Space-Query Tradeoffs in Range Subgraph Counting and Listing

Shiyuan Deng, Shangqi Lu, Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
Hong Kong, China
{*sydeng,sqlu,taoyf*}*@cse.cuhk.edu.hk*

January 9, 2023

Abstract

This paper initializes the study of *range subgraph counting* and *range subgraph listing*, both of which are motivated by the significant demands in practice to perform graph analytics on subgraphs pertinent to only selected, as opposed to all, vertices. In the first problem, there is an undirected graph G where each vertex carries a real-valued attribute. Given an interval q and a pattern Q, a query counts the number of occurrences of Q in the subgraph of G induced by the vertices whose attributes fall in q. The second problem has the same setup except that a query needs to enumerate (rather than count) those occurrences with a small delay. In both problems, our goal is to understand the tradeoff between *space usage* and *query cost*, or more specifically: (i) given a target on query efficiency, how much pre-computed information about G must we store? (ii) Or conversely, given a budget on space usage, what is the best query time we can hope for? We establish a suite of upper- and lower-bound results on such tradeoffs for various query patterns.

This research was supported in part by GRF Projects 14207820, 14203421, and 14222822 from HKRGC.

1 Introduction

Consider $G = (V, E)$ as a *data graph* and Q as a *pattern graph.* A subgraph of G, if isomorphic to Q, is said to be an *occurrence* of Q. The goal of *pattern searching* is to either list the occurrences of Q or to count the number of them. Both are fundamental problems in computer science and have attracted considerable attention in the past few decades.

This paper studies pattern searching in vertex-induced subgraphs. Here, a query selects a subset $U \subseteq V$ of vertices and needs to count/list the occurrences of Q in G', where G' is the subgraph of G induced by U. Note that if an occurrence uses any vertex outside U, the occurrence should not be counted/listed. Trivially, one can answer the query by first generating G' and then counting/listing Q in G' "from scratch", but this does not leverage the power of *preprocessing.* Instead, our goal is to store G in a data structure that can answer all queries with non-trivial guarantees. It is intriguing to investigate how much we can minimize the query time subject to a space budget, and conversely, how much space we must consume to achieve a target query time.

Vertex selection in database systems is done with a predicate q, which determines U as $\{v \in V \mid v \text{ satisfies } q\}$. Concentrating on *range predicates*, the problems we consider are:

Problem 1 (Range Subgraph Counting). $G = (V, E)$ is an undirected graph where each vertex $v \in V$ carries a real-valued *attribute* A_v. For an interval $q = [x_1, x_2]$, define $V_q = \{v \in V \mid x_1 \le A_v \le x_2\}$ and G_q as the subgraph of G induced by V_q. Let Q be a connected (only one connected component) pattern graph with $O(1)$ vertices. Given an interval q, a query returns the number of occurrences of Q in G_q. The pattern Q is fixed for all queries.

Problem 2 (Range Subgraph Listing). Same setup except that a query reports the occurrences of Q in G_q.

Universal Notations. Several notations will apply throughout the paper. Set $n = |V|$ and $m = |E|$. Symbol $\omega < 2.37286$ [1] represents the matrix multiplication exponent. The notations $\tilde{O}(.)$ and $\tilde{\Omega}(.)$ hide a factor polylogarithmic to the underlying problem's parameters.

1.1 Motivation

Practical Applications. Subgraph patterns are important for understanding the characteristics of a data graph G, as has been documented in a long string of papers, e.g., [2,3,8,10,11,17,18,24–28, 30,33,36–38,50]. In practice, analysts are interested in not only patterns from the whole G but also those pertinent only to selected vertices. Consider a social network G where each vertex represents an individual. A graph's *clustering coefficient* [49], a popular measurement in network science, is the ratio between the number of triangles (3-cliques[1]) and the number of wedges (2-paths[2]). The coefficient of G, however, is just a single value revealing little about the features of specific demographic groups. It is more informative to, for example, compare the coefficients of (i) the subgraph of G induced by people with ages $\in [20, 30]$, and (ii) that induced by age $\in [60, 70]$. A step further, by putting together the coefficients induced by "age $\in [i \cdot 10, (i + 1) \cdot 10]$" for each $i \in [1, 10]$, one obtains an interesting comparison across different age groups. Refined analysis can then concentrate on the pattern occurrences of a target group. The power of the above analysis owes to queries of Problem 1 and 2 with *arbitrary* selection ranges. Designing effective data structures is essential to avoid lengthy response time.

[1]An ℓ-*clique* is a clique with ℓ vertices.

[2]An ℓ-*path* is a path with ℓ edges.

Problem	**Pattern Q**	**Space**	**Query**	**Remark**
1 (cnt)	any fixed Q	$O(n^2)$	$\tilde{O}(1)$	near optimal†
1	wedge	$\tilde{O}(m^2/\lambda^2)$	$\tilde{O}(\lambda)$	for any $\lambda \in [1, \sqrt{m}]$, near optimal†
1 (lower bound)	wedge	$\tilde{O}(m^{2-\delta}/\lambda^2)$ impossible	$\tilde{O}(\lambda)$	for $\lambda \in [1, \sqrt{m}]$ and any $\delta > 0$, subj. to strong set disjointness conj.
1	ℓ-clique	$O(m)$	$\tilde{O}(1)$	
2 (rep)	any fixed Q	$\tilde{O}(m + m^{\rho^*}/\Delta)$	delay $\tilde{O}(\Delta)$	for any $\Delta \geq 1$, ρ^* = frac. edge covering num. of Q
2	triangle	$O(m)$	delay $\tilde{O}(1 + (m^*)^{\frac{\omega-1}{\omega+1}})$	m^* = num. of edges in at least one triangle in G_q
2	ℓ-star	$O(m)$	delay $\tilde{O}(1)$	near optimal
2	2ℓ-cycle	$\tilde{O}(\#P_\ell)$	delay $\tilde{O}(1)$	$\#P_\ell$ = num. of ℓ-paths in G

Remark: "near optimal" means no polynomial improvement (i.e., n^δ for arbitrary small constant $\delta > 0$) possbile. The near optimality marked with † is subject to the strong set disjointness conjecture.

Table 1: A summary of our results

Importance of Space-Query Tradeoffs. One should not confuse the space-query tradeoff with the tradeoff between *preprocessing time* and *query cost*, as has been extensively studied on join algorithms [5, 12, 20–23, 35, 41–43, 45]. Both tradeoffs are important, but they matter in different ways. Unlike preprocessing time, which is "one-time cost" (because a structure, once built, can be used forever), the space consumption is permanent. In other words, the space-query tradeoff has a (much) more durable effect on the underlying database system. However, in spite of their importance, the space-query tradeoffs on joins have received surprisingly little attention: we are aware of only a single paper [20], which, as will be discussed in Section 1.3, does not consider query predicates (or equivalently, only one query, which always outputs the entire join, exists) and concerns only reporting (but not counting). Our work can be thought of as a step in the same direction as [20] because, as explained in Section 5, subgraph searching can be cast as a join problem (in fact, some of our results are explicitly about joins), and actually the first step on predicate-driven queries and counting.

Finally, it is worth mentioning that a useful structure, no matter how little space it occupies, must be constructible in polynomial time. This is true for all the structures developed in our paper. In fact, each of our structures can be built with at most the time needed to find all the occurrences of the query pattern Q, ignoring polylog factors.

1.2 Our Contributions

Table 1 summarizes the main results of this paper. Next, we will explain the results in detail.

1.2.1 Problem 1

Wedges. We will show:

Theorem 1.1. *Consider Problem 1 with Q = wedge. For any real value $\lambda \in [1, \sqrt{m}]$, there is a structure of $\tilde{O}(m^2/\lambda^2)$ space that answers a query in $\tilde{O}(\lambda)$ time.*

The space-query tradeoff may look disappointing. After all, wedge counting is easy in *one-off* computation: we can count the number of wedges in G using $O(n + m)$ time. It is natural to wonder whether the space in Theorem 1.1 is necessary. We answer the question by showing that any substantial improvement to Theorem 1.1 will yield a major breakthrough on *set disjointness*:

Set Disjointness. The data is a collection of $s \geq 2$ sets S_1, S_2, ..., S_s. Given distinct set ids $a, b \in [1, s]$, a query returns whether $S_a \cap S_b$ is empty.

Let $N = \sum_{i=1}^{s} |S_i|$ be the input size of set disjointness. Given any $\lambda \in [1, \sqrt{N}]$, there is a simple structure of $O(N^2/\lambda^2)$ space with $O(\lambda)$ query time (see Appendix B). Improving the tradeoff by a polynomial factor even for one arbitrary λ has been a long-standing open problem. The *strong set disjointness conjecture* [31, 32] states that a structure with query time λ must use $\tilde{\Omega}(N^2/\lambda^2)$ space for any $\lambda \geq 1$. We will prove:

Theorem 1.2. *Consider Problem 1 with Q = wedge. Fix any $\lambda \in [1, \sqrt{m}]$ and any constant $\delta > 0$. Suppose that we can obtain a structure of $\tilde{O}(m^{2-\delta}/\lambda^2)$ space answering a query in $\tilde{O}(\lambda)$ time. Then, for any set disjointness input of size N, we can obtain a structure of $\tilde{O}(N^{2-\delta}/\lambda^2)$ space answering a query in $\tilde{O}(\lambda)$ time (thus breaking the strong set disjointness conjecture).*

Cliques. We will show:

Theorem 1.3. *For Problem 1 with Q = ℓ-clique, there is a structure of $O(m)$ space answering a query in $\tilde{O}(1)$ time.*

Counting triangles ($\ell = 3$) appears harder than counting wedges: in one-off computation, the fastest known algorithm for the former takes $O(m^{\frac{2\omega}{\omega+1}})$ time. It is thus surprising to see Q = triangle easier than Q = wedge in Problem 1. From Theorem 1.1 and 1.3, one sees that the problem of calculating the clustering coefficient (see Section 1.1) of G_q for any q boils down to counting the wedges in G_q. Effectively, this implies optimal settlement of that problem (subject to the strong set disjointness conjecture), which bears practical significance due to the popularity of clustering coefficients.

