional lower bounds for several problems such as approximate distance oracles [3, 2]. Using the data structure for 2-Set Disjointness, a sophisticated data structure for $k$-Reachability problem was also constructed by [7]. In this problem, we are given as an input a directed sparse graph $G = (V, E)$ for preprocessing. Given two vertices $u, v$, the goal is to answer whether there exists a path of length $k$ between $u$ and $v$. For every $k \geq 1$, the tradeoff between space $S$ and answering time $T$ admits the tradeoff $S \cdot T^{2/(k-1)} = O(|E|^2)$. This tradeoff was also conjectured to be optimal by [7]. The authors also studied several other interesting queries over triangles (such as detecting whether an edge participates in a triangle or not) and provided conditional lower bounds assuming the optimality of 2-Set Disjointness data structure.

**Our Contribution.** In this paper, we continue the investigation between the preprocessing space requirement and answering time over boolean queries. Our main insight is the observation that all of the problems mentioned above can be cast as query answering problem over databases. This observation allows for a unified treatment to develop algorithms within a single framework. In order to develop a unified framework, we leverage the formalism of join query evaluation over static databases by the data management community that has a long line of research on the design and analysis of algorithms that minimize the total runtime of query execution in terms of the input and output size [19, 15, 13]. More specifically, we use the formalism of conjunctive queries to introduce the novel notion of boolean adorned queries where the values of some variables in the query is fixed by the user (denoted by an access pattern), and the goal is to evaluate the boolean query. This formulation is general enough to capture all of the problems mentioned above allowing us to develop a unified algorithm that recovers all existing results known for 2-Set Disjointness (and its generalization $k$-Set Disjointness), $k$-Reachability and edge triangle detection where each problem becomes an instantiation of the introduced framework. Casting these problems as a database join query evaluation task, we can are able to leverage the powerful tool of worst-case optimal joins [14]. We summarize our main technical contribution below.

1. Our first main result is a unified algorithm that captures all known algorithms for problems mentioned above as boolean conjunctive queries. Using the concept of adorned queries, we present a novel data structure that creates a compressed representation based on an access pattern specified by the user. More importantly, it can be tuned to tradeoff space for answering time, thus exploring the full continuum between optimal space and optimal runtime. At the one extreme, the data structure achieves constant answering time; At the other extreme it uses linear space, but provides provable worst guarantees.

2. Directly applying the first main result may sometimes lead to suboptimal tradeoffs since it does not take into account structural properties of the query. Our second main result combines the data structure with a special type of tree decompositions that takes into account the query structure and the access pattern.

3. Using our main result, we improve upon the space-time tradeoff for all $k$-Reachability queries where $k \geq 4$, thus proving that the data structure from [7] is suboptimal. This is the first non-trivial improvement for the $k$-reachability problem.

4. Finally, we present two lower bounds for optimality of $k$-Set Disjointness using a recursive argument by assuming the optimality of $\ell$-Set Disjointness where $\ell > k$. We also refute the conjecture for edge triangle detection from [7] for some answering time values.
Organization. We introduce the basic terminology and problem setting in Section 2 and Todo: add. Section 4 presents our main results for full adorned boolean queries. Section 5 presents our improved result for path queries. We discuss the lower bounds and related work in Section 6 and 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.

2.1 Conjunctive Queries

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

\[ Q(y) = R_1(x_1), R_2(x_2), \ldots, R_n(x_n). \]

Here, the symbols \( y, x_1, \ldots, x_n \) are vectors that contain variables or constants. Each \( R_i \) is a relation name, and the expressions \( Q(y), R_1(x_1), R_2(x_2), \ldots, R_n(x_n) \) are called atoms. The atom \( Q(y) \) is the head of the query, while the atoms \( R_i(x_i) \) from the body. The variables in the head are a subset of the variables that appear in the body. A CQ is full if every variable in the body appears also in the head, and it is boolean if the head contains no variables, i.e., it is of the form \( Q() \). We will typically use the symbols \( x, y, z, \ldots \) to denote variables, and \( a, b, c, \ldots \) to denote constants. Given an input database \( D \), we use \( Q(D) \) to denote the result of the query \( Q \) over the database.

In this paper, we will consider CQs that have no constants and no repeated variables in the same atom. Such a query can be represented equivalently as a hypergraph \( \mathcal{H}_Q = (\mathcal{V}_Q, \mathcal{E}_Q) \), where \( \mathcal{V}_Q \) is the set of variables, and for each hyperedge \( F \in \mathcal{E}_Q \) there exists a relation \( R_F \) with variables \( F \). We will use \( |R_F| \) to denote the number of tuples in relation \( R_F \).

