Higher Lower Bounds from the 3SUM Conjecture*

Tsvi Kopelowitz† University of Michigan

Seth Pettie ‡ University of Michigan

Ely Porat § Bar-Ilan University

July 10, 2015

Abstract

The 3SUM conjecture has proven to be a valuable tool for proving conditional lower bounds on dynamic data structures and graph problems. This line of work was initiated by Pătraşcu (STOC 2010) who reduced 3SUM to an offline SetDisjointness problem. However, the reduction introduced by Pătraşcu suffers from several inefficiencies, making it difficult to obtain *tight* conditional lower bounds from the 3SUM conjecture.

In this paper we address many of the deficiencies of Pătraşcu's framework. We give new and efficient reductions from 3SUM to offline SetDisjointness and offline SetIntersection (the reporting version of SetDisjointness) which leads to polynomially higher lower bounds on several problems. Using our reductions, we are able to show the essential optimality of several algorithms, assuming the 3SUM conjecture.

- Chiba and Nishizeki's $O(m\alpha)$-time algorithm (SICOMP 1985) for enumerating all triangles in a graph with arboricity/degeneracy α is essentially optimal, for any α.
- Bjørklund, Pagh, Williams, and Zwick's algorithm (ICALP 2014) for listing t triangles is essentially optimal (assuming the matrix multiplication exponent is $\omega = 2$).
- Any static data structure for SetDisjointness that answers queries in constant time must spend $\Omega(N^{2-o(1)})$ time in preprocessing, where N is the size of the set system.

These statements were unattainable via Pătraşcu's reductions.

We also introduce several new reductions from 3SUM to pattern matching problems and dynamic graph problems. Of particular interest are new conditional lower bounds for dynamic versions of Maximum Cardinality Matching, which introduce a new technique for obtaining amortized lower bounds.

1 Introduction

Data structure lower bounds come in two varieties: conditional and unconditional. The strongest unconditional lower bounds (in the cell probe model) are either poly-logarithmic [32, 26] or on extreme tradeoffs between sub-logarithmic update time and large query time; see [34]. Pătraşcu [35]

*Supported by NSF grants CCF-1217338 and CNS-1318294 and a grant from the US-Israel Binational Science Foundation. This research was performed in part at the Center for Massive Data Algorithmics (MADALGO) at Aarhus University, which is supported by the Danish National Research Foundation grant DNRF84.

†Email: kopelot@gmail.com

‡Email: pettie@umich.edu.

§Email: porately@gmail.com

proposed an approach for proving polynomial *conditional* lower bounds (CLBs) based on the conjectured hardness of the 3SUM problem, via a new intermediate problem he called Convolution3SUM. The integer versions of the 3SUM and Convolution3SUM problems are defined as follows. Given a set $A \subset \mathbb{Z}$, the 3SUM problem is to decide if there is a triple $(a, b, c) \in A^3$ of distinct elements such that $a + b = c$. The Convolution3SUM problem is, given a vector $A \in \mathbb{Z}^n$, to decide if there is a pair $(i, j) \in [n]^2$ for which $A(i) + A(j) = A(i + j)$. Here $[n] = \{0, 1, \ldots, n - 1\}$ is the first n naturals.

It was originally conjectured [16] that 3SUM required $\Omega(n^2)$ time. However, there are now known to be $O(n^2/\operatorname{polylog}(n))$ 3SUM algorithms for both integer [5] and real [22] inputs. The modern *3SUM Conjecture* [35] is that the time complexity of the problem is $\Omega(n^{2-o(1)})$, even in expectation.

3SUM and Set Disjointness. Pătraşcu [35] gave a reduction from 3SUM to Convolution3SUM showing that the 3SUM conjecture implies that Convolution3SUM also requires $\Omega(n^{2-o(1)})$ time.

Theorem 1.1 (Pătraşcu [35])**.** *Define $T_{3S}(n)$ and $T_{C3S}(n)$ to be the randomized (Las Vegas) complexities of 3SUM and Convolution3SUM on instances of size n. For any parameter k, $T_{3S}(n) = O(n^2/k + (k^3 + k^2 \log n) \cdot T_{C3S}(n/k))$.*

Pătraşcu then reduced Convolution3SUM to the offline SetDisjointness problem where we are given a set C of elements, two families A and B of subsets of C, and q pairs of subsets $(S, S') \in A \times B$, and we are interested in determining for each of the q pairs whether they are disjoint or not. The following two Theorems summarize the reductions by Pătraşcu (the second Theorem is implicit in Section 2.3 of [35]).

Theorem 1.2 (Pătraşcu [35])**.** *Let $g(n)$ be such that Convolution3SUM requires $\Omega(\frac{n^2}{g(n)})$ expected time. For any $\sqrt{n \cdot g(n)} \ll n^\epsilon \ll n/g(n)$ let $\mathbb{A}$ be an algorithm for the offline SetDisjointness problem where $|C| = n$, $|A| = |B| = \Theta(n^{1/2+\epsilon})$, each set in $A \cup B$ has at most $O(n^{1-\epsilon})$ elements from C, each element in C appears in $\sqrt{n}$ sets from A and $\sqrt{n}$ sets from B, and $q = \Theta(n^{1+\epsilon})$. Then $\mathbb{A}$ requires $\Omega(\frac{n^2}{g(n)})$ expected time.*

Theorem 1.3 (Pătraşcu [35])**.** *Let $g(n)$ be such that Convolution3SUM requires $\Omega(\frac{n^2}{g(n)})$ expected time. For any $\sqrt{n \cdot g(n)} \ll n^\epsilon \ll n/g(n)$ let $\mathbb{A}$ be an algorithm for the offline SetDisjointness problem where $|C| = \Theta(n^{2-2\epsilon})$, $|A| = |B| = \Theta(n^{1/2+\epsilon} \log n)$, each set in $A \cup B$ has at most $O(n^{1-\epsilon})$ elements from C, each element in C appears in $\Theta(n^{2\epsilon-1/2})$ sets from A and $\Theta(n^{2\epsilon-1/2})$ sets from B, and $q = \Theta(n^{1+\epsilon} \log n)$. Then $\mathbb{A}$ requires $\Omega(\frac{n^2}{g(n)})$ expected time.*

The 3SUM conjecture posits that $g(n) = n^{o(1)}$, implying that $1/2 < \epsilon < 1$. Using this case, Pătraşcu provided CLBs for triangle enumeration, set-disjointness, and various other dynamic graph and data structure problems. Pătraşcu's results led to a surge of research on CLBs that assume the 3SUM conjecture. Vassilevska-Williams and Williams [39] showed that finding a triangle of zero weight in a weighted graph requires cubic time. Abboud, Vassilevska-Williams, and Weimann [2] proved that the Local Alignment problem, which is of great importance in computational biology, cannot be solved in truly sub-quadratic time. Amir, Chan, Lewenstein, and Lewenstein [4] proved that for the Jumbled Indexing problem on a text with n integers (from a large enough alphabet), either the preprocessing time needs to be truly quadratic or the query time needs to be truly linear. Abboud and Vassilevska Williams [1] showed several CLBs conditioned on popular conjectures of hardness. In particular, they showed, conditioned on the 3SUM conjecture, that data structure versions of

(s,t)-reachability, Strong Connectivity, Subgraph Connectivity, Bipartite Perfect Matching, and variations of a problem known as Pagh's problem all require non-trivial polynomial preprocessing, query, or update time. Recently, Abboud, Vassilevska Williams, and Yu [3] used a different approach for reducing 3SUM to various dynamic graph problems.

We emphasize that the results in [4] and [39] implicitly make use of Theorem 1.1 by directly reducing from Convolution3SUM (rather than 3SUM) to the target problem, and the results in [1] make use of Theorem 1.2.

Limitations of Pătraşcu's Reductions. Pătraşcu's reductions suffer from four limitations

- The first limitation is that the number of SetDisjointness queries is rather large (at least $\omega(n^{3/2})$) making it impossible to get any $\omega(n^{1/2})$ lower bound per query.
- The second limitation is that the size N of the set systems in the offline SetDisjointness problem are also rather large ($N = \Omega(n^{3/2})$) so it is impossible to get lower bounds that are $\Omega(N^{4/3})$ as a function of N.
- The third limitation is that the sets in the offline SetDisjointness instances are very sparse. In Pătraşcu's framework all sets have size at most $O(\sqrt{|C|})$, so it is not clear how to get CLBs for the dense case of many graph problems.
- Finally, Pătraşcu's reduction from 3SUM to offline SetDisjointness is only meaningful if the true complexity of 3SUM is $\omega(n^{13/7})$.[1]

Regarding the first two limitations: we would like to have much more control over the size of the set system and the number of queries. The third limitation seems intrinsic to *any* reduction to SetDisjointness where the sets are essentially random, since, by the birthday paradox, random $\omega(\sqrt{|C|})$-size sets will almost surely intersect, giving no useful information per query. The fourth limitation gets at the issue of *robustness*: how valuable are reductions from 3SUM if the 3SUM conjecture turns out to be false (but not by much)? The possibility of a truly subquadratic 3SUM algorithm seemed remote a few years ago but given recent developments [38, 22, 8] is not so absurd. Chan and Lewenstein [8] showed numerous special cases of 3SUM can be solved in truly subquadratic time.

1.1 New Results: A More Versatile Lower Bound Framework

We overcome the limitations of Pătraşcu's framework by giving efficient reductions directly from 3SUM to SetDisjointness and from 3SUM to SetIntersection (which we define shortly). By avoiding Convolution3SUM as an intermediate problem, our reduction gives non-trivial lower bounds if the true complexity of 3SUM is just $\Omega(n^{3/2+\Omega(1)})$.

For clarity's sake, we present our new framework in terms of a new constant γ instead of the constant ϵ of Theorem 1.2. Notice that in Pătraşcu's frameworks $1/2 < \epsilon < 1$ while here $0 < \gamma < 1$. The first theorem reduces 3SUM to offline SetDisjointness and is proven in Section 2.

[1] To see this, note that Theorem 1.2 is only applicable if $g(n) = O(n^{1/3})$, i.e., Convolution3SUM requires $\Omega(n^{5/3})$ time. It is consistent with Theorem 1.1 that Convolution3SUM can be solved in $O(n^{5/3})$ time while 3SUM can be solved in $O(n^{13/7})$ time. Furthermore, even if 3SUM were proved to be $\Omega(n^{5/3})$ unconditionally, this would imply no superlinear lower bound on Convolution3SUM (via Theorem 1.1) or to any of the problems that Convolution3SUM is reduced to.

Reductions to Set Disjointness/Intersection

Reduction	$\lvert A\rvert, \lvert B\rvert$	$\lvert C\rvert$	q	Remarks
Conv3SUM $\to$ SetDisjointness	$n^{1/2+\epsilon}$	n	$n^{1+\epsilon}$	$\sqrt{ng(n)} \ll n^{\epsilon} \ll n/g(n)$
Conv3SUM $\to$ SetDisjointness	$n^{1/2+\epsilon}\log n$	$n^{2-2\epsilon}$	$n^{1+\epsilon}\log n$	
3SUM $\to$ SetDisjointness	$n\log n$	$n^{2-2\gamma}$	$n^{1+\gamma}\log n$	$0<\gamma<1$ and $\delta>0$
3SUM $\to$ SetIntersection	$n^{\frac{1}{2}(1+\delta+\gamma)}$	$n^{1+\delta-\gamma}$	$n^{1+\gamma}$	Total output size: $O(n^{2-\delta})$

Figure 1: The first two reductions from Convolution3SUM are from Pătraşcu [35]. The third and fourth reductions from 3SUM are new.

Theorem 1.4. *Let $f(n)$ be such that 3SUM requires expected time $\Omega(\frac{n^2}{f(n)})$. For any constant $0<\gamma<1$ let $\mathbb{A}$ be an algorithm for offline SetDisjointness where $|C| = \Theta(n^{2-2\gamma})$, $|A| = |B| = \Theta(n\log n)$, each set in $A\cup B$ has at most $O(n^{1-\gamma})$ elements from C, and $q = \Theta(n^{1+\gamma}\log n)$. Then $\mathbb{A}$ requires $\Omega(\frac{n^2}{f(n)})$ expected time.*

We also consider the offline SetIntersection problem where the input is the same as in SetDisjointness, except that we need to enumerate all elements in the q intersections. We prove the following theorem in Section 2. Note that in this case we allow $\gamma = 0$.