Arbitrary Subgraphs. We will show:

Theorem 1.4. *For any Q, there is a structure for Problem 1 that uses $O(n^2)$ space and answers a query in $\tilde{O}(1)$ time.*

The above result is difficult to improve: reducing the space by an n^δ factor for any constant $\delta > 0$ breaks the strong set disjointness conjecture. To explain, assume $n = O(m)$.[3] If there was a structure of $O(n^{2-\delta}) = O(m^{2-\delta})$ space and $\tilde{O}(1)$ query time, applying the structure to Q = wedge would yield a breakthrough on set disjointness by way of Theorem 1.2. The reader should note that the hardness comes from producing a guarantee on all Q; it is possible to do better for special patterns (Theorem 1.3). The hardness thus endows Q = wedge with unique significance in Problem 1. Theorem 1.4 further implies that Problem 1 under Q = wedge is the hardest when G is the sparsest: $m = o(n^{1+\epsilon})$ for any constant $\epsilon > 0$. To see why, set $m = n^{1+\epsilon}$, which gives $n^2 = m^{\frac{2}{1+\epsilon}}$. Since $\frac{2}{1+\epsilon} = 2 - \frac{2\epsilon}{1+\epsilon}$, Theorem 1.4 yields a structure of $O(m^{2-\delta})$ space and $\tilde{O}(1)$ query time with $\delta = \frac{2\epsilon}{1+\epsilon}$, improving Theorem 1.1 by a polynomial factor at $\lambda = \tilde{O}(1)$.

[3]Discard "isolated" vertices with no incident edges.

1.2.2 Problem 2

A listing query ensures a *delay* Δ if it reports a new occurrence of Q or declares "no more occurrences" within Δ time after the previous occurrence[4].

Arbitrary Subgraphs. We will show:

Theorem 1.5. *For any Q and $\Delta \geq 1$, there is a structure for Problem 2 that uses $\tilde{O}(m + m^{\rho^*}/\Delta)$ space and has a query delay of $\tilde{O}(\Delta)$, where ρ^* is the fractional edge covering number of Q.*

Imagine assigning each edge of Q a non-negative weight such that (i) for each vertex of Q, all its incident edges receive a combined weight at least 1 and (ii) the total weight of all edges is minimized. The fractional edge covering number ρ^* of Q is the total weight of an optimal assignment. The maximum number of occurrences of Q in G is $O(m^{\rho^*})$ [4] and the bound is tight in the worst case.

Our structure actually settles a problem on natural joins:

> **Range Join.** Let $\mathcal{R}$ be a set of $O(1)$ relations each with $O(1)$ real-valued attributes. Denote by *join*$(\mathcal{R})$ the natural join result on the relations in $\mathcal{R}$. Given an interval $q = [x_1, x_2]$, a query reports all the tuples $t \in$ *join*$(\mathcal{R})$ such that every attribute of t falls in q.

Let N be the total number of tuples in the relations of $\mathcal{R}$. For any $\Delta \geq 1$, we give a structure of $\tilde{O}(N + N^{\rho^*}/\Delta)$ space answering a query with an $\tilde{O}(\Delta)$ delay. Here, the fractional edge covering number ρ^* is with respect to the join's hypergraph (details deferred to Section 5).

The challenge behind Theorem 1.5 is to design a structure that works for all Q. It is possible to do better for specific Q. Next, we present three examples that are not only important subproblems themselves but also illustrate different techniques.

Triangles. We will show:

Theorem 1.6. *For Problem 2 with $Q =$ triangle, there is a structure of $O(m)$ space answering a query with an $\tilde{O}(1 + (m^*)^{\frac{\omega-1}{\omega+1}})$ delay, where m^* is the number of edges appearing in at least one reported triangle.*

The fractional edge covering number ρ^* is 1.5 for $Q =$ triangle. To ensure $\tilde{O}(m)$ space, Theorem 1.5 needs to set $\Delta = \sqrt{m}$. As $\frac{\omega-1}{\omega+1} < 0.408$, Theorem 1.6 achieves a polynomial improvement in delay. The reader should note that the value m^* in Theorem 1.6 never exceeds m but can be much less (this happens when there are few triangles to list). Problem 2 with $Q =$ triangle and q fixed to $(-\infty, \infty)$ was used as a motivating problem in the previous work of [20], which described a structure of $O(m)$ space with a delay $\tilde{O}(\sqrt{m})$ and is thus strictly improved by Theorem 1.6.

ℓ-Stars. An *ℓ-star* is a tree with ℓ leaves and one non-leaf vertex (a wedge is a 2-star). We will show:

Theorem 1.7. *For Problem 2 where $Q = \ell$-star, there is a structure of $O(m)$ space answering a query with an $\tilde{O}(1)$ delay.*

As a corollary, for any interval q, $O(m)$ space suffices to *detect* the presence of an ℓ-star in G_q using $\tilde{O}(1)$ time. For $Q =$ wedge, this means that the hardness manifested by Theorem 1.2 is indeed due to *counting*.

2ℓ-Cycles[5]**.** We will show:

[4]The reader may assume that a dummy occurrence is always output at the beginning of a query algorithm.
[5]A cycle with 2ℓ vertices.

Theorem 1.8. *For Problem 2 with $Q = 2\ell$-cycle where $\ell \geq 2$, there is a structure of $\tilde{O}(\#P_\ell)$ space answering a query with an $\tilde{O}(1)$ delay, where $\#P_\ell$ is the number of ℓ-paths in G.*

The fractional edge covering number ρ^* is ℓ for a 2ℓ-cycle. Theorem 1.5 needs $\tilde{O}(m^\ell)$ space to achieve an $\tilde{O}(1)$ delay. The space in Theorem 1.8 is significantly better. For $\ell = 2$ ($Q = 4$-cycle), the space is $\tilde{O}(nm)$ which is the maximum number of wedges in G. For $\ell > 2$, the space is $\tilde{O}(m^{\lceil(\ell+1)/2\rceil})$ which is the maximum number of ℓ-paths in G.

1.3 Related Work

The preceding sections have covered the most relevant existing results. We will now proceed to discuss other related work.

Pattern searching has been extensively studied in one-off computation. We refer the reader to [3, 8, 10, 17, 18, 27, 28, 30, 37, 50] and [2, 11, 17, 24–26, 33, 36, 38, 39], as well as the references therein, for algorithms on counting and listing, respectively. Those algorithms can be applied in Problem 1 and 2 after G_q has been generated. Our focus in this work is to avoid a full generation of G_q because doing so can take $\Omega(m)$ time.

In the other extreme, one can precompute the set S of occurrences of Q in G. The size of S is $O(m^{\rho^*})$ (AGM bound), assuming that Q has a constant size. By resorting to standard computational geometry techniques [19], we can store S in structures of $\tilde{O}(m^{\rho^*})$ space to answer a query of Problem 1 in $\tilde{O}(1)$ time and a query of Problem 2 with an $\tilde{O}(1)$ delay. For Problem 1, Theorem 1.4 achieves a better space bound on every Q with $\rho^* \geq 2$. When $\rho^* < 2$, Q has at most three vertices: a 1-path (single edge), a wedge, or a triangle. We have resolved the wedge and triangle cases (Theorem 1.1 and 1.3), while Problem 1 is trivial for $Q =$ 1-path. For Problem 2, Theorem 1.5 captures the above extreme idea as a special case with $\Delta = \tilde{O}(1)$ and offers a tunable space-query tradeoff.

A *relational event graph*, introduced by Bannister et al. [6], is a graph $G = (V, E)$ where every *edge* $e \in E$ carries a real-valued timestamp. For an interval $q = [x_1, x_2]$, let G_q^{edge} be the subgraph of G induced by all the edges whose timestamps are covered by q. A pattern searching query counts/lists the occurrences of a pattern Q in G_q^{edge}. See [6, 14, 15] for several data structures designed for such queries. Similar as it sounds, pattern searching on a relational event graph is drastically different from Problem 1 and 2 such that there is little overlap — in neither results nor techniques — between our solutions and those in [6, 14, 15].

Delay minimization is an important topic in the literature of joins and conjunctive queries; see [5, 9, 12, 13, 20–23, 34, 35, 41, 43–45] and their references. Regarding our problems, we are not aware of previous work giving a result better than what has already been mentioned. Our formulation of range join listing (Section 1.2.2) suggests that the presence of query predicates can pose new challenges on joins (also conjunctive queries) from the indexing's perspective. Deep and Koutris [20] proved a result equivalent to Theorem 1.5 (up to an $\tilde{O}(1)$ factor) on Problem 2, but only in the special scenario where a query concerns the whole G, i.e., fixing the query range q to $(-\infty, \infty)$.

2 Preliminaries

In this section, we will describe several technical tools to be deployed in our solutions.

Structures for Multidimensional Points. We will utilize some well-known geometry data structures as introduced below. The reader does not need to be bothered with the details of these structures because we will apply them as "black boxes". Let P be a set of n points in d-dimensional space $\mathbb{R}^d$ where d is a constant. Given a rectangle q of the form $[x_1, y_1] \times [x_2, y_2] \times ... \times [x_d, y_d]$, a *range reporting* query enumerates the points in $P \cap q$. We can create a *range tree* [7,19] on P, which uses $\tilde{O}(n)$ space and permits us to answer such a query with an $\tilde{O}(1)$ delay. When $d = 2$, we can replace the range tree with a *Chazelle's structure* [16] which retains the aforementioned query performance but reduces the space consumption to $O(n)$.

We will also need *range sum* queries on P in the scenario where each point in P is 2D (i.e., $d = 2$) and carries a real-valued *weight*. Given a rectangle $q = [x_1, y_1] \times [x_2, y_2]$, such a query reports the total weight of the points in $P \cap q$. We can again build a Chazelle's structure of [16] on P which occupies $O(n)$ space and answers a query in $\tilde{O}(1)$ time.

From "Delays with Duplicates" to "Delays under Distinctness". Let us consider a duplicate-removal scenario often encountered in designing algorithms with small delays. Suppose that we have an algorithm $\mathcal{A}$ for enumerating a set S of elements. With a delay of Δ, $\mathcal{A}$ can report an element $e \in S$, but cannot guarantee that e has never been reported before. The good news, on the other hand, is that $\mathcal{A}$ can output the same element at most α times for some $\alpha \geq 1$.

By modifying a *buffering technique* in [47], we can convert $\mathcal{A}$ into an algorithm that enumerates only the *distinct* elements of S with a delay of $O(\alpha \cdot \Delta \log |S|)$. Conceptually, divide the execution of $\mathcal{A}$ into *epochs*, each of which runs for $\alpha \cdot \Delta$ time[6]. As $\mathcal{A}$ runs, we use a *buffer* B to stash the set of distinct elements that have been found by $\mathcal{A}$ but not yet reported. Every time $\mathcal{A}$ finds an element $e \in S$, we check whether e has ever existed in B (this takes $O(\log |S|)$ time, using a binary search tree maintained on all the elements that have ever been found so far). If so, e is ignored; otherwise, it is added to B. At the end of each epoch, we output an arbitrary element from B and remove it from B. Finally, after $\mathcal{A}$ has terminated, we simply output the remaining elements in B.