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:

\[ P_k(x_1, x_{k+1}) = R(x_1, x_2), R(x_2, x_3), \ldots, R(x_k, x_{k+1}). \]

Join Size Bounds. Let \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \) be a hypergraph, and \( S \subseteq \mathcal{V} \). A weight assignment \( u = (u_F)_{F \in \mathcal{E}} \) is called a fractional edge cover of \( S \) if \((i)\) for every \( F \in \mathcal{E}, u_F \geq 0 \) and \((ii)\) for every \( x \in S, \sum_{F \in \mathcal{E}} u_F \geq 1 \). The fractional edge cover number of \( S \), denoted by \( \rho^*_\mathcal{H}(S) \) is the minimum of \( \sum_{F \in \mathcal{E}} u_F \) over all fractional edge covers of \( S \). We write \( \rho^*(\mathcal{H}) = \rho^*_\mathcal{H}(\mathcal{V}) \).

Tree Decompositions. Let \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \) be a hypergraph of a CQ \( Q \). A tree decomposition of \( \mathcal{H} \) is a tuple \((\mathcal{T}, (B_t)_{t \in \mathcal{V}(\mathcal{T})})\) where \( \mathcal{T} \) is a tree, and every \( B_t \) is a subset of \( \mathcal{V} \), called the bag of \( t \), such that

1. each edge in \( \mathcal{E} \) is contained in some bag; and
2. for each variable \( x \in \mathcal{V} \), the set of nodes \( \{ t \mid x \in B_t \} \) form a connected subtree of \( \mathcal{T} \).

The fractional hypertree width of a decomposition is defined as \( \max_{x \in \mathcal{V}(\mathcal{T})} \rho^*(B_x) \), where \( \rho^*(B_t) \) is the minimum fractional edge cover of the vertices in \( B_t \). The fractional hypertree width of a query \( Q \), denoted \( \text{fhw}(Q) \), is the minimum fractional hypertree width among all tree decompositions of its hypergraph. We say that a query is acyclic if \( \text{fhw}(Q) = 1 \).
**Computational Model.** To measure the running time of our algorithms, we will use the uniform-cost RAM model [8], where data values as well as 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 over a query result, we will use the concept of **adorned queries**. In an adorned query, each variable in the head of the query is associated with a binding type, which can be either **bound** (b) or **free** (f). We denote this as $Q^\eta(x_1, \ldots, x_k)$, where $\eta \in \{b,f\}^k$ 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 instantiation of the bound variables to constants, $x_1 \leftarrow a_1, \ldots, x_\ell \leftarrow a_\ell$ is called an access request: in this case, we need to return the result of the query where we have replaced each bound variable $x_i$ with the corresponding constant $a_i$.

In the next few examples, we demonstrate how several data structure problems can be captured using the concept of adorned queries.

**Example 2 (Set Disjointness and Set Intersection).** In the set disjointness problem, we are given $m$ sets $S_1, \ldots, S_m$ over the same universe $U$. Let $N = \sum_{i=1}^m |S_i|$ be the total size of all 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:

$$Q^{bb}(y,z) = R(x,y), R(x,z).$$

An access request in this case specifies two sets $y = S_i, z = S_j$, and computes the (boolean) query $Q() = R(x,S_i), R(x,S_j)$.

In the related set intersection problem, given a pair of indexes $(i,j), 1 \leq i,j \leq m$, we instead want to enumerate the elements in the intersection $S_i \cap S_j$. This is captured through the following adorned query:

$$Q^{bdf}(y,z,x) = R(x,y), R(x,z).$$

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

$$Q^{bk}(y_1, \ldots, y_k) = R(x,y_1), \ldots, R(x,y_k).$$

**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 $S(x,y)$ (which means that there is an edge from $u$ to $v$), we can capture this problem through the following adorned query:

$$Q^{bk}(x_1, x_{k+1}) = S(x_1, x_2), S(x_2, x_3), \ldots, S(x_k, x_{k+1}).$$

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 whether, given an edge $(u, v)$ as the request, $(u, v)$ participates in a triangle or not. This task can be expressed as the adorned query

$$Q_{\triangle}^{bb}(x, z) = R(x, y), R(y, z), R(x, z).$$

In the reporting variant of the problem, the goal is to also enumerate all triangles in which $(u, v)$ participates. In our framework, this can be expressed using the adorned query $Q_{\triangle}^{bf}(x, z, y) = R(x, y), R(y, z), R(x, z)$.

We say that an adorned query is boolean if every head variable is bound. In this case, the answer for every access request is boolean, i.e., yes or no. Both the $k$-set disjointness problem and the $k$-reachability problem are captured by boolean adorned queries.

3.2 Problem Statement

Given an adorned query $Q^n$ 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 $\eta$ as fast as possible. In other words, an algorithm can be split into two phases:

- **Preprocessing phase**: it computes a data structure that needs space $S$. We will often need to calculate the time to construct the data structure, denoted $P$.
- **Answering phase**: given an access request, we compute the answer using the data structure. We denote by $T$ the time needed to compute any access request.

In this work, our goal is to study the relationship between the space of the data structure $S$ and the answering time $\delta$ for a given adorned query $Q^n$. We will focus on boolean adorned queries, where the output is a single bit (yes/no).