Theorem 1.5. *Let $f(n)$ be such that 3SUM requires expected time $\Omega(\frac{n^2}{f(n)})$. For any constants $0\le\gamma<1$ and $\delta>0$, let $\mathbb{A}$ be an algorithm for offline SetIntersection where $|C| = \Theta(n^{1+\delta-\gamma})$, $|A| = |B| = \Theta(\sqrt{n^{1+\delta+\gamma}})$, each set in $A\cup B$ has at most $O(n^{1-\gamma})$ elements from C, $q = \Theta(n^{1+\gamma})$, and the total size of the set intersections of these q pairs is $O(n^{2-\delta})$ in expectation. Then $\mathbb{A}$ requires $\Omega(\frac{n^2}{f(n)})$ expected time.*

These theorems eliminate the need to use Convolution3SUM as a stepping stone (Theorem 1.1) when proving lower bounds via SetDisjointness/SetIntersection. Nonetheless, the Convolution3SUM problem is still useful and its exact relationship to the 3SUM problem an interesting open question. The more constrained structure of Convolution3SUM makes it easier to use in some CLBs; see [4, 39]. We are also able to prove (in Appendix A) that the randomized complexities of 3SUM and Convolution3SUM differ by at most a logarithmic factor.

Theorem 1.6. *Define $T_{3S}(n)$ and $T_{C3S}(n)$ to be the randomized (Las Vegas) complexities of 3SUM and Convolution3SUM on instances of size n. Then $T_{3S}(n) = O(\log n\cdot T_{C3S}(n))$.*

1.2 Techniques and Implications

Our new framework borrows liberally from the techniques of Pătraşcu's framework, particularly the use of almost linear hash functions. However, in order to obtain our improvements we assemble the building blocks in a new and simpler way. The implications of our new framework are significant. To start off, since q can be made arbitrarily close to n (by having γ approach 0), it is now possible to obtain much higher CLBs for the query time of SetDisjointness. Similarly, since the size of the offline SetDisjointness can be made small (by having γ approach 0), it is now possible to obtain higher CLBs in terms of the size of the offline SetDisjointness instance.

Finally, as is illustrated in Section 3, using the new framework it is now possible to obtain CLBs for graph problems which apply to all edge densities. Such reductions make use of Theorem 1.5. In particular, this implies that using clever techniques, such as fast matrix multiplication, cannot lead to faster algorithms for the dense case of such problems, such as enumerating all of the triangles in a dense graph.

To illustrate the advantages of the new framework, we briefly show how we are able to obtain higher CLBs for the fundamental problem of *online* SetDisjointness, and then continue in the body of this paper to describe new and better CLBs for many old and new problems. A corresponding discussion on CLBs for the online SetIntersection problem is given in Appendix B.

A Higher Lower Bound for Online SetDisjointness. It is obvious that online SetDisjointness solves offline SetDisjointness. We phrase the CLBs in terms of N: the sum of set sizes. Define t_p to be the preprocessing time of an online SetDisjointness structure and t_q its query time.

Using Pătraşcu's reduction from Convolution3SUM to SetDisjointness we have $N = \Theta(n^{1.5})$, there are $\Theta(n^{1+\epsilon}) = \Theta(N^{(2+2\epsilon)/3})$ queries that need to be answered. Thus we obtain the following lower bound tradeoff: $t_p + N^{(2+2\epsilon)/3} t_q = \Omega\left(\frac{N^{4/3}}{g(N^{2/3})}\right)$. The 3SUM conjecture implies $g(x) = x^{o(1)}$. If, for example, we only allow linear preprocessing, letting ϵ tend to $1/2$ gives a query lower bound of $\Omega(N^{\frac{1}{3}-o(1)})$. If we demand constant time queries, we obtain a lower bound of $\Omega(N^{\frac{4}{3}-o(1)})$ on the preprocessing time. We show next how our new framework provides better tradeoffs.

Theorem 1.7. *Assume the 3SUM conjecture. For any $0 < \gamma < 1$, any data structure for SetDisjointness has*

$$t_p + N^{\frac{1+\gamma}{2-\gamma}} t_q = \Omega\left(N^{\frac{2}{2-\gamma}-o(1)}\right).$$

Proof. Using Theorem 1.4, we have $N = \Theta(n^{2-\gamma}\log n)$, and the number of queries to answer is $\Theta(n^{1+\gamma}\log n) = \tilde{\Theta}(N^{\frac{1+\gamma}{2-\gamma}})$. By the 3SUM conjecture answering these queries takes time $\Omega(n^{2-o(1)}) = \Omega(N^{\frac{2}{2-\gamma}-o(1)})$. □

Theorem 1.7 implies, for example, that if we only allow linear preprocessing time, then by making γ tend to zero the query time must be $\Omega(N^{\frac{1}{2}-o(1)})$. This CLB is comparable with the data structure of Cohen and Porat [11] where $t_p = O(N\sqrt{N})$ and $t_q = O(\sqrt{N})$ (see also [25]). Furthermore, if we only allow constant query time, then by making γ tend to 1 the preprocessing time must be $\Omega(N^{2-o(1)})$, matching that of the trivial preprocessing algorithm that computes all answers in advance.

1.3 Triangle Enumeration

In the triangle enumeration problem one wishes to enumerate all triangles in a graph G with n vertices and m edges. Itai and Rodeh [20] were the first to obtain a $O(m^{1.5})$ time algorithm for listing all triangles in a graph. This was improved by Chiba and Nishizeki [10] who provided an algorithm that lists all triangles in $O(m\alpha)$ time where $\alpha = \alpha(G)$ is the arboricity[2] of G, which is always at most $O(\sqrt{m})$.

[2]The *arboricity* of an undirected graph $G = (V, E)$ is defined as $\alpha(G) = \max_{U \subseteq V\,:\,|U|\geq 2} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$ where $E(U)$ is the set of edges induced by U. The arboricity is also the minimum number of forests that E can be partitioned into [29, 30].

Triangle Enumeration Bounds

Authors	Time Bound	Remarks
Itai & Rodeh	$O(m^{3/2})$	
Chiba & Nishizeki	$O(m\alpha)$	α = arboricity
Pătraşcu	$\Omega(m^{4/3-o(1)})$	$t \approx m = n^{1.5+o(1)}$ triangles
Bjørklund, Pagh, Williams & Zwick	$O(m^{\frac{2\omega}{\omega+1}} + m^{\frac{3(\omega-1)}{\omega+1}} t^{\frac{3-\omega}{\omega+1}})$	
	$O(n^{\omega} + n^{\frac{3(\omega-1)}{5-\omega}} t^{\frac{2(3-\omega)}{5-\omega}})$	
	$O(m^{4/3+o(1)} + t \cdot (\frac{m}{t^{2/3}}))$	Assuming $\omega = 2$
	$O(n^{2+o(1)} + t \cdot (\frac{n}{t^{1/3}}))$	Assuming $\omega = 2$
	$\Omega(t \cdot (\frac{m}{t^{2/3}})^{1-o(1)})$	Assuming QES Conjecture
	$\Omega(t \cdot (\frac{n}{t^{2/3}})^{1-o(1)})$	Assuming QES Conjecture
Kopelowitz, Pettie & Porat	$O(m + m\frac{\alpha \log\log n}{\log n} + t)$	Randomized, w.h.p.
new	$\Omega(m\alpha^{1-o(1)})$	every arboricity α
	$\Omega(\min\{m^{3/2-o(1)}, t \cdot (\frac{m}{t^{2/3}})^{1-o(1)}\})$	Assuming 3SUM Conjecture
	$\Omega(\min\{n^{3-o(1)}, t \cdot (\frac{n}{t^{1/3}})^{1-o(1)}\})$	Assuming 3SUM Conjecture

Figure 2: The Bjørklund et al. [6] lower bounds assume the QES Conjecture, which states that the brute force algorithm for solving quadratic equations over finite fields is essentially optimal.

An interesting question regarding the runtime of such algorithms is whether the $O(m\alpha)$ runtime stems *solely* from the size of the output (the number of triangles), or if there is something intrinsic to the triangle enumeration problem that demands such effort. Recently in [25], we were able to asymptotically break the $\Omega(m\alpha)$ bound by showing that enumerating t triangles takes $O(m + m\alpha/\frac{\log n}{\log\log n} + t)$ expected time in the RAM model. While this shows a poly-log improvement over Chiba and Nishizeki's algorithm, the question of whether a *polynomial* improvement (in either m or α) is obtainable still remains.

Pătraşcu showed that conditioned on the 3SUM conjecture, there exists a graph with $t = O(m)$ triangles, for which listing all triangles must take $\Omega(m^{4/3-o(1)})$ time. A careful examination of Pătraşcu's proof shows that the arboricity of this graph is indeed roughly $m^{1/3}$, implying the essential optimality of Chiba and Nishizeki's algorithm and of [25], at least for one particular arboricity. However, Pătraşcu's CLB does not extend to any $\alpha \gg m^{1/3}$. This left open the possibility of clever algorithms that dramatically improve the $\Omega(m\alpha)$ bound on dense graphs.

Using our new framework we prove an $\Omega(m\alpha^{1-o(1)})$ CLB for triangle enumeration, for all arboricities $1 \ll \alpha \ll m^{1/2}$. We emphasize that the number of triangles in these instances is polynomially smaller than $m\alpha$, implying that the hardness is *not* due to the time required to report the output. Thus, the Chiba-Nishizeki algorithm and the algorithm in [25] are essentially optimal for the entire spectrum of arboricities. The proof of Theorem 1.8 is given in Section 3.

Theorem 1.8. *Assume the 3SUM conjecture. For any constants $0 < x \leq 1$ and $0 < y \leq 1$ such that $x \leq 2y$, there exists a graph with n vertices, m edges, and arboricity $\alpha = \Theta(n^x) = \Theta(m^y)$ with $t = O(m\alpha^{1-\Omega(1)})$ triangles, such that listing all triangles requires $\Omega(m\alpha^{1-o(1)})$ expected time.*

For a comparison between the known upper and conditional lower bounds, see Figure 2.

Output-sensitive triangle enumeration algorithms. Another approach for enumerating triangles considers output-sensitive algorithms for triangle enumeration. A recent algorithm of Bjørklund,

Pagh, Williams, and Zwick [6] shows that if the matrix multiplication exponent is $\omega = 2$, then listing only t triangles takes $\tilde{O}(\min\{n^2 + nt^{2/3}, m^{4/3} + mt^{1/3}\})$ time. Notice that this runtime can be expressed as paying either $\frac{n}{t^{1/3}}$ or $\frac{m}{t^{2/3}}$ time per triangle.

We prove that, assuming the 3SUM conjecture and assuming $\omega = 2$, this per triangle cost is essentially optimal, for any graph with arboricity at least $m^{1/3}$. This lower bound is obtained by considering the extreme case of listing all triangles in the graph, which by Theorem 1.8 requires $\Omega(m\alpha^{1-o(1)})$ expected time, combined with controlling the number of triangles in the graph so that $t = \alpha^3$. Such a result seems to be unobtainable using Pătraşcu's framework, since the corresponding graphs in his framework have arboricity at most $m^{1/3}$. Thus we are able to prove the following.

Theorem 1.9. *Assume the 3SUM conjecture. Then any algorithm for listing t triangles whose runtime is expressed in terms of the number of edges m must take $\Omega(\min\{m^{3/2-o(1)}, t \cdot (\frac{m}{t^{2/3}})^{1-o(1)}\})$ expected time. If its runtime is expressed in terms of the number of vertices n it must take $\Omega(\min\{n^{3-o(1)}, t \cdot (\frac{n}{t^{1/3}})^{1-o(1)}\})$ expected time.*

In particular, Theorem 1.9 implies that if we do not spend $\Omega(m^{3/2})$ time for listing just t triangles (which is enough time to report all triangles), then the time per triangle must be $\Omega((\frac{m}{t^{2/3}})^{1-o(1)})$.

1.4 New Lower Bounds

We prove polynomial CLBs, conditioned on the 3SUM conjecture, for data structure versions of the following problems: Document Retrieval problems, Maximum Cardinality Matching in bipartite graphs (improving known CLBs [1]), d-failure Connectivity Oracles, and Distance Oracles for Colors. The new CLB for Maximum Cardinality Matching is of particular interest since it introduces new techniques for obtaining amortized lower bounds, and so we describe it in more detail; See Section 4 and Appendix G. The rest of the CLBs are in the Appendix.