B always contains at least one element at the end of each epoch. To see why, consider the end of the t-th epoch for some $t \geq 1$. At this moment, $\mathcal{A}$ has been running for $t \cdot \alpha \cdot \Delta$ time and therefore must have reported $t \cdot \alpha$ elements, which may not be distinct. However, as each element can be reported at most α times, there must be at least t (distinct) ones among those $t \cdot \alpha$ elements. Since we have reported only $t - 1$ elements in the preceding epochs, B must still have at least one element at the end of epoch t. It is now straightforward to verify that the modified algorithm has a delay of $O(\alpha \cdot \Delta \log |S|)$ in enumerating the distinct elements of S.

3 Problem 1: Matching Upper and Lower Bounds

This section will establish the conditional lower bound in Theorem 1.2 and its matching upper bound in Theorem 1.4. Our discussion on the upper bound will also establish Theorem 1.3. Throughout the paper, we will assume that the vertices of G have distinct attribute values. The assumption loses no generality because one can break ties by vertex id.

3.1 Lower Bound

Suppose that Problem 1 under $Q =$ wedge admits a structure that uses $\tilde{O}(m^{2-\delta}/\lambda^2)$ space and answers a query in $\tilde{O}(\lambda)$ time for some $\lambda \geq 1$. We will design a structure for set disjointness that uses $\tilde{O}(N^{2-\delta}/\lambda^2)$ space and answers a query in $\tilde{O}(\lambda)$ time. Recall that the data input to set disjointness consists of $s \geq 2$ sets $S_1, ..., S_s$ with a total size of N. Define $\mathcal{U} = \bigcup_{i=1}^{s} S_i$.

[6]Recall that "time" in the RAM model is defined as the number of atomic operations (e.g., addition, multiplication, comparison, accessing a memory word, etc.) executed. Each epoch is essentially a sequence of $\alpha \cdot \Delta$ such operations.

Create a graph $G = (V, E)$ as follows. V has $2s + |\mathcal{U}|$ vertices, including $2s$ *set vertices* and $|\mathcal{U}|$ *element vertices*. Each set S_i ($i \in [1, s]$) defines two set vertices, whose attribute values are set to i and $s + i$, respectively. Each element in $\mathcal{U}$ defines an element vertex with the same attribute value $s + 1/2$. Set E contains $2N$ edges: for each element $e \in S_i$, add to E two edges each between the element vertex of e and a set vertex of S_i. Now, create a Problem-1 structure under Q = wedge on G. The structure occupies $\tilde{O}(N^{2-\delta}/\lambda^2)$ space.

Consider a set disjointness query with set ids a and b. Assuming w.l.o.g. $a < b$, we issue four Problem-1 wedge-counting queries on G with intervals $q_1 = [a, s + b]$, $q_2 = [a + 1, s + b]$, $q_3 = [a, s+b-1]$, and $q_4 = [a+1, s+b-1]$, respectively. Let $c_1, c_2, ..., c_4$ be the counts returned. We declare $S_a \cap S_b$ non-empty if and only if $c_1 - c_2 - c_3 + c_4 > 0$. The query time is $\tilde{O}(\lambda)$. Appendix A proves the algorithm's correctness. This completes the proof of Theorem 1.2.

3.2 Upper Bound

Next, we will attack Problem 1 by allowing Q to be an arbitrary pattern graph. Consider any occurrence of Q in G. Let u (resp. v) be the vertex in this occurrence with the smallest (resp. largest) attribute. We *register* the occurrence at the pair (u, v). Denote by $c_{u,v}$ the number of occurrences registered at (u, v).

For a query with $q = [x_1, x_2]$, an occurrence registered at (u, v) appears in G_q (i.e., the subgraph of G induced by V_q) if and only if $A_u \geq x_1$ and $A_v \leq x_2$. We can therefore convert the problem to *range sum* on 2D points. For each pair $(u, v) \in V \times V$, create a point (A_u, A_v) with weight $c_{u,v}$. Let P be the set of points created; clearly, $|P| = O(n^2)$. The query result is simply the total weight of all the points in P covered by the rectangle $[x_1, \infty) \times (-\infty, x_2]$ (a range sum operation). We can store P in a Chazelle's structure (see Section 2) that occupies $O(|P|) = O(n^2)$ space and performs a range sum operation in $\tilde{O}(1)$ time. This establishes Theorem 1.4.

Improvement for Cliques. The space of our structure can be lowered to $O(m)$ when Q is a clique. The crucial observation is that registering an occurrence at (u, v) implies $\{u, v\} \in E$. We add to P only the points (A_u, A_v) with a non-zero $c_{u,v}$ (points with zero weights do not affect a range sum operation). This reduces the size of P to at most m and, hence, the space of the Chazelle's structure to $O(m)$. We thus complete the proof of Theorem 1.3.

4 Problem 1: Wedges

The section will explain how to achieve the guarantees in Theorem 1.1 for Problem 1 under Q = wedge. We will represent a wedge occurrence in $G = (V, E)$ as *wedge*(u, v, w) where u, v, and w are vertices in V, and $\{u, v\}$ and $\{v, w\}$ are edges in E. Let us introduce a slightly different problem:

Colored Range Wedge Counting. Define $G = (V, E)$ and A_v for each $v \in V$ as in Problem 1. Each vertex in V is colored black or white. Given an interval q, a query returns the number of occurrences *wedge*(u, v, w) such that $A_u \in q$, $A_w \in q$, and v is black.

Note that no requirements exist on A_v and the colors of u and w.

Let $\mathcal{C}$ be a set of subsets of V. We call $\mathcal{C}$ a *canonical collection* if

- (P4-1) each vertex of V appears in $\tilde{O}(1)$ subsets in $\mathcal{C}$;

- (P4-2) for any interval q, we can partition V_q (i.e., the set of vertices in V with attribute values in q) into $\tilde{O}(1)$ disjoint subsets, each being a member of $\mathcal{C}$. The ids of these subsets can be obtained in $\tilde{O}(1)$ time.

It is rudimentary to find a canonical collection $\mathcal{C}$ satisfying $\sum_{U \in \mathcal{C}} |U| = \tilde{O}(n)$.[7] We will work with such a $\mathcal{C}$ henceforth. In Appendix B, we prove:

Lemma 4.1. *Consider the colored range wedge counting problem. For any real value $\lambda \in [1, \sqrt{m}]$, there is a structure of $\tilde{O}(m^2/\lambda^2)$ space that answers a query in $\tilde{O}(\lambda)$ time.*

Equipped with the above, we now return to Problem 1 with Q = wedge.

Structure. For each $U \in \mathcal{C}$ (where U is a subset of V), we create a graph G_U by adding edges in three steps:

1. Initialize G_U as an empty graph with no vertices and edges.
2. For every vertex $u \in U$, we add all its edges in G (i.e., the original data graph) to G_U. The addition of an edge $\{u, v\}$ creates vertex v in G_U if v is not present in G_U yet.
3. Finally, color a vertex in G_U black if it comes from U, or white otherwise.

We now build a structure of Lemma 4.1 on G_U, which uses $\tilde{O}(|E_U|^2/\lambda^2)$ space where E_U is the set of edges in G_U. By Property P4-1, each edge $\{u, v\}$ of G can be added to the E_U of $\tilde{O}(1)$ subsets $U \in \mathcal{C}$. It thus follows that $\sum_{U \in \mathcal{C}} |E_U| = \tilde{O}(m)$. The structures of all $U \in \mathcal{C}$ occupy $\tilde{O}(m^2/\lambda^2)$ space in total.

Query. Consider now a (Problem-1) query with interval q. By Property P4-2, in $\tilde{O}(1)$ time we can pick $h = \tilde{O}(1)$ members $U_1, ..., U_h$ from $\mathcal{C}$ to partition V_q. For each $i \in [1, h]$, issue a colored range wedge counting query with interval q on G_{U_i}. We return the sum of the h queries' outputs. The overall query time is $h \cdot \tilde{O}(\lambda) = \tilde{O}(\lambda)$.

To verify correctness, first observe that every *wedge*(u, v, w) counted by the colored query on G_{U_i} satisfies: $A_u \in q$, $A_w \in q$ (definition of colored range wedge counting), and $A_v \in q$ (because v being black means $v \in U_i \subseteq V_q$). Conversely, every occurrence *wedge*(u, v, w) satisfying $\{A_u, A_v, A_w\} \subseteq q$ is counted only once: by the colored query on G_{U_i} where U_i is the only subset (among all $i \in [1, h]$) containing v. Indeed, for any U_j with $j \neq i$, v is either absent in G_{U_j} or is white; in neither case can the wedge be counted. Correctness now follows.

5 Problem 2: Arbitrary Subgraphs

We now proceed to tackle Problem 2 for an arbitrary query pattern Q. We will, in fact, solve the range join problem defined in Section 1.2.2. As shown in Appendix D, it is relatively easy to convert our structure to prove Theorem 1.5.

For a relation $R \in \mathcal{R}$ (recall that $\mathcal{R}$ is the set of input relations; see Section 1.2.2) its scheme, *scheme*(R), is the set of attributes in R. Let $\mathcal{X} = \bigcup_{R \in \mathcal{R}}$ *scheme*(R). The input size N can now be

[7] It suffices to build a binary search tree T on the vertices' attribute values. Each node in T defines a subset in $\mathcal{C}$, which consists of every $v \in V$ whose attribute A_v is stored in the node's subtree. It is well known (see, e.g., [46]) that, for any interval q, there exist $O(\log n)$ *canonical nodes* in T whose subtrees are disjoint and together contain all and only the attribute values in q. Those nodes can be found in $O(\log n)$ time and satisfy Property P4-2 with respect to V_q.

expressed as $\sum_{R \in \mathcal{R}} |R|$. We will assume, w.l.o.g., that (i) the relations in $\mathcal{R}$ have distinct schemes, (ii) N is a power of 2, and (iii) each attribute $X \in \mathcal{X}$ has a domain $dom(X)$ comprising the integers in $[1, N]$. Given an interval $q = [x_1, x_2]$, a query lists every tuple t in $join(\mathcal{R})$ — the natural join result on $\mathcal{R}$ — satisfying $t[X] \in q$ for all $X \in \mathcal{X}$, where $t[X]$ is the tuple's value under attribute X. We want to design a structure of small space to answer such queries with a small delay.

It will be convenient to work with a hypergraph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ where $\mathcal{E} = \{scheme(R) \mid R \in \mathcal{R}\}$. Given an edge $e \in \mathcal{E}$, we use R_e to denote the (only) relation $R \in \mathcal{R}$ whose scheme is e. For a function W that assigns a non-negative weight $W(e)$ to every $e \in \mathcal{E}$, its *lump-sum* is $\sum_{e \in \mathcal{E}} W(e)$. The function W is a *fractional edge covering* if $\sum_{e \in \mathcal{E}: X \in e} W(e) \geq 1$ holds on every attribute $X \in \mathcal{X}$. The *fractional edge covering number* ρ^* of $\mathcal{G}$ is the smallest lump-sum of all fractional edge coverings. Henceforth, we will use W to represent an optimal assignment function with lump-sum ρ^*.

