Space-Time Tradeoffs for Answering Boolean Conjunctive Queries

Shaleen Deep
Department of Computer Sciences, University of Wisconsin-Madison, Madison, Wisconsin, USA
shaleen@cs.wisc.edu

Xiao Hu
Department of Computer Sciences, Duke University, Durham, North Carolina, USA
xh102@cs.duke.edu

Paraschos Koutris
Department of Computer Sciences, University of Wisconsin-Madison, Madison, Wisconsin, USA
paris@cs.wisc.edu

Abstract
In this paper, we investigate space-time tradeoffs for answering boolean queries. The goal is to create a data structure in an initial preprocessing phase, which is then used to solve an algorithmic task. Previous work has developed data structures that tradeoff space usage for answering time and has proved conditional space lower bounds for several queries of practical interest. Our main insight is to exploit the formalism of relation algebra by casting the problems as answering join queries over a relational database. Using the notion of boolean adorned queries and access patterns, we propose a unified framework that captures several widely studied algorithmic problems. Our main contribution is four-fold. First, we present an algorithm that recovers existing space-time tradeoffs for several problems. The algorithm is based on a novel application of join size bounds to bound the space usage of our data structure. Second, we combine our data structure with the notion of query decompositions to further improve the tradeoffs. We show that our algorithm is also applicable to queries that contain negation. Third, we show that the conjectured space-time lower bound for path queries is false by proposing an algorithm that improves the state-of-the-art. Finally, we prove new conditional space-time lower bounds for star and path queries.

1 Introduction
Recent work has made remarkable progress in developing data structures and algorithms for answering set intersection problems [10], reachability oracles and directed reachability [4, 3, 8], histogram indexing [7, 16], 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 such problems is the study of tradeoffs between the space $S$ necessary for the data structure and the answering time $T$. For example, consider the 2-Set Disjointness problem: given a collection of $m$ sets $S_1, \ldots, S_m \subseteq U$, we want to create a data structure such that for any pair of integers $1 \leq i, j \leq m$, we answer whether $S_i \cap S_j$ is empty or not. Previous work [8, 10] has shown that the space-time tradeoff for 2-Set Disjointness is captured by $S \cdot T^2 = N^2$, where $N$ is the total size of all sets. The data structure obtained is conjectured to be optimal [10], 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 several other data structure problems.

In this paper, we observe that we can cast many of the above problems as answering Conjunctive Queries (CQs) over a relational database. CQs are a powerful class of relational queries with widespread applicability in data analytics and graph exploration [29, 28, 9].

For example, if $R(x, y)$ encodes the fact that element $x$ belongs to set $y$, 2-Set Disjointness
can be captured by the CQ \( q(y_1, y_2) = R(x, y_1), R(x, y_2) \). As another example, consider the
\( k\)-Reachability problem [10], where we are given as an input a directed 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 \( v \). As we will see later in the paper, \( k\)-Reachability can also be
naturally captured by a CQ.

The insight of casting data structure problems as CQs over a database allows for a
unified treatment for developing algorithms within a single framework. In particular, we can
leverage sophisticated tools and techniques that have been developed by the data management
community through a long line of research on efficient join algorithms [30, 22, 20]. These
techniques include worst-case optimal joins [21] and tree decompositions [11, 25]. The use of
these techniques has been a subject of previous work [1, 12, 9, 23] for enumerating query
results under static and dynamic settings. In this paper, we build upon the aforementioned
techniques to develop a unified framework that allows us to obtain general space-time tradeoffs
for any boolean CQ (a boolean CQ is one that outputs only true/false). As a consequence,
we recover the state-of-the-art tradeoffs for several data structure problems (e.g., 2-Set
Disjointness and its generalization \( k\)-Set Disjointness and \( k\)-Reachability) as special cases of
the general tradeoff. For some problems, such as the edge triangles detection problem, we
even obtain improved tradeoffs. Naturally, our space-time tradeoffs also apply to novel data
structure problems.

**Our Contribution.** We summarize our main technical contributions below.

1. We propose a unified framework that captures several widely studied data structure
problems. More specifically, we use the formalism of Conjunctive Queries and use the
notion of boolean adorned queries, where the values of some variables in the query are
fixed by the user (denoted as an access pattern), and the goal is to evaluate the boolean
query. We then show how several problems, such as 2-Set Disjointness and \( k\)-Reachability,
are instantiations of this framework.