Maximum Cardinality Matching. In the *Dynamic Maximum Cardinality Matching problem* we are interested in maintaining a dynamic graph $G = (V, E)$, with $n = |V|$ and $m = |E|$, to support maximum cardinality matching (MCM) queries, which report the size of the current MCM. When both insertions and deletions are supported we say that G is fully dynamic, while if only insertions are supported we say that G is *incremental*. The trivial algorithm for updating an MCM takes $O(m)$ time by finding an augmenting path. Sankowski [36] gave a fully dynamic algorithm with an amortized time bound of $O(n^{1.495})$ based on fast matrix multiplication. In the bipartite vertex-addition model, where vertices on one side of the graph arrive online with all of their edges, Bosek, Leniowski, Sankowski, and Zych [7] recently showed how to maintain a maximum cardinality matching whose total update time is $O(m\sqrt{n})$ time.

Abboud and Vassilevska-Williams [1] showed that, based on the 3SUM conjecture, in a fully dynamic graph and any $1/6 \leq \alpha \leq 1/3$, either the preprocesing time is $\Omega(m^{4/3-o(1)})$, the amortized update time is $\Omega(m^{\alpha-o(1)})$, or the amortized query time is $\Omega(m^{2/3-\alpha-o(1)})$. In our setting we will require the size of the MCM to be reported after each update, and so the CLB of [1] implies that if the preprocessing time t_p is $O(m^{4/3-\Omega(1)})$ then the update time t_u is $\Omega(m^{1/3-o(1)})$. Using Theorem 1.4 we are able to prove the following in Appendix G:

Theorem 1.10. *Assume the 3SUM conjecture. For any $0 < \gamma < 1$ and any fully dynamic MCM algorithm, even if the preprocessing phase is given an MCM of the initial graph,*

$$t_p + m^{\frac{1+\gamma}{2-\gamma}} t_u = \Omega(m^{\frac{2}{2-\gamma}-o(1)}).$$

Moreover, the same bound holds even for the class of approximate *MCM algorithms that report the size of some matching without length-7 augmenting paths.*

By having γ approach 0 the implication of Theorem 1.10 is that if we require $t_p = O(m)$ then the update time must be $\Omega(m^{1/2-o(1)})$. This slightly improves on the results of Abboud and Vassilevska-Williams [1].[3] Guaranteeing the absence of short augmenting paths is one way to achieve a provably good approximation to the MCM; see [31].

The incremental case. In this setting we consider an initially empty graph G and so there is no preprocessing phase. Abboud and Vassilevska-Williams [1] show that their lower bounds for fully dynamic MCM extend to incremental MCM, but only for *worst case* time bounds. They mention the difficulty in obtaining amortized lower bounds for the incremental case using their approach, as they simulate deletions by rolling the structure back after each insertion to its state prior to the insertion. The worst-case lower bounds of Abboud and Vassilevska-Williams [1] can be phrased in terms of $\hat{n}$, the number of *vertices* when an operation takes place. Either the update or query is $\Omega(\hat{n}^{1/2-o(1)})$. We show in Appendix G that if we allow the graph to grow with each query, it is straightforward to obtain an amortized expected $\Omega(\hat{n}^{1/3-o(1)})$ lower bound.

We focus on improving the amortized lower bound for incremental MCM in terms of $\hat{n}$, using Theorem 1.4. Our strategy to answer SetDisjointness queries using an incremental dynamic MCM algorithm. The construction has the property that queries can be simulated by two vertex insertions and two edge insertions. There are two ways to undo these four insertions, (i) rolling back the state of the data structure to its original state, or (ii) inserting two more vertices and two more edges. By dynamically choosing which of (i) or (ii) to employ we can control the total number of vertices that end up in the graph and get better lower bounds as a function of $\hat{n}$.

Theorem 1.11 is proved in Section 4.

Theorem 1.11. *Assume the 3SUM conjecture. Any algorithm for incremental MCM has amortized expected update time of* $\Omega(\hat{n}^{\sqrt{2}-1-o(1)}) = \Omega(\hat{n}^{0.414-o(1)})$ *where* $\hat{n}$ *is the number of vertices in the graph following the update.*

2 The Improved Framework - Theorems 1.4 and 1.5

Let $\mathcal{H}$ be a family of hash functions from $[u] \to [m]$. $\mathcal{H}$ is called *linear* if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$, we have $h(x) + h(x') = h(x + x') \pmod m$. $\mathcal{H}$ is called *balanced* if for any set $S = \{x_1, \ldots, x_n\} \subset [u]$, and for any $i \in [m]$ we have $|\{x \in S : h(x) = i\}| \leq \frac{3n}{m}$. For the proofs of our new framework we will be assuming the existence of pair-wise independent families of hash functions that are *magically* linear and balanced. In reality there will be minor violations of linearity and balancedness: $h(x) + h(x')$ may differ from $h(x + x')$ by a constant, and some hash values will receive more than $3n/m$ elements. In Appendix C we exhibit almost linear, almost balanced hash functions and sketch the modifications to the reduction needed to accommodate them.

Combined proof of Theorem 1.4 and Theorem 1.5. Since the proofs of both theorems follow a similar path we describe them together. To simplify the exposition we consider the variant of 3SUM where we are looking for three elements x, y, z such that $x - y = z$. Let $R = n^\gamma$. Let $Q = (5n/R)^2$ for Theorem 1.4 and let $Q = (n^{1+\delta}/R)$ for Theorem 1.5. Without loss of generality we assume that

[3] They [1] also get CLBs on approximate MCM algorithms that eliminate short augmenting paths, but from different conjectures concerning the complexity of triangle detection and combinatorial BMM.

$\sqrt{Q}$ is an integer. Finally, we assume that the input is drawn from an integer universe $U \subseteq [2^w]$, where $w = \Omega(\log n)$ is the machine's word length.

We pick a random hash function $h_1 : U \to [R]$ from a family that is linear and balanced. Using h we create R buckets $\mathcal{B}_1, \cdots, \mathcal{B}_R$ such that $\mathcal{B}_i = \{x : h_1(x) = i\}$. Since h_1 is balanced, each bucket contains at most $3n/R$ elements. This bucketing is similar to Pătraşcu's reduction [35].

Next, we pick a random hash function $h_2 : U \to [Q]$ where h_2 is chosen from a pair-wise independent and linear family. For each bucket $\mathcal{B}_i$ we create $2\sqrt{Q}$ shifted sets as follows: for each $0 \leq j < \sqrt{Q}$ let $\mathcal{B}^{\uparrow}_{i,j} = \{h_2(x) + j \cdot \sqrt{Q} \,(\mathrm{mod}\, Q) \,|\, x \in \mathcal{B}_i\}$ and $\mathcal{B}^{\downarrow}_{i,j} = \{h_2(x) - j \,(\mathrm{mod}\, Q) \,|\, x \in \mathcal{B}_i\}$.

Next, for each $z \in A$ we want to determine if there exist x and y in A such that $x - y = z$. To do this we utilize the linearity of h_1 and h_2, which implies that $h_1(x) - h_1(y) = h_1(z) \,(\mathrm{mod}\, R)$ and $h_2(x) - h_2(y) = h_2(z) \,(\mathrm{mod}\, Q)$. Thus, if $x \in \mathcal{B}_i$ then y must be in $\mathcal{B}_{i-h_1(z) (\mathrm{mod}\, R)}$. To this end, for each $i \in [R]$ we would like to intersect $\mathcal{B}_i$ with $\mathcal{B}_{i-h_1(z) (\mathrm{mod}\, R)} + z$ in order to find candidate pairs of x and y. Denote by $h_2^{\uparrow}(z) = \lfloor \frac{h_2(z)}{\sqrt{Q}} \rfloor$ and $h_2^{\downarrow}(z) = h_2(z) (\mathrm{mod}\, \sqrt{Q})$. Due to the linearity of h_2, the intersection of the sets $\mathcal{B}_i$ and $\mathcal{B}_{i-h_1(z) (\mathrm{mod}\, n^{\gamma})} + z$ is determined precisely by the intersection of the sets $\mathcal{B}^{\uparrow}_{i,h_2^{\uparrow}(z)}$ and $\mathcal{B}^{\downarrow}_{i-h_1(z) (\mathrm{mod}\, R), h_2^{\downarrow}(z)}$. If the two sets are disjoint then there is no candidate pair. If the sets are not disjoint, then it is possible that this is due to a 3SUM a solution, but we may have false positives. Notice that the number of set pairs whose intersection we need to examine is $O(nR)$ since once we pick z (n choices) and i (R choices) the rest is implicit.

Set z and let $k = h_2(z)$. Since h_2 is pair-wise independent and linear then for any pair $x, y \in U$ where $x \neq y$ we have that if $x - y \neq z$ then $\Pr[h_2(x) - k = h_2(y)] = \Pr[h_2(x - y) = h_2(z)] = \frac{1}{Q}$. This is where the proofs of the two theorems deviate.

Details for Theorem 1.4. Since each bucket contains at most $3n/R$ elements, the probability of a false positive due to two buckets $\mathcal{B}_i$ and $\mathcal{B}_j$ is $\Pr[h_2(\mathcal{B}_i) - k \cap h_2(\mathcal{B}_j)] \leq (\frac{3n}{R})^2 \frac{1}{Q} = \frac{9}{25}$. In order to reduce the probability of a false positive to be polynomially small, we repeat the process with $O(\log n)$ different choices of h_2 functions (but using the same h_1). This blows up the number of sets by a factor of $O(\log n)$, but not the universe. If the sets intersect under all $O(\log n)$ choices of h_2 then we can spend $O(n/R)$ time to find x and y within buckets $\mathcal{B}_i$ and $\mathcal{B}_j$, which is either part of a 3SUM witness (and the algorithm halts), or a false positive, which only occurs with probability $1/\mathrm{poly}(n)$.

Details for Theorem 1.5. We bound the expected output size from all of the intersections. Since each pair of buckets imply at most $(\frac{3n}{R})^2$ pairs of elements, the expected size of their intersection is $E[|h_2(\mathcal{B}_i) - k \cap h_2(\mathcal{B}_j)|] = (\frac{3n}{R})^2 \frac{1}{Q} = O(\frac{n^{1-\delta}}{R})$. Thus, the expected size of the output of all of the $O(nR)$ intersections is $O(nR \frac{n}{Rn^{\delta}}) = O(n^{2-\delta})$. For each pair in an intersection we can verify in constant time if together with the current z they form a 3SUM witness.

Final details. To summarize, for SetDisjointness (SetIntersection)we create a total of $O(R\sqrt{Q} \log n)$ sets ($O(R\sqrt{Q})$ sets). These sets can be partitioned into two families A and B where all of the $\uparrow$-type sets are in A and all of the $\downarrow$-type sets are in B. All of the intersections we are interested in are between a set from A and a set from B. The universe C of the elements in the sets is of size $O(Q)$. The number of queries is $O(nR \log n) = O(n^{1+\gamma} \log n)$ ($O(nR) = O(n^{1+\gamma})$). □

3 Enumerating Triangles

Following Pătraşcu [35] we express a SetIntersection instance as a tripartite graph in which triangles are in one-to-one correspondence with the elements output by SetIntersection queries.

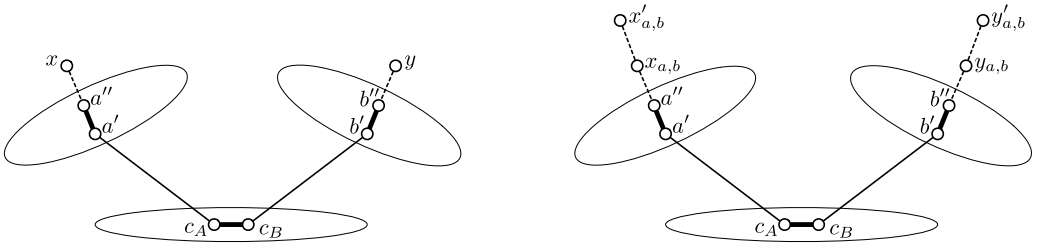
Proof. (of Theorem 1.8) The SetIntersection instance of Theorem 1.5 is interpreted as a graph G on vertex set $A \cup B \cup C$. Each element in C has edges to the sets in A and B that contain it, and the edges between A and B correspond to the SetIntersection queries. So, $|C| = \Theta(n^{1+\delta-\gamma})$, $|A| = |B| = \Theta(\sqrt{n^{1+\delta-\gamma}})$, there are $\Theta(n^{1+\gamma})$ edges between A and B, and at most $O(n^{1-\gamma})$ edges between each vertex in $A \cup B$ and elements in C. Thus the total number of edges between $A \cup B$ and C is at most $O(n\sqrt{n^{1+\delta-\gamma}})$. The expected number of triangles due to false positives in this graph is $O(n^{2-\delta})$.