4 General Space-Time Tradeoffs

4.1 Space-Time Tradeoffs via Worst-case Optimal Algorithms

Let $Q^n$ be an adorned query, and let $\mathcal{H} = (V, E)$ be the corresponding hypergraph. Let $V_b$ denote the bound variables in the head of the query. For any fractional edge cover $u$ of $V$, we define the slack of $u$ as:

$$\alpha := \min_{x \in V \setminus V_b} \left( \sum_{F: x \in F} u_F \right).$$

In other words, the slack is the maximum quantity by which we can reduce the fractional cover $u$ so that it remains a fractional edge cover of the non-bound variables in the query. Hence $\{u_F/\alpha\}_{F \in E}$ fractionally covers the nodes in $V \setminus V_b$. Note that we always have $\alpha(S) \geq 1$.

Example 6. Consider $Q_{b \ldots b}(y_1, \ldots, y_k) = R_1(x, y_1), R_2(x, y_2), \ldots R_k(x, y_k)$. We construct the fractional edge cover where $u_i = 1$ for every $i = 1, \ldots, k$. Observe that the slack in this case is $\alpha = k$, since the fractional edge cover where $\hat{u}_i = u_i / k = 1/k$ still covers the only non-bound variable, which is $x$.

We can now state our first main theorem.

Theorem 7. Let $Q^n$ be a boolean adorned query with hypergraph $(V, E)$. Let $u$ be any fractional edge cover of $V$. Then, for any input database $D$, we can construct a data structure that answers any access request in time $O(T)$ and takes space

$$S = O\left( |D| + \prod_{F \in E} |R_F|^{u_F/T \alpha} \right).$$
Proof. Let \( V_b = \{x_1, \ldots, x_k\} \). Recall that an access request \( a = (a_1, \ldots, a_k) \) corresponds to the query \( Q[\sigma_{x_1 = a_1, \ldots, x_k = a_k}/x_1, \ldots, x_k] \); in other words, we substitute each occurrence of a bound variable \( x_i \) with the constant \( a_i \). Define the hypergraph \( \mathcal{H}_b = (V_b, \mathcal{E}_b) \), where \( \mathcal{E}_b = \{F \cap V_b \mid F \in \mathcal{E}\} \). We say that an access request \( a \) is valid if it is an answer for the query \( Q_b \) corresponding to \( \mathcal{H}_b \), i.e. \( a \in Q_b(D) \). We can construct hash indexes of linear size \( O(|D|) \) during the preprocessing phase so that we can check whether any access request is valid in constant time \( O(1) \).

For every relation \( R_F \) in the query, let \( R_F(a) = \sigma_{x_1 = a_1, \ldots, x_k} R_F \). In other words, \( R_F(a) \) is the subrelation that we obtain once we filter out the tuples that satisfy the selection condition implied by the access request.

If \( \alpha \) is the slack for the fractional edge cover \( u \), define \( \hat{u}_F = u_F/\alpha \) for every \( F \in \mathcal{E} \). As we have discussed earlier, \( \hat{u} = \{\hat{u}_F\}_{F \in \mathcal{E}} \) is a fractional edge cover for the query \( Q[a_1/x_1, \ldots, a_k/x_k] \); indeed, it is necessary 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 \( Q[a_1/x_1, \ldots, a_k/x_k] \) with running time

\[
T(a) = \prod_{F \in \mathcal{E}} |R_F(a)|^{u_F/\alpha}.
\]

We can now describe the preprocessing phase and the data structure we build. The data structure simply creates a hash index. Let \( J \) be the set of valid access requests such that \( T(a) > T \). For every \( a \in J \), we add to the hash index a key-value entry, where the key is \( a \) and the value the (boolean) answer to the access request \( Q[a_1/x_1, \ldots, a_k/x_k] \).

We claim that the answer time using the above data structure is at most \( O(T) \). Indeed, we first check whether \( a \) 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. If \( a \) 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 \( T(a) < T \) and hence we can compute the answer to the access request in time \( O(T) \).

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

\[
T \cdot |J| \leq \sum_{a \in J} T(a) = \sum_{a \in J} \prod_{F \in \mathcal{E}} |R_F(a)|^{u_F/\alpha} \\
= \sum_{a \in J} 1^{1-1/\alpha} \cdot \left( \prod_{F \in \mathcal{E}} |R_F(a)|^{u_F} \right)^{1/\alpha} \\
\leq \left( \sum_{a \in J} 1 \right)^{1-1/\alpha} \cdot \left( \sum_{a \in J} \prod_{F \in \mathcal{E}} |R_F(a)|^{u_F} \right)^{1/\alpha} \\
\leq |J|^{1-1/\alpha} \cdot \prod_{F \in \mathcal{E}} |R_F|^{u_F/\alpha}
\]

Here, 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 [15]. By rearranging the terms, we can now obtain the desired bound. 

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

Example 8. Continuing our running example for this section, for $Q_{bb}(y_1, \ldots, y_k) = R_1(x, y_1), R_2(x, y_2), \ldots, R_k(x, y_k)$ we obtain the following improved tradeoff: $S \cdot T^k = O(|D|^k)$. We should note here that this result matches the best-known space-time tradeoff for the $k$-set disjointness problem [7]. (Note that in $k$-set disjointness, all atoms use the same relation symbol $R$, so $|R_i| = |D|$ for every $i = 1, \ldots, k$.)

