Towards Optimal Set-Disjointness and Set-Intersection Data Structures

Tsvi Kopelowitz
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kopelot@gmail.com

Virginia Vassilevska Williams
EECS Department, MIT, Cambridge, MA, USA
virgi@mit.edu

Abstract

In the online *set-disjointness* problem the goal is to preprocess a family of sets $\mathcal{F}$, so that given two sets $S, S' \in \mathcal{F}$, one can quickly establish whether the two sets are disjoint or not. If $N = \sum_{S \in \mathcal{F}} |S|$, then let N^p be the preprocessing time and let N^q be the query time. The most efficient known combinatorial algorithm is a generalization of an algorithm by Cohen and Porat [TCS'10] which has a tradeoff curve of $p + q = 2$. Kopelowitz, Pettie, and Porat [SODA'16] showed that, based on the 3SUM hypothesis, there is a conditional lower bound curve of $p + 2q \geq 2$. Thus, the current state-of-the-art exhibits a large gap.

The online *set-intersection* problem is the reporting version of the online set-disjointness problem, and given a query, the goal is to report all of the elements in the intersection. When considering algorithms with N^p preprocessing time and $N^q + O(op)$ query time, where op is the size of the output, the combinatorial algorithm for online set-disjointess can be extended to solve online set-intersection with a tradeoff curve of $p + q = 2$. Kopelowitz, Pettie, and Porat [SODA'16] showed that, assuming the 3SUM hypothesis, for $0 \leq q \leq 2/3$ this curve is tight. However, for $2/3 \leq q < 1$ there is no known lower bound.

In this paper we close both gaps by showing the following:

- For online set-disjointness we design an algorithm whose runtime, assuming $\omega = 2$ (where ω is the exponent in the fastest matrix multiplication algorithm), matches the lower bound curve of Kopelowitz et al., for $q \leq 1/3$. We then complement the new algorithm by a matching conditional lower bound for $q > 1/3$ which is based on a natural hypothesis on the time required to detect a triangle in an unbalanced tripartite graph. Remarkably, even if $\omega > 2$, the algorithm matches the lower bound curve of Kopelowitz et al. for $p \geq 1.73688$ and $q \leq 0.13156$.
- For set-intersection, we prove a conditional lower bound that matches the combinatorial upper bound curve for $q \geq 1/2$ which is based on a hypothesis on the time required to enumerate all triangles in an unbalanced tripartite graph.
- Finally, we design algorithms for detecting and enumerating triangles in unbalanced tripartite graphs which match the lower bounds of the corresponding hypotheses, assuming $\omega = 2$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Set-disjointness data structures, Triangle detection, Triangle enumeration, Fine-grained complexity, Fast matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.74

Category Track A: Algorithms, Complexity and Games

Funding *Tsvi Kopelowitz*: Tsvi Kopelowitz is supported by ISF grant no. 1926/19, and by BSF grant no. 2018364.
Virginia Vassilevska Williams: Virginia Vassilevska Williams is supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

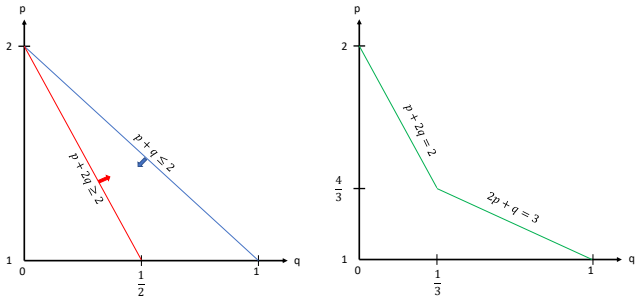

Figure 1 The left graph depicts the gap between the upper bound and lower bound curves for online SetDisjointness prior to this work. The lower bound tradeoff (red curve) is based on the 3SUM hypothesis and the upper bound tradeoff (blue curve) is a variation of the algorithm of Cohen and Porat [14]. The right graph depicts the optimal online SetDisjointness tradeoff (green curve) shown in this paper, assuming that $\omega = 2$.

1 Introduction

In the *online SetDisjointness* problem the goal is to preprocess a family F of subsets from universe U such that given a query pair $(S, S') \in \mathcal{F} \times \mathcal{F}$, one can quickly establish whether S and S' are disjoint or not. The *online SetIntersection* problem is the reporting version of the online SetDisjointness problem, where given a query pair $(S, S') \in \mathcal{F} \times \mathcal{F}$ one must enumerate all of the elements in $S \cap S'$.

Set-disjointness problems at large, including both online SetDisjointness and online SetIntersection, are fundamental algorithmic problems, and have many applications, for example, in information retrieval [14, 24, 20], graph related problems [28, 5, 29, 33, 34], and data structures [27, 16, 29]. Moreover, both problems have played a crucial role in obtaining conditional lower bounds (CLB) in fine-grained complexity. Specifically, many CLBs that are based on the 3SUM hypothesis[1] are reductions from 3SUM to versions of SetDisjointness or SetIntersection, which are further reduced to other algorithmic problems [32, 1, 29, 27, 5, 4, 22, 23]. The Boolean matrix multiplication (BMM) problem can be interpreted as online SetDisjointness with the requirement that the answers to all of the queries must be computed and stored during the preprocessing phase. Thus, the "combinatorial" BMM hypothesis[2] and the CLBs that follow [1, 37, 10] are closely related to online SetDisjointness. Another example is the orthogonal vectors (OV) hypothesis[3] [1, 37], which can be interpreted as asking whether a given family of sets contains two disjoint sets.

[1] The 3SUM hypothesis states that in the Word RAM model of computation with $O(\log n)$ bit words, determining whether a set of n integers contains three that sum to 0, requires $n^{2-o(1)}$ time.

[2] This hypothesis roughly states that algorithms for $n \times n$ BMM that are simple and do not use Strassen-like techniques must take $n^{3-o(1)}$ time in the Word RAM Model with $O(\log n)$ bit words.

[3] The OV hypothesis states that in the Word RAM model with $O(\log n)$ bit words, an algorithm that can decide whether a set of n binary vectors of dimension d contains two orthogonal vectors, must take $n^{2-o(1)}d^{O(1)}$ time.

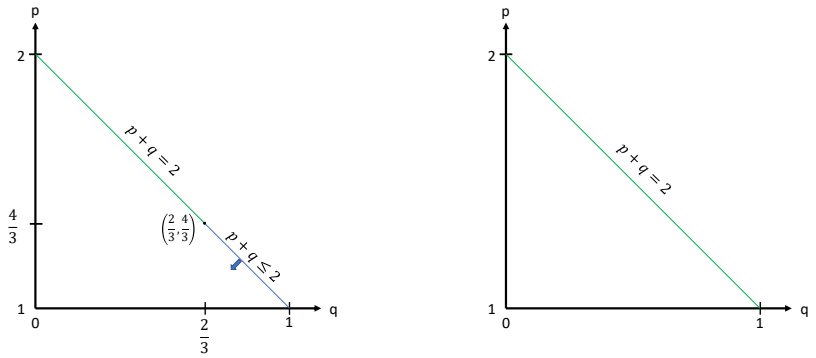

Figure 2 The left graph depicts the gap between the upper bound and lower bound curves for online SetIntersection prior to this work. The green curve (for $q \leq \frac{2}{3}$) is optimal, while the blue curve is an upper bound with no matching lower bound. The right graph depicts the optimal online SetIntersection tradeoff (green curve) shown in this paper, for all values of q.