We prove that enumerating all triangles essentially requires $\Omega(M\alpha)$ time for any feasible combination of N (the number of vertices), M (the number of edges), and α. By feasible we mean that $\alpha = \Theta(M^x) = \Theta(N^y)$, $y \leq 2x$. We assume that each vertex has a degree of at least 2 (since it is easy to filter out other vertices) and then $x \leq y$. Notice that proving a lower bound for $\alpha = \Theta(N^x)$ implies a lower bound for $\Theta(N^{x'})$ for any constant $x' < x$ since one can add singleton vertices.

For our lower bound proof it suffices to consider the case where $n^{1+\gamma} \geq n\sqrt{n^{1+\delta-\gamma}}$, and so $\gamma \geq 1/3 + \delta/3$. Furthermore, this implies that $|A| = |B| > |C|$. Thus, $N = \Theta(\sqrt{n^{1+\delta+\gamma}})$ and $M = \Theta(n^{1+\gamma})$. Our aim is to show that α is at most $O(n^{1-\gamma})$, since for each edge we must spend at least $\Omega(n^{1-\gamma})$ time (assuming the 3SUM conjecture), and so if $\alpha \leq O(n^{1-\gamma})$ we conclude that the total runtime is at least $\Omega(M\alpha)$. However, this may not be the case in G, so we devise a *triangle-preserving* reduction to a new graph G' with N' vertices and $M' = O(M)$ edges such that there is an injective function between triangles in the original graph and triangles in G'. To bound the arboricity α' of G' we show that there exists an orientation of G' with max out-degree $O(n^{1-\gamma})$. It is well known that the maximum out-degree in any orientation must be at least $\Omega(\alpha)$ (see [24]).

Denote by $E(u, V)$ the set of edges between a vertex u and a set of vertices V. Consider a vertex $a \in A$. Since $|E(a, C)| = O(n^{1-\gamma})$, if $|E(a, B)| = O(n^{1-\gamma})$ then we orient all of the edges of a to leave a. However, it is possible that $E(a, B)$ is too large. To deal with this, we create $\lceil \frac{|E(a,B)|}{n^{1-\gamma}} \rceil$ copies of a. The neighbors of a in B are arbitrarily partitioned into $\lceil \frac{|E(a,B)|}{n^{1-\gamma}} \rceil$ sets of size at most $O(n^{1-\gamma})$, and the i^{th} copy of a has as its neighbors the i^{th} set in the partition. All of the edges touching copies of a are oriented outwards from those copies. Furthermore, each copy of a has outgoing edges towards the $O(n^{1-\gamma})$ neighbors of a in C. Thus, the out-degree of each copy of a is at most $O(n^{1-\gamma})$. By orienting all of the edges between B and C to leave B, the out-degree of any vertex in this orientation is at most $O(n^{1-\gamma})$, and so the arboricity of this new graph is at most $\alpha' = O(n^{1-\gamma})$. It is straghtforward to see that this new graph G' is a triangle-perserving graph of G. The number of edges of this graph is $M' = \Theta(M)$ since we only increase the number of edges by adding new edges between copies and C, but each such edge can be charged to an edge between A and B in the initial graph. Also, since there are $\Theta(n^{1+\gamma})$ edges between A and B, the number of copies that are created is at most $O(n^{1+\gamma}/(n^{1-\gamma})) = O(n^{2\gamma})$. Hence, the number of vertices in the new graph is $N' = N + n^{2\gamma}$. Finally, since $\gamma \geq 1/3 + \delta/3$ we have $\frac{1+\delta+\gamma}{2} \leq 2\gamma$ and so $N' = O(n^{2\gamma})$.

To summarize, we have obtained a graph with $M' = \Theta(n^{1+\gamma})$ edges, $N' = O(n^{2\gamma})$ vertices, and $\alpha' = O(n^{1-\gamma})$. Thus, enumerating all of the triangles in $O(M'(\alpha')^{1-\Omega(1)}) = O(n^{2-\Omega(1)})$ time will contradict the 3SUM conjecture. We will now show that this lower bound holds for the entire spectrum of possible polynomial dependencies of α' on N' and M'.



(A) (B)

Figure 3: An illustration of the graph before the set intersection query $a \cap b = \emptyset?$. There is a unique perfect matching before the query: matched edges are drawn thick and unmatched ones thin. Dashed edges are inserted in the course of the query. (A) For the worst case bound we insert edges $(x, a''), (y, b'')$, check if the size of the MCM has increased (implying $a \cap b \neq \emptyset$), then delete them. (B) For the amortized bound we insert new vertices $x_{a,b}, x'_{a,b}, y_{a,b}, y'_{a,b}$ insert edges $(x_{a,b}, a''), (y_{a,b}, b'')$, then check whether the size of the MCM has increased, then insert edges $(x'_{a,b}, x_{a,b}), (y'_{a,b}, y_{a,b})$. Depending on the actual time of these operations, we either do nothing or roll back all edge and vertex insertions.

Recall that we always have $M' \leq N'\alpha'$. Since we can always increase the number of vertices, it is enough to prove that the lower bound holds for all combinations of $M' = N'\alpha'$. This is exactly the case here, since $M' = n^{1+\gamma} = n^{2\gamma}n^{1-\gamma} = N'\alpha'$. Furthermore, we capture the entire spectrum of polynomial dependencies of α' in terms of M' and N'. To see this for M' notice that $\alpha' = {M'}^{\frac{1-\gamma}{1+\gamma}} = M'^x$. As γ admits values between 1/3 and 1 (exclusive), x admits values between 1/2 and 0 (exclusive). Similarly, $\alpha' = {N'}^{\frac{1-\gamma}{2\gamma}} = N'^y$, so y admits values between 0 and 1 (exclusive). □

4 Maximum Cardinality Matching - Theorem 1.11

In this section n denotes the size of the 3SUM instance and N and M denote the number of vertices and edges in the graph on which we compute maximum cardinality matchings.

Due to space considerations we provide here a proof of Theorem 1.11. The discussion of the other results is in Appendix G.

In amortized analysis we are want to bound the total cost of a sequence $S = (\sigma_1, \cdots \sigma_k)$ of k operations. A function f assigns valid amortized costs if $\sum_{i=1}^{k} f(\sigma_i)$ is an upper bound on the empirical cost $\sum_{i=1}^{k} \text{cost}(\sigma_i)$. We prove that if N_i is the number of vertices after σ_i, then $\sum_{i=1}^{k} \text{cost}(\sigma_i)$ is $\Omega(\sum_{i=1}^{k} N_i^{\sqrt{2}-1-o(1)})$, that is, any amortization function f that is a function of the current number of vertices $\hat{N}$ has $f(\hat{N}) = \Omega(\hat{N}^{\sqrt{2}-1-o(1)})$.

Consider the following instance of the incremental MCM problem which is created from an instance of the offline SetDisjointness problem. For each $c \in C$ we create two vertices c_A and c_B with an edge between them. We say that c_A and c_B are copies of c. For each $a \in A$ ($b \in B$) we create two vertices a' and a'' (b' and b'') with an edge between them, and for each $c \in a$ ($c \in b$)there is an edge between a' and c_A (b' and c_B). We say that a' and a'' (b' and b'') are copies of a (b). We also add 2 additional vertices, x and y.

The initialization of this graph is implemented by inserting all of the edges one at a time using

the incremental MCM algorithm. This initial graph has $\Theta(n + n^{2-2\gamma})$ vertices and $\Theta(n^{2-\gamma})$ edges. Notice that this initial graph (without the extra 2 vertices) has a unique perfect matching with the edges between copies. To implement a SetDisjointness query between a and b we add edges (x, a'') and (y, b''). See Figure 3(A). Now, a and b are disjoint iff there is no augmenting path after adding the two edge (details in Appendix G). Thus an increase in the MCM implies that $a \cap b \neq \emptyset$. In order to facilitate additional SetDisjointness queries we undo the *effect* of adding $(x, a''), (y, b'')$, using one of the following two approaches.

Rollback. One approach is to delete the two edges that are added. An incremental data structure can always support deletions of the last element that was inserted by keeping track of the memory modifications that took place during the last insertions, and reversing them in the same time bound. Notice that this approach does not blend well with amortized time bounds since we have no a priori bound on the maximum time per operation.

Creating Perfect Matchings. The second approach is to add another two edges per SetDisjointness query for a total of 4 edges, and create four separate dummy vertices $x_{a,b}, x'_{a,b}, y_{b,a}$, and $y'_{b,a}$ associate with each SetDisjointness query on $a \in A$ and $b \in B$. For the SetDisjointness query we add edges $(x_{a,b}, a'')$ and $(y_{b,a}, b'')$ to the graph and as before the MCM increases iff the sets a and b intersect. See Figure 3(B). Then, we add edges $(x_{a,b}, x'_{a,b})$ and $(y_{b,a}, y'_{b,a})$ which guarantee that the resulting graph has a perfect matching that is comprised of the perfect matching of the graph prior to the insertion of the 4 edges together with edges $(x_{a,b}, x'_{a,b})$ and $(y_{b,a}, y'_{b,a})$. The MCM has increased by 2 after the insertion of the 4 edges, regardless of whether the sets intersect or not. The downside of this approach is that the number of vertices grows with the number of SetDisjointness queries, leading to weaker lower bounds in terms of the number of vertices.

Combining the Two. Assume that the amortized cost of each edge or vertex insertion is $\hat{N}^\alpha$ for some constant $\alpha > 0$, where $\hat{N}$ is the number of vertices in the graph when the insertion takes place. To answer a SetDisjointness query we first add the four vertices and four edges, thereby creating a perfect matching. If the insertion time of these vertices and edges is less than $9\hat{N}^\alpha$ (within $\hat{N}^\alpha$ of the budget for 8 insertions) then we rollback the insertion. Otherwise, we leave the four edges and continue to the next SetDisjointness query. Intuitively, our goal with this combined method is to guarantee that the graph does not grow by too much, while maintaining the amortized cost (to obtain a higher lower-bound). SetDisjointness queries for which we perform a rollback cost $O(\hat{N}^\alpha)$ time each. By assumption the subsequence of remaining operations (those inserts used to set up the initial graph and subsequent inserts not rolled back) has amortized cost $O(\hat{N}^\alpha)$.

Next we bound the number of vertices at the end of the process, denoted by N. After the graph setup there is $O(n^{2-\gamma}(n + n^{2-2\gamma})^\alpha)$ credit for performing expensive insertions later. For our proof we will focus on $\gamma \leq 1/2$ and so the amount of credit becomes $O(n^{2-\gamma}n^\alpha)$. Each expensive SetDisjointness query uses up at least $\hat{N}^\alpha$ of that credit, and so the total credit used during all of the expensive insertions is at least $\Omega(\sum_{i=0}^{N}(n+i)^\alpha) = \Omega(N^{1+\alpha})$. Since we can never be in credit debt, we have that $n^{2-\gamma}n^\alpha \geq \Omega(N^{1+\alpha})$ and so $N \leq O(n^{\frac{2-\gamma+\alpha}{1+\alpha}})$.

The number of cheaper insertions that we rolled back is $O(n^{1+\gamma})$. Each one of these costs at most N^α. So the total time of the entire sequence of operations which solves 3SUM is $O(n^{1+\gamma}N^\alpha + N^{1+\alpha}) \leq O(n^{1+\gamma+\frac{(2-\gamma+\alpha)\alpha}{1+\alpha}} + n^{2-\gamma+\alpha})$. Given the 3SUM conjecture this runtime cannot be $O(n^{2-\Omega(1)})$, so up to a $o(1)$ term we must have $2 \leq \max\{1+\gamma+\frac{(2-\gamma+\alpha)\alpha}{1+\alpha}, 2-\gamma+\alpha\}$. The two terms are equal when $\gamma = \frac{1}{2+\alpha}$ and then $2 \leq 2-\gamma+\alpha$ implying that $\alpha \geq \gamma = \frac{1}{2+\alpha}$. Thus, α must be at least $\sqrt{2}-1 > 0.414$.