We next present a few more applications of Theorem 7.

Edge Triangles Detection. For the boolean version of the problem, it was shown in [7] that – conditioned on the strong set disjointness conjecture – any data structure that achieves answer time $T$ needs space $S = \Omega(|E|^2/T^2)$.

A matching upper bound can be constructed by setting a fractional edge cover of $u(R, S, T) = (1, 1, 0)$. The slack in this case is $\alpha = 2$. Thus, Theorem 7 can be applied to achieve query time $T$ using space $S = O(|E|^2/T^2)$. However, there exists a different fractional edge cover than can achieve a better space-time tradeoff. Observe that $u(R, S, T) = (1/2, 1/2, 1/2)$ is also a valid fractional edge cover with slack $\alpha = 1$. Thus, Theorem 7 can be applied to obtain the following corollary.

Corollary 9. Given a graph $G = (V, E)$, there exists a data structure that achieves answer time $O(T)$ for the edge triangles detection problem and takes space $S = O(|E|^{3/2}/T)$.

The data structure from Theorem 7 is always better when $T \leq O(\sqrt{|E|})$. Hence, the conditional lower bound from [7] is not correct. We should note here that this does not imply that the strong set disjointness conjecture is false, since we have observed an error in the reduction used in [7].

Square Detection. We can consider other graph patterns in addition to triangles. For example, the adorned query below checks whether a given edge belongs in a square pattern in a graph $G = (V, E)$.

$$Q_{bb}^b(x_1, x_2) = R(x_1, x_2), R(x_2, x_3), R(x_3, x_4), R(x_4, x_1).$$

Considering the fractional edge cover that assigns a weight of 1/2 to each hyperedge, with slack $\alpha = 1$, we obtain a tradeoff with time $O(T)$ and space $S = O(|E|^2/T)$.

4.2 Space-Time Tradeoffs via Tree Decompositions

Consider the adorned query corresponding to the $k$-reachability problem, $Q_{bb}(x_1, x_{k+1}) = \delta_1(x_1, x_2), \ldots, \delta_k(x_k, x_{k+1})$. For this query, the best fractional edge cover will be of size $\lceil(k+1)/2\rceil$, and $\alpha = 1$. Applying Theorem 7, this gives us a tradeoff $S \cdot T = |D|^{\lceil(k+1)/2\rceil}$, which is far from optimal. In this section, we will show how we can leverage tree decompositions to further improve general space-time tradeoffs.

Again, let $Q^b$ be an adorned query, and let $H = (V, E)$ be the corresponding hypergraph. Let $C \subseteq V$. A $C$-connex tree decomposition of $H$ is a pair $(T, A)$, where (i) $T$ is a tree

---

1 All answering times $T > \sqrt{|E|}$ are trivial to achieve using linear space.
(a) Query decomposition for 5-path query. (b) C-connex query decomposition with \( C = \{x_1, x_6\} \).

**Figure 1** The hypergraph \( \mathcal{H} \) for a path query of length 5, along with two tree decompositions. The decomposition on the left is unconstrained, and the decomposition on the right \( C = \{x_1, x_6\} \). The variables in \( C \) are colored red, and the grey nodes are the ones in the set \( A \).

The variables in \( C \) are colored red, and the grey nodes are the ones in the set \( A \).

Next, we define a parameterized notion of width for the \( V_b \)-connex tree decomposition. The width is parameterized by a function \( \delta \) 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 \( B_t \) is now defined as:

\[
\rho_t(\delta) = \min_{u} \left( \sum_{F} u_F - \delta(t) \cdot \alpha \right)
\]

where \( u \) is a fractional edge cover of the bag \( B_t \), and \( \alpha \) is the slack (on the bound variables of the bag). The \( \delta \)-width of the decomposition is then defined as \( \max_{t \in A} \rho_t(\delta) \). Finally, we define the \( \delta \)-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 11.** Let \( Q^\eta \) be a boolean adorned query with hypergraph \( \mathcal{H} = (V, E) \). Consider any \( V_b \)-connex tree decomposition of \( \mathcal{H} \). For some parametrization \( \delta \) of the decomposition, let \( f \) be its \( \delta \)-width, and \( h \) be its \( \delta \)-height.
Then, for any input database $D$, we can construct a data structure that answers any access request in time $T = O(|D|^{h})$ in space $S = O(|D| + |D|^{f})$.

Proof. We first present the construction the data structure. Let $T$ denote the $V_{f}$-connex tree decomposition with $f$ as its $\delta$-width, and $h$ as its $\delta$-height. We apply Theorem 7 to each bag (except the root bag) in $T$ with the following parameters: (i) $H^{t} = (V^{t}, E^{t})$ where $V^{t} = B_{t}$ and $E^{t} = E_{B_{t}}$; (ii) $V_{f}^{t} = \text{anc}(t) \cap B_{t}$; and fractional edge cover $\mu$ corresponding to bag $B_{t}$. The space requirement for the data structure corresponding to bag $B_{t}$ is $S = O(|D| + |D|^{\rho_{f}(\delta)} \leq O(|D| + |D|^{f})$. Recall that the data structure stores a list of all access requests $a$ defined over $V_{f}^{t}$ such that $T(a) > O(|D|^{h})$. We will store the data structure only for all children of the root bag.