Measuring efficiency. We measure the efficiency of algorithms for online SetDisjointness and online SetIntersection in terms of $N = \sum_{S \in \mathcal{F}} |S|$. For online SetDisjointness, let N^p and N^q be the preprocessing and query time, respectively. For online SetIntersection, let N^p and $N^q + O(op)$ be the preprocessing and query time, respectively, where op is the size of the output.

When discussing tradeoffs between p and q we make the standard assumption that the preprocessing phase must scan the input at least once, and so $p \geq 1$. Moreover, there is no advantage in allowing $p > 2$ since for $p = 2$ it is straightforward to obtain a constant query time. Thus we assume that $1 \leq p \leq 2$. Similarly, a trivial query algorithm is to scan the entire instance in $O(N)$ time, so we assume that $0 \leq q \leq 1$.

A brief history and the gaps. A variation of the algorithm of Cohen and Porat [14] for online SetDisjointness has $p + q = 2$ (see Section 2.1). A straightforward variation of this algorithm also solves online SetIntersection with $p + q = 2$ (see Section 2.1). To our knowledge, for any values of p and q, there is no published algorithm with a better upper-bound tradeoff.

Regarding lower bounds, assuming the 3SUM hypothesis, Pătraşcu [32] proved that for online SetDisjointness whenever $1 \leq p < 4/3$ we have $q \geq 1/3$. Pătraşcu's CLB is fairly limited with regard to the range of options for p and q. The CLB tradeoff was later improved by Kopelowitz, Pettie and Porat [29] to $p + 2q \geq 2$ for the full range of p and $0 \leq q \leq 1/2$. Kopelowitz et al. [29] also showed that, assuming the 3SUM hypothesis, for online SetIntersection, whenever $4/3 \leq p < 2$ and $0 \leq q \leq 2/3$ we have $p + q \geq 2$. Thus, the combinatorial algorithm is tight for $q \leq \frac{2}{3}$.

In both problems, a large gap remains; see Figures 1 and 2. The goal of this paper is to close the gaps for both problems.

1.1 Our Results

In this paper we take a step towards closing the gaps for both online SetDisjointness and online SetIntersection as follows.

New algorithm for online SetDisjointness. For online SetDisjointness we design an algorithm that utilizes fast matrix multiplication (FMM) (see [15, 36, 35, 31]) and, assuming $\omega = 2$ (where ω is the exponent in the fastest matrix multiplication algorithm), matches the lower bound curve of Kopelowitz et al. [29] for $q \leq 1/3$. The algorithm borrows some ideas from the fast sparse matrix multiplication algorithm of Yuster and Zwick [38], and is stated in the following theorem whose proof appears in Section 2.2.

▶ **Theorem 1.** *There exists an algorithm for the online SetDisjointness problem where*

$$\begin{aligned} &p + \tfrac{2}{\omega-1}q = 2, && \text{for } 0 \leq q \leq \tfrac{\omega-1}{\omega+1} \\ &\tfrac{2}{\omega-1}p + q = 1 + \tfrac{2}{\omega-1}, && \text{for } \tfrac{\omega-1}{\omega+1} \leq q \leq 1 \end{aligned}.$$

If $\omega > 2$, the time bounds of Theorem 1 can be improved using fast rectangular matrix multiplication (FRMM) (see [30]). In particular, if we denote by $\omega(1, 1, k)$ the exponent of n in the time required to multiply an $n \times n^k$ matrix by an $n^k \times n$ matrix, then the following corollary is straightforward from the proof of Theorem 1 (see Section 2.2).

▶ **Corollary 2.** *There exists an algorithm for the online SetDisjointness problem where*

$$p = (1 - q) \cdot \omega(1, 1, \frac{2 - p}{1 - q}).$$

We note that, since $\omega(1, 1, k) = 2$ for $k \leq 0.30298$ [30, 21], for the range of $p \geq \frac{4}{2.30298} = 1.73688$ and $q \leq \frac{0.30298}{2.30298} = 0.13156$, the tradeoff becomes $p + 2q = 2$, which is optimal by the 3SUM conjecture.

Unbalanced triangle detection. We complement our new algorithm with a matching CLB for the case of $q \geq 1/3$ which is based on the problem of detecting a triangle in an unbalanced tripartite graph.

▶ **Problem 3** (Unbalanced Triangle Detection)**.** *In the* Unbalanced Triangle Detection *(UTD) problem the goal is to determine whether an undirected tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle or not, where $m_1 = |E \cap (A \times B)|$, $m_2 = |E \cap (B \times C)|$, $m_3 = |E \cap (C \times A)|$, and $m_1 \leq m_2 \leq m_3$.*

▶ **Hypothesis 4** (UTD hypothesis)**.** *Assuming $\omega = 2$, any algorithm for the UTD problem in the word RAM model with $O(\log m_3)$ bit words, for any $m_1 \leq m_2 \leq m_3$, has time cost*

$$\Omega\left((m_1 \cdot m_2)^{\frac{1}{3}}(m_3)^{\frac{2}{3}-o(1)}\right).$$

The UTD hypothesis is the natural extension of the popular Triangle Detection hypothesis [1, 11, 26, 32] which states that, assuming $\omega = 2$, the current best known running time $O(m^{4/3})$ for triangle detection in m-edge graphs is optimal, up to $m^{o(1)}$ factors . To see this, just set $m_1 = m_2 = m_3$ in the UTD hypothesis and the Triangle Detection Hypothesis is obtained by noting that when it comes to triangle problems, without loss of generality, the input graph is tripartite[4].

[4] The reduction works by creating 3 copies of the vertex set, each of which is an independent set, and placing copies of the original edges between copies of vertices (but each vertex copy is in a different copy of the vertex set). Each triangle in the original graph appears 6 times.

We remark an important subtlety in the UTD hypothesis: in order to disprove the hypothesis, it is enough to design an algorithm that beats the hypothesized lower bound for a single combination of m_1, m_2 and m_3. Nevertheless, the CLBs that we prove based on the UTD hypothesis hold even if we restrict the UTD hypothesis to be true for the restricted cases of $m_2 = m_3$.

UTD algorithm. In Section 3 We design a new algorithm for UTD which matches the lower bounds of the UTD hypothesis if $\omega = 2$. The algorithm is a natural (albeit not exactly straightforward) extension of the best known algorithms for triangle detection [3].