References

[1] Amir Abboud and Virginia 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] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP (1)*, pages 39–51, 2014.

[3] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 41–50, 2015.

[4] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP (1)*, pages 114–125, 2014.

[5] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

[6] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP (1)*, pages 223–234, 2014.

[7] Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 384–393, 2014.

[8] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 31–40, 2015.

[9] Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In Leah Epstein and Paolo Ferragina, editors, *ESA*, volume 7501 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2012.

[10] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[11] Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010.

[12] M. Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.

[13] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algor.*, 25(1):19–51, 1997.

[14] R. Duan and S. Pettie. Connectivity oracles for failure prone graphs. In *Proceedings 42nd ACM Symposium on Theory of Computing*, pages 465–474, 2010.

[15] Johannes Fischer, Travis Gagie, Tsvi Kopelowitz, Moshe Lewenstein, Veli Mäkinen, Leena Salmela, and Niko Välimäki. Forbidden patterns. In *LATIN*, 2012.

[16] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

[17] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In *Automata, Languages, and Programming - 38th International Colloquium, ICALP (2)*, 2011.

[18] Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. String retrieval for multi-pattern queries. In *SPIRE*, 2010.

[19] Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Document listing for queries with excluded pattern. In *CPM*, 2012.

[20] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[21] Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, triangles. *Algorithmica*, pages 1–18, 2014.

[22] Allan Grønlund Jørgensen and Seth Pettie. Threesomes, degenerates, and love triangles. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 621–630, 2014.

[23] B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

[24] Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *ICALP (2)*, pages 532–543, 2014.

[25] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic set intersection. In *Proceedings 14th Int'l Symposium on Algorithms and Data Structures (WADS)*, 2015.

[26] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proceedings 44th Symposium on Theory of Computing (STOC)*, pages 85–94, 2012.

[27] Kasper Green Larsen, J. Ian Munro, Jesper Sindahl Nielsen, and Sharma V. Thankachan. On hardness of several string indexing problems. In *CPM*, 2014.

[28] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In David Eppstein, editor, *SODA*, pages 657–666. ACM/SIAM, 2002.

[29] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees in finite graphs. *Journal of the London Mathematical Society*, 36(1):445–450, 1961.

[30] C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39(1):12, 1964.

[31] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th ACM Symposium on Theory of Computing, STOC*, pages 745–754, 2013.

[32] M. Pătraşcu and E. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006.

[33] M. Pătraşcu and M. Thorup. Planning for fast connectivity updates. In *Proceedings 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007.

[34] Mihai Patrascu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 559–568, 2011.

[35] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *STOC*, pages 603–610. ACM, 2010.

[36] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings 8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 118–126, 2007.

[37] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[38] R. Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, 2014. Technical report available as arXiv:1312.6680.

[39] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

A Proof of Theorem 1.6

Definition A.1. (Universality and Linearity) *Let $\mathcal{H}$ be a family of hash functions from $[u] \to [m]$.*

1. *$\mathcal{H}$ is called* c-universal *if for any distinct $x, x' \in [u]$,*

$$\Pr_{h \in \mathcal{H}}(h(x) = h(x')) \leq \frac{c}{m}.$$

2. *$\mathcal{H}$ is called* almost linear *if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$,*

$$h(x) + h(x') \in h(x + x') + \{-1, 0\} \pmod m.$$

In our application any $O(1)$-universal almost linear hash function suffices.

Theorem A.1. *(Dietzfelbinger, Hagerup, Katajainen, and Penttonen 1997 [13]) Let u and m be powers of two, with $m < u$. The family $\mathcal{H}_{u,m}$ is 2-universal and almost linear, where*

$$\mathcal{H}_{u,m} = \{h_a : [u] \to [m] \mid a \in [u] \text{ is an odd integer}\}$$
$$\text{and } \; h_a(x) = (ax \bmod u) \operatorname{div} (u/m).$$

Because the modular arithmetic and division are by powers of two, the hash functions of Theorem A.1 are very easy to implement using standard multiplication and shifts. If $u = 2^w$, where w is the number of bits per word, and $m = 2^s$, the function is written in C as `(a*x) >> (w-s)`. Dietzfelbinger et al. [13] proved that it is 2-universal. It is clearly almost linear.

A.1 Hashing and Coding Preliminaries

The reduction in the next section makes use of any constant rate, constant relative distance binary code. The expander codes of Sipser and Spielman [37] are sufficient for our application.

Theorem A.2. *(See Sipser and Spielman [37]) There is a constant $\epsilon > 0$ such that for any sufficiently large $\delta > \delta(\epsilon)$, there is a binary code $C : \{0,1\}^N \to \{0,1\}^{\delta N}$ such that for any $x, y \in \{0,1\}^N$, the Hamming distance between $C(x)$ and $C(y)$ is at least $\epsilon \cdot \delta N$. Moreover, $C(x)$ can be computed in $O(\delta N)$ time.*

A.2 The Reduction

Let $[u]\backslash\{0\} = [2^w]\backslash\{0\}$ be the universe. It is convenient to assume that 0 is excluded from A, but this is without loss of generality since all witnesses involving 0 can be enumerated in $O(n \log n)$ time by sorting A. Choose L hash functions $(h_i)_{i \in [L]}$ independently from $\mathcal{H}_{u,m}$, where $m = 2^{\lceil \log n \rceil}$ is the least power of two larger than n. Ideally a hash function will map A injectively into the buckets $[m]$, or at least put a constant load on each bucket, but this cannot be guaranteed. Some buckets will be overloaded and the items in them discarded.

Definition A.2. (Overloaded Buckets, Discarded Elements) *For each $i \in [L]$ and $j \in [m]$ define*

$$\text{BUCKET}_i(j) = \{x \in A \mid h_i(x) = j\}$$

to be the set of elements hashed by h_i to the jth bucket. The truncation of this bucket is defined as

$$\textsc{bucket}_i^\star(j) = \begin{cases} \textsc{bucket}_i(j) & \text{if } |\textsc{bucket}_i(j)| \le T, \text{ and} \\ \emptyset & \text{otherwise,} \end{cases}$$

where $T = O(1)$ is a constant threshold to be determined. If $\textsc{bucket}_i^\star(j) = \emptyset$ we say that the elements of $\textsc{bucket}_i(j)$ were discarded by h_i*. An element is called* bad *if it is discarded by a $4/T$-fraction of the hash functions.*

Lemma A.3. *The probability that an element is bad is at most $\exp\left(-\frac{2L}{3T}\right)$.*

Proof. Since each h_i is 2-universal, the expected number of other elements in x's bucket is, by linearity of expectation, at most $2(n-1)/m < 2$. By Markov's inequality the probability that x is discarded by h_i is less than $2/T$. Let X be the number of hash functions that discard x, so $\mathbb{E}(X) < 2L/T$. By definition x is bad if $X > 4L/T > 2 \cdot \mathbb{E}(X)$. Since the hash functions were chosen independently, by a Chernoff bound, $\Pr(x \text{ is bad}) < \exp\left(-\frac{2L}{3T}\right)$. □

We will set $T = O(1)$ and $L = \Theta(\log n)$ to be sufficiently large so that the probability that no elements are bad is $1 - 1/\operatorname{poly}(n)$. We proceed under the assumption that there are no bad elements.

Lemma A.4. *Suppose there are no bad elements with respect to $(h_i)_{i\in[L]}$. For any three $a, b, c \in A$, there are more than $\left(1 - \frac{12}{T}\right) L$ indices $i \in [L]$ such that h_i discards none of $\{a, b, c\}$.*

Proof. Each of a, b, c is discarded by less than $4L/T$ hash functions, so none are discarded by at least $L - 12L/T$ hash functions. □

Let $\delta > 1, \epsilon > 0$ be the parameters of Theorem A.2, where $N = \lceil \log n \rceil$ and $L = \delta N$. We assign each $x \in A$ an L-bit codeword C_x such that any two C_x, C_y disagree in at least ϵL positions.

We will make $8TL$ calls to an Convolution3SUM algorithm on vectors $\{A_\ell\}_{\ell\in[L]\times\{-1,0\}\times\{0,1\}\times[2T]}$, each of length $14m = O(n)$. For reasons that will become clear we index the calls by tuples $\ell = (i, \alpha, \beta, \gamma) \in [L] \times \{-1, 0\} \times \{0, 1\} \times [2T]$. The first coordinate i of ℓ identifies the hash function. The second coordinate α indicates that we are looking for witnesses $a, b, a + b \in A$ for which $h_i(a) + h_i(b) = h_i(a+b) + \alpha \pmod m$. A natural way to define A_ℓ creates multiple copies of elements but can lead to a situation where there are false positives: we may have $A_\ell(p) + A_\ell(q) = A_\ell(p + q)$ and yet this is not a witness for the original 3SUM instance because $A_\ell(p) = A_\ell(q)$.[4] In each call to Convolution3SUM we look for witnesses where each element can play the role of either "p" or "q" in the example above, but not both; all elements will be eligible to play the role of "$p + q$." The parity of $C_x(i)$ XOR β tells us which roles x is allowed to play, where β is the third coordinate of ℓ. The fourth coordinate γ of ℓ effects a cyclic shift of the order of elements within a bucket.

Each vector A_ℓ is partitioned into $2m$ contiguous *blocks*, each of length $7T$. Many of the locations of A_ℓ are filled with a dummy value ∞, which is some sufficiently large number that cannot be part of any witness, say $2\max(A) + 1$. The elements of the jth bucket each appear three times in A_ℓ, twice in the first half and once in the second.

[4] This minor bug appears in Pătraşcu's reduction from 3SUM to Convolution3SUM.

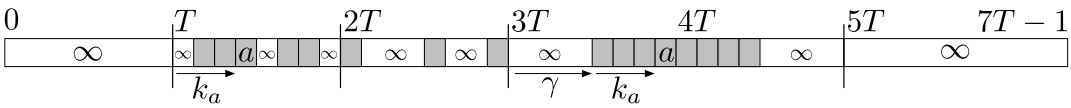


Figure 4: Block j in A_ℓ occupies positions $j(7T)$ through $(j+1)(7T)-1$. In the first half of A_ℓ, a block is partitioned into five intervals. The first interval covers positions 0 through $T-1$ and is always filled with a dummy value ∞. The second and third intervals run, respectively, from positions T through $2T-1$ and positions $2T$ through $3T-1$. They contain those elements $x \in \textsc{bucket}^\star_i(j-\alpha)$ for which $C_x(i) \textsc{ xor } \beta$ is, respectively, 0 and 1. The fourth interval runs from positions $3T$ through $5T-1$ and contains *all* members of $\textsc{bucket}^\star_i(j-\alpha)$, cyclically shifted by γ. The last interval, from positions $5T$ through $7T-1$, is always filled with dummies. The composition of a block j in the *second* half of A_ℓ is similar, except that the second and third intervals (positions T through $3T-1$) contain only dummies, and the fourth interval contains all members of $\textsc{bucket}^\star_i((j-\alpha) \bmod m)$.

Order the elements of $\textsc{bucket}^\star_i(j)$ arbitrarily as $(x(i,j,k))_{k\in[T]}$, where $x(i,j,k)$ does not exist if $k \geq |\textsc{bucket}^\star_i(j)|$. Define the vector $A_{(i,\alpha,\beta,\gamma)}$ as follows.

$$A_{(i,\alpha,\beta,\gamma)}(j(7T)+t) = \begin{cases} x(i,j,k) & \text{when } t = T+k,\ k\in[T], \text{ and } C_{x(i,j,k)}(i) \textsc{ xor } \beta = 0, \\ x(i,j,k) & \text{when } t = 2T+k,\ k\in[T], \text{ and } C_{x(i,j,k)}(i) \textsc{ xor } \beta = 1, \\ x(i,(j-\alpha) \bmod m, k) & \text{when } t = 3T + ((k+\gamma) \bmod 2T) \text{ and } k\in[T], \\ \infty & \text{in all other cases.} \end{cases}$$

The last case applies when j, k, or t is out of range or if the given element, say $x(i,j,k)$, does not exist because $|\textsc{bucket}^\star_i(j)| \leq k$. See Figure 4.