2. Our first main result (Theorem 7) is an algorithm that constructs 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 optimal answering time. At the one extreme, the data
structure achieves constant answering time by explicitly storing all possible answers.
At the other extreme, the data structure stores nothing, and we execute each request
from scratch. We show how we can recover existing and new tradeoffs using this general
framework. As a consequence, we also refute a lower bound conjecture for the edge
triangles detection problem from [10].

3. Directly applying 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 11) combines the data structure with a particular type of tree decompositions
that takes into account the query structure and the access pattern to improve space
efficiency. We then show that the algorithm is also applicable to boolean CQ with
negation.

4. In addition to our two main results, we explicitly improve upon the best-known space-time
tradeoff for the \( k\)-Reachability problem when \( k \geq 4 \). For every \( k \geq 2 \), this tradeoff is
\( S \cdot T^{2/(k-1)} = O(|E|^2) \), where \( |E| \) is the number of edges in the graph. This tradeoff was
conjectured to be optimal by [10], and used to conditionally prove other lower bounds on
space-time tradeoffs. We show that for some regime of answer time \( T \), the tradeoff can
be improved to \( S \cdot T^{2/(k-2)} = O(|E|^2) \), thus making the conjecture false. This is the first
non-trivial improvement for \( k\)-Reachability.
5. Finally, we show a reduction between lower bounds for the problem of $k$-Set Disjointness for different values of $k$. The $k$-Set Disjointness problem generalizes 2-Set Disjointness to computing the intersection between $k$ given sets (instead of just 2).

2 Preliminaries

In this section we present the basic notation and terminology.

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 $H_Q = (V_Q, E_Q)$, where $V_Q$ is the set of variables, and for each hyperedge $F \in E_Q$ there exists a relation $R_F$ with variables $F$. We will use $|R_F|$ to denote the number of tuples in relation $R_F$ and $|D|$ to represent the size of the database (to be used interchangeably with $|E|$ representing the total number of edges across all relations in the database).

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

A CQ with negation ($CQ^\neg$) is a CQ where some of the atoms can be negative, i.e., $\neg R_i(x_i)$ is allowed. For $Q \in CQ^\neg$, we denote by $Q^+$ the conjunction of the positive atoms in $Q$. A $CQ^\neg$ is said to be safe if every variable of the query appears in at least some positive atom. In this paper, we only consider safe $CQ^\neg$, a standard assumption [27, 19] to ensure that query result is well-defined and not domain dependent.

Join Size Bounds. Let $H = (V, E)$ be a hypergraph, and $S \subseteq V$. A weight assignment $u = (u_F)_{F \in E}$ is called a fractional edge cover of $S$ if (i) for every $F \in E$, $u_F \geq 0$ and (ii) for every $x \in S$, $\sum_{F \ni x, u_F \geq 1}$. The fractional edge cover number of $S$, denoted by $\rho^*(S)$ is the minimum of $\sum_{F \in E} u_F$ over all fractional edge covers of $S$. We write $\rho^*(H) = \rho^*_H(V)$. In a celebrated result, Atserias, Grohe and Marx [5] proved that for every fractional edge cover $u$ of $V$, the size of a natural join is bounded using the following inequality, known as the AGM inequality:

$$|\mathcal{N}_{F \in E} R_F| \leq \prod_{F \in E} |R_F|^{u_F}$$

The above bound is constructive [22, 20]: there exist worst-case algorithms that compute the join $\mathcal{N}_{F \in E} R_F$ in time $O(\prod_{F \in E} |R_F|^{u_F})$ for every fractional edge cover $u$ of $V$.

Tree Decompositions. Let $H = (V, E)$ be a hypergraph of a CQ $Q$. A tree decomposition of $H$ 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 $V$, called the bag of $t$, such that
1. each edge in $E$ is contained in some bag; and
2. for each variable $x \in V$, the set of nodes $\{t \mid x \in B_t\}$ form a connected subtree of $T$.

The fractional hypertree width of a decomposition is defined as $\max_{t \in V(T)} \rho^*(B_t)$, 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 $fhw(Q)$, is the minimum fractional hypertree width among all tree decompositions of its hypergraph. We say that a query is acyclic if $fhw(Q) = 1$.