We now describe the query answering algorithm. Let $\mathcal{C} = \{x_{1}, \ldots, x_{k}\}$ and access request $a = (a_{1}, \ldots, a_{k})$. We first need to check whether $a$ 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 $O(|D|^{f})$ during the preprocessing phase. If the access request is valid, the second step is to check whether $Q(a)$ is true or false. Let $\mathcal{P}$ denote the set of bags that are children of root bag. Then, for each bag $B_{t} \in \mathcal{P}$, we check whether $B_{t}(a) = \pi_{B_{t} \cap \mathcal{C}}(a)$ is stored in the data structure for bag $\mathcal{B}$. If it is stored, it means that $B_{t}(a)$ is heavy. If the entry for $B_{t}(a)$ is false in the data structure, we can output no since we know that no output tuple can be formed by the subtree rooted at bag $B_{t}$.

If there is no entry for $B(a)$, this means that $T(\mathcal{B}(a)) \leq O(|D|^{\delta(t)})$. Now we can evaluate the join for the bag by fixing $V_{f}^{t}$ as $B_{t}(a)$. If no output is generated, the algorithm output false since no output tuple can be formed by subtree rooted at $B_{t}$. If there is output generated, then there can be at most $O(|D|^{\delta(t)})$ tuples. For each of these tuples, we recursively proceed to the children of bag $B_{t}$ and repeat the algorithm. In the worst case, all bags in $T$ 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 largest root to leaf path, i.e the $\delta$-height of the decomposition. Thus, $T = O(|D|^{\sum_{t \in \mathcal{P}_{\delta}(\delta)}}) = O(|D|^{h})$.

The function $\delta$ allows us to tradeoff between time and space. If we set $\delta(t) = 0$ for every node in the tree, then the $\delta$-height becomes $O(1)$, while the $\delta$-width becomes equal to the fractional hypetree width of the decomposition. As we increase the values of $\delta$ in each bag, the $\delta$-height increases so the answer time $T$ increases, while the $\delta$-width decreases and so the space decreases.

Additionally, we note that the tradeoff from Theorem 11 is always 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.

Example 12. Consider again the running example of the path query of length 5. Since the bag $\mathcal{B}_{t} = \{x_{1}, x_{6}\}$ is in $A$, we have to assign $\delta(t_{1}) = 0$. For the bag $\mathcal{B}_{r_{2}} = \{x_{1}, x_{2}, x_{5}, x_{6}\}$, the only valid fractional edge cover assigns a weight of 1 to both $R_{1}, R_{3}$ and has slack 1. Hence, if we assign $\delta(t_{2}) = \tau$ for some parameter $\tau$, the width of the bag is $2 - \tau$. For the bag $\mathcal{B}_{t_{3}} = \{x_{2}, x_{3}, x_{4}, x_{5}\}$, the only fractional cover also assigns a weight of 1 to both $R_{2}, R_{4}$, with slack 1 again. Assigning $\delta(t_{3}) = \tau$, the width becomes $2 - 2\tau$ for $t_{3}$ as well.

Hence, the $\delta$-width of the tree decomposition is $2 - \tau$, while the $\delta$-height is $2\tau$. Plugging this in Theorem 11 gives us a tradeoff with time $T = O(|E|^{2\tau})$ and space $S = O(|E| + |E|^{2-\tau})$, which matches the state-of-the-art tradeoff from [7].
In fact, we can generalize the above argument to show that we can recover the state-of-the-art tradeoff from [7] for any length $k$: time $T = O(|E|^{(k-1)/2})$ and space $S = O(|E|+|E|^{2-\tau})$.

**Square Detection.** As another application of the theorem, consider a variant of the square detection problem. Here, we are given two vertices and we want to check whether they occur in two opposite corners of a square. In other words,

$$Q^{bb}(x_1, x_3) = R(x_1, x_2), R(x_2, x_3), R(x_3, x_4), R(x_4, x_1).$$

Applying Theorem 7 gives a tradeoff with time $O(T)$ and space $O(|E|^2/T)$. But we can obtain a better tradeoff using Theorem 11.

Indeed, consider the tree decomposition where we have a root bag $t_1$ with $B_{t_1} = \{x_1, x_3\}$, and two children of $t_1$ with bags $B_{t_2} = \{x_1, x_2, x_3\}$ and $B_{t_3} = \{x_1, x_3, x_4\}$. For bag $t_2$, we can see that if we assign a weight of 1 to both hyperedges, we get a slack of 2. Hence, if $\delta(t_2) = \tau$, the $\delta$-width is $2 - 2\tau$. Similarly for $t_3$, we assign $\delta(t_3) = \tau$, for a $\delta$-width with $2 - 2\tau$. Applying Theorem 11, 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 improves from $O(|E|^2/T)$ to $O(|E|^2/T^2)$.