▶ **Theorem 5.** *There exists an algorithm for the UTD problem whose time cost is*

$$O\left(m_3 + (m_1 \cdot m_2)^{\frac{\omega-1}{\omega+1}} (m_3)^{\frac{2}{\omega+1}}\right).$$

CLB for online SetDisjointness. In Section 4 we prove a CLB for online SetDisjointness which is conditioned on the UTD hypothesis. The CLB, which matches the upper bound of Theorem 1 for $q \geq \frac{1}{3}$, assuming $\omega = 2$, is summarized in the following theorem.

▶ **Theorem 6.** *Assuming $\omega = 2$, any algorithm for online SetDisjointness that has $\frac{1}{3} \leq q < 1$ must obey $2p + q \geq 3$, unless the UTD hypothesis is false.*

Assuming that $\omega = 2$, Theorems 6 and 1 combined with the 3SUM CLB of Kopelowitz et al. [29] provide a (conditionally) optimal curve, as depicted in Figure 1.

Unbalanced triangle enumeration. For set-intersection, we prove a conditional lower bound that matches the combinatorial upper bound curve for $q \geq 1/2$ which is based on a hypothesis on the time required to enumerate all triangles in an unbalanced tripartite graph.

▶ **Problem 7** (Unbalanced Triangle Enumeration)**.** *In the* Unbalanced Triangle Enumeration *(UTE) problem the goal is to enumerate all triangles in a given undirected tripartite graph $G = (A \cup B \cup C, E)$, where $m_1 = |E \cap (A \times B)|$, $m_2 = |E \cap (B \times C)|$, $m_3 = |E \cap (C \times A)|$, and $m_1 \leq m_2 \leq m_3$.*

▶ **Hypothesis 8** (UTE hypothesis)**.** *Assuming $\omega = 2$, any algorithm for the UTE problem on a graph with t triangles must have time cost $t^{\frac{1}{3}}(m_1 m_2 m_3)^{\frac{1}{3}} m_3^{-o(1)}$ in the word RAM model with $O(\log m_3)$ bit words.*

Similar to the UTD hypothesis, the UTE hypothesis implies that, assuming $\omega = 2$, the current best known running time of $O(m + m^{\frac{4}{3}} + t^{\frac{1}{3}} m)$ for triangle enumeration in m-edge graphs by Bjørklund et al. [9] is optimal, up to $m^{o(1)}$ factors (just set $m_1 = m_2 = m_3$).

UTE algorithm. In Section 5 we design a new algorithm for UTE which, assuming $\omega = 2$, matches the lower bounds of the UTE hypothesis. The algorithm is a natural (albeit not exactly straightforward) extension of the best known algorithms for output-sensitive triangle enumeration [9].

▶ **Theorem 9.** *There exists an algorithm for UTE on graphs with at most t triangles whose time cost is*

$$\tilde{O}\left(m_3 + m_3^{\frac{2}{\omega+1}} (m_1 m_2)^{\frac{\omega-1}{\omega+1}} + t^{\frac{3-\omega}{\omega+1}} (m_1 m_2 m_3)^{\frac{\omega-1}{\omega+1}}\right).$$

CLB for online SetIntersection In Section 6, we prove the following CLB for SetIntersection based on the UTE hypothesis, which matches the algorithm of Section 2.1 for $q \geq 1/2$.

▶ **Theorem 10.** *Any algorithm for online SetIntersection that has* $\frac{1}{2} \leq q < 1$ *must obey* $p + q \geq 2$*, unless the UTE hypothesis is false.*

1.2 More Related Work

SetDisjointness and SetIntersection. Many existing set intersection data structures, e.g., [17, 7, 6], work in the comparison model in which sets are represented as sorted lists or arrays. The benchmark in this model is the minimum number of comparisons needed to answer a query. Bille, Pagh, and Pagh [8] used word-packing techniques to evaluate expressions of set intersections and unions. Their query algorithm finds the intersection of k sets with a total of n elements in $O(n/\frac{w}{\log^2 w} + k \cdot op)$ time, where op is the size of the output and w is the size of a machine word. Cohen and Porat [13] designed a *static* $O(N)$-space data structure for answering online SetIntersection queries in $O(\sqrt{N(1 + |S \cap S'|)})$ time. Kopelowitz, Porat and Pettie [28] designed an incremental algorithm for online SetDisjointness where both queries and element insertions into sets cost $O(\sqrt{\frac{N}{\log n/\log\log n}})$ time.

Kopelowitz, Porat and Pettie [28] also designed a fully dynamic algorithm for both online SetDisjointness and online SetIntersection which uses M words of space, each update costs $O(\sqrt{M \log N})$ expected time, each SetIntersection query costs $O(\frac{N\sqrt{\log N}}{\sqrt{M}}\sqrt{op+1})$ expected time where op is the size of the output, and each online SetDisjointness query costs $O(\frac{N\sqrt{\log N}}{\sqrt{M}} + \log N)$ expected time. The relationship between the space usage and query time was also investigated by Afshani and Neilsen [2] and Goldstein et al. [23].

Triangle Enumeration. Itai and Rodeh [25] showed that all t triangles in a graph could be enumerated in $O(m^{3/2})$ time. Thirty years ago Chiba and Nishizeki [12] generalized [25] to show that $O(m\alpha)$ time suffices, where α is the *arboricity* of the graph. Kopelowitz, Pettie, and Porat [28] proved that enumerating t triangles takes $O(m\lceil\alpha/\frac{\log n}{\log\log n}\rceil + t)$ time. Eppstein et al. [19] designed an algorithm for the w-bit word RAM model running in $O(m\lceil\alpha/\frac{w}{\log w}\rceil + t)$ time.

The fastest algorithm for triangle enumeration in general m-edge, n-node graphs is the algorithm of Bjørklund, Pagh, Williams, and Zwick [9] which if $\omega = 2$, runs in $\tilde{O}(\min\{n^2 + nt^{2/3}, m^{4/3} + mt^{1/3}\})$ time.

Duraj et al. [18] showed that the related problem of establishing for each edge e in a graph the number of triangles that contain e is equivalent to several natural range query problems.

2 SetDisjointness Algorithms

Before we describe our new algorithm, we begin by presenting a simple algorithm for online SetDisjointness which is a variation of the algorithm of Cohen and Porat [14], and has a efficiency tradeoff of $p + q = 2$. This algorithm is a building block for our new algorithm.

2.1 Heavy-Light Decomposition

▶ **Lemma 11.** *There exists an algorithm for the online SetDisjointness problem with* $p+q = 2$.

Proof. Let $0 \leq \alpha \leq 1$. A set S is said to be *light* if $|S| \leq N^\alpha$, and *heavy* otherwise. Notice that the number of heavy sets is at most $N^{1-\alpha}$.