Computational Model. To measure the running time of our algorithms, we will use the uniform-cost RAM model [13], 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. Each relation (or materialized view) $R$ over schema $\mathcal{X}$ is implemented via a data structure that stores entries $t \in R$ and requires space $O(|R|)$. This data structure can look up, insert, and delete entries in constant time. For a schema $S \subseteq \mathcal{X}$, we use an index structure that for some $t$ defined over $S$ can (i) check if $t \in \pi_S(R)$, (ii) return $|\sigma_{S=1}(R)|$ in constant time; and (iii) insert and delete index entries in constant time.

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 that was introduced by [26]. 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 a 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: $Q^{bbf}(y, z, x) = R(x, y), R(x, z)$.

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:

\[ Q_{bb}(y_1, \ldots, y_k) = R(x, y_1), \ldots, R(x, y_k). \]

**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 \( x \) to \( y \)), we can capture this problem through the following adorned query:

\[ Q_{bb}^b(x_1, x_{k+1}) = S(x_1, x_2), S(x_2, x_3), \ldots, S(x_k, x_{k+1}). \]

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

\[ Q_{\triangle_{bb}}^b(x, z) = R(x, y), R(y, z), R(x, z). \]

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

\[ Q_{\triangle_{bb}}^{df}(x, y, z) = R(x, y), R(y, z), R(x, z). \]

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.

### 3.2 Problem Statement

Given an adorned query \( Q^\eta \) 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 \( S \).
- **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 \( T \) for a given adorned query \( Q^\eta \). 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 \( Q^\eta \) 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:

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

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

---

1 We will omit the parameter \( u \) from the notation of \( \alpha \) whenever it is clear from the context.
Example 6. Consider $Q^{b_1,b_2,\ldots,b_k}(y_1,\ldots,y_k) = R_1(x,y_1), R_2(x,y_2),\ldots, R_k(x,y_k)$. We construct the fractional edge cover where $u_i = 1$ for every $i = 1,\ldots,k$. Observe that the slack in this case is $\alpha(u) = k$, since the fractional edge cover $\tilde{u}$, where $\tilde{u}_i = u_i/k = 1/k$ still covers the only non-bound variable, which is $x$.

We can now state our first main theorem.

Theorem 7. Let $Q^b$ 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

$$S = O\left(|D| + \prod_{F \in E} |R_F|^{|u| / T^\alpha}\right).$$

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 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 the slack is larger than one, we can obtain much better tradeoffs.

Example 8. Continuing our running example for this section, for $Q^{b_1,b_2,\ldots,b_k}(y_1,\ldots,y_k) = R_1(x,y_1), R_2(x,y_2),\ldots, R_k(x,y_k)$ 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 [10]. (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 [10] 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 $T \leq \sqrt{|E|}$. Hence, the conditional lower bound from [10] 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 [10].

---

2 All answering times $T > \sqrt{|E|}$ are trivial to achieve using linear space by using the data structure for $T' = \sqrt{T}$ 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 sized structures. In those settings, $T > \sqrt{|E|}$ is useful.
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

\[ Q(x_1, x_2, x_3, x_4) = R(x_1, x_2), R(x_2, x_3), R(x_3, x_4), R(x_4, x_1). \]

Considering the adornned query corresponding to the \( k \)-reachability problem, \( Q_{bb}(x_1, x_{k+1}) = S_1(x_1, x_2), \ldots, S_b(x_k, x_{k+1}). \) 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 \( ST = |D|^{(k+1)/2} \), 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 \( Q^n \) 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 decomposition of \( H \), and (ii) \( A \) is a connected subset of the tree nodes such that the union of their bags is exactly \( C \). For our purposes, we choose \( C = V_b \). Given a \( V_b \)-connex tree decomposition, we orient the tree from some node in \( A \). We then define the bound variables for the bag \( t \), \( V'_t \), as the variables in \( B_t \) 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, \( V'_t = B_t \setminus V'_b \).

\( \blacktriangleright \) **Example 10.** Consider the 5-path query \( Q_{bb}(x_1, x_6) = R_1(x_1, x_2), \ldots, R_5(x_5, x_6) \). Here, \( x_1 \) and \( x_6 \) are the bound variables. Figure 1a shows the unconstrained query decomposition for \( Q_{bb}(x_1, x_6) \) while Figure 1b shows the \( C \)-connex decomposition where \( C = A = \{x_1, x_6\} \).

The root bag contains the bound variables \( x_1 \) and \( x_6 \). Bag \( B_{t_2} \) contains \( x_1 \) and \( x_6 \) as bound variables and \( x_2, x_5 \) are the non-bound variables. \( x_2, x_5 \) then act as bound variables for \( B_{t_3} \) and \( x_3, x_4 \) are the non-bound variables.