## 5 Path Queries

In this section, we will present an algorithm for the adorned query $P^{bb}_k(x_1, x_{k+1}) = R_1(x_1, x_2), \ldots, R_k(x_k, x_{k+1})$ that improves upon the conjectured optimal solution. Before we present the improved algorithm, we first state the upper bound on the tradeoff between space and query time.

- **Theorem 13 (due to [7]).** There exists a data structure for solving $P^{bb}_k(x_1, x_{k+1})$ with space $S$ and answering time $T$ such that $S \cdot T^{2/(k-1)} = O(D^2)$.

Note that for $k = 2$, the problem is equivalent to SetDisjointness with the space/time tradeoff as $S \cdot T^2 = O(N^2)$. [7] also conjectured that the tradeoff is essentially optimal.

- **Conjecture 14.** Any data structure for $P^{bb}_k(x_1, x_{k+1})$ with answering time $T$ must use space $S = \Omega(|D|^2/T^{2/(k-1)})$.

When $k$ is a non-constant, Conjecture 14 tells us that we need $\Theta(|D|^2)$ space to answer queries in constant time. Using Conjecture 14, [7] also showed a result that shows the optimality of approximate distance oracles.

Our main result in this section is to show that Theorem 13 can be improved upon and that Conjecture 14 is false.

The first observation is that the tradeoff in Theorem 13 is only applicable when $T \leq |D|$. Indeed, we can always answer any boolean path query in linear time using breadth-first search. Surprisingly, it is also possible to improve Theorem 13 for the regime of small answering time as well. In what follows, we will show the improvement for paths of length 4; we will generalize the algorithm for any length in the next section.

- **Lemma 15.** There exists a data structure for solving $P^{bb}_4(x_1, x_5)$ with space $S$ and answering time $T \leq \sqrt{|D|}$ such that $S \cdot T = O(|D|^2)$.

For $k = 4$, Theorem 13 gives us the tradeoff $S \cdot T^{2/3} = O(|D|^2)$ which is always worse than the tradeoff in Lemma 15. We next present our algorithm in detail.
**Preprocessing Phase.** Consider $F_{bb}^b(x_1, x_2) = R(x_1, x_2), S(x_2, x_3), T(x_3, x_4), U(x_4, x_5)$. Let $\Delta$ be a degree threshold. We say that a constant $a$ is heavy if its frequency on attribute $x_3$ is greater than $\Delta$ in both relations $R$ and $T$. In other words, $a$ is heavy if $|\sigma_{x_3=a}(S)| > \Delta$ and $|\sigma_{x_3=a}(T)| > \Delta$. We distinguish two cases based on whether a constant for $x_3$ is heavy or light. Let $L_{\text{heavy}}(x_3)$ denote the unary relation that contains all heavy values, and $L_{\text{light}}(x_3)$ the one that contains all light values. Observe that we can compute both of these relations in time $O(|D|)$. We compute the following two views:

$$V_1(x_1, x_3) = R(x_1, x_2), S(x_2, x_3), L_{\text{heavy}}(x_3).$$

$$V_2(x_3, x_5) = L_{\text{heavy}}(x_3), T(x_3, x_4), U(x_4, x_5).$$

We view the stores as a hash index that, given a value of $x_1$ (or $x_3$), it returns all matching values of $x_3$. Both views require space $O(|D|^2/\Delta)$. Indeed, notice that $|L_{\text{heavy}}| \leq |D|/\Delta$. Since we can construct a fractional edge cover for $V_1$ by assigning a weight of 1 to $R, L_{\text{heavy}},$ this gives us an upper bound of $|D| \cdot (|D|/\Delta)$ for the query output. The same argument holds for $V_2$.

We also compute the following view for the light values:

$$V_3(x_2, x_4) = S(x_2, x_3), L_{\text{light}}(x_3), T(x_3, x_4).$$

This view requires space $O(|D| \cdot \Delta)$, since the degree of the light constants is at most $\Delta$. We can now rewrite the original query as

$$F_{bb}^b(x_1, x_5) = R(x_1, x_2), V_3(x_2, x_4), U(x_4, x_5).$$

The rewritten query is a three path query. Hence, we can apply Theorem 7 to create a data structure with answering time $T = O(|D|/\Delta)$ and space $S = O(|D|^2/|D|/\Delta) = O(|D| \cdot \Delta)$. **Query Answering.** Given an access request, we first check whether there exists a 4-path that goes through some heavy value in $L_{\text{heavy}}(x_3)$. This can be done in time $O(|D|/\Delta)$ using the views $V_1$ and $V_2$. Indeed, we obtain at most $O(|D|/\Delta)$ values for $x_3$ using the index for $V_1$, and $O(|D|/\Delta)$ values for $x_3$ using the index for $V_3$. We then intersect the results in time $O(|D|/\Delta)$. If we find no such 4-path, we check for a 4-path that uses a light value for $x_3$. From the data structure we have constructed in the preprocessing phase, we can do this in time $O(|D|/\Delta)$. **Tradeoff Analysis.** From the above, we can compute the answer in time $T = O(|D|/\Delta)$. From the analysis in the preprocessing phase, the space needed is $S = O(|D|^2/\Delta + |D| \cdot \Delta)$. Thus, whenever $\Delta \geq \sqrt{|D|}$, the space becomes $S = O(|D| \cdot \Delta)$, completing our analysis.