The algorithm stores each set via a lookup table, and precomputes the answers to all $O(N^{2-2\alpha})$ heavy pairs (pairs of heavy sets). Specifically, for a heavy pair S and S', the algorithm checks for each element $e \in S$ whether $e \in S'$. The sets S and S' are disjoint if and only if all of the tests fail. Notice that during the precomputation, an element in a heavy set is looked up at most $N^{1-\alpha}$ times, once for each heavy set. Since the number of elements in all heavy sets is at most N, the total preprocessing cost is $O(N^{2-\alpha})$ time and $p = 2 - \alpha$.

For the query, if both of the queries sets are heavy then the answer is obtained from the precomputed information, and if at least one query set is light then the algorithm scans the at most N^{α} elements in the light set to test whether any of these elements are in the other set (regardless of whether the other set is heavy or light). Thus, the query cost is $O(N^{\alpha})$ and $q = \alpha$. Finally, $p + q = 2 - \alpha + \alpha = 2$ as required. ◀

SetIntersection Algorithm. It is fairly straightforward to convert the algorithm of Lemma 11 to also solve online SetIntersection with $p + q = 2$: for each pair of heavy sets, instead of storing only an indication of whether the sets are disjoint or not the algorithm stores the entire intersection.

2.2 Improved algorithm

▶ **Theorem 1.** *There exists an algorithm for the online SetDisjointness problem where*

$$\begin{array}{ll} p + \frac{2}{\omega-1}q = 2, & \text{for } 0 \le q \le \frac{\omega-1}{\omega+1} \\ \frac{2}{\omega-1}p + q = 1 + \frac{2}{\omega-1}, & \text{for } \frac{\omega-1}{\omega+1} \le q \le 1 \end{array}.$$

Proof. The algorithm is similar to the algorithm in the proof of Lemma 11, but with a faster method for precomputing all of the answers for heavy pairs. Thus, assume without loss of generality that there are at most $N^{1-\alpha}$ sets, where α is taken from the proof of Lemma 11.

For every element $e \in U$, let $f_e = |\{S \in \mathcal{F} : e \in S\}|$ be the number of sets in $\mathcal{F}$ that contain e. For a parameter $0 \le \beta \le 1$, an element $e \in U$ is said to be *frequent* if $f_e > N^{\beta}$, and *rare* otherwise. Let F be the set of frequent elements and let R be the set of rare elements. Notice that $|F| \le N^{1-\beta}$ and $\sum_{e \in R} f_e \le N$.

For each rare element e, there are at most $O((f_e)^2) = O(N^{2\beta})$ heavy pairs that contain e in their intersection, and enumerating these pairs costs $O((f_e)^2)$ time. In order to efficiently enumerate these pairs for all rare elements, the algorithm computes for each element e a list of sets that contain e by scanning the entire instance in linear time. Given these lists, the algorithm enumerates all of the heavy pairs that have a rare element in their intersection in $O(\sum_{e \in R}(f_e)^2) = O(\sum_{e \in R} N^{\beta} f_e) = O(N^{\beta} \sum_{e \in R} f_e) = O(N^{1+\beta})$ time.

The algorithm is now left with the task of establishing which heavy pairs have at least one frequent element in their intersection. However, enumerating all heavy pairs that have a frequent element in their intersection is too expensive. In order to reduce the time cost, the algorithm constructs a Boolean matrix M such that the columns of M correspond to frequent elements and the rows of M correspond to characteristic vectors of the heavy sets after removing all of the rare elements. Thus, the size of M is $|H| \times |F|$ where H is the set of heavy sets. Let $P = M \cdot M^T$ where the product is a Boolean product. Notice that $p_{i,j} = 1$ if and only if there exists an element e such that the both $m_{i,e} = 1$ and $m^T_{e,j} = m_{j,e} = 1$. Thus, the non-zero entries of P exactly correspond to the heavy pairs that have a frequent element in their intersection. The time cost of computing P using FMM is

$$O\left(\frac{|H|^2|F|}{\min(|H|,|F|)^{3-\omega}}\right) = O\left(N^{\omega-2\alpha-\beta+(3-\omega)(\max(\alpha,\beta))}\right).$$

Thus, the total preprocessing time is $N^p = O\left(N^{1+\beta} + N^{\omega-2\alpha-\beta+(3-\omega)(\max(\alpha,\beta))}\right)$, which is minimized whenever $1+\beta = \omega - 2\alpha - \beta + (3-\omega)(\max(\alpha,\beta))$, and then $p = 1+\beta$. Recall that the query time is $O(N^\alpha)$ and so $q = \alpha$.

If $\alpha \leq \beta$, then $\beta = 1 - \frac{2\alpha}{\omega-1}$, the preprocessing time is given by $p = 1+\beta = 2 - \frac{2q}{\omega-1}$, and so $p + \frac{2}{\omega-1}q = 2$. Notice that in order for this case to hold, it must be that $\alpha \leq \beta = 1 - \frac{2\alpha}{\omega-1}$ implying that $q = \alpha \leq \frac{\omega-1}{\omega+1}$.

If $\alpha > \beta$, then $\alpha = 1 - \frac{2\beta}{\omega-1}$, the query time is given by $q = \alpha = 1 - \frac{2(p-1)}{\omega-1}$, and so $\frac{2}{\omega-1}p + q = 1 + \frac{2}{\omega-1}$. Notice that in order for this case to hold, it must be that $\alpha = 1 - \frac{2\beta}{\omega-1} \geq 1 - \frac{2\alpha}{\omega-1}$ implying that $q = \alpha \geq \frac{\omega-1}{\omega+1}$. ◀

▶ **Corollary 2.** *There exists an algorithm for the online SetDisjointness problem where*

$$p = (1-q)\cdot\omega(1,1,\frac{2-p}{1-q}).$$

Proof. When using FRMM, the cost to compute P is $O\left(|H|^{\omega(1,1,\log_{|H|}\frac{|F|}{|H|})}\right) = O\left(N^{(1-\alpha)\cdot\omega(1,1,\frac{1-\beta}{1-\alpha})}\right)$ time. Thus, the total preprocessing time is $N^p = O\left(N^{1+\beta} + N^{(1-\alpha)\cdot\omega(1,1,\frac{1-\beta}{1-\alpha})}\right)$, which is minimized whenever $1+\beta = (1-\alpha)\cdot\omega(1,1,\frac{1-\beta}{1-\alpha})$, and so $p = (1-q)\cdot\omega(1,1,\frac{2-p}{1-q})$. ◀

3 Unbalanced Triangle Detection Algorithm

▶ **Theorem 5.** *There exists an algorithm for the UTD problem whose time cost is*

$$O\left(m_3 + (m_1\cdot m_2)^{\frac{\omega-1}{\omega+1}}(m_3)^{\frac{2}{\omega+1}}\right).$$

Proof. The algorithm uses three positive integer parameters to be set later: τ_A, τ_B, and τ_C. A vertex $a \in A$ is said to be light if the number of edges $(a,b) \in E \cap (A \times B)$ is at most τ_A, and heavy otherwise. Thus, the number of heavy nodes in A is at most $\frac{m_1}{\tau_A}$. A vertex $b \in B$ is said to be light if the number of edges $(b,c) \in E \cap (B \times C)$ is at most τ_B, and heavy otherwise. Thus, the number of heavy nodes in B is at most $\frac{m_2}{\tau_B}$. A vertex $c \in C$ is said to be light if the number of edges $(c,a) \in E \cap (C \times A)$ is at most τ_C, and heavy otherwise. Thus, the number of heavy nodes in C is at most $\frac{m_3}{\tau_C}$.