Lemma A.5. (No False Negatives) *Suppose $a, b, a+b \in A$ is a witness to the 3SUM instance A. For some $\ell = (i,\alpha,\beta,\gamma)$, this is also a witness in the Convolution3SUM instance A_ℓ.*

Proof. Set the threshold $T = 12/\epsilon = O(1)$. By Lemma A.4 there are more than $L(1-12/T) = L(1-\epsilon)$ indices $i \in [L]$ such that none of $\{a, b, a+b\}$ are discarded by h_i. Moreover, by the properties of the error correcting code (Theorem A.2) there are at least ϵL indices i for which $C_a(i) \neq C_b(i)$, which implies that both criteria are satisfied for at least one i. Fix any such i.

Let $j_a = h_i(a), j_b = h_i(b)$, and $j_{a+b} = h_i(a+b)$ be the bucket indices of a, b, and $a+b$. Let k_a, k_b, k_{a+b} be their positions in those buckets, that is, $a = x(i, j_a, k_a)$ and $b = x(i, j_b, k_b)$, and $a+b = x(i, j_{a+b}, k_{a+b})$. Without loss of generality $j_a \leq j_b$. Let $\beta = C_a(i)$, so $C_a(i) \textsc{ xor } \beta = 0$ and $C_b(i) \textsc{ xor } \beta = 1$. Let $\alpha \in \{-1, 0\}$ be such that $h_i(a) + h_i(b) \equiv h_i(a+b) + \alpha \pmod m$.

In the vector $A_{(i,\alpha,\beta,\gamma)}$,

- a is at position $j_a(7T) + T + k_a$, because $C_a(i) \textsc{ xor } \beta = 0$,
- b is at position $j_b(7T) + 2T + k_b$, because $C_b(i) \textsc{ xor } \beta = 1$,
- and since $j_{a+b} \equiv j_a + j_b - \alpha \pmod m$, $a+b$ is at position $(j_a+j_b)(7T) + 3T + ((k_{a+b}+\gamma) \bmod 2T)$.

Thus, for $\gamma = (k_a + k_b - k_{a+b}) \bmod 2T$, the triple $(a, b, a+b)$ forms a witness for the Convolution3SUM vector A_ℓ. □

Lemma A.6. (No False Positives) *If $(a, b, a+b)$ is a witness in some Convolution3SUM instance A_ℓ, it is also a witness in the original 3SUM instance A.*

Proof. None of $\{a, b, a+b\}$ can be the dummy ∞ in A_ℓ, so they must all be members of A. The only way it cannot be an witness for 3SUM is if $b = a$, that is, $(a, a, 2a)$ is not a triple of distinct numbers. If a is not discarded, it appears at exactly three positions in A_ℓ. Regardless of the bit $C_a(i)$, a appears at both

$$\begin{aligned} & A_\ell((j_a + \alpha)(7T) + 3T + ((k_a + \gamma) \bmod 2T)) \\ \text{and } & A_\ell((m + j_a + \alpha)(7T) + 3T + ((k_a + \gamma) \bmod 2T)), \end{aligned}$$

for some $k_a \in [T]$ and $\gamma \in [2T]$.

Depending on the parity of $C_x(i)$ XOR β, a also appears at either

$$\begin{aligned} & A_\ell(j_a(7T) + T + k_a) \\ \text{or } & A_\ell(j_a(7T) + 2T + k_a). \end{aligned}$$

For $(a, a, 2a)$ to be a Convolution3SUM witness we would need $2a$ to appear either at

$$\begin{aligned} & A_\ell\big((2j_a + \alpha)(7T) + 4T + k_a + ((k_a + \gamma) \bmod 2T)\big) \\ \text{or } & A_\ell\big((2j_a + \alpha)(7T) + 5T + k_a + ((k_a + \gamma) \bmod 2T)\big). \end{aligned}$$

However, in both of those positions A_ℓ is ∞ by definition. See Figure 4. □

A.3 Conclusions

We have shown that the randomized (Las Vegas) complexities of 3SUM and Convolution3SUM are equivalent up to a logarithmic factor. Since hashing plays such an essential role in the reduction, it would be surprising if our construction could be efficiently derandomized, or if it could be generalized to show that 3SUM and Convolution3SUM over the reals are essentially equivalent.

The $O(\log n)$-factor gap in Theorem 1.6 stems from our solution to two technical difficulties, (i) ensuring that all triples appear in lightly loaded buckets with respect to a large fraction of the hash functions, and (ii) ensuring that no non-3SUM witnesses $(a, a, 2a)$ occur as witnesses in any Convolution3SUM instance. We leave it as an open problem to show that 3SUM and Convolution3SUM are asymptotically equivalent, without the $O(\log n)$-factor gap.

B The Reporting Version of Set Intersection

Theorem B.1. *Assume the 3SUM conjecture. For any $0 \le \gamma < 1$, $\delta > 0$, any data structure for SetIntersection with expected preprocessing time t_p, amortized expected query time t_q, and the amortized expected time to report an element in the output is t_r has*

$$t_p + N^{\frac{2(1+\gamma)}{3+\delta-\gamma}} t_q + N^{\frac{2(2-\delta)}{3+\delta-\gamma}} t_r = \Omega\left(N^{\frac{4}{3+\delta-\gamma} - o(1)}\right).$$

Notice that, if we only allow linear preprocessing time and a constant reporting time, then by making γ and δ arbitrarily small we obtain a query time lower bound of $\Omega(N^{\frac{2}{3}-o(1)})$. In this case the expected size of output during each query is $O(N^{\frac{2}{3}-\Omega(1)})$.

Proof. We use the same reduction as the one in the proof of Theorem 1.7. Using Theorem 1.5, we have $N = \Theta(n^{1-\gamma}\sqrt{n^{1+\delta-\gamma}}) = \Theta(n^{\frac{3+\delta-\gamma}{2}})$, the number of queries is $\Theta(n^{1+\gamma}) = \Theta(N^{\frac{2(1+\gamma)}{3+\delta-\gamma}})$, and the total size of the output is $\Theta(n^{2-\delta}) = \Theta(N^{\frac{2(2-\delta)}{3+\delta-\gamma}})$. Thus, we obtain the following lower bound tradeoff:

$$t_p + N^{\frac{2(1+\gamma)}{3+\delta-\gamma}} t_q + N^{\frac{2(2-\delta)}{3+\delta-\gamma}} t_r = \Omega(n^{2-o(1)}) = \Omega\left(N^{\frac{4}{3+\delta-\gamma}-o(1)}\right).$$

□

C Almost Linear and Almost Balanced Hashing

We will now describe how to overcome the assumption that there exist pair-wise independent hash functions that are *magically* linear and balanced. In this section we use a more general definition of *almost linear* than the one used in Section A or by Baran et al. [5] and Pătraşcu [35]. Whereas Section A required any $O(1)$-universal almost linear hash family, here we require a pairwise independent (and exactly 1-universal) almost linear hash family. The hash family [13] proposed by Baran et al. [5] and Pătraşcu [35] is almost linear (under Section A's definition) but it is only known to be 2-universal [13, Lem. 2.4] and is definitely not pairwise independent. This issue was also noted in [21].

A family $\mathcal{H}$ of hash functions from $[u] \to [m]$ is called *almost linear* if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$, either $h(x)+h(x') = h(x+x')+c_h \pmod m$, or $h(x)+h(x') = h(x+x')+c_h+1 \pmod m$, where c_h is some integer that depends only on h. Given a hash function $h \in \mathcal{H}$ we say that a value $i \in m$ is heavy for set $S = \{x_1, \ldots, x_n\} \subset [u]$ if $|\{x \in S : h(x) = i\}| > \frac{3n}{m}$. $\mathcal{H}$ is called *almost balanced* if for any set $S = \{x_1, \ldots, x_n\} \subset [u]$, the expected number of elements from S that are hashed to heavy values is $O(m)$.

We will show that there exists a family $\mathcal{H}$ of pairwise independent hash functions that is almost linear and almost balanced, which is suitable for use in the reductions of Theorems 1.4 and 1.5. Each step in the proofs of Theorems 1.4 and 1.5 that used linearity can be replaced by two parallel steps making use of the almost linearity of $\mathcal{H}$. The reduction algorithm must consider both options $h(x) + h(x') = h(x + x') + c_h \pmod m$, and $h(x) + h(x') = h(x + x') + c_h + 1 \pmod m$. This bifurcation of the possible cases only blows up the running time by a factor of four: whereas before we assumed h_1 and h_2 were perfectly linear, we now have to entertain two options for h_1 hash values and two options for h_2 hash values.

The balance assumption is overcome by directly verifying all of the elements that are assigned a heavy value to see if they are part of a solution for 3SUM. This takes $O(n)$ time per element. Since the expected number of such elements in is $O(n^\gamma)$ the total time spent on such elements is $O(n^{1+\gamma})$ which is the same as the number of intersections in the reductions, and so we can ignore this cost in our analysis. For the rest of the elements (those that are not assigned to heavy values) we proceed as in the proofs of Theorem 1.4 and 1.5.

The Hash Family. Baran et. al. [5] showed that any 1-universal family of hash functions is almost balanced; see [21] for an additional proof. Rather than use the family of [13], we use one analyzed by Dietzfelbinger [12].

Theorem C.1. *([12, Theorem 3]) The family $\mathcal{H}_{u,m,r}$ defined below is pairwise independent and*

hence 1-universal whenever $r = km$ for some $k \geq u/2$, and u, m, and r are all powers of 2.

$$\mathcal{H}_{u,m,r} = \{h_{a,b} : [u] \to [m] \mid a \in [r] \text{ is an odd integer and } b \in [r]\}$$
$$\text{and } h_{a,b}(x) = (ax + b \bmod r) \operatorname{div} (r/m)$$

Since $\mathcal{H}_{u,m,r}$ is 1-universal it is also almost balanced [5, 21]. We need to also prove that it is almost linear.

Lemma C.2. *The family $\mathcal{H}_{u,m,r}$ is almost linear, with $c_{h_{a,b}} = (b - 1 \bmod r) \operatorname{div} (r/m)$.*

Proof. Consider two elements $x, x' \in [u]$. Consider the function $g(x) = g_{a,b}(x) = ax + b \bmod r$. Then $g(x) + g(x') \bmod r = a(x + x') + 2b \bmod r = g(x + x') + b \bmod r$. Cutting off the least significant bits in the computation by considering $h = h_{a,b}$ instead of $g_{a,b}$ can only affect a missing carry in the computation, and so either $(h(x) + h(x') \bmod m) = (h(x + x') + b \bmod m)$ or $(h(x) + h(x') \bmod m) = (h(x + x') + b - 1 \bmod m)$. Hence by setting $c_h = b - 1 \bmod m$ we have that $\mathcal{H}_{u,m,r}$ is almost linear. □

D d-Failure Connectivity

In the *d-Failure Connectivity Oracle* problem we wish to preprocess an undirected graph $G = (V, E)$ and some integer parameter $d > 0$ in order to support: (1) updates to G in which a set F of up to d vertices are deleted and (2) connectivity queries in which we determine whether a given s and t are in the same connected component of the graph induced by $V(G) \backslash F$.

In Duan and Pettie's [14] d-failure connectivity structure the preprocessing and deletion times are $O(mn^{1/c} \operatorname{poly}(\log(n)))$ and $O(d^{2c+4} \operatorname{poly}(\log n))$, where $c \geq 1$ is an integer parameter. The query time is $O(d)$, independent of c.

Theorem D.1. *Assume the 3SUM conjecture. For any $1/2 \leq \gamma < 1$ suppose there is a d-failure connectivity structure for $d^{\frac{2-\gamma}{2-2\gamma}}$-edge, $d^{\frac{1}{2-2\gamma}}$-vertex graphs with expected preprocessing time t_p, amortized expected deletion time t_d, and amortized expected query time t_q. Then,*

$$t_p + d^{\frac{1}{2-2\gamma}} \cdot t_d + d^{\frac{1+\gamma}{2-2\gamma}} \cdot t_q = \Omega(d^{\frac{1}{1-\gamma} - o(1)}).$$

This lower bound shows that with preprocessing and deletion times similar to [14], the time to answer a connectivity query must be $\Omega(\sqrt{d}/f(n))$, where the complexity of 3SUM is $O(n^2/f(n))$. In particular, if f is polylogarithmic, then the query time must be $\tilde{\Omega}(\sqrt{d})$. The connectivity oracles of [33, 14, 23] answer queries in $O(\log \log n)$ time after d *edge* deletions, independent of d. Our lower bound precludes such a $\tilde{O}(1)$ query time for d *vertex* deletions.