### 5.1 General Path Queries

We can now use Theorem 13 to improve the space/time tradeoff for all path queries of length greater than four as well. Then general idea is to use the four path query as a base case and create a recursive data structure.

Let $S(i,j)$ denote the space requirement of path subquery $F_{bb}^b(x_1, x_j) = R_i(x_i, x_{i+1}) \times \cdots \times R_{j-1}(x_{j-1}, x_j) \times T(i,j)$ be its answering. Our goal is to find $S(1, k+1)$ and $T(1, k+1)$.

Similar to the idea from previous section, we set $\sqrt{|N/\Delta|}$ as the degree threshold for any valuation $x_{k+1}$ and $x_1$. For all heavy $x_1$ and $x_{k+1}$ pairs, we maintain a matrix of size $O(\sqrt{|D| \cdot \Delta} \cdot \sqrt{|N| \cdot \Delta}) = O(|D| \cdot \Delta)$. We now recursively construct the data structure for
Theorem 13
Theorem 16
BFS

Figure 2 Space/time tradeoff for path query of length $k$. The curve in blue shows the tradeoff obtained from Theorem 13. The highlighted portion in brown shows the improved tradeoff using BFS. The red curve is the new tradeoff obtained using Theorem 16. The green portion of the original curve is still the best possible when Theorem 16 is not applicable.

the case when either $x_1$ or $x_{k+1}$ is light. Without loss of generality, let us assume that $x_{k+1}$ is light. For each neighbor valuation $x_k$ of $x_{k+1}$, we call the data structure for subquery $P^{bb}_{(1,k)}$ that uses space $S_{(1,k)}$. Since we are only concerned with light valuations, the answering time $\sqrt{|D|/\Delta} \cdot T_{(1,k)}$.

This gives us the following recurrence equations,

\[
S_{(1,k+1)} = O(|D| \cdot \Delta) + S_{(1,k)} + S_{(2,k+1)}
\]

\[
T_{(1,k+1)} = \sqrt{|D|/\Delta} \cdot T_{(1,k)} + \sqrt{|N|/\Delta} \cdot T_{(2,k+1)} + O(\sqrt{|D|/\Delta})
\]

The recursion terminates when the path subquery becomes of length four and we can use the data structure from Lemma 15. This gives us $S_{(1,k+1)} = O(|D|/\Delta)^{(k-2)/2}$.

\textbf{Theorem 16.} There exists a data structure for solving $P^{bb}_{(x_1,x_{k+1})}$ with space $S = O(|D|^2/\Delta + |D| \cdot \Delta)$ and answering time $T = O((|D|/\Delta)^{(k-2)/2})$ for all $k \geq 4$.

6 Lower Bounds

6.1 Star Queries

In this section, we present conditional lower bounds for $k$-Set Disjointness problem using conditional optimality of $\ell$-Set Disjointness where $\ell < k$. First, we review the known results from [7] starting with the conjecture for $k$-Set Disjointness.

\textbf{Conjecture 17.} Any data structure for $k$-Set Disjointness problem that answers queries in time $T$, must use space $S = \Omega(|D|^k/T^k)$.

Conjecture 17 was shown to be conditionally optimal using the conjectured lower bound for another problem called $(k+1)$-Sum Indexing problem. The lower bound for $(k+1)$-Sum Indexing problem was subsequently showed to be false which implies that Conjecture 17 is still an open problem. Conjecture 17 can be further generalized to the case when input relations are of unequal sizes as follows.

\textbf{Conjecture 18.} Any data structure for $k$-Set Disjointness problem that answers queries in time $T$, must use space $S = \Omega(\Pi_{i=1}^{|R_i|}|R_i|^k/T^k)$. 
We now state the main result for star queries.

**Theorem 19.** Suppose that any data structure for $k$-Set Disjointness problem that answers queries in time $T$, uses space $S = \Omega((\Pi_{i=1}^{k} |R_i|/T^k))$. Then, for $\ell$-Set Disjointness problem where $2 \leq \ell < k$, it must be the case that any data structure that answers queries in time $T$, uses space $S = \Omega((\Pi_{i=1}^{\ell} |R_i|/T^\ell))$.

