Matching Triangles and Basing Hardness on an Extremely Popular Conjecture*

Amir Abboud[1], Virginia Vassilevska Williams[1], and Huacheng Yu[1]

[1]Computer Science Department, Stanford University

Abstract

Due to the lack of unconditional polynomial lower bounds, it is now in fashion to prove conditional lower bounds in order to advance our understanding of the class P. The vast majority of these lower bounds are based on one of three famous hypotheses: the 3-SUM conjecture, the APSP conjecture, and the Strong Exponential Time Hypothesis. Only circumstantial evidence is known in support of these hypotheses, and no formal relationship between them is known. In hopes of obtaining "less conditional" and therefore more reliable lower bounds, we consider the conjecture that *at least one* of the above three hypotheses is true. We design novel reductions from 3-SUM, APSP, and CNF-SAT, and derive interesting consequences of this very plausible conjecture, including:

- Tight $n^{3-o(1)}$ lower bounds for purely-combinatorial problems about the triangles in unweighted graphs.
- New $n^{1-o(1)}$ lower bounds for the amortized update and query times of dynamic algorithms for single-source reachability, strongly connected components, and Max-Flow.
- New $n^{1.5-o(1)}$ lower bound for computing a set of n st-maximum-flow values in a directed graph with n nodes and $\tilde{O}(n)$ edges.
- There is a hierarchy of natural graph problems on n nodes with complexity n^c for $c \in (2,3)$.

Only slightly non-trivial consequences of this conjecture were known prior to our work. Along the way we also obtain new conditional lower bounds for the Single-Source-Max-Flow problem.

*A preliminary version of this paper has appeared in STOC 2015.

1 Introduction

A central goal in theoretical computer science is to understand the exact complexity of natural computational problems. For many such problems, $O(n^c)$ time algorithms are known, for some constant $c > 1$, and a proof that $O(n^{c-\varepsilon})$ algorithms, for some $\varepsilon > 0$, do not exist is highly desirable. Unfortunately, obtaining such "truly super-linear" unconditional lower bounds for problems in P seems far beyond the current state of the art in complexity. The urgency of such lower bounds, due to both intellectual curiosity and practical relevance, has led researchers to settle for conditional lower bounds. A reductions-based approach, which can be viewed as a refinement of NP-hardness, has been gaining popularity, with many recent results providing satisfactory answers to our urgent needs for lower bounds. In this approach, one assumes that a certain famous problem with an $O(n^a)$ time upper bound that has resisted improvements for many years, requires $n^{a-o(1)}$ time, and derives $n^{b-o(1)}$ lower bounds for other problems.

Most known conditional lower bounds for the exact polynomial time complexity of problems are based on one of the following three popular conjectures, regarding fundamental problems in computational geometry, graph algorithms, and satisfiability. See Appendix A for background on these conjectures and a brief survey of the known conditional lower bounds.

The 3-SUM Conjecture: *There is no algorithm that can check whether a list of n numbers contains three that sum to zero (the 3-SUM problem) in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$.*

The APSP Conjecture: *There is no algorithm that can compute all pairs shortest paths (APSP) on n node graphs with edge weights in $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$.*

Strong Exponential Time Hypothesis (SETH): *For every $\delta > 0$ there is an integer $k \geq 3$ such that k-SAT on n variables and $O(n)$ clauses cannot be solved in $2^{(1-\delta)n}\text{poly}(n)$ time.*

No formal relationship is known between these conjectures, and as far as we know, any subset of them can be true. A loose relationship between SETH and 3-SUM was shown by Pătraşcu and Williams [59]: if k-SUM can be solved in $n^{o(k)}$ time, then the weaker conjecture ETH[1] is false, and since SETH implies ETH, SETH must be false as well. We remark that the conjectured lower bounds are assumed to hold even against randomized algorithms.

Lower bounds based on a weaker conjecture. In this work we search for the weakest hypothesis that is still useful for proving interesting lower bounds for natural polynomial time problems. An obvious candidate is the assumption that at least *one* of the popular conjectures is true.

Conjecture 1. *At least one of the 3-SUM conjecture, the APSP conjecture, or SETH, is true.*

Conjecture 1 seems much more believable than any of the above three conjectures, since to refute it, it must be the case that all three of the computational geometry, graph algorithms, and exact algorithms communities have missed breakthrough algorithms for their core problems. Given the great popularity of each of these conjectures individually, Conjecture 1 is *extremely popular*.

Conjecture 1 is especially useful when trying to prove limitations to powerful new algorithmic tools. The recent groundbreaking tools of Laplacian system solvers [67] and interior-point methods [46, 55] are a great example. These new tools have allowed for celebrated algorithmic improvements over longstanding upper bounds for different versions of Max-Flow, including recent best-paper award winners [54, 66, 51, 47, 23]. With such a powerful tool at hand, one might consider a lower bound based on the hardness of a single problem, e.g. APSP, as a challenge to refute the APSP conjecture with the tool, rather than an impossibility result. However, lower bounds based on Conjecture 1 can be more safely regarded as impossibility results - in fact, such lower bounds are at least as believable as any other known conditional lower bounds for a problem in P.

[1]ETH stipulates that there is some $\varepsilon > 0$ such that 3SAT is not in $2^{\varepsilon n}\text{poly } n$ time.