Proof. We reduce the SetDisjointness problem to the d-failure connectivity as follows. We make use of Theorem 1.4 and set $d = O(n^{2-2\gamma}) = |C|$. Construct a tripartite graph $G = (V, E)$ on vertices $V = A \cup B \cup C$ and edges

$$E = \{(a, c) \mid c \in a\} \cup \{(b, c) \mid c \in b\}$$

We now need to answer $n^{1+\gamma} = O(d^{\frac{1+\gamma}{2-2\gamma}})$ SetDisjointness queries using a black-box data structure for d-failure connectivity on G. For each $a \in A$ separately we perform up to d deletions and then answer *all* SI queries involving a using connectivity queries. To do this we delete all vertices in C

that correspond to elements not in a and let $G[a]$ be the resulting graph. Notice that in $G[a]$, a is only connected to sets in $A \cup B$ that intersect a. We can therefore answer any SetDisjointness query "$a \cap b = \emptyset$?" by asking one connectivity query in $G[a]$.

Observe that G is an M-edge, N-vertex graph where $N = |A|+|B|+|C| = O(n \log n) = \tilde{O}(d^{\frac{1}{2-2\gamma}})$ and $M = O(n^{2-\gamma}) = O(d^{\frac{2-\gamma}{2-2\gamma}})$. Thus, $t_p + (n \log n) \cdot t_d + n^{1+\gamma} \cdot t_q = \Omega(n^{2-o(1)})$. Substituting $n = \Omega(d^{\frac{1}{2-2\gamma}})$ completes the proof. □

E Distance Oracles for Colors

Distance Oracles for Colors. Let S be a set of points in some metric with distance function $d(\cdot,\cdot)$, where each point $p \in S$ has some associated colors $C(p) \subset [\ell]$. For $c \in [\ell]$ we denote by $P(c)$ the set of points from S with color c. We generalize d so that the distance between a point p and a color c is denoted by $d(p,c) = \min_{q \in P(c)}\{d(p,q)\}$. In the *(Approximate) Distance Oracles for Vertex-Labeled Graphs problem* [17] [9] we are interested in preprocessing S so that given a query of a point q and a color c we can return $d(q,c)$ (or some approximation). We further generalize d so that the distance between two colors c and c' is denoted by $d(c,c') = \min_{p \in P(C)}\{d(p,C')\}$. In the *Distance Oracle for Colors problem* we are interested in preprocessing S so that given two query colors c and c' we can return $d(c,c')$. In the *Approximate Distance Oracle for Colors problem* we are interested in preprocessing S and some constant $\alpha > 1$ so that given two query colors c and c' we can return some value $\hat{d}$ such that $d(c,c') \leq \hat{d} \leq \alpha d(c,c')$.

For a text T and a pattern P let $L(P,T)$ be the set of locations in which P occurs in T. A special case of the Distance Oracle for Colors problem is the *Snippets problem* in which one is given a text T of length N to preprocess so that given pattern queries $P_1, P_2, \cdots, P_k$ one can quickly compute $\min_{1 \leq i \leq N}\{\max_{1 \leq j \leq k}\{\min_{o \in L(P_i,T)} |o - j|\}\}$. In words, we are interested in the location in the text which minimizes the distance to maximum distance to any queried pattern . This problem is of interest for search engines where one is interested in demonstrating the relevance of documents or webpages to queried patterns. A common method of demonstrating such relevance is by providing a so called snippet of the document in which the queried patterns appear close to each other.

We show evidence of the hardness of the Distance Oracle for Colors problem and the Approximate Distance Oracle for Colors problem by focusing on the 1-D case (the snippets).

Theorem E.1. *Assume the 3SUM conjecture. Suppose there is a 1-D Distance Oracle for Colors with constant stretch $\alpha \geq 1$ algorithm for an array of size N with expected preprocessing time t_p and amortized expected query time t_q. Then for any constant $0 < \gamma < 1$*

$$t_p + N^{\frac{1+\gamma}{2-\gamma}} \cdot t_q(N) = \Omega(N^{\frac{2}{2-\gamma}-o(1)}).$$

Notice that by making γ as small as possible then if the preprocessing time is linear we obtain a lower bound $t_q = \Omega(N^{1/2-o(1)})$, even in expectation. Furthermore, if the preprocessing time is truly subquadratic then the query time lower bound can be forced to be a polynomial by making γ large enough.

Proof. We use Theorem 1.4 and reduce offline SetDisjointness to the Colored Distance problem as follows. Let $A \cup B = \{S_1, \cdots, S_{\Theta(n)}\}$. For each S_i we define a unique color c_i. For an element $e \in C$ let $|e|$ denote the number of subsets containing e and notice that in expectation $\sum_{e \in C} |e| = \Theta(n^{2-\gamma})$. Since each element in C appears in at most $O(n)$ subsets, we partition C into $\Theta(\log n)$ parts where

the i^{th} part P_i contains all of the elements $e \in C$ such that $2^{i-1} < |e| \leq 2^i$. An array X_i is constructed from $P_i = \{e_1, \cdots e_{|P_i|}\}$ by assigning an interval $I_j = [f_j, \ell_j]$ in X_i to each $e_j \in P_i$ such that no two intervals overlap. Every interval I_j contains a list of all of the colors of subsets in $A \cup B$ that contain e_j. This implies that $|I_j| = |e_j| \leq 2^i$. Furthermore, for each e_j and e_{j+1} we separate I_j from I_{j+1} with a dummy color d listed $2^i + 1$ times at locations $[\ell_j + 1, f_{j+1} - 1]$. Finally, we pad each X_i so that its size is $N = \Theta(n^{2-\gamma})$. This is always possible since $\sum_{e \in C} |e| = \Theta(n^{2-\gamma})$.

We can now simulate a SetDisjointness query on subsets $(S_i, S_j) \in A \times B$ by performing a colored distance query on colors c_i and c_j in each of the $\Theta(\log n)$ arrays. There exists a P_i for which the two points returned from the query are at distance strictly less than $2^i + 1$ if and only if there is an element in C that is contained in both S_i and S_j. The number of SetDisjointness queries that need to be decided is $t = \Theta(n^{1+\gamma})$. For the CLB we restrict our attention to the array on which the algorithm spends the most time to preprocess and perform all of the queries. This array has size $N = \Theta(n^{2-\gamma})$. So the total runtime is at most a $O(\log n)$ factor of the time spent on this array. Thus,

$$\begin{aligned}
\log n(t_p + n^{1+\gamma} \cdot t_q) &= O(\log N(t_p + N^{\frac{1+\gamma}{2-\gamma}} \cdot t_q)) \\
&= \Omega(N^{\frac{2}{2-\gamma} - o(1)}) \\
&= \Omega(n^{2-o(1)}).
\end{aligned}$$

Finally, notice that the lower bound also holds for the approximate case, as for any constant α the reduction can overcome the α approximation by separating intervals using $\alpha 2^i + 1$ listings of d. □

F Document Retrieval Problems with Multiple Patterns

Two Patterns Document Retrieval. In the *Document Retrieval problem* [28] we are interested in preprocessing a collection of documents $X = \{D_1, \cdots, D_k\}$ where $N = \sum_{D \in X} |D|$, so that given a pattern P we can quickly report all of the documents that contain P. Typically, we are interested in run time that depends on the number of documents that contain P and not in the total number of occurrences of P in the entire collection of documents. In the *Two Patterns Document Retrieval problem* we are given two patterns P_1 and P_2 during query time, and wish to report all of the documents that contain both P_1 and P_2. We consider two versions of the Two Patterns Document Retrieval problem. In the decision version we are only interested in detecting if there exists a document that contains both patterns. In the reporting version we are interested in enumerating all documents that contain both patterns.

All known solutions for the Two Patterns Document Retrieval problem with non trivial preprocessing use at least $\Omega(\sqrt{N})$ time per query [28],[11],[18],[19]. In a recent paper, Larsen, Munro, Nielsen, and Thankachan [27] show lower bounds for the Two Patterns Document Retrieval problem conditioned on the hardness of boolean matrix multiplication. We provide some additional evidence of hardness conditioned on the 3SUM conjecture.

It is straightforward to see that the appropriate versions of the two pattern document retrieval problem solve the corresponding versions of the intersection problems. In particular, this can be obtained by creating an alphabet Σ which corresponds to all of the sets in $A \cup B$. For each $c \in C$ we create a document that contains the characters corresponding to the sets that contain c. The intersection between $a \in A$ and $b \in B$ directly corresponds to all the documents that contain both

a and b. Thus, all of the lower bound tradeoffs for intersection problems are lower bound tradeoffs for the two pattern document retrieval problem.

Theorem F.1. *Assume the 3SUM conjecture. For any $0 < \gamma < 1$, any data structure for the Two Patterns Document Retrieval problem for a collection of documents X where $N = \sum_{D \in X} |D|$, with expected preprocessing time t_p and amortized expected query time t_q.*

$$t_p + N^{\frac{1+\gamma}{2-\gamma}} t_q = \Omega\left(N^{\frac{2}{2-\gamma} - o(1)}\right).$$

Theorem F.2. *Assume the 3SUM conjecture. For any $0 \leq \gamma < 1$, $\delta > 0$, any data structure for the Two Patterns Document Retrieval algorithm for a collection of documents X where $N = \sum_{D \in X} |D|$, with expected preprocessing time t_p, the amortized expected query time t_q, and amortized expected time to report each document in the output t_r has*

$$t_p + N^{\frac{2(1+\gamma)}{3+\delta-\gamma}} t_q + N^{\frac{2(2-\delta)}{3+\delta-\gamma}} t_r = \Omega\left(N^{\frac{4}{3+\delta-\gamma} - o(1)}\right).$$

Forbidden Pattern Document Retrieval. In the *Forbidden Pattern Document Retrieval problem* [15] we are also interested in preprocessing the collection of documents but this time given a query P^+ and P^- we are interested in reporting all of the documents that contain P^+ and do not contain P^-. Here too we consider a decision version and a reporting version.

All known solutions for the Forbidden Pattern Document Retrieval problem with non trivial preprocessing use at least $\Omega(\sqrt{N})$ time per query [15] [19]. In a recent paper, Larsen, Munro, Nielsen, and Thankachan [27] show lower bounds for the Forbidden Pattern Document Retrieval problem conditioned on the hardness of boolean matrix multiplication. We provide some additional evidence of hardness conditioned on the 3SUM conjecture.

Theorem F.3. *Assume the 3SUM conjecture. For any $0 < \gamma < 1$, any data structure for the Forbidden Pattern Document Retrieval algorithm for a collection of documents X where $N = \sum_{D \in X} |D|$, with expected preprocessing time t_p and amortized expected query time t_q has*

$$t_p + N^{\frac{1+\gamma}{3-2\gamma}} t_q = \Omega(N^{\frac{2}{3-2\gamma} - o(1)}).$$

Proof. Similar to the proof of Theorem F.1 we set $\Sigma = A \cup B$. However this time for each c we create a document that contains all of the characters corresponding to sets from A that contain c and sets from B that do not contain c.

Using Theorem 1.4, we have $N = \Theta(n^{3-2\gamma})$, and the number of queries to answer is $\Theta(n^{1+\gamma}) = \Theta(n^{1+\gamma}) = \Theta(N^{\frac{1+\gamma}{3-2\gamma}})$. Thus we obtain the following lower bound tradeoff:

$$t_p + N^{\frac{1+\gamma}{3-2\gamma}} t_q = \Omega(n^{2-o(1)}) = \Omega(N^{\frac{2}{3-2\gamma} - o(1)}).$$

□

Notice that if we only allow linear preprocessing time then by making γ arbitrarily small we obtain a query time lower bound of $\Omega(N^{\frac{1}{3}-o(1)})$.