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

![Diagram](https://via.placeholder.com/150)

(a) Query decomposition for 5-path query. (b) \( C \)-connex query decomposition with \( C = \{x_1, x_6\} \).

**Figure 1** The hypergraph \( H \) 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 is a \( C \)-connex decomposition with \( C = \{x_1, x_6\} \). The variables in \( C \) are colored red, and the grey nodes are the ones in the set \( A \).
a bag \( B_t \) is now defined as: 
\[
\rho_t(\delta) = \min_u \left( \sum_{F \in \delta} u_F - \delta(t) \cdot \alpha \right)
\]
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.

\[\textbf{Theorem 11.} \text{ Let } Q^b \text{ be a boolean adorned query with hypergraph } H = (V, E). \text{ Consider any } V_3 \text{-connex tree decomposition of } H. \text{ For some parametrization } \delta \text{ of the decomposition, let } f \text{ be its } \delta \text{-width, and } b \text{ be its } \delta \text{-height.}
\]

Then, for any input database \( D \), we can construct a data structure that answers any access request in time \( T = O(|D|^b) \) in space \( S = O(|D| + |D|^{f}) \).

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. Due to lack of space, we refer the reader to Appendix B for more optimizations that can be applied.

**Example 12.** Consider again the running example of the path query of length 5. Since the bag \( B_{t_1} = \{x_1, x_6\} \) is in \( A \), we have to assign \( \delta(t_1) = 0 \). For the bag \( B_{t_2} = \{x_1, x_2, x_5, x_6\} \), 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 \( B_{t_3} = \{x_2, x_3, x_4, x_5\} \), 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 \( 2 - \tau \) 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 [10].

**k-Reachability.** It is straightforward generalize the above argument to show that we can recover the state-of-the-art tradeoff from [10] for any length \( k \): time \( T = O(|E|^{(k-1)/2\tau}) \) 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 \( O(|E|^2/T^2) \).
4.3 Extension to CQs with Negation

In this section, we present a simple extension of our result to adorned boolean CQs with negation. Given a query $Q \in CQ^-$, we build the data structure from Theorem 11 for query $Q$ but impose the constraint on the decomposition that the leaf node(s) contains no free variables and all variables of every negated relation appear as bound variables in some leaf node(s). It is easy to see that such a decomposition always exists. Indeed, we can fix the root bag to be $C = V_0$, its child bag with free variables as $\text{vars}(Q^+) \setminus C$ and bound variables as $C$, and the leaf bag with bound variables as $\text{vars}(Q^-)$ and no free variables. Observe that the bag containing $Q^+$ free variables can be covered by only using the positive atoms since $Q$ is safe. The intuition behind this approach is the following: during the query answering phase, we wish to find the join result over all variables $V_t$ before reaching the leaf nodes. Then, we can perform checks to see whether there the tuples satisfy the negated atoms or not, a constant time operation. We show an application of this algorithm for open triangle detection (see Appendix C for more examples).

**Open Triangle Detection.** Consider the query $Q^{bb}(x_2, x_3) = R(x_1, x_2), \neg S(x_2, x_3), T(x_1, x_3)$.

Here, $Q^-$ is $\neg S(x_2, x_3)$, $Q^+$ is $R(x_1, x_2), T(x_1, x_3)$ and the corresponding adorned view is $Q^{bb}(x_2, x_3) = R(x_1, x_2), T(x_1, x_3)$. Observe that $\{x_2, x_3\} \subseteq \text{vars}(Q^-)$. We can now apply Theorem 11 to obtain the tradeoff $S = O(|E|^2/r^2)$ and $T = O(r)$ with root bag $C = \{x_2, x_3\}$, its child bag with $V_0 = C$ and $V_t = \{x_1\}$, and the leaf bag to be $V_k = C$ and $V_t = \emptyset$. Given an access request $x_2 \leftarrow a, x_3 \leftarrow b$, we can check whether the materialized access requests contains an answer for $(a, b)$. If not, then we traverse the decomposition and evaluating the join to find if there exists an answer for $x_3$. For the last bag, we simple check whether $(a, b) \notin S$ in $O(1)$ time.