The section's main result is:

Theorem 5.1. *For the range join problem (see Section 1.2.2), given any $\Delta \geq 1$, there is a structure of $\tilde{O}(N + N^{\rho^*}/\Delta)$ space that answers a query with an $\tilde{O}(\Delta)$ delay.*

5.1 A Generalization of the AGM Bound

The classical AGM bound [4] states that $|join(\mathcal{R})| \leq \prod_{e \in \mathcal{E}} |R_e|^{W(e)}$. Next, we will present a more general version of this inequality.

Set $d = |\mathcal{X}|$ and impose an arbitrary ordering on the d attributes: $X_1, X_2, ..., X_d$. Given intervals $I_1, I_2, ..., I_d$ where $I_i \subseteq dom(X_i)$ for each $i \in [1, d]$, define $B(I_1, ..., I_d)$ as the d-dimensional box $I_1 \times ... \times I_d$. For a relation $R \in \mathcal{R}$, we use $R \ltimes B(I_1, ..., I_d)$ to represent the set of tuples $t \in R$ such that $t[X_i] \in I_i$ for every i satisfying $X_i \in scheme(R)$.

We prove in Appendix C:

Lemma 5.2. *Let $\boldsymbol{\mathcal{I}}_i$, $i \in [1, d]$, be a set of disjoint intervals in $dom(X_i)$. Then:*

$$\sum_{I_1 \in \boldsymbol{\mathcal{I}}_1} \sum_{I_2 \in \boldsymbol{\mathcal{I}}_2} ... \sum_{I_d \in \boldsymbol{\mathcal{I}}_d} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_d)|^{W(e)} \leq \prod_{e \in \mathcal{E}} |R_e|^{W(e)}. \tag{1}$$

To see how (1) captures the AGM bound, consider the special $\boldsymbol{\mathcal{I}}_i$ with size $|dom(X_i)|$, namely, each interval in $\boldsymbol{\mathcal{I}}_i$ is a value in $dom(X_i)$ and vice versa. Thus, $|R_e \ltimes B(I_1, ..., I_d)|$ is either 0 or 1 such that the left hand side of (1) is precisely $|join(\mathcal{R})|$. The real power of (1), however, comes from allowing $\boldsymbol{\mathcal{I}}_i$ to be an arbitrary set of disjoint intervals, a feature crucial for us to prove Theorem 5.1.

A remark is in order about why Lemma 5.2 is not trivial. It would be if the term $\prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_d)|^{W(e)}$ in (1) was replaced by the output size of the join on the relations in $\{R_e \ltimes B(I_1, ..., I_d) \mid e \in \mathcal{E}\}$. By the AGM bound, the term $\prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_d)|^{W(e)}$ is an *upper bound* on the size of the join $\{R_e \ltimes B(I_1, ..., I_d) \mid e \in \mathcal{E}\}$. The non-trivial goal is to show that the summation of all those *upper* bounds (i.e., the left hand side of (1)) still cannot exceed $\prod_{e \in \mathcal{E}} |R_e|^{W(e)}$.

5.2 Range Join

This subsection serves as a proof of Theorem 5.1. Given an $\ell \geq 0$, we call an interval a *level-ℓ dyadic interval* if it has the form $[i \cdot 2^\ell + 1, (i + 1) \cdot 2^\ell]$ for some integer $i \geq 0$. Because N is a power of 2, for each $\ell \in [0, \log_2 N]$, we can partition $[1, N]$ into $N/2^\ell$ disjoint level-ℓ dyadic intervals.

A *dyadic combination* is a sequence of d dyadic intervals $(I_1, ..., I_d)$; recall that $d = |\mathcal{X}|$. The combination defines a (natural) join instance on the relations in $\{R_e \ltimes B(I_1, ..., I_d) \mid e \in \mathcal{E}\}$. We will denote the instance as $\mathcal{R}_{I_1,...,I_d}$. Define

$$\text{AGM}(I_1, ..., I_d) \;\; = \;\; \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_d)|^{W(e)}. \tag{2}$$

The AGM bound assures us that $|join(\mathcal{R}_{I_1,...,I_d})| \le \text{AGM}(I_1, ..., I_d)$.

Structure. A dyadic combination $(I_1, ..., I_d)$ with a non-empty $join(\mathcal{R}_{I_1,...,I_d})$ is said to be *heavy* if $\text{AGM}(I_1, ..., I_d) > \Delta$, or *light* otherwise. For each heavy combination, we build a structure of [20] that can enumerate the tuples in $join(\mathcal{R}_{I_1,...,I_d})$ with an $\tilde{O}(\Delta)$ delay. The structure's space is bounded by $O(\text{AGM}(I_1, ..., I_d)/\Delta)$.[8]

We argue that the structures on all the heavy (dyadic) combinations use $\tilde{O}(N^{\rho^*}/\Delta)$ space in total. Fix d arbitrary level numbers $\ell_1, ..., \ell_d$ each between 0 and $\log_2 N$. For $i \in [1, d]$, let $\boldsymbol{\mathcal{I}}_i$ be the set of all level-ℓ_i dyadic intervals. The total space occupied by the structures of all heavy combinations $(I_1, ..., I_d) \in \boldsymbol{\mathcal{I}}_1 \times ... \times \boldsymbol{\mathcal{I}}_d$ is

$$\frac{1}{\Delta} \sum_{(I_1,...,I_d) \in \boldsymbol{\mathcal{I}}_1 \times ... \times \boldsymbol{\mathcal{I}}_d} \text{AGM}(I_1, ..., I_d). \tag{3}$$

up to an $\tilde{O}(1)$ factor. The above includes a term for every light combination but such terms can only over-estimate the space. Each $\boldsymbol{\mathcal{I}}_i$ is a set of disjoint intervals in $dom(X_i)$. Applying the definition in (2) and Lemma 5.2, we can see that (3) is bounded by N^{ρ^*}/Δ, noticing that the right hand side of (1) is at most N^{ρ^*}.

In the above analysis, we have fixed a set of $\ell_1, ..., \ell_d$. As each ℓ_i has $O(\log N)$ choices, all together there are $O(\log^d N) = \tilde{O}(1)$ different sets of $\ell_1, ..., \ell_d$. We can now conclude that the overall space is $\tilde{O}(N^{\rho^*}/\Delta)$.

Finally, we need a hash table to check in constant time whether a dyadic combination is heavy. The hash table occupies $\tilde{O}(N^{\rho^*}/\Delta)$ space because our earlier analysis implies a bound $\tilde{O}(N^{\rho^*}/\Delta)$ on the number of heavy dyadic combinations. The overall space of our entire structure is therefore $\tilde{O}(N + N^{\rho^*}/\Delta)$, where the term $\tilde{O}(N)$ counts the space for storing the relations of $\mathcal{R}$.

Query. Consider a range join query with interval $q = [x_1, x_2]$. We consider, w.l.o.g., that x_1 and x_2 are integers in $[1, N]$. In $\tilde{O}(1)$ time, we can partition the box $B(\underbrace{q, ..., q}_{t})$ into $O(\log^d N) = \tilde{O}(1)$ disjoint boxes, each in the form $B(I_1, ..., I_d)$ where $(I_1, ..., I_d)$ is a dyadic combination; we say that $(I_1, ..., I_d)$ is *canonical* for q. The query result is

$$\bigcup_{\text{canonical } (I_1, ..., I_d)} join(\mathcal{R}_{I_1,...,I_d}).$$

The results $join(\mathcal{R}_{I_1,...,I_d})$ of all the canonical $(I_1, ..., I_d)$ are disjoint. If a canonical $(I_1, ..., I_d)$ is heavy, we enumerate $join(\mathcal{R}_{I_1,...,I_d})$ with an $\tilde{O}(\Delta)$ delay using the structure of [20] on $(I_1, ..., I_d)$. Otherwise, we apply a worst-case optimal join algorithm [39, 40, 48] to compute $join(\mathcal{R}_{I_1,...,I_d})$.

[8]Strictly speaking, the space should also account for the relations in $\mathcal{R}_{I_1,...,I_d}$. In our context, it suffices to store the relations of $\mathcal{R}$ once and generate the relations in $\mathcal{R}_{I_1,...,I_d}$ when answering a query. Appendix D has additional details about [20].

The algorithm finishes in $\tilde{O}(\text{AGM}(I_1, ..., I_d))$ time, which is $\tilde{O}(\Delta)$ by definition of light dyadic combination. Our algorithm guarantees a delay of $\tilde{O}(\Delta)$. This completes the proof of Theorem 5.1.

Remark. In [36], Khamis et al. used dyadic intervals in their algorithm for one-off computation of *join*$(\mathcal{R})$. Their main technical issue was to select "good" dyadic boxes (i.e., boxes of the form $B(I_1, ..., I_d)$) to cover the tuples in *join*$(\mathcal{R})$ once. That issue is non-existent in our context, where the primary obstacle is to argue that the total space given in (3) is affordable. We overcame the obstacle using Lemma 5.2, which, though perpahs no longer surprising given all the existing variations of the AGM bound, deserves a careful treatment that, we believe, has not appeared before.

6 Problem 2: Triangles

This section will describe a structure for Problem 2 under Q = triangle. We will first attack, in Section 6.1 and 6.2, two fundamental problems whose solutions are vital to establishing Theorem 1.6, the proof of which is presented in Section 6.3.

6.1 The Range Triangle Edges Problem

This subsection will discuss the following standalone problem.

Range Triangle Edges (RTE). Let G be an undirected graph with m edges. Given an interval $q = [x_1, x_2]$, a query returns: (i) all the edges appearing in at least one triangle of G_q; and (ii) $\Theta(m^*)$ triangles where m^* is the number of edges reported in (i).

We will develop a structure of $O(m)$ space that can answer a query in $\tilde{O}(m^*)$ time. Furthermore, the query can enumerate the m^* edges and the $\Theta(m^*)$ triangles both with a delay Δ.

Let us represent a triangle occurrence in G as *triangle*(u, v, w) where u, v, and w are the triangle's vertices. Ordering is important: we will always adhere to the convention $A_u < A_v < A_w$. Given an interval q, we denote by E_q^* the set of edges showing up in at least one triangle of G_q. Hence, $m^* = |E_q^*|$. If *triangle*(u, v, w) appears in G_q, we call $\{u, v\}$ a type-1 edge, $\{v, w\}$ a type-2 edge, and $\{u, w\}$ a type-3 edge. The total number of edges of all three types is between m^* and $3m^*$.[9]. Next, we explain how to extract the edges of each type in G_q.