Light vertices. For each light $a \in A$ the algorithm enumerates all pairs of edges touching a where one edge touches a vertex in B and the other edge touches a vertex in C, and for each such pair the algorithm tests (in constant time) whether the pair is part of a triangle. If there exists a triangle that contains a light $a \in A$ then one of the enumerated pairs must be two edges from this triangle and thus the algorithm will detect this triangle. Let $d_B(a)$ be the number of edges of a whose other endpoint is in B, and let $d_C(a)$ be the number of edges of a whose other endpoint is in C. Thus, the total number of pairs for a given $a \in A$ is $d_B(a)\cdot d_C(a)$. Notice that $\sum_{a\in A} d_C(a) = m_3$, and recall that since a is light then $d_B(a) \leq \tau_A$. Thus, the time cost for testing whether there exists a triangle with a light $a \in A$ vertex is

$$O(\sum_{a\in A} d_B(a)\cdot d_C(a)) \leq O(\tau_A \sum_{a\in A} d_C(a)) = O(\tau_A \cdot m_3).$$

Similarly, the algorithm checks whether there is exists a triangle with a light $b \in B$ or a light $c \in C$ in $O(\tau_B\cdot m_1 + \tau_C \cdot m_2)$. Thus, the total cost of detecting whether there exists a triangle with at least one light vertex is $O(m_3 + \tau_A\cdot m_3 + \tau_B\cdot m_1 + \tau_C\cdot m_2)$ time, where the first m_3 term comes for the necessity of scanning the entire graph.

Heavy vertices. If there is no triangle that contains at least one light vertex, then there can only be a triangle with three heavy vertices. Here the algorithm utilizes the upper bound on the number of heavy vertices in each part of the tripartite graph. Without loss of generality, let $a_1, a_2, \ldots, a_{\frac{m_1}{\tau_A}}$ be the set of heavy vertices in A, let $b_1, b_2, \ldots, b_{\frac{m_2}{\tau_B}}$ be the set of heavy vertices in B, and let $c_1, c_2, \ldots, c_{\frac{m_3}{\tau_C}}$ be the set of heavy vertices in A. Let L be a $\frac{m_1}{\tau_A} \times \frac{m_2}{\tau_B}$ Boolean matrix where $q_{i,j} = 1$ if and only if $(a_i, b_j) \in E$. Similarly, let R be a $\frac{m_2}{\tau_B} \times \frac{m_3}{\tau_C}$ Boolean matrix where $r_{i,j} = 1$ if and only if $(b_i, c_j) \in E$, and let T be a $\frac{m_1}{\tau_A} \times \frac{m_3}{\tau_C}$ Boolean matrix where $t_{i,j} = 1$ if and only if $(a_i, c_j) \in E$.

The algorithm computes $Z = (L \cdot R) \bigwedge T$ where the first operator is a BMM and the second operator is an entry-wise AND.

▷ Claim 12. $Z \neq 0$ if and only if there exists a triangle in G whose vertices are all heavy.

Proof. Let $X = L \cdot R$. If there exists an entry $z_{i,j} = 1$ then $x_{i,j} = t_{i,j} = 1$. Since $t_{i,j} = 1$, then by definition $(a_i, c_j) \in E$. Since $x_{i,j} = 1$, then there must exist an integer $1 \leq k \leq \frac{m_2}{\tau_B}$ such that $l_{i,k} = 1$ and $r_{k,j} = 1$, and so $(a_i, b_k), (b_k, c_j) \in E$, implying that the triangle (a_i, b_k, c_j) is in G, and all of the vertices of this triangle are heavy.

For the other direction, suppose that the triangle (a_i, b_k, c_j) is in G, and all of the vertices of this triangle are heavy. Then in particular $l_{i,k} = r_{k,i} = t_{i,j} = 1$. Thus, it must be that $x_{i,j} = 1$ and so $z_{i,j} = 1$. ◁

The cost of computing Z is dominated by the cost of computing the BMM of L and R, which is

$$O\left(\frac{m_1 m_2 m_3}{\tau_A \cdot \tau_B \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}}\right).$$

Time cost. The total time cost is

$$O\left(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2 + \frac{m_1 m_2 m_3}{\tau_A \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}}\right).$$

The time cost is minimized when the last four[5] terms in the summation are all equal:

$$\tau_A \cdot m_3 = \tau_B \cdot m_1 = \tau_C \cdot m_2 = \frac{m_1 m_2 m_3}{\tau_A \cdot \tau_B \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}}.$$

Since $m_1 \leq m_2 \leq m_3$ it must be that $\tau_A \leq \tau_C \leq \tau_B$, and so $\frac{m_2}{\tau_B} \leq \frac{m_3}{\tau_C}$. Moreover, $m_2 \tau_A \leq m_2 \tau_C = m_1 \tau_B$ and so $\frac{m_2}{\tau_B} \leq \frac{m_1}{\tau_A}$. Thus, $\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}) = \frac{m_2}{\tau_B}$. By plugging in $\tau_B = \frac{m_3}{m_1}\tau_A$ and $\tau_C = \frac{m_3}{m_2}\tau_A$, we have

$$\begin{aligned}\tau_A \cdot m_3 &= \frac{m_1 m_2 m_3}{\tau_A \cdot \tau_B \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}} \\ &= \frac{m_1 m_2 m_3}{\tau_A \cdot \frac{m_3}{m_1}\tau_A \cdot \frac{m_3}{m_2}\tau_A \cdot (\frac{m_2}{\frac{m_3}{m_1}\tau_A})^{3-\omega}} = \frac{(m_1 m_2)^{\omega-1}}{\tau_A^{\omega} \cdot (m_3)^{\omega-2}},\end{aligned}$$

and so $\tau_A = (\frac{m_1 m_2}{m_3})^{\frac{\omega-1}{\omega+1}}$. Finally, the time cost is

$$O(m_3 + \tau_A \cdot m_3) = O(m_3 + (m_1 m_2)^{\frac{\omega-1}{\omega+1}} (m_3)^{\frac{2}{\omega+1}}).$$

[5] The reason for focusing only on the last four terms and not on the first term is that the last four terms contain parameters which we can control, while the first term m_3 is always set.