5 Path Queries

In this section, we will present an algorithm for the adorned query $P^{bb}(x_1, x_{k+1}) = R_1(x_1, x_2), \ldots, R_k(x_k, x_{k+1})$ 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 [10]).** There exists a data structure for solving $P^{bb}(x_1, x_{k+1})$ 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 $S \cdot T^2 = O(N^2)$. [10] also conjectured that the tradeoff is essentially optimal.

**Conjecture 14.** Any data structure for $P^{bb}(x_1, x_{k+1})$ 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, [10] 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 useful 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.
5.1 Paths of Length 4

Lemma 15. There exists a parameterized data structure for solving $P^{bb}_{k}(x_1, x_5)$ that uses space $S$ and answering time $T \leq \sqrt{|D|}$ that satisfies the tradeoff $S \cdot T = O(|D|^2)$.

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 $P^{bb}_{4}(x_1, x_5) = R(x_1, x_2), S(x_2, x_3), T(x_3, x_4), U(x_4, x_5)$. 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 $S$ and $T$; otherwise, it is light. 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 $\mathcal{L}_{\text{heavy}}(x_3)$ denote the unary relation that contains all heavy values, and $\mathcal{L}_{\text{light}}(x_3)$ the one that contains all light values. Observe that we can compute both of these relations in time $O(|D|)$ by simply iterating over the active domain of variable $x_3$ and checking the degree in relations $S$ and $T$. We compute the following two views:

$$V_1(x_1, x_3) = R(x_1, x_2), S(x_2, x_3), \mathcal{L}_{\text{heavy}}(x_3)$$
$$V_2(x_3, x_5) = \mathcal{L}_{\text{heavy}}(x_3), T(x_3, x_4), U(x_4, x_5).$$

We store the views as a hash index that, given a value of $x_1$ (or $x_5$), returns all matching values of $x_3$. Both views take space $O(|D|^2/\Delta)$. Indeed, $|\mathcal{L}_{\text{heavy}}| \leq |D|/\Delta$. Since we can construct a fractional edge cover for $V_1$ by assigning a weight of 1 to $R$, $\mathcal{L}_{\text{heavy}}$, 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: $V_3(x_2, x_4) = S(x_2, x_3), \mathcal{L}_{\text{light}}(x_3), T(x_3, x_4)$. 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 $P^{bb}_{k}(x_1, x_5) = R(x_1, x_2), V_3(x_2, x_4), U(x_4, x_5)$.

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 $\mathcal{L}_{\text{heavy}}(x_3)$. 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 $x_5$ using the index for $V_3$. We then intersect the results in time $O(|D|/\Delta)$ by iterating over the $O(|D|/\Delta)$ values for $x_3$ and checking if the bound values for $x_1$ and $x_5$ from a tuple in $V_1$ and $V_2$ respectively. 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.

5.2 General Path Queries

We can now use the algorithm for the 4-path query to improve the space-time tradeoff for general path queries of length greater than four.

Theorem 16. Let $D$ be an input instance, and $\Delta \geq \sqrt{|D|}$. There exists a data structure for solving $P^{bb}_k(x_1, x_{k+1})$ with space $S = O(|D| \cdot \Delta)$ and answer time $T = O((|D|/\Delta)^{(k-2)/2})$ for all $k \geq 4$.

The space-time tradeoff we obtain from the above theorem is $S \cdot T^{2/(k-2)} = O(|D|^2)$, but it holds only for $T \leq |D|^{(k-2)/4}$. To compare this with the tradeoff of $S \cdot T^{2/(k-1)} = O(|D|^2)$
we obtain from Theorem 13, it is instructive to look at Figures 2a and 2b, which plot the space-time tradeoffs for \( k = 4 \) and \( k = 6 \) respectively. For \( k = 4 \), we can see that the new tradeoff is better for \( T \leq |D|^{1/2} \). Once we exceed \( |D|^{1/2} \), it is still better to use the data structure from Theorem 16 until the tradeoff from Theorem 13 becomes better. For \( k = 6 \), the switch point also happens at \( T = |D|^{1/2} \) but requires more space. In general as the path length \( k \) grows, and the tradeoff line will become flatter.

6 Lower Bounds

In this section, we study lower bounds for adorned star and path queries. We first present conditional lower bounds for the \( k \)-Set Disjointness problem using the conditional optimality of \( \ell \)-Set Disjointness where \( \ell < k \). First, we review the known results from [10] starting with the conjecture for \( k \)-Set Disjointness.

\[ \text{Conjecture 17} \] 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. The lower bound for \((k+1)\)-Sum Indexing was subsequently showed to be false [17], 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.

\[ \text{Conjecture 18.} \] Any data structure for \( Q_{b_1 \ldots b}^{y_1 \ldots y_k}(x_1, \ldots, x_\ell) = R_1(x, y_1), \ldots, R_\ell(x, y_\ell) \) that answers queries in time \( T \) must use space \( S = \Omega(\Pi_{i=1}^\ell |R_i|/T^\ell) \).