Type 1 and 2. We will discuss only type 1 because type 2 is symmetric. For each edge $\{u, v\}$ in G (assume, w.l.o.g., $A_u < A_v$), identify a *sentinel* vertex w^* for $\{u, v\}$ as follows:

- w^* = null if G has no occurrence of the form *triangle*(u, v, w);
- otherwise, w^* has the smallest attribute among all the vertices w making a triangle occurrence *triangle*(u, v, w) in G.

Consider any interval $q = [x_1, x_2]$. Observe that $\{u, v\}$ is a type-1 edge for q if and only if $x_1 \le A_u$ and $A_{w^*} \le x_2$. This motivates us to convert type-1 edge retrieval to range reporting on 2D points (introduced in Section 2). Towards the purpose, create a set P of points, which has a point (A_u, A_{w^*}) for every $\{u, v\}$ whose sentinel w^* is not null. Attach edge $\{u, v\}$ to the point (A_u, A_{w^*}) so that the former can be fetched along with the latter. The size of P is at most m. Given $q = [x_1, x_2]$, we can find all the type-1 edges by enumerating the points of P inside the

[9]An edge can be of different types in various triangle occurrences.

rectangle $[x_1, \infty) \times (-\infty, x_2]$. Hence, we can store P in a Chazelle's structure (see Section 2) that has $O(|P|) = O(m)$ space and ensures an $\tilde{O}(1)$ delay in reporting the type-1 edges of any q.

Type 3. A similar approach works for type 3. Let $\{u, w\}$ be an edge appearing in at least one occurrence *triangle*(u, v, w) in G. It is a type-3 edge of $q = [x_1, x_2]$ if and only if $x_1 \leq A_u$ and $A_w \leq x_2$. By adapting the earlier discussion in a straightforward manner, we conclude that there is a structure of $O(m)$ space allowing us to retrieve all the type-3 edges with an $\tilde{O}(1)$ delay.

Listing $\Theta(m^*)$ Triangles. The above has explained how to retrieve E_q^*, but an RTE query still needs to report $\Theta(m^*)$ triangles. Next, we remedy the issue by slightly modifying our solution so far.

Recall that, in dealing with type 1, we attached the edge $\{u, v\}$ to the point (A_u, A_{w^*}) generated from the edge. Now, we attach *triangle*(u, v, w^*) to (A_u, A_{w^*}) as well. This way, when (A_u, A_{w^*}) is found, we obtain both $\{u, v\}$ and *triangle*(u, v, w^*) for free. After applying the same idea to type-2 and type-3, we can assert that, whenever the query algorithm finds a type-1, -2, or -3 edge, it must have also found a triangle in G_q. Therefore, the algorithm can report the triangles in G_q with an $\tilde{O}(1)$ delay, although the same triangle may be reported up to three times[10]. By applying the duplicate-removal technique in Section 2, we now have an algorithm that can enumerate $\Theta(m^*)$ distinct triangles with an $\tilde{O}(1)$ delay. The number of distinct triangles reported is at least $m^*/3$ and at most $3m^*$.

6.2 The Small-Delay Triangle Listing Problem

In this subsection, we will concentrate on a standalone problem defined as follows.

> **Small-Delay Triangle Listing (SDTL).** G is an undirected graph with m edges, each of which appears in at least one triangle. We are given $\Omega(m)$ *free triangles* and $O(m)$ *forbidden triangles*. Design an algorithm to enumerate all the triangles of G — except for the forbidden ones — with a small delay (free triangles must be enumerated). No preprocessing is allowed.

We will settle the problem with an algorithm of delay $\tilde{O}(m^{\frac{\omega-1}{\omega+1}})$.

Suppose that G has OUT triangles in total. Our starting point is an algorithm of Bjorklund et al. [11] which is able to list k triangles in $\alpha \cdot m^{\frac{3(\omega-1)}{\omega+1}} k^{\frac{3-\omega}{\omega+1}}$ time, where $\alpha = \tilde{O}(1)$, for a parameter $k \in [\Omega(m), \text{OUT}]$. As far as the algorithm of [11] is concerned, we can consider OUT known because it can be found in $O(m^{2\omega/(\omega+1)})$ time [3] which is $O(m^{\frac{3(\omega-1)}{\omega+1}} k^{\frac{3-\omega}{\omega+1}})$. The algorithm of Bjorklund et al. does not have a small delay, but we will turn it into one that does.

We run the algorithm of Bjorklund et al. [11] with geometrically-increasing k and, in each run, report only some, but not all, of the triangles. How many triangles are reported in each run is decided strategically to keep the delay small. Let S_{no}^0 be the set of forbidden triangles and S_{yes}^0 the set of free triangles in the beginning. Set $k_0 = |S_{no}^0| + |S_{yes}^0|$. When running the algorithm of [11] for the i-th time, we set its parameter k to $k_i = \min\{3^i k_0, \text{OUT}\}$. We enforce the invariant that, when run i starts, there are always a set S_{no}^{i-1} of forbidden triangles and a set S_{yes}^{i-1} of free triangles. The set S_{yes}^{i-1} will be reported with a small delay during the i-th run (details to be clarified shortly).

[10]An occurrence *triangle*(u, v, w) can be reported only when $\{u, v\}$, $\{v, w\}$, or $\{u, w\}$ is output as a type-1, -2, or -3 edge, respectively.

Specifically, suppose that the i-th run finds a set S^i_{raw} of k_i triangles (some of which have been output in previous runs). We generate the forbidden and free sets for the next run as follows:

$$S^i_{no} = S^{i-1}_{no} \cup S^{i-1}_{yes} \text{ and then } S^i_{yes} \quad = \quad S^i_{raw} \setminus S^i_{no}.$$

Run i finishes in $\alpha \cdot m^{\frac{3(\omega-1)}{\omega+1}} k_i^{\frac{3-\omega}{\omega+1}}$ time. We instruct the run to output a triangle from S^{i-1}_{yes} every

$$\alpha \cdot \frac{m^{\frac{3(\omega-1)}{\omega+1}} k_i^{\frac{3-\omega}{\omega+1}}}{|S^{i-1}_{yes}|} \tag{4}$$

atomic operations. We will show $|S^{i-1}_{yes}| = \Omega(k_i)$, with which the delay in (4) can be bounded as:

$$\tilde{O}\left(\frac{m^{\frac{3(\omega-1)}{\omega+1}}}{k_i^{\frac{2\omega-2}{\omega+1}}}\right) = \tilde{O}\left(m^{\frac{\omega-1}{\omega+1}}\right) \tag{5}$$

where the equality used $k_i \geq k_0 = \Omega(m)$.

For $i = 1$, $|S^{i-1}_{yes}| = \Omega(k_0)$ follows directly from the definition of the SDTL problem (i.e., we have $\Omega(m)$ free triangles to start with). To prove $|S^{i-1}_{yes}| = \Omega(k_i)$ for $i \geq 2$, we derive:

$$|S^{i-1}_{no}| \leq |S^0_{no}| + |S^0_{yes}| + \sum_{j=1}^{i-2} |S^j_{raw}| = k_0 + \sum_{j=1}^{i-2} 3^j \cdot k_0 = \sum_{j=0}^{i-2} 3^j \cdot k_0 < \frac{3^{i-1}k_0}{2}.$$

Therefore:

$$|S^{i-1}_{yes}| \quad \geq \quad |S^{i-1}_{raw}| - |S^{i-1}_{no}| > k_{i-1} - 3^{i-1}k_0/2 = k_{i-1}/2 = \Omega(k_i).$$

We now conclude that the delay of our algorithm is as given in (5).

6.3 Proof of Theorem 1.6

We are ready to explain how to solve Problem 2 with Q = triangle. In preprocessing, we build an RTE structure (Section 6.1) on G. Now, consider a (Problem-2) query with interval q. We start by issuing an RTE query to retrieve E^*_q, i.e., the set of edges appearing in at least one triangle of G_q. This, in effect, generates G^*_q, which is the subgraph of G_q induced by the edges in E^*_q. In addition, the RTE query has also enumerated a set S of $\Theta(m^*)$ triangles in G_q, where $m^* = |E^*_q|$. The size of S falls in $[\frac{m^*}{3}, 3m^*]$.

Our remaining mission is to enumerate the triangles in G^*_q that are outside S. Note that G^*_q is a graph with m^* edges and at least $\Theta(m^*)$ triangles. This motivates us to convert the mission to the SDTL problem, which has been solved in Section 6.2. However, the SDTL problem requires $\Theta(m^*)$ free triangles and $O(m^*)$ forbidden triangles as part of the input. Unfortunately, we do not seem to have these triangles at the moment.

We overcome this obstacle by, interestingly, dividing S into S_{yes} and S_{no}, such that S_{yes} (resp. S_{no}) serves as the set of free (resp. forbidden) triangles. Recall that the RTE query algorithm, denoted as $\mathcal{A}$, is designed to enumerate an edge in E^*_q with a delay $\Delta = \tilde{O}(1)$ and a triangle in S also with a delay Δ. Therefore, it must finish within $t_{max} = \max\{\Delta \cdot (|E^*_q|+1), \Delta \cdot (|S|+1)\} \leq \Delta \cdot (3m^*+1)$ time. We can now apply the buffering technique in Section 2 with $\alpha = 18$ to turn $\mathcal{A}$ into an algorithm that outputs a triangle at the end of each epoch, which has a length 18Δ. The total number of epochs is at most $\frac{t_{max}}{18\Delta} \leq \frac{3m^*+1}{18}$. Thus, when $\mathcal{A}$ finishes, we have output at most $(3m^*+1)/18$ triangles, whereas the buffer B (defined in Section 2) still has at least $|S| - \frac{3m^*+1}{18} = \Theta(m^*)$ triangles. We can, thus, set S_{yes} to the content of B when $\mathcal{A}$ finishes, and S_{no} to the set of triangles already output.

We can now apply the SDTL algorithm on G_q^* and, thus, complete the proof of Theorem 1.6.

7 Problem 2: Near-Constant Delays

This section will focus on two instances of Problem 2 where it is possible to achieve $\tilde{O}(1)$ delays with space substantially smaller than Theorem 1.5. We will discuss first $Q = \ell$-star in Section 7.1 and then $Q = 2\ell$-cycle in Section 7.2. We will focus on explaining how to enumerate a perhaps-not-distinct occurrence with an $\tilde{O}(1)$ delay, while ensuring each occurrence to be output only a constant number of times. Owing to the duplicate-removal method in Section 2, we can modify the algorithms to enumerate only *distinct* occurrences with $\tilde{O}(1)$ delays.