Notice that with an $O(m_3)$ time preprocessing we can ensure that every vertex in A has at least one edge to B and to C (process the vertices in B and C similarly), by removing any vertex (and its incident edges) without this property. This procedure never removes any triangles, and ensures that $m_1 \geq \max\{|A|, |B|\}$, $m_2 \geq \max\{|C|, |B|\}$, and $m_3 \geq \max\{|A|, |C|\}$. As $m_3 \leq |A| \cdot |C| \leq m_1 \cdot m_2$, $\tau_A \geq 1$. Similarly, $\tau_B, \tau_C \geq 1$, so the thresholds used by the algorithm make sense. ◀

4 Optimal Conditional Lower Bound for SetDisjointness

▶ **Theorem 6.** *Assuming $\omega = 2$, any algorithm for online SetDisjointness that has $\frac{1}{3} \leq q < 1$ must obey $2p + q \geq 3$, unless the UTD hypothesis is false.*

Proof. To prove the theorem we first describe a reduction from the UTD problem to the online SetDisjointness problem by describing an algorithm for UTD that uses an algorithm for online SetDisjointness as a black box. We then show that if the online SetDisjointness algorithm obeys $2p + q = 3 - \epsilon$ for $\frac{1}{3} \leq q \leq 1$, for some constant $\epsilon > 0$, then there exists an algorithm contradicting the UTD Hypothesis.

Reduction from UTD to online SetDisjointness. Given an instance $G = (A \cup B \cup C, E)$ of UTD, for each $x \in A \cup B$ define the set S_x to be the set of vertices from C that are neighbors of x. All of the sets are given as input for the preprocessing phase of the online SetDisjointness algorithm. Notice that the sum of the sizes of the sets is exactly $N = m_2 + m_3 = \Theta(m_3)$, since each edge touching a vertex in C contributes exactly one element to exactly one of the sets. Next, for each of the m_1 edges $(a, b) \in E \cup (A \times B)$, the algorithm performs an online SetDisjointness query on S_a and S_b. If any of the queries returns a false (meaning that the intersection of the two sets is not empty) then the algorithm returns that there is a triangle in G, and otherwise, the algorithm returns that there is no triangle in G.

▷ Claim 13. There exists an edge $(a, b) \in E \cap (A \times B)$ such that $S_a \cap S_b \neq \emptyset$ if and only if G contains a triangle.

Proof. If there exists a triangle $(a, b, c) \in A \times B \times C$ in G, then both S_a and S_b contain c. Thus, $S_a \cap S_b \neq \emptyset$. For the other direction, if there exists an edge $(a, b) \in E \cap (A \times B)$ such that $S_a \cap S_b \neq \emptyset$ then there exists some $c \in S_a \cap S_b$ which implies that $(a, c), (b, c) \in E$, and so (a, b, c) is a triangle in G. ◁

Finally, the time cost of solving UTD is $O(N^p + m_1 \cdot N^q) = O((m_3)^p + m_1 \cdot (m_3)^q)$.

The lower bound. Suppose that there exists an online SetDisjointness algorithm with $\frac{1}{3} \leq q < 1$ and $2p + q = 3 - \epsilon$ for some positive $\epsilon > 0$. By rearranging,

$$3(p-1) = \frac{3(1-q)}{2} - \frac{3}{2}\epsilon.$$

Thus, there exist constant positive numbers x and $\epsilon_q = \frac{3}{4}\epsilon$ such that

$$3(p-1) + \epsilon_q = x = \frac{3(1-q)}{2} - \epsilon_q.$$

Notice that, since $q \geq \frac{1}{3}$, then $x < \frac{3(1-q)}{2} \leq 1$. Moreover, by rearranging, there exists a constant $\epsilon' = \frac{1}{3}\epsilon_q$ such that

$$p \leq \frac{x+3}{3} - \epsilon'$$

and

$$q + x \leq \frac{x+3}{3} - \epsilon'.$$

Thus, since the UTD hypothesis holds for any combination of m_1, m_2 and m_3 (as long as $m_1 \leq m_2 \leq m_3$), we set $m_1 = (m_3)^x$ and $m_2 = m_3$ (recall that $x < 1$). The UTD hypothesis states that the time cost for solving UTD on this setting of m_1, m_2 and m_3 is $\Omega\left((m_3)^{\frac{x+3}{3}}\right)$, while the time cost of solving UTD using the reduction is

$$O((m_3)^p + m_1 \cdot (m_3)^q) = O((m_3)^p + (m_3)^{x+q}) = O\left((m_3)^{\frac{x+3}{3}-\epsilon'}\right),$$

thereby obtaining a contradiction. ◀

5 Unbalanced Triangle Enumeration

We begin with an algorithm which does not care about the number of triangles in the input.

▶ **Lemma 14.** *There exists an algorithm for the UTE problem whose time cost is*

$$O\left(m_3 + \sqrt{m_1 \cdot m_2 \cdot m_3}\right).$$

Proof. The algorithm uses the same definition and treatment of light vertices as in the algorithm in the proof of Theorem 5, but instead of stopping once a triangle is found, the algorithm continues until all triangles that contain at least one light vertex are enumerated. Recall that this process costs $O(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2)$ time.

Heavy vertices. Since there can be at most $\frac{m_1}{\tau_A}$ heavy vertices in A, at most $\frac{m_2}{\tau_B}$ heavy vertices in B, and at most $\frac{m_3}{\tau_C}$ heavy vertices in C, there can be at most $\frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}$ triangles whose vertices are *all* heavy. Thus, for each triplet of a heavy vertex $a \in A$, a heavy vertex $b \in B$, and a heavy vertex $c \in C$, the algorithm spends constant time looking up whether (a, b, c) is a triangle or not. The time cost of enumerating all such triplets is $O\left(\frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}\right)$.

Time cost. The total time cost is

$$O(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2 + \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}).$$

The time cost is minimized when the last four terms in the summation are all equal:

$$\tau_A \cdot m_3 = \tau_B \cdot m_1 = \tau_C \cdot m_2 = \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}.$$

By plugging in $\tau_B = \frac{m_3}{m_1}\tau_A$ and $\tau_C = \frac{m_3}{m_2}\tau_A$, we have

$$\begin{aligned}
\tau_A \cdot m_3 &= \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C} \\
&= \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \frac{m_3}{m_1}\tau_A \cdot \frac{m_3}{m_2}\tau_A} \\
&= \frac{(m_1 \cdot m_2)^2}{(\tau_A)^3 \cdot m_3}.
\end{aligned}$$

Therefore, $\tau_A = \sqrt{\frac{m_1 \cdot m_2}{m_3}}$, and the total time cost is $O(m_3 + \tau_A \cdot m_3) = O(m_3 + \sqrt{m_1 \cdot m_2 \cdot m_3})$.