Note that when \( R_1 = \cdots = R_\ell = R \), we obtain the original conjecture. We now state the main result for star queries.

\[ \text{Theorem 19.} \] Suppose that any data structure for \( Q_{b_1 \ldots b}^{y_1 \ldots y_k}(x_1, \ldots, y_\ell) \) that answers queries in time \( T \) uses space \( S = \Omega(\Pi_{i=1}^\ell |R_i|/T^\ell) \). Then, for \( Q_{b_1 \ldots b}^{y_1 \ldots y_k}(x_1, \ldots, y_\ell) \) where \( 2 \leq \ell < k \), it must be the case that any data structure that answers queries in time \( T \), uses space \( S = \Omega(\Pi_{i=1}^\ell |R_i|/T^\ell) \).
Theorem 19 creates a hierarchy for $k$-Set Disjointness, where the optimality of smaller set disjointness instances depends on larger set disjointness instances. Next, 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 $\sigma : T = \Omega(|D|^2)$. Then, for $P_2^{bb}$, it must be the case that any data structure that uses space $S = O(|D|^2/T^2)$, the answering time is $\Omega(T)$.

Using a similar argument, it can be shown that the conditional optimality of Theorem 16 for $k = 4$ implies that $\sigma : T = \Omega(|D|^2)$ tradeoff for $P_3^{bb}$ is also optimal (but only for the range $T \leq \sqrt{|D|}$ when the result is applicable).

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 [24] 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. [8] also considered the problem of set intersection and presented a data structure that can answer boolean set intersection queries which is conditionally optimal [10]. 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. [10] 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. A different line of work has considered the problem of enumerating query results of a non-boolean query. [8] 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 (all variables in the query are either b or f) over Conjunctive Queries [9].

Our work extends the results to the setting where the join variable is 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. In the non-static setting, [6] initiated the study of answering conjunctive query results under updates. More recently, [14] 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. [15] presented a tradeoff between preprocessing time and delay for enumerating the results of any (not necessarily full) hierarchical queries under static and dynamic settings.

8 Conclusion

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 and show conditional lower bounds.
References


A Missing Proofs

Proof of Theorem 7. Let \( \mathcal{V}_b = \{ x_1, \ldots, x_k \} \). Recall that an access request \( a = (a_1, \ldots, a_k) \) corresponds to the query \( Q[a_1/x_1, \ldots, a_k/x_k] \); in other words, we substitute each occurrence of a bound variable \( x_i \) with the constant \( a_i \). Define the hypergraph \( \mathcal{H}_b = (\mathcal{V}_b, \mathcal{E}_b) \), where \( \mathcal{E}_b = \{ F \cap \mathcal{V}_b \mid F \in \mathcal{E} \} \). 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 \( R_F(a) = \sigma_{x_i=a_i,x_i\in\mathcal{V}_b\cap F}(R_F) \). 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 [22]. By rearranging the terms, we can now obtain the desired bound.

Proof of Theorem 11. We introduce some new notation. Let \( \mathcal{T} = (T,A) \) denote the \( \mathcal{V}_b \)-connex tree decomposition with \( f \) as its \( \delta \)-width, and \( h \) as its \( \delta \)-height. For each node
We now describe the query answering algorithm. Let $C = \{x_1, \ldots, x_k\}$ 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 $O(|D|)$ during the preprocessing phase. If the access request is valid, the second step is to check whether $Q(a)$ is true or false. Let $P$ denote the set of bags that are children of root bag. Then, for each bag $B_i \in P$, we check whether $\pi_{V_b}(a) \in \mathcal{L}(t)$. If it is stored, it means that $\pi_{V_b}(a)$ is heavy. If the entry for $\pi_{V_b}(a)$ is false in the data structure, we can output false immediately since we know that no output tuple can be formed by the subtree rooted at bag $B_i$.