Theorem F.4. *Assume the 3SUM conjecture. For any $0 \leq \gamma < 1$, $\delta > 0$, any data structure for the reporting version of the Forbidden Pattern Document Retrieval algorithm for a collection of*

documents X where $N = \sum_{D \in X} |D|$, with expected preprocessing time t_p, the amortized expected query time t_q, and amortized expected time to report each document in the output t_r has

$$t_p + N^{\frac{1+\gamma}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}} t_q(N) + N^{\frac{2-\delta}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}} t_r(N) = \Omega(N^{\frac{2}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}-o(1)}).$$

Notice that allowing only linear preprocessing time and a constant reporting time, then by making γ and δ arbitrarily small we obtain a query time lower bound of $\Omega(N^{\frac{2}{3}-o(1)})$. Notice that in this case the expected output size per query is $O(N^{\frac{2}{3}-\Omega(1)})$.

Proof. Our proof is similar to the proof of Theorem F.3, only this time we use Theorem 1.5.. So we have $N = \Theta(n^{1+\delta-\gamma}\sqrt{n^{1+\delta-\gamma}}) = \Theta(n^{\frac{3}{2}(1+\delta-\frac{\gamma}{3})})$, the number of queries is $\Theta(n^{1+\gamma}) = \Theta(N^{\frac{1+\gamma}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}})$, and the total size of the output is $\Theta(n^{2-\delta}) = \Theta(N^{\frac{2-\delta}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}})$. Thus, we obtain the following lower bound tradeoff:

$$t_p + N^{\frac{1+\gamma}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}} t_q(N) + N^{\frac{2-\delta}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}} t_r(N) = \Omega(n^{2-o(1)}) = \Omega(N^{\frac{2}{\frac{3}{2}(1+\delta-\frac{\gamma}{3})}-o(1)}).$$

□

G A Full Version of Section 4

Fully Dynamic Maximum Cardinality Matching We consider the task of maintaining the cardinality of the maximum matching in a fully dynamic graph $G(V, E)$ where edges may be added or removed from G. To simplify the exposition, we assume that a query is performed after each update.

Consider the following instance of the fully dynamic MCM problem which is created from an instance of the offline SetDisjointness problem. For each $c \in C$ we create two vertices c_A and c_B with an edge between them. We say that c_A and c_B are copies of c. For each $a \in A$ we create two vertices a' and a'' with an edge between them, and for each $c \in a$ there is an edge between a' and c_A. We say that a' and a'' are copies of a. For each $b \in B$ we create two vertices b' and b'' with an edge between them, and for each $c \in b$ there is an edge between b' and c_B. We say that b' and b'' are copies of b.

The initialization of this graph is implemented by inserting all of the edges one at a time using the fully dynamic MCM algorithm. This initial graph has $\Theta(n + n^{2-2\gamma})$ vertices and $\Theta(n^{2-\gamma})$ edges. We also add 2 additional vertices, x and y, that are used during the implementation of the set intersections. Before we continue in the description of the reduction, it is important to notice that our initial graph (without the extra 2 vertices) has a unique perfect matching where we match each pair of vertices that are copies of the same entity. This perfect matching is of course also a MCM.

As mentioned, we make use of 2 dummy vertices x, y. To implement a SetDisjointness query we first add an edge from x to a'' and an edge from y to b''. See Figure 3(A) (in the Appendix). There are two cases to consider. In the first case, there exists an element $c \in a \cap b$. In this case before the addition of the new edges our graph has a path from a'' to b'' which is $(a'', a', c_A, c_B, b', b'')$. This path has 5 edges with 3 of them being in the perfect matching that we assume exists by induction. Adding the edges (x, a'') and (y, b'') creates an augmenting path, and so the MCM has increased due to the insertion of the two edges. On the other hand, if the SetDisjointness is empty then there

is no augmenting path in the graph. This is because paths from x to y must have two consecutive edges that are unmatched.

Thus, the MCM increases due to the insertion of the two edges if and only if the two sets intersect. In order to facilitate additional SetDisjointness queries we then delete the two edges from the graph, thereby reverting back to the original graph with the original perfect matching and the original size of the MCM.

The total number of insertions and deletions performed is $n^{1+\gamma} = n^{1+\gamma}$. If $t_u(\hat{n}, m)$ is the amortized expected update time (either insertion or deletion), then by the 3SUM conjecture and Theorem 1.4 we have that $t_u(n+n^{2-2\gamma}, n^{2-\gamma}) \cdot (n^{1+\gamma}+n^{2-\gamma}) = \Omega(n^{2-o(1)})$. Given that Theorem 1.4 holds for any constant $0 < \gamma < 1$ the highest lower bound is obtained by setting $\gamma = \frac{1}{2}$ and so $t_u(\hat{n}, m) = \Omega(m^{1/3-o(1)})$ and $t_u(\hat{n}, m) = \Omega(\hat{n}^{1/2-o(1)})$.

G.1 Incremental version

We now consider the task of maintaining the MCM of an incremental graph $G(V, E)$ where edges and vertices may be added to G. Again, to simplify the exposition, we assume that a query is performed after each update. The difficulty compared to the fully dynamic case is that we can no longer remove edges from the graph in order to facilitate additional SetDisjointness queries. We discuss two techniques for overcoming this difficulty, and then combine the two techniques to obtain our strongest results.

Rollback. The first technique we discuss is the *rollback* technique which takes advantage of the fact that in the fully dynamic reduction we only delete edges right after they were inserted. So an edge deletion can be simulated by tracking all of the changes made due to the previous insertion, and then rolling back the data structures used to their previous state prior to the insertion of the edge. The time of the deletion process is the same as the time of the insertion process. We do however need to use extra space for tracking the changes made during an insertion, but the amount of space is sub-linear as we are interested in times that are sub-linear per update.

The downside of this approach is that the lower bound now only holds in the worst-case, and not for an amortized analysis. We conclude that the rollback technique immediately implies a $t_u(\hat{n}, m) = \Omega(n^{1/2-o(1)})$ worst-case expected time lower bound from the reduction of the fully dynamic case.

Creating Perfect Matchings. The second technique we consider comes from the following observation. Consider a SetDisjointness query that we compute using the framework of the fully dynamic reduction, without the deletion of the edge. The issue we run into if we just ignore deletions (and do not execute them) we may run into false positives. This happens because an augmenting path may be created by two edge insertions where each insertion is due to a different SetDisjointness query.

To overcome this, we guarantee that after each SetDisjointness query is performed the resulting graph has a perfect matching. This will imply that augmenting paths during a SetDisjointness query can only be created due to the two edges inserted for that query. The perfect matching is created by adding another two edges per SetDisjointness query for a total of 4 edges, and creating four separate dummy vertices $x_{a,b}, x'_{a,b}, y_{b,a}$, and $y'_{b,a}$ for each SetDisjointness query on sets $a \in A$ and $b \in B$. For the SetDisjointness query we add edges $(x_{a,b}, a'')$ and $(y_{b,a}, b'')$ to the graph and just like in the fully dynamic case we know that the MCM increases if and only if the sets a and b intersect. See Figure 3(B). After we decide if the sets intersect or not, we add edges $(x_{a,b}, x'_{a,b})$ and

$(y_{b,a}, y'_{b,a})$. These two additional edges guarantee that the resulting graph has a perfect matching. This perfect matching is comprised of the perfect matching of the graph prior to the insertion of the 4 edges together with edges $(x_{a,b}, x'_{a,b})$ and $(y_{b,a}, y'_{b,a})$. The MCM has also increased by 2 after the insertion of the 4 edges, regardless of whether the sets intersect or not.

The downside of this method is that the number of vertices increases as we perform more and more SetDisjointness queries. Furthermore, being that this is an amortized lower bound, it is difficult to assign part of the cost of the entire sequence to the set up of the graph and the rest of the cost to the SetDisjointness queries. Since we perform $\Theta(n^{1+\gamma})$ SetDisjointness queries, the resulting graph has $N = \Theta(n^{1+\gamma})$ vertices and $M = \Theta(n^{1+\gamma} + n^{2-\gamma})$ edges. The highest lower bound is obtained by setting $\gamma = 1/2$. So, we have that the amortized expected time per insertion is $t_u(\hat{n}, m) = \Omega(n^{1/2-o(1)}) = \Omega(N^{1/3-o(1)})$. Thus the exponent in the dependency of the lower-bound on the number of vertices in the graph goes down from 1/2 to 1/3.

Combining the Two. To obtain our higher amortized lower bound we combine the two approaches of the rollbacks and creating perfect matchings. In particular, we will always begin by adding four edges as before, thereby creating a perfect matching. If the time of the insertion of the four edges is low (to be defined soon) then we will rollback the insertion. Otherwise, we leave the four edges and continue to the next SetDisjointness query. Intuitively, our goal with this combined method is to guarantee that the graph does not grow by too much, while maintaining the amortized cost (to obtain a higher lower-bound).

Before delving into the detailed proof of the lower bound, recall that when we are interested in the amortized cost of an operation, we are actually interested in the cost of a sequence of operations. The so called amortized cost of each operation σ can be viewed as a function $f(\sigma)$ that assigns each operation a value (which may depend on various parameters), so that the time of a sequence $S = (\sigma_1, \cdots \sigma_k)$ of k operations is always at most $\sum_{i=1}^{k} f(\sigma_i)$. The lower bound that we prove is that if n_i is the number of vertices in G during the i^{th} operation, then $\sum_{i=1}^{k} f(\sigma_i) \geq \Omega(\sum_{i=1}^{k} n_i^{\sqrt{2}-1}) \geq \Omega(\sum_{i=1}^{k} n_i^{0.414})$. In other words, there is no algorithm for incremental MCM in which the amortized cost of each insertion is $O(\hat{n}^{0.414-\Omega(1)})$ where $\hat{n}$ is the size of the graph during the insertion.

Now to the lower bound. Assume for contradiction that the amortized cost of each insertion is $\hat{n}^{\alpha}$ (ignoring constant coefficients) for some constant $\alpha > 0$, where $\hat{n}$ is the size of the graph during that insertion. Recall that we want to prove a lower bound on α. Our threshold for deciding when to rollback edge insertions is $4\hat{n}^{\alpha}$. If the time of an insertion is less than $4\hat{n}^{\alpha}$ then we rollback; otherwise we do not. For the time, notice that SetDisjointness queries for which we perform a rollback will end up costing $O(\hat{n}^{\alpha}) \leq O(N^{\alpha})$ time each. The cost of the edge insertions due to SetDisjointness queries for which we do not perform rollbacks, together with the edge insertions performed to setup the graph each cost an amortized $O(\hat{n}^{\alpha})$.

The next step will be to express N as a function of n, by taking advantage of the fact that after the setup of the graph, every four edge insertions implementing a SetDisjointness query that remain in the graph must have time that is expensive in a worst-case sense. Notice that after the graph setup there is $O(n^{2-\gamma}n^{\alpha})$ credit for performing expensive insertions later, as the graph has $m = \Theta(n^{2-\gamma})$ edges at the start. Each expensive SetDisjointness query uses up at least $\Omega(\hat{n}^{\alpha})$ of that credit, and so the total credit used during all of the expensive insertions is at least $\Omega(\sum_{i=i}^{N}(n+i)^{\alpha}) = \Omega(N^{1+\alpha})$. Since we can never be in credit debt, we have that $n^{2-\gamma}n^{\alpha} \geq \Omega(N^{1+\alpha})$ and so $N \leq O(n^{\frac{2-\gamma+\alpha}{1+\alpha}})$.

The number of cheaper insertions that we rolled back is $O(n^{1+\gamma}) = O(n^{1+\gamma})$. Each one of these costs at most N^{α}. So the total time of the entire sequence of operations which solves 3SUM

is $O(n^{1+\gamma}N^{\alpha} + N^{1+\alpha}) \leq O(n^{1+\gamma+\frac{(2-\gamma+\alpha)\alpha}{1+\alpha}} + n^{2-\gamma+\alpha})$. Given the 3SUM conjecture this runtime cannot be $O(n^{2-\Omega(1)})$ and so we must have that $2 \leq \max\{1+\gamma+\frac{(2-\gamma+\alpha)\alpha}{1+\alpha}, 2-\gamma+\alpha\}$. If we set $\gamma = \frac{1}{2+\alpha}$ the two terms are equal, and then $2 \leq 2-\gamma+\alpha$ implying that $\alpha \geq \gamma = \frac{1}{2+\alpha}$. Thus, α must be at least $\sqrt{2}-1 > 0.414$.