Similarly to the UTD algorithm, notice that with an $O(m_3)$ time preprocessing we can ensure that every vertex in A has at least one edge to B and to C (process the vertices in B and C similarly), by removing any vertex (and its incident edges) without this property. This procedure never removes any triangles, and ensures that $m_1 \geq \max\{|A|, |B|\}$, $m_2 \geq \max\{|C|, |B|\}$, and $m_3 \geq \max\{|A|, |C|\}$. As $m_3 \leq |A| \cdot |C| \leq m_1 \cdot m_2$, $\tau_A \geq 1$. Similarly, $\tau_B, \tau_C \geq 1$, so the thresholds the algorithm uses make sense. ◀

Now we give an algorithm for UTE, assuming that the input graph has t triangles. Notice that our algorithm for UTD can count the number of triangles, so we can assume that we know t.

▶ **Theorem 9.** *There exists an algorithm for UTE on graphs with at most t triangles whose time cost is*

$$\tilde{O}\left(m_3 + m_3^{\frac{2}{\omega+1}}(m_1 m_2)^{\frac{\omega-1}{\omega+1}} + t^{\frac{3-\omega}{\omega+1}}(m_1 m_2 m_3)^{\frac{\omega-1}{\omega+1}}\right).$$

Proof. Let L, D_1, D_2, D_3 be parameters to be chosen later; we will make sure that all of these parameters are at least 1. Recall that $m_1 = E \cap (A \times B), m_2 = E \cap (B \times C), m_3 = E \cap (A \times C)$.

For every $a \in A$ with at most D_1 neighbors in C, list all triangles through a by going through all pairs of neighbors of a. The total time over all low-degree $a \in A$ is $O(m_1 D_1)$ time. Similarly, in $O(m_2 D_2)$ time list all triangles through all $b \in B$ with at most D_2 neighbors in A and in $O(m_3 D_3)$ time list all triangles through all $c \in C$ with at most D_3 neighbors in B.

Now let us set $D_1 = \frac{m_3 D_3}{m_1}$ and $D_2 = \frac{m_3 D_3}{m_2}$. As $m_3 \geq m_1, m_2$, $D_1, D_2 \geq 1$. This makes the total time so far $O(m_3 D_3)$.

Any triangle (a, b, c) that has not been listed must have that a has at least D_1 neighbors in C, b has at least D_2 neighbors in A and c has at least D_3 neighbors in B. Thus we can restrict to a subset A' of A of size at most $n_A = m_1/D_2 = (m_1 m_2)/(m_3 D_3)$, a subset B' of B of size at most $n_B = m_2/D_3$ and a subset C' of C of size at most $n_C = m_3/D_1 = m_1/D_3$. Notice that

$$n_A = (m_1/D_3) \cdot (m_2/m_3) \leq (m_1/D_3) = n_C \leq (m_2/D_3) = n_B.$$

Bjørklund et al. [9] give an $\tilde{O}(L^{3-\omega} n^{\omega})$ time algorithm that given a tripartite graph G' with n nodes in each partition, every edge e of G' the algorithm lists L triangles that contain e, for some parameter $L \geq 1$, or all triangles containing e if e is in fewer than L triangles. Our algorithm reduces to this balanced case.

Since $n_A \leq n_C \leq n_B$, the algorithm splits the larger partitions into parts of size roughly n_A, thereby obtaining $(n_B n_C)/n_A^2$ instances of balanced graphs, where every partition has n_A vertices. On each one of these instances the algorithm executes the algorithm of [9] to list up to L triangles for every edge in the remaining graph. The running time of this step is

$$\tilde{O}\left(n_B n_C n_A^{\omega-2} L^{3-\omega}\right) = \tilde{O}\left(\frac{(m_1 m_2)^{\omega-1}}{D_3^{\omega} m_3^{\omega-2}} L^{3-\omega}\right).$$

To minimize the total runtime we set

$$m_3 D_3 = \frac{(m_1 m_2)^{\omega-1}}{D_3^{\omega} m_3^{\omega-2}} L^{3-\omega}.$$

This sets $D_3 = L^{\frac{3-\omega}{\omega+1}}(m_1 m_2/m_3)^{\frac{\omega-1}{\omega+1}}$. Notice that as long as $L \geq 1$, $D_3 \geq 1$, just as with the UTD algorithm the algorithm can execute an $O(m_3)$ time preprocessing phase to make sure that $m_3 \leq m_1 m_2$.

With this setting of D_3, the runtime of this step becomes

$$O\left(m_3^{\frac{2}{\omega+1}}(m_1m_2)^{\frac{\omega-1}{\omega+1}}L^{\frac{3-\omega}{\omega+1}}\right).$$

Now, set $L = \max\{1, 6t/m_3\}$. The total runtime of the algorithm so far becomes:

$$\tilde{O}\left(m_3 + m_3^{\frac{2}{\omega+1}}(m_1m_2)^{\frac{\omega-1}{\omega+1}} + (m_1m_2m_3)^{\frac{\omega-1}{\omega+1}}T^{\frac{3-\omega}{\omega+1}}\right).$$

The only triangles remaining are those through edges that are contained in more than L triangles. As the number of triangles is t and since each triangle has 3 edges, the total number of edges whose triangles the algorithm has not found is at most $3t/L$. Notice that from the earlier steps of the algorithm we know for every edge e how many triangles contain e. Thus, the algorithm also knows the $3t/L$ edges that are left.

If $L = 1$ and so $6t/m_3 \leq 1$, we must have $t \leq m_3/6$ and so $3t/L = 3t \leq m_3/2$. Otherwise, if $L = 6t/m_3$, then we also get $3t/L = m_3/2$. In both cases, the total number of remaining edges is at most $m_3/2$, and so the algorithm recurses, applying the same steps but on an unbalanced graph with at most m_1, m_2 and $m_3/2$ edges and still at most t triangles. When the number of edges becomes constant, the algorithm solves the problem via brute force.

Since in each recursive step the current largest edge set shrinks by a factor of 2, the number of recursive steps is $O(\log n)$ and we at most tack on a logarithmic factor to the runtime. ◀

6 Optimal Conditional Lower Bound for SetIntersection

▶ **Theorem 10.** *Any algorithm for online SetIntersection that has $\frac{1}{2} \leq q < 1$ must obey $p + q \geq 2$, unless the UTE hypothesis is false.*

Proof. To prove the theorem we first describe a reduction from the UTE problem to the online SetIntersection problem by describing an algorithm for UTE that uses an algorithm for online SetIntersection as a black box. We then show that if the online SetIntersection algorithm obeys $p + q \geq 2 - \epsilon$ for $\frac{1}{2} \leq q < 1$ and some constant $\epsilon > 0$, then there exists an algorithm contradicting the UTE Hypothesis.

Reduction from UTE to online SetIntersection. The reduction is the same as the reduction given in the proof of Theorem 6, except that instead of using online SetDisjointness, the reduction algorithm uses SetIntersection. Specifically, the reduction algorithm does not stop after it is established that two sets are not disjoint. Instead, for each of the m_1 edges $(a, b) \in E \cup (A \times B)$, the algorithm performs an online SetIntersection query, and for each c in the output the algorithm enumerates triangle (a, b, c). The correctness of the reduction follows from the following claim.