If there is no entry for $\pi_{V_b}(a)$ in $\mathcal{L}(t)$, this means that answering time of evaluating the join at node $t$ is $T \leq O(|D|^{|E^t|})$. Thus, we can evaluate the join for the bag by fixing $V_b^t$ as $\pi_{V_b^t}a$. If no output is generated, the algorithm outputs false since no output tuple can be formed by subtree rooted at $B_i$. If there is output generated, then there can be at most $O(|D|^{|E^t|})$ tuples. For each of these tuples, we recursively proceed to the children of bag $B_i$ 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 longest root to leaf path, i.e the $\delta$-height of the decomposition. Thus, $T = O(|D|^{\sum_{t \in P} \Delta(t)}) = O(|D|^h)$.

**Proof of Theorem 16.** Fix some $\Delta \geq \sqrt{|D|}$. We construct the data structure for a path of length $k$ recursively. We use the 4-path as the base of the recursion, with answer time $T_4 = |D|/\Delta$ and space $S_4 = |D| \cdot \Delta$.

For the recursive step, similar to the idea from previous section, we set $\sqrt{|D|}/\Delta$ as the degree threshold for any constant that variables $x_1$ and $x_{k+1}$ can take. Let $L^1_{\text{heavy}}, L^{k+1}_{\text{heavy}}$ be unary relations that store the heavy values for $x_1, x_{k+1}$ respectively. We compute and store the result of the query

$$V(x_1, x_{k+1}) = L^1_{\text{heavy}}(x_1), R_1(x_1, x_2), \ldots, R_k(x_k, x_{k+1}), L^{k+1}_{\text{heavy}}(x_{k+1}).$$

This view has size bounded by $(|D|/\sqrt{|D|/\Delta})^2 = |D| \cdot \Delta$. We now consider the following
two adorned queries:

\[ V_{bb}^{1}(x_2, x_{k+1}) = R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1}). \]

\[ V_{bb}^{2}(x_1, x_k) = R_1(x_1, x_2), \ldots, R_{k-1}(x_{k-1}, x_k). \]

Both of these correspond to the \((k-1)\)-path case, so we can recursively use the data structure for this. Let \(S_k, T_k\) be the space and time for paths of length \(k\). For space, we have the following recursive formula:

\[ S_k = |D| \cdot \Delta + S_{k-1} \]

By materializing the join of all relations in each view. Let view \(V_i\) contain the join of all relations in each view. Let view \(V_i\) contain the join of all relations assigned to view \(V_i\).

We have now reduced the \(k\)-star query where all \(k\) variables are heavy into an instance of \(\ell\)-star query where the input relations are \(V_1, \ldots, V_\ell\). Suppose that there exists a data structure 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

\[ S = o(\Pi_{i=1}^{\ell} |V_i|/T^\ell) = o((\Pi_{i=1}^{\ell} \Pi_{R \in J_i} |R_i|/T^{|k_i|-1} \cdot (1/T^\ell)) = o(\Pi_{i=1}^{k} |R_i|/T^k) \]

which contradicts the space lower bound for \(k\)-star.

**Proof of Theorem 19.** Let \(\Delta = T\) be the degree threshold for the \(k\) bound variables \(y_1, \ldots, y_k\).

If any of the \(k\) variables is light (i.e \(|\sigma_{y_i=a}[y_i]R_i(y_i, x)| \leq \Delta\)), 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 \(V_i\) contain the join of \(k\) relations. Then, \(|V_i| = O((\Pi_{R \in J_i} |R_i|/T^{k_i-1}))\) where \(J_i\) is the set of all relations assigned to view \(V_i\).

Proof of Theorem 20. Let \(\Delta = T\) be the degree threshold for all vertices. If both bound variables in \(P_{bb}^3\) 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 \(P_{bb}^3(x_2, x_4) = R_2(x_2, x_3), R_3(x_3, x_4)\).

Suppose that there exists a data structure with space \(O(|D|^2/T^2)\) for \(P_{bb}^3(x_2, x_4)\) and answering time \(o(T)\). Then, we can answer \(P_{bb}^3\) with light \(x_1\) in time \(o(T^2)\). This improves the tradeoff for \(P_{bb}^3\) since the product of space usage and answering time is \(o(|D|^2)\) for any non-constant \(T\), a contradiction.
B Optimizing Tree Decompositions

In this section, we discuss two key optimizations that lead to improved tradeoffs. The first optimization relates to the space usage of the root bag. Observe that the space requirement of Theorem 11 was defined as the maximum over all bags except $A$ (which is the connected subset of tree nodes whose union is exactly equal to $C$). Our key observation is that if the space budget $S = O(|D|^{\max_{x \in A} \rho_x})$, then we can achieve an answering time of $T = O(1)$. This is because we have materialized all possible valid access requests (and their true/false answer) and similar to the query answering phase, given an access request, we can check in constant time whether the request is valid and find its boolean answer.