**Proof.** Let $\Delta = T$ be the degree threshold for the $k$ bound variables $y_1, \ldots, y_k$ in $Q^{bb}(y_1, \ldots, y_k)$. If any of the $k$ variables is light, then we can check whether the intersection between $k$ sets is empty or not in time $O(T)$ by indexing all relations in a linear time preprocessing phase. The remaining case is when all $k$ variables are heavy. We now create $\ell$ views $V_1, \ldots, V_\ell$ by arbitrarily partitioning the $k$ relations into the $\ell$ views followed by materializing the join of all relations in each view. Let view $V_i$ contains the join of $k_i$ relations. Then, $|V_i| = O((\Pi_{R \in J_i} |R|/T^{k_i-1}))$ where $J_i$ is the set of all relations assigned to view $V_i$.

We have now reduced the $k$-Set Disjointness problem where all $k$ variables are heavy into an instance of $\ell$-Set Disjointness problem where the input relations are $V_1, \ldots, V_\ell$. Suppose that there exists a data structure for $\ell$-Set Disjointness that can answer queries in time $T$ using space $S = o((\Pi_{i=1}^{\ell} |R_i|/T^\ell))$. Then, we can use such a data structure for answering the original query where all variables are heavy. The space used by this oracle is

$$S = o((\Pi_{i=1}^{\ell} |V_i|/T^\ell)) = o((\Pi_{i=1}^{\ell} |R_i|/T^{k_i-1}) \cdot (1/T^\ell))$$

which contradicts the space lower bound for $k$-Set Disjointness.

**Theorem 19** creates a hierarchy for $k$-Set Disjointness problem where the optimality of smaller set disjointness instances depends on larger set disjointness instances. More importantly, **Theorem 19** does not require any assumption regarding the optimality of other problems such as $(k + 1)$-Sum Indexing.

### 6.2 Path Queries

In this section, we show conditional lower bounds on the space requirement of path queries. We begin by proving a simple result for optimality of $P_2^{bb}$ (which is equivalent to 2-Set Disjointness) assuming the optimality of $P_3^{bb}$ query.

**Theorem 20.** Suppose that any data structure for $P_3^{bb}$ that answers queries in time $T$, uses space $S$ such that $S \cdot T = \Omega(|D|^2)$. Then, for $P_2^{bb}$, it must be the case that any data structure that uses space $S = \Theta(|D|^2/T^2)$, the answering time is $\Omega(T)$.

**Proof.** Let $\Delta = T$ be the degree threshold for all vertices. If both bound variables valuations in $P_3^{bb}$ are heavy, then we can answer the query in constant time using space $\Theta(|D|^2/T^2)$ by materializing the answers to all heavy-heavy queries. In the remaining cases, at least one of the bound valuations is light. Without loss of generality, suppose $x_1$ is light. Then, we can make $\Delta$ calls to the oracle for query $P_2^{bb}(x_2, x_4) = R_2(x_2, x_3), R_3(x_3, x_4)$.

Suppose that there exists a data structure with space usage $\Theta(|D|^2/T^2)$ for $P_2^{bb}(x_2, x_4)$ and answering time $o(T)$. Then, we can answer $P_3^{bb}$ with light $x_1$ in time $o(T^2)$. This improves the tradeoff for $P_3^{bb}$ since the product of space usage and answering time is $o(|D|^2)$ for any non-constant $T$, a contradiction.

Using a similar argument, it can be shown that the conditional optimality of **Theorem 16** for $k = 4$ implies that $S \cdot T = \Omega(|D|^2)$ tradeoff for $P_3^{bb}$ is also optimal.
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 [16] 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. [6] also considered the problem of set intersection and presented a data structure that can answer boolean set intersection queries which is conditionally optimal [7]. There also exist another line work that looks at the problem of approximate distance oracles. Agarwal et al. [3, 2] 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. [7] used Theorem 13 to conjecture that the tradeoff \( S \times T^{2/(k-1)} = O(|D|^2) \) is optimal which would also imply that stretch-(1+1/k) oracle tradeoff is also optimal. Our results show that Theorem 13 can be further improved which means that the optimality of stretch-(1+1/k) oracle remains an open question.

There has also been some work on the enumeration of non-boolean query results. [6] presented a data structure to enumerate the intersection of two sets with guarantees on the total answering time. [4] initiated the study of answering conjunctive query results under updates. More recently, [9] presented an algorithm for counting the number of triangles under updates. [10] presented a tradeoff between preprocessing time and delay for enumerating the results of any (not necessarily full) hierarchical queries under static and dynamic settings. Enumerating the results of arbitrary conjunctive queries under static and dynamic settings still remains an open problem.

8 Conclusion and Open Problems

In this paper, we investigated the tradeoffs between answering time and space required by the 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 interests. We then apply our main result to improve upon the state-of-the-art algorithms to answer boolean queries over the four path query which is subsequently used to improve the tradeoff for all path queries of length greater than four. Finally, we show conditional lower bounds on star queries and path queries that does not require any additional assumptions. There are several questions that remain open. We describe the 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.

More general space-time bounds. The first question is to study the tradeoff between space vs answering time (and delay guarantees) for arbitrary non-boolean hierarchical queries and path queries. Using some of our techniques, it may be possible to smartly materialize a certain subset of joins that could be used to achieve answering time guarantees by interleaving with join computation.
References