▷ Claim 15. For every edge $(a, b) \in E \cap (A \times B)$ there is a bijection between every $c \in S_a \cap S_b$ and every triangle in G containing (a, b).

Proof. If there exists a triangle $(a, b, c) \in A \times B \times C$ in G, then both S_a and S_b contain c, and so $c \in S_a \cap S_b$. For the other direction, for every edge $(a, b) \in E \cap (A \times B)$ and every $c \in S_a \cap S_b$, the edges (a, c) and (b, c) must be in E, and so (a, b, c) is a triangle in G. ◁

As in the proof of Theorem 6, the sum of the sizes of the sets in the online SetIntersection instance is exactly $N = m_2 + m_3 = \Theta(m_3)$, and the size of the output is $O(t)$. Finally, the time cost of solving UTE is $O(N^p + m_1 \cdot N^q + t) = O((m_3)^p + m_1 \cdot (m_3)^q + t)$.

The lower bound. Suppose that there exists an online SetIntersection algorithm with $\frac{1}{2} \leq q < 1$ and $p + q \leq 2 - \epsilon$ for some positive $\epsilon > 0$. Without loss of generality, assume that $\epsilon < 2 - 2q$, which is okay since $q < 1$.

By rearranging, $2p - 2 \leq 2 - 2q - 2\epsilon$. Thus, there exists a constant positive number x such that

$$2p - 2 + \epsilon \leq x \leq 2 - 2q - \epsilon.$$

Notice that since $q \geq \frac{1}{2}$ then $x < 2 - 2q \leq 1$. Moreover, by rearranging, there exists a constant $\epsilon' > 0$ such that $p \leq 1 + \frac{x}{2} - \epsilon'$ and $q + x \leq 1 + \frac{x}{2} - \epsilon'$.

Since the UTE hypothesis holds for any combination of m_1, m_2 and m_3 (as long as $m_1 \leq m_2 \leq m_3$), we set $m_3 = m_2$ and $m_1 = (m_3)^x$. Moreover, let $t = (m_3)^y$ where $y = 1 + \frac{x}{2} - \epsilon'$.

Notice that the maximum number of triangles in an unbalanced tripartite graph with edge set sizes m_1, m_2, m_3 is $\sqrt{m_1 m_2 m_3} = m_3^{1+x/2}$, so that the number of triangles we need to list here is just a bit smaller than this.

The UTE hypothesis states that the time cost for solving UTE on this setting of m_1, m_2 and m_3 is

$$\Omega\left(t^{\frac{1}{3}}(m_1 m_2 m_3)^{\frac{1}{3}}\right) = \Omega\left(m_3^{\frac{2+x+y}{3}}\right) = \Omega\left(m_3^{(1+\frac{x}{2})-\frac{1}{3}\epsilon'}\right).$$

However, the time cost of solving UTE using the reduction is

$$\begin{aligned} O((m_3)^p + m_1 \cdot (m_3)^q + t) &= O((m_3)^p + (m_3)^{x+q} + (m_3)^y) \\ &= O((m_3)^{1+\frac{x}{2}-\epsilon'}) << \Omega((m_3)^{1+\frac{x}{2}-\frac{\epsilon'}{3}}), \end{aligned}$$

where the last transition is due to $\epsilon' > 0$. Thus, we have obtained a contradiction. ◀

References

1. A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 434–443, 2014.
2. P. Afshani and J. Sindahl Nielsen. Data structure lower bounds for document indexing problems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*, pages 93:1–93:15, 2016.
3. N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.
4. A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part I*, pages 114–125, 2014.
5. A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *27th International Symposium on Algorithms and Computation, ISAAC*, pages 12:1–12:12, 2016.
6. R. A. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In *Combinatorial Pattern Matching, 15th Annual Symposium, CPM*, pages 400–408, 2004. doi:10.1007/978-3-540-27801-6_30.
7. J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 390–399, 2002.
8. Philip Bille, Anna Pagh, and Rasmus Pagh. Fast evaluation of union-intersection expressions. In *Algorithms and Computation, 18th International Symposium, ISAAC*, pages 739–750, 2007.

9 A. Bjorklund, R. Pagh, V. Vassilevska Williams, and U. Zwick. Listing triangles. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part I*, pages 223–234, 2014.

10 T. M. Chan, S. Durocher, K. Green Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory Comput. Syst.*, 55(4):719–741, 2014.

11 K. Chatterjee, W. Dvorák, M. Henzinger, and A. Svozil. Algorithms and conditional lower bounds for planning problems. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS*, pages 56–64. AAAI Press, 2018.

12 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

13 H. Cohen and E. Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010. doi:10.1016/j.tcs.2010.06.002.

14 H. Cohen and E. Porat. On the hardness of distance oracle for sparse graph. *CoRR*, abs/1006.1117, 2010. arXiv:1006.1117.

15 D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.

16 P. Davoodi, M. H. M. Smid, and F. van Walderveen. Two-dimensional range diameter queries. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium*, pages 219–230, 2012.

17 E. D. Demaine, A. López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 743–752, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338634.

18 Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams. Equivalences between triangle and range query problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 30–47. SIAM, 2020.

19 D. Eppstein, M. T. Goodrich, M. Mitzenmacher, and M. R. Torres. 2-3 Cuckoo filters for faster triangle listing and set intersection. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 247–260, 2017.

20 J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela, and N. Välimäki. Forbidden patterns. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium*, pages 327–337, 2012.

21 F. Le Gall and F. Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1029–1046, 2018.

22 I. Goldstein, T. Kopelowitz, M. Lewenstein, and E. Porat. How hard is it to find (honest) witnesses? In *24th Annual European Symposium on Algorithms, ESA*, pages 45:1–45:16, 2016.

23 I. Goldstein, T. Kopelowitz, M. Lewenstein, and E. Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures - 15th International Symposium, WADS*, pages 421–436, 2017.

24 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *J. ACM*, 61(2):9:1–9:36, 2014.

25 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

26 Z. Jafargholi and E. Viola. 3sum, 3xor, triangles. *Algorithmica*, 74(1):326–343, 2016.

27 T. Kopelowitz and R. Krauthgamer. Color-distance oracles and snippets. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 24:1–24:10, 2016.

28 T. Kopelowitz, S. Pettie, and E. Porat. Dynamic set intersection. In *Proceedings 14th Int'l Symposium on Algorithms and Data Structures (WADS)*, pages 470–481, 2015.

29 T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1272–1287, 2016.

30 F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 514–523, 2012.

31 F. Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

32 M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 603–610, 2010.

33 M. Patrascu and L. Roditty. Distance oracles beyond the Thorup-Zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014.

34 M. Patrascu, L. Roditty, and M. Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 738–747, 2012.

35 A. Stothers. On the complexity of matrix multiplication. *Ph.D. Thesis, U. Edinburgh*, 2010.

36 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 887–898, 2012.

37 V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians*, pages 3431–3475, 2018.

38 R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. on Algorithms*, 1(1):2–13, 2005.