Previous results. Besides the large number of lower bounds that are based on a single conjecture, there are few examples of lower bounds that are based on two of the conjectures. The works of Pătraşcu [57], and Vassilevska and Williams [73] prove that if a triangle of total weight 0 in an edge-weighted graph on n nodes can be found in $O(n^{3-\varepsilon})$ time, the both the 3-SUM and APSP conjectures would be refuted. More recently, Abboud, Vassilevska Williams, and Weimann [6] show that an $O(n^{2-\varepsilon})$ algorithm for the Local Alignment problem from Bioinformatics refutes both the 3-SUM conjecture and SETH. No non-trivial lower bounds were known under Conjecture 1.

1.1 Our results

Using a large collection of new reductions and algorithms we obtain interesting consequences of Conjecture 1.

Intermediate problems. Our main contribution is in the identification of two innocent-looking graph problems, which we call Triangle-Collection and Δ-Matching-Triangles, that allow for *tightly efficient* reductions from each of our hard problems. Let $\Delta \geq 1$ be an integer.

Definition 1.1 (Triangle-Collection). *Given a node-colored graph G, is it true that for all triples of distinct colors a, b, c there is a triangle (x, y, z) in G in which x has color a, y has color b, and z has color c?*
(Does the set of all triangles in the graph "collect" all triples of colors?)

Definition 1.2 (Δ-Matching-Triangles). *Given a node-colored graph G, is there a triple of distinct colors a, b, c such that there are at least Δ triangles (x, y, z) in G in which x has color a, y has color b, and z has color c?*
(Are there Δ triangles with "matching" colors?)

An equivalent way to define these problems is: given a k-partite graph G on n nodes, Triangle-Collection asks whether every triple of partitions has a triangle among them, and Δ-Matching-Triangles asks whether there is a triple of partitions with at least Δ triangles among them.

Note that an $O(n^3)$ algorithm for each of these problems is trivial and the output is a single bit, yet the following theorem shows that if an $O(n^{3-\varepsilon})$ algorithm existed for some $\varepsilon > 0$, we would have groundbreaking algorithms for 3-SUM, APSP, *and* CNF-SAT! It is quite surprising that these simple problems are hiding such "hardness" to be a bottleneck for these three famous problems (and many others by the known reductions).

Theorem 1.1. *Conjecture 1 implies that Triangle-Collection and Δ-Matching-Triangles, with $\omega(1) < \Delta < n^{o(1)}$, on graphs with n nodes cannot be solved in $O(n^{3-\varepsilon})$ time, for any $\varepsilon > 0$.*

Observe that the Δ-Matching-Triangles problem with $\Delta = 1$ simply asks if there is a triangle in the graph and can therefore be solved in $O(n^{\omega})$ time, where $\omega < 2.373$ is the matrix multiplication exponent [74, 34]. However, when Δ increases to $\omega(1)$ it must require $n^{3-o(1)}$ time under Conjecture 1. It is natural to wonder what is the complexity of the problem when $\Delta > 1$ is a constant. Studying this question, we have discovered a surprising hierarchy of n-node graph problems with increasing complexities, starting at $n^{\omega \pm o(1)}$ and approaching $n^{3 \pm o(1)}$. These results are presented at the end of this section.

Besides allowing us to give a tight lower bound for a natural combinatorial problem based on the extremely weak Conjecture 1, the Triangle-Collection problem serves as a good intermediate problem for obtaining other good from Conjecture 1. The simplicity and purely-combinatorial nature of the problem allow for simple reductions to other (more well-studied) problems, while the tightness of the lower bound means that no efficiency is lost by reducing from it.

New lower bounds for dynamic problems. We consider variants of classic dynamic problems such as Single-Source-Reachability(#SSR), Strongly Connected Components(#SCC), Subgraph Connectivity(#SS-Sub-Conn), and Max-Flow. For example, #SSR asks to maintain the number of nodes reachable from a single source in a dynamic directed graph. See Appendix A for background on dynamic algorithms and the many known algorithms for these problems.

After a sequence of reductions from 3-SUM by Pătraşcu [57] that was later optimized by Abboud and Vassilevska [5] and by Kopelowitz, Pettie and Porat [48], we can conclude that the above problems require $n^{\frac{2}{3}-o(1)}$ amortized update if the 3-SUM conjecture holds. This lower bound does not match the known upper bounds, and in fact, Abboud and Vassilevska [5] show that there is a higher $n^{1-o(1)}$ lower bound under SETH. However, obtaining a higher lower bound from the 3-SUM conjecture using Pătraşcu's approach seems impossible, due to certain inefficiencies in the reduction, and obtaining a higher lower bound from 3-SUM has remained an open question. No lower bound for these problems was known under the APSP conjecture.

We give simple reductions from Triangle-Collection to these classic dynamic problems to obtain linear $n^{1-o(1)}$ lower bounds on the amortized update times, under our very weak Conjecture 1. The tightness of our reduction from 3-SUM to the purely-combinatorial Triangle-Collection problem allows us to overcome the $n^{\frac{2}{3}-o(1)}$ barrier for lower bounds under the 3-SUM conjecture.

Theorem 1.2. *Conjecture 1 implies that any dynamic algorithm for #SSR, #SCC, #SS-Sub-Conn, and Max-Flow requires either amortized $n^{1-o(1)}$ update or query times, or $n^{3-o(1)}$ preprocessing time.*

Our lower bound for dynamic Max-Flow hints on a barrier for efficient Max-Flow algorithms: changing one edge of the input, corresponding to one constraint in the linear program, will make the algorithm spend linear time to recompute the optimal solution