7.1 ℓ-Stars

Recall that an ℓ-star is a tree with only one non-leaf node, which we will refer to as the star's *center*. Consider a query with interval q. We refer to a node u as a *q-center* if G_q has at least one ℓ-star occurrence with u as the center. Once u is found, it becomes a trivial matter to enumerate all the ℓ-stars having u as the center with an $\tilde{O}(1)$ delay. Specifically, we can first (use a binary search tree to) retrieve all the neighbors v of u in G satisfying $A_v \in q$. From those neighbors, any ℓ distinct vertices form an ℓ-star together with u (as the center). It is rudimentary to ensure an $\tilde{O}(1)$ delay in enumerating all those stars.

Next, we concentrate on designing a structure to enumerate the q-centers with an $\tilde{O}(1)$ delay. Consider an arbitrary ℓ-star in G with center u. Sort the star's $\ell+1$ vertices in ascending order of attribute and look for the position of u. If u is the r-th smallest, we will refer to the star as a *rank-r ℓ-star* and u as a *rank-r q-center*.

Now, fix an $r \in [1, \ell+1]$. We will describe a structure to support the following operation:

> Given an interval q, find all the rank-r q-centers, i.e., all vertices $u \in V$ s.t. G_q has a rank-r ℓ-star with u as the center.

Consider any rank-r ℓ-star in G having u as the center. Let us write out the star's vertices as $v_1, ..., v_{r-1}, u, v_{r+1}, ..., v_\ell$ in ascending order of attribute. For a $q = [x_1, x_2]$, the ℓ-star appears in G_q if and only if $x_1 \leq A_{v_1}$ and $A_{v_\ell} \leq x_2$. Refer to v_1 as a *left r-sentinel* of u and to v_ℓ as a *right r-sentinel* of u. From all the left r-sentinels of u (one from each rank-r ℓ-star with center u), identify the one v_1^* with the largest attribute. Similarly, from all the right r-sentinels of u, identify the one v_ℓ^* with the smallest attribute. Observe that u is a rank-r q-center if and only if $x_1 \leq A_{v_1^*}$ and $A_{v_\ell^*} \leq x_2$. We can therefore convert the retrieval of rank-r q-centers into range reporting on 2D points (review Section 2), in the same way as illustrated in Section 6.1. Following Section 2, we can create a Chazelle's structure on n points — each point created for a vertex $u \in V$ in the way explained — that has $O(n)$ space and, given any q, can list the rank-r q-centers with an $\tilde{O}(1)$ delay. This completes the proof of Theorem 1.7.

7.2 2ℓ-Cycles

We will start with an assumption: all queries specify a fixed $q = (-\infty, \infty)$, namely, there is effectively only one query, which enumerates all the 2ℓ-cycles in G. The assumption allows us to explain the core ideas with the minimum technical details and will be removed eventually.

Queries with $q = (-\infty, \infty)$. Given a 2ℓ-cycle occurrence, we refer to the vertex u in the cycle having the smallest attribute as the occurrence's *anchor*. Let v be the vertex in the cycle such that cutting the cycle at u and v gives two ℓ-paths connecting u and v. We will refer to v as the occurrence's *inverse anchor*, the pair (u, v) as an *anchor pair*, and the two aforementioned paths as *cycle ℓ-paths*. The number of cycle ℓ-paths is at most $\#P_\ell$ (recall that $\#P_\ell$ is the total number of ℓ-paths).

The problem may appear deceivingly simple: can't we answer a query by simply concatenating, for each anchor pair (u, v), every two cycle ℓ-paths from u to v? This does not work because the two cycle ℓ-paths may share common vertices other than u and v, in which case the concatenation does not yield a 2ℓ-cycle! This motivates a crucial notion: two cycle ℓ-paths are *interior disjoint* if they (i) have the same anchor pair (u, v), and (ii) do not share any common vertex except u and v. Concatenating two cycle ℓ-paths from u to v spawns a 2ℓ-cycle if and only if those paths are interior disjoint. The challenge we are facing at this moment is the following problem.

> Design a structure to support the following operation: given a cycle ℓ-path π from anchor u to inverse anchor v, list all the cycle ℓ-paths interior disjoint with π with an $\tilde{O}(1)$ delay.

We will overcome the challenge with a structure of $\tilde{O}(\#P_\ell)$ space.

Our main observation is that the operation can be converted to range reporting on $(\ell-1)$-dimensional points (review Section 2). To explain, let us consider any cycle ℓ-path π from anchor u to inverse anchor v. After excluding u and v, the path has $\ell - 1$ vertices, which we list as $w_1, w_2, ..., w_{\ell-1}$ in ascending order of attribute[11]. Convert π into an $(\ell-1)$-dimensional point $(A_{w_1}, ..., A_{w_{\ell-1}})$. Let $P_{u,v}$ be the set of points thus obtained from all the cycle ℓ-paths with (u, v) as the anchor pair.

Now, consider another cycle ℓ-path π' from u to v. List the vertices of π' other than u and v as $w'_1, w'_2, ..., w'_{\ell-1}$ also in ascending order of attribute. If π' is interior disjoint with π, each $A_{w'_i}$ ($i \in [1, \ell-1]$) must fall in one of the ℓ open intervals:

$$(-\infty, A_{w_1}), (A_{w_1}, A_{w_2}), ..., (A_{w_{\ell-2}}, A_{w_{\ell-1}}), (A_{w_{\ell-1}}, \infty). \tag{6}$$

Therefore, $(A_{w'_1}, ..., A_{w'_{\ell-1}})$ — the point converted from π' — must fall in one of the following $\ell^{\ell-1} = O(1)$ rectangles: $q_1 \times q_2 \times ... \times q_{\ell-1}$, where each q_i ($i \in [1, \ell-1]$) is taken independently from one of the intervals in (6). As per Section 2, by creating a range tree on $P_{u,v}$ of $\tilde{O}(|P_{u,v}|)$ space, we can enumerate all the points in such a rectangle with an $\tilde{O}(1)$ delay.

The conclusion from the above is that, for each anchor pair (u, v), we can create a range tree of $\tilde{O}(|P_{u,v}|)$ space which, given any cycle ℓ-path cycle π from u to v, permits the enumeration of every cycle ℓ-path π', which is interior disjoint with π, with an $\tilde{O}(1)$ delay. The structures of all the anchor pairs use in total $\sum_{\text{anc. pair } (u,v)} \tilde{O}(|P_{u,v}|) = \tilde{O}(\#P_\ell)$ space.

With the challenge conquered, listing all the 2ℓ-cycles becomes an easy matter. We simply look at each cycle ℓ-path π, retrieve every ℓ-path π' interior disjoint with π, and make a cycle by concatenating π and π'. The delay in cycle reporting is $\tilde{O}(1)$ (each 2ℓ-cycle can be reported twice).

[11] The order should not be confused with the order by which the vertices appear in π.

Arbitrary Queries. Next, we remove the constraint $q = (-\infty, \infty)$ and tackle queries with arbitrary q. A new issue now arises: a query can no longer afford to look at all the cycle ℓ-paths. We say that a cycle ℓ-path from anchor u to inverse anchor v *contributes* to G_q if it makes a 2ℓ-cycle in G_q with another interior disjoint cycle ℓ-path. We need a way to list only the contributing cycle ℓ-paths.

Fix any cycle ℓ-path π with anchor pair (u, v). Let S_π be the set of 2ℓ-cycles in G that include π and have (u, v) as the anchor pair. Take an arbitrary cycle from S_π. By definition of anchor, u has the smallest attribute among the cycle's vertices. Let w be the vertex in the cycle with the largest attribute. For $q = [x_1, x_2]$, the cycle appears in G_q if and only if $x_1 \le A_u$ and $A_w \le x_2$. Let w^* be the vertex with the smallest attribute among all such w's. It becomes evident that π contributes to the G_q of $q = [x_1, x_2]$ if and only if $x_1 \le A_u$ and $A_{w^*} \le x_2$. We can therefore convert the retrieval of contributing cycle ℓ-paths to range reporting on 2D points, using the method in Section 6.1. The resulting structure (a Chazelle's structure) stores a point converted from every cycle ℓ-path and uses $O(\#P_\ell)$ space. Give any q, we can list the cycle ℓ-paths contributing to G_q with an $\tilde{O}(1)$ delay.

Suppose that we have found a contributing cycle ℓ-path π with anchor pair (u, v). As before, we proceed to find the cycle ℓ-paths π' interior disjoint with π. The new requirement here, however, is that π' needs to be contributing as well. Recall that, in the $q = (-\infty, \infty)$ scenario, we converted the task to range reporting on $(\ell - 1)$-dimensional points. To deal with arbitrary $q = [x_1, x_2]$, we will increase the dimension by one.

To explain, in a fashion like before, let us list out the vertices of π — after excluding u and v — as $w_1, ..., w_{\ell-1}$ in ascending order of attribute. Denote by $w_{\max}$ the vertex in π with the largest attribute ($w_{\max}$ can be v). Convert π to an ℓ-dimensional point $(A_{w_1}, ..., A_{w_{\ell-1}}, A_{w_{\max}})$. Let $(A_{w'_1}, ..., A_{w'_{\ell-1}}, A_{w'_{\max}})$ be the point converted from π' in the same manner. As we already know $A_u \in [x_1, x_2]$ (recall that π is a contributing path), π' is a path we want if and only if it satisfies the conditions below:

- $A_{w'_i}$ $(1 \le i \le \ell - 1)$ falls in one of the ℓ intervals in (6);
- $A_{w'_{\max}} \le x_2$.

Thus, the point $(A_{w'_1}, ..., A_{w'_{\ell-1}}, A_{w'_{\max}})$ must fall in one of the following $\ell^{\ell-1} = O(1)$ rectangles: $q_1 \times q_2 \times ... \times q_{\ell-1} \times (-\infty, x_2]$, where each q_i $(i \in [1, \ell - 1])$ is an interval taken independently from (6). By the above reasoning, for each anchor pair (u, v), we create a set $P_{u,v}$ of ℓ-dimensional points, each converted from a cycle ℓ-path with anchor pair (u, v), and then build a range tree on $P_{u,v}$. The range trees of all anchor pairs use $\sum_{\text{anc. pair } (u,v)} \tilde{O}(|P_{u,v}|) = \tilde{O}(\#P_\ell)$ space in total.

We now elaborate on the overall algorithm for answering a (Problem-2) query with parameter q. First, enumerate all the cycle ℓ-paths contributing to G_q with an $\tilde{O}(1)$ delay; call this the *outer enumeration*. Every time such a path π — say with anchor pair (u, v) — is obtained, we suspend outer enumeration and utilize the range tree on $P_{u,v}$ to find all the paths π' discussed previously with an $\tilde{O}(1)$ delay. Upon the delivery of a π', concatenate it with π and output the 2ℓ-cycle obtained. After exhausting all such π', we resume outer enumeration. This concludes the proof of Theorem 1.8.