Example 21. Consider the query $Q_{x,y,z} = R(x,y), S(y,z), T(x,z), U(p,q), V(q,r), W(p,r), D(x,p), E(y,p), F(r,z)$ as shown in Figure 3. Applying our main result on this query gives a tradeoff $S = O(|E|^3/\tau)$ and $T = O(\tau)$ by assigning an edge cover of $1/2$ to the relations forming the two triangles. The $C$-connex decomposition will consist of the root bag containing variables $C = \{x,y,z\}$ and a child bag containing free variables $p, q, r$ and bound variables $x, y, z$. However, note that materializing the root bag which consists of relation $R, S, T$ takes space only $|E|^{3/2}$. Thus, the tradeoff is only useful when $S = O(|E|^{3/\tau}) \leq |E|^{3/2}$.

Our second optimization relates to complete materialization of all tuples over free variables in a bag. Consider a bag $t \notin A$ in the $C$-connex decomposition. Theorem 11 sets space usage of $t$ as $S = O(|D|^{\rho(t)})$ and answering time of $T = O(|D|^{\rho(t)})$. The key observation is that one can achieve $S = O(|D|^{\rho(t)}(\mathcal{V}))$ and $T = O(|D|^{\rho(t)}(\mathcal{V}))$, where $\rho(t)(\mathcal{V})$ is the fractional edge cover number over the free variables in $t$, by materializing the join over free variables of bag $t$ and storing a bit that tells whether a tuple $s$ participates in some join for the bags below it in the decomposition. Thus, in the query answering phase, we can check whether some tuple $s$ is true/false by iterating over all materialized tuples and return the answer. We demonstrate this with the following example.

Example 22. Consider the query $Q_{x_1,x_2,x_3,x_4} = R(x_1,y), S(x_2,y), T(y,z), U(x_3,z), V(x_4,z)$. The $C$-connex decomposition contains two bags: the root bag with variables $C = \{x_1,x_2,x_3,x_4\}$ and a child bag containing free variables $y, z$ and bound variables $x_1, x_2, x_3, x_4$. The tradeoff obtained is $S = O(|E|^4/\tau^2)$ (note that slack $\alpha = 2$) and $T = O(\tau)$. However, applying our optimization here, we observe that the fractional edge cover number for free variables $y, z$ is $1$. Thus, during the query answering phase, given an access request,
we can iterate over all \( O(|E|) \) tuples in relation \( T \) and check whether the access request and some tuple \( s \in T \) join or not, a constant time operation for each \( s \). Thus, we can achieve answering time \( T = O(|E|) \) using linear space \( S = O(|E|) \), which is an improvement over the tradeoff obtained directly from Theorem 11.

Thus, for each bag \( t \notin A \), we can set \( \rho_t(\delta) = \rho_t^*(V_t) \) and \( \delta(t) = \rho_t^*(V_t) \) to achieve the best possible width and height for the decomposition.

\[ \text{Figure 4 } C\text{-connex decomposition for Example 23} \]

The next example shows the application of the algorithm to adorned path queries containing negation.

\[ \text{Example 23.} \text{ Consider the query } Q_{bb}(x_1, x_6) = R(x_1, x_2), \lnot S(x_2, x_3), T(x_3, x_4), \lnot U(x_4, x_5), V(x_5, x_6). \text{ We use the decomposition as shown in Figure 4. We can now apply Theorem 11 to obtain the tradeoff } S = O(|D|^3/\tau) \text{ and } T = O(\tau). \text{ Both leaf bags only require linear space since a single atom covers the variables. Given an access request } x_1 \leftarrow a, x_6 \leftarrow b, \text{ we can check whether the data structure materialized any information for this request. If not, we proceed to the query answering phase and find at most } O(\tau) \text{ answers after evaluating the join in the middle bag. For each of these answers, we can now check in constant time whether the tuples formed by values for } x_2, x_3 \text{ and } x_4, x_5 \text{ are not present in relations } S \text{ and } U \text{ respectively.} \]

For adorned queries where \( V_b \subseteq \text{vars}(Q^-) \), 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 11 using the query \( Q^+ \).