Appendix

A Correctness of the Reduction in Section 3.1

In our construction, S_i ($i \in [1, s]$) corresponds to two set vertices with attribute values i and $i + s$, respectively. To facilitate derivation, we make a copy of each set: define $S_i = S_{i-s}$ for each $i \in [s + 1, 2s]$. In the rest of the proof, we hold the view that each S_i ($i \in [1, 2s]$) corresponds to only one set vertex, the one with attribute value i.

Consider a wedge occurrence with vertices u, v, and w where the edges are $\{u, v\}$ and $\{v, w\}$. We classify it as one of the two types below:

- (type e-s-e) u and w are element vertices and v is a set vertex;
- (type s-e-s) u and w are set vertices and v an element vertex.

Lemma A.1. *For any interval $q = [x, y]$ satisfying $1 \le x < s + 1/2 < y \le 2s$, we have*

- *the number of e-s-e wedges in G_q is $\sum_{i\in[x,y]} \binom{|S_i|}{2}$;*
- *the number of s-e-s wedges in G_q is $\sum_{i\in[x,y],j\in[i+1,y]} |S_i \cap S_j|$.*

Proof. To prove the first bullet, define an *e-s-e tuple* as (e_1, S_i, e_2) where $i \in q$ and e_1 and e_2 are distinct elements in S_i. The number of such tuples is $\sum_{i=x}^{y} \binom{|S_i|}{2}$. Our construction ensures a one-one correspondence between e-s-e tuples and e-s-e wedges in G_q.

To prove the second bullet, define an *s-e-s tuple* as (S_i, e, S_j) where $x \le i < j \le y$ and $e \in S_i \cap S_j$. The number of such tuples is $\sum_{i\in[x,y],j\in[i+1,y]} |S_i \cap S_j|$. Our construction ensures a one-one correspondence between s-e-s tuples and s-e-s wedges in G_q. □

To find out whether $S_a \cap S_b$ is empty, our reduction issues four Problem-1 queries with intervals $q_1 = [a, s + b]$, $q_2 = [a + 1, s + b]$, $q_3 = [a, s + b - 1]$, and $q_4 = [a + 1, s + b - 1]$, respectively. The above lemma is applicable to all these intervals. For $i \in [1, 4]$, let c'_i (resp. c''_i) be the number of e-s-e (resp. s-e-s) wedges in G_{q_i}; this means that c_i, the total number of wedges in G_{q_i}, equals $c'_i + c''_i$. According to Lemma A.1, we have:

$$\begin{aligned} & c'_1 - c'_2 - c'_3 + c'_4 \\ = & \sum_{i\in[a,s+b]} \binom{|S_i|}{2} - \sum_{i\in[a+1,s+b]} \binom{|S_i|}{2} - \sum_{i\in[a,s+b-1]} \binom{|S_i|}{2} + \sum_{i\in[a+1,s+b-1]} \binom{|S_i|}{2} \\ = & \ 0 \end{aligned}$$

and

$$\begin{aligned} & c''_1 - c''_2 - c''_3 + c''_4 \\ = & \left(\sum_{\substack{i\in[a,s+b] \\ j\in[i+1,s+b]}} |S_i \cap S_j| - \sum_{\substack{i\in[a+1,s+b] \\ j\in[i+1,s+b]}} |S_i \cap S_j| \right) - \\ & \left(\sum_{\substack{i\in[a,s+b-1] \\ j\in[i+1,s+b-1]}} |S_i \cap S_j| - \sum_{\substack{i\in[a+1,s+b-1] \\ j\in[i+1,s+b-1]}} |S_i \cap S_j| \right) \end{aligned}$$

$$\begin{aligned}
&= \sum_{j\in[a+1,s+b]} |S_a \cap S_j| - \sum_{j\in[a+1,s+b-1]} |S_a \cap S_j| \\
&= |S_a \cap S_{s+b}| \\
&= |S_a \cap S_b|.
\end{aligned}$$

We thus conclude that $c_1 - c_2 - c_3 + c_4 = |S_a \cap S_b|$.

B Proof of Lemma 4.1

Let us first consider a variant of the set disjointness problem.

Weighted Set Intersection Size. We have $s \geq 2$ sets S_1, S_2, ..., S_s. Each S_i ($i \in [1, s]$) is associated with a function $weight_{S_i}$ which assigns to each element $e \in S_i$ a value $weight_{S_i}(e)$. Given distinct set ids $a, b \in [1, s]$, a query returns

$$size(S_a, S_b) = \sum_{e\in S_a\cap S_b} weight_{S_a}(e) \cdot weight_{S_b}(e). \tag{7}$$

Let $N = \sum_{i=1}^{s} |S_i|$. For any $\lambda \in [1, \sqrt{N}]$, it is straightforward to build a structure of $O(N^2/\lambda^2)$ space answering a query in $O(\lambda)$ time. Call S_i ($i \in [1, s]$) a *large* set if $|S_i| > \lambda$, or a *small* set otherwise. The number of large sets is at most N/λ. For each pair $(i, j) \in [1, s] \times [1, s]$, $i \neq j$, such that S_i and S_j are both large, we store $size(S_i, S_j)$; the space needed is $O(N^2/\lambda^2)$. Given a query with parameters a and b, return $size(S_a, S_b)$ directly if S_a and S_b are both large. Otherwise, assume, w.l.o.g., that S_a is small. We compute $S_a \cap S_b$ in $O(\lambda)$ time using a hash table (for each $e \in S_a$, check if $e \in S_b$). The result $size(S_a, S_b)$ can then be obtained easily.

Equipped with the above, next we describe a structure for the colored range wedge counting problem to prove Lemma 4.1.

Structure. First obtain a canonical collection $\mathcal{C}$ of V (defined in Section 4) satisfying $\sum_{U\in\mathcal{C}} |U| = \tilde{O}(n)$. For each $U \in \mathcal{C}$ — recall that U is a subset of V — construct a weighted set as follows:

- S_U = the set of black vertices adjacent to at least one vertex in U;
- for each $b \in S_U$, $weight_{S_U}(b)$ = the number of vertices in U adjacent to b.

These weighted sets constitute an instance of the weighted set intersection size problem. Build a structure described earlier on the instance using the given parameter λ. The lemma below implies that the structure occupies $\tilde{O}(m^2/\lambda^2)$ space.

Lemma B.1. $\sum_{U\in\mathcal{C}} |S_U| = \tilde{O}(m)$.

Proof. Each $b \in S_U$ is adjacent to a vertex $u \in U$. Pay a dollar to the edge $\{b, u\}$ for each such pair (b, u). Since an edge can receive a dollar only if it has a vertex in U, it can receive up to two dollars[12]. $|S_U|$ is no more than the number of dollars paid. Do the above for all $U \in \mathcal{C}$. Each edge in G can receive $\tilde{O}(1)$ dollars in total because every vertex appears in $\tilde{O}(1)$ subsets in $\mathcal{C}$ (Property P4-1 of $\mathcal{C}$; see Section 4). □

[12]Two is possible: this happens when b and u are both black and both appear in U.

For any distinct $U, U' \in \mathcal{C}$, define $\mathit{size}(S_U, S_{U'})$ as in (7). On the other hand, for each $U \in \mathcal{C}$, define

$$\mathit{size}(S_U, S_U) = \sum_{b \in S_U} \binom{\mathit{weight}_{S_U}(b)}{2}.$$

We store the value $\mathit{size}(S_U, S_U)$ for all U. The total space is $\tilde{O}(m^2/\lambda^2)$.

Before proceeding, the reader should note the following subtle fact about the function $\mathit{size}(.,.)$:

Fact B-1: $\mathit{size}(S_U, S_{U'})$ is the number of occurrences $\mathit{wedge}(u, v, w)$ in G such that $u \in S_U$, $w \in S_{U'}$, and v is black.

The fact holds even if $U = U'$.

Query. Given a query with interval q, in $\tilde{O}(1)$ time we can pick $h = \tilde{O}(1)$ members $U_1, ..., U_h$ from $\mathcal{C}$ that form a partition of V_q (Property P4-2 of $\mathcal{C}$). The query returns

$$\sum_{i,j \in [1,h] : i \le j} \mathit{size}(S_{U_i}, S_{U_j}). \tag{8}$$

Each $\mathit{size}(S_{U_i}, S_{U_j})$ is either explicitly stored or can be obtained from the weighted set intersection size structure in $O(\lambda)$ time. The overall query time is therefore $\tilde{O}(\lambda)$.

Fact B-1 and $U_1, ..., U_h$ forming a partition of V_q assure us that (8) counts only occurrences $\mathit{wedge}(u, v, w)$ in G such that $A_u \in q$, $A_w \in q$, and v is black. To complete the correctness argument, we still need to show that (8) counts every such occurrence exactly once. Indeed, there exist unique $a, b \in [1, h]$ such that $a \le b$, $u \in U_a$, and $w \in U_b$. The wedge is counted only by the term in (8) with $i = a$ and $j = b$.

C Proof of Lemma 5.2

Let us first review Hölder's Inequality. Fix some positive integers α and β. Let

- $x_{i,j}$, for each $i \in [1, \alpha]$ and $j \in [1, \beta]$, be non-negative real numbers;
- y_j, for each $j \in [1, \beta]$, be non-negative real numbers satisfying $\sum_{j=1}^{\beta} y_j \ge 1$.

Under the convention $0^0 = 0$, Hölder's inequality states that:

$$\sum_{i=1}^{\alpha} \prod_{j=1}^{\beta} x_{i,j}^{y_j} \le \prod_{j=1}^{\beta} \left(\sum_{i=1}^{\alpha} x_{i,j} \right)^{y_j}. \tag{9}$$

A proof can be found in [29].

We now return to the context of Lemma 5.2. Given any $j \in [1, d-1]$ and $(I_1, I_2, ..., I_j) \in \boldsymbol{\mathcal{I}}_1 \times ... \boldsymbol{\mathcal{I}}_j$, we will use $B(I_1, I_2, ..., I_j)$ as a short-form for the d-dimensional box

$$B(I_1, ..., I_j, \mathit{dom}(X_{j+1}), ..., \mathit{dom}(X_d)).$$

As a special case, define $B(\emptyset) = B(\mathit{dom}(X_1), ..., \mathit{dom}(X_d))$.

Lemma C.1. *For any $j \in [1, d]$, we have*

$$\sum_{I_j \in \boldsymbol{\mathcal{I}}_j} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \leq \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_{j-1})|^{W(e)}.$$

Proof. Define

$$\mathcal{E}_j = \{e \in \mathcal{E} \mid X_j \in e\}.$$

Since $\sum_{e \in \mathcal{E}_j} W(e) \geq 1$ (W is a fractional edge covering), from Hölder's inequality (9) we have

$$\begin{aligned}
& \sum_{I_j \in \boldsymbol{\mathcal{I}}_j} \prod_{e \in \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \\
\leq \; & \prod_{e \in \mathcal{E}_j} \Big(\sum_{I_j \in \boldsymbol{\mathcal{I}}_j} |R_e \ltimes B(I_1, ..., I_j)| \Big)^{W(e)} \\
\leq \; & \prod_{e \in \mathcal{E}_j} \Big| R_e \ltimes B\big(I_1, ..., I_{j-1}, dom(X_j)\big) \Big|^{W(e)} \\
= \; & \prod_{e \in \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_{j-1})|^{W(e)} \qquad (10)
\end{aligned}$$

where the second inequality used the fact that $\boldsymbol{\mathcal{I}}_j$ is a set of disjoint intervals in $dom(X_j)$.

For each $e \in \mathcal{E} \setminus \mathcal{E}_j$, $R_e \ltimes B(I_1, ..., I_j)$ does not depend on I_j and can be rewritten as $R_e \ltimes B(I_1, ..., I_{j-1})$. We can thus derive:

$$\begin{aligned}
& \sum_{I_j \in \boldsymbol{\mathcal{I}}_j} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \\
= & \sum_{I_j \in \boldsymbol{\mathcal{I}}_j} \Big(\prod_{e \in \mathcal{E} \setminus \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \cdot \prod_{e \in \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \Big) \\
= & \prod_{e \in \mathcal{E} \setminus \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \cdot \sum_{I_j \in \boldsymbol{\mathcal{I}}_j} \prod_{e \in \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_j)|^{W(e)} \\
\leq & \prod_{e \in \mathcal{E} \setminus \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_{j-1})|^{W(e)} \cdot \prod_{e \in \mathcal{E}_j} |R_e \ltimes B(I_1, ..., I_{j-1})|^{W(e)} \\
= & \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_{j-1})|^{W(e)}.
\end{aligned}$$

where the inequality used (10). □

We can prove Lemma 5.2 with d applications of Lemma C.1:

$$\begin{aligned}
& \sum_{I_1 \in \boldsymbol{\mathcal{I}}_1} \dots \sum_{I_d \in \boldsymbol{\mathcal{I}}_d} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_d)|^{W(e)} \\
\le \;& \sum_{I_1 \in \boldsymbol{\mathcal{I}}_1} \dots \sum_{I_{d-1} \in \boldsymbol{\mathcal{I}}_{d-1}} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_{d-1})|^{W(e)} \\
\le \;& \sum_{I_1 \in \boldsymbol{\mathcal{I}}_1} \dots \sum_{I_{d-2} \in \boldsymbol{\mathcal{I}}_{d-2}} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1, ..., I_{d-2})|^{W(e)} \\
\le \;& \dots \\
\le \;& \sum_{I_1 \in \boldsymbol{\mathcal{I}}_1} \prod_{e \in \mathcal{E}} |R_e \ltimes B(I_1)|^{W(e)} \\
\le \;& \prod_{e \in \mathcal{E}} |R_e|^{W(e)}.
\end{aligned}$$

D Proof of Theorem 1.5

The reader should read this proof after having finished Section 5. The basic idea is to convert Problem 2 to range join. Let $\mathcal{X}$ (resp. $\mathcal{E}$) be the set of vertices (resp. edges) in the pattern graph Q. The reader should not confuse $\mathcal{X}$ and $\mathcal{E}$ with V and E: the latter two are defined on the data graph G. For each edge $e \in \mathcal{E}$, construct a relation R_e with two attributes by inserting, for each edge $\{u, v\}$ in G, two tuples (u, v) and (v, u). This defines a join instance $\mathcal{R} = \{R_e \mid e \in \mathcal{E}\}$ with input size $N = 2m \cdot |\mathcal{E}| = O(m)$.

Every occurrence of Q corresponds to a constant number of tuples in *join*$(\mathcal{R})$. Motivated by this, given a Problem-2 query with interval q, we issue a range join query on $\mathcal{R}$ with q, which guarantees retrieving all the occurrences. The issue, however, is that not every tuple in *join*$(\mathcal{R})$ gives rise to an occurrence. To see this, consider Q = 4-cycle and, hence, $\mathcal{R}$ has four relations with schemes (X_1, X_2), (X_2, X_3), (X_3, X_4), and (X_4, X_1), respectively. Let $\{u, v\}$ be an arbitrary edge in E; tuples (u, v), (v, u), (u, v), and (v, u) exist in the four relations, respectively. Thus, *join*$(\mathcal{R})$ contains a tuple (u, v, u, v) that does not correspond to any occurrence.

The issue can be eliminated by slightly modifying the structure of [20], which we review next. Consider an arbitrary set $\mathcal{R}$ of relations (with any number of attributes) defined in Section 5. Deep and Koutris [20] proved the existence of a set $\mathcal{B}$ of boxes such that:

- each box has the form $B(I_1, ..., I_d)$ where I_i is an interval in $dom(X_i)$ for $i \in [1, d]$;
- the boxes are disjoint and their union is $B(dom(X_1),\ dom(X_2),\ ..., dom(X_d))$;
- for each box $B(I_1, ..., I_d)$, the join instance $\mathcal{R}_{I_1,...,I_d}$ has a non-empty result;
- each box $B(I_1, ..., I_d)$ satisfies $\text{AGM}(I_1, ..., I_d) \le \Delta$;
- $|\mathcal{B}| = O(N^{\rho^*}/\Delta)$.

The structure of [20] simply stores $\mathcal{B}$ itself and uses $O(N^{\rho^*}/\Delta)$ space[13]. To enumerate *join*$(\mathcal{R})$, the algorithm of [20] looks at each $B(I_1, ..., I_d) \in \mathcal{B}$ and applies a worst-case optimal join algorithm [39,40,48] to compute *join*$(\mathcal{R}_{I_1,...,I_d})$ in $\tilde{O}(\text{AGM}(I_1, ..., I_d)) = \tilde{O}(\Delta)$ time. This guarantees a delay of $\tilde{O}(\Delta)$.

[13]Obviously, the relations of $\mathcal{R}$ also need to be stored separately.

We now adapt the structure to list all the occurrences of Q in G (fixing $q = (-\infty, \infty)$). Construct $\mathcal{R}$ from G and Q as before. Apply [20] to find a set $\mathcal{B}$ with all the properties explained earlier. Then, inspect each box $B(I_1, ..., I_d) \in \mathcal{B}$ in turn and remove it from $\mathcal{B}$ if all the occurrences of Q producible from $join(\mathcal{R}_{I_1,...,I_d})$ can already be produced from the boxes inspected earlier. The size of $\mathcal{B}$ can only decrease and therefore is still bounded by $O(N^{\rho^*}/\Delta)$. To find the occurrences, apply a worst-case optimal join algorithm on each box in $\mathcal{B}$. As each box generates at least one new occurrence, we guarantee a delay of $\tilde{O}(\Delta)$.

To support (Problem-2) queries with arbitrary q, use the adapted structure to replace that of [20] in the solution presented in Section 5.2. All the analysis still holds through. We thus complete the proof of Theorem 1.5.

References

[1] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[4] Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

[5] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic*, pages 208–222, 2007.

[6] Michael J Bannister, Christopher DuBois, David Eppstein, and Padhraic Smyth. Windows into relational events: Data structures for contiguous subsequences of edges. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 856–864, 2013.

[7] Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters (IPL)*, 8(5):244–251, 1979.

[8] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2315–2332, 2021.

[9] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 303–318, 2017.

[10] Andreas Bjorklund, Petteri Kaski, and Lukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. *ACM Transactions on Algorithms*, 13(4):48:1–48:26, 2017.

[11] Andreas Bjorklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 223–234, 2014.

[12] Nofar Carmeli and Markus Kroll. On the enumeration complexity of unions of conjunctive queries. *ACM Transactions on Database Systems (TODS)*, 46(2):5:1–5:41, 2021.

[13] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 393–409, 2020.

[14] Farah Chanchary and Anil Maheshwari. Time windowed data structures for graphs. *J. Graph Algorithms Appl.*, 23(2):191–226, 2019.

[15] Farah Chanchary, Anil Maheshwari, and Michiel Smid. Querying relational event graphs using colored range searching data structures. *Discrete Applied Mathematics*, 286:51–61, 2020.

[16] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988.

[17] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985.

[18] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 210–223, 2017.

[19] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[20] Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 307–322, 2018.

[21] Arnaud Durand. Fine-grained complexity analysis of queries: From decision to counting and enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 331–346, 2020.

[22] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007.

[23] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 121–131, 2014.

[24] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters (IPL)*, 51(4):207–211, 1994.

[25] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999.

[26] David Eppstein, Maarten Loffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 6506, pages 403–414, 2010.

[27] Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. *SIAM J. Discret. Math.*, 29(3):1322–1339, 2015.

[28] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences (JCSS)*, 78(3):698–706, 2012.

[29] Ehud Friedgut. Hypergraphs, entropy, and inequalities. *Am. Math. Mon.*, 111(9):749–760, 2004.

[30] Pierre-Louis Giscard, Nils M. Kriege, and Richard C. Wilson. A general purpose algorithm for counting simple cycles and simple paths of any length. *Algorithmica*, 81(7):2716–2737, 2019.

[31] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures Workshop (WADS)*, pages 421–436. Springer, 2017.

[32] Isaac Goldstein, Moshe Lewenstein, and Ely Porat. On the hardness of set disjointness and set intersection with bounded universe. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 7:1–7:22, 2019.

[33] Chinh T. Hoang, Marcin Kaminski, Joe Sawada, and R. Sritharan. Finding and listing induced paths and cycles. *Discrete Applied Mathematics*, 161(4-5):633–641, 2013.

[34] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 375–392, 2020.

[35] Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 297–308, 2013.

[36] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 213–228, 2015.

[37] Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters (IPL)*, 74(3-4):115–121, 2000.

[38] Jaroslav Nesetril and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[39] Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018.

[40] Hung Q. Ngo, Christopher Re, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013.

[41] Dan Olteanu and Jakub Zavodny. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):2:1–2:44, 2015.

[42] Saladi Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015.

[43] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 151–163, 2018.

[44] Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Rec.*, 44(1):10–17, 2015.

[45] Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 68, pages 20:1–20:16, 2017.

[46] Yufei Tao. Algorithmic techniques for independent query sampling. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2022.

[47] Yufei Tao and Ke Yi. Intersection joins under updates. *Journal of Computer and System Sciences (JCSS)*, 124:41–64, 2022.

[48] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

[49] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[50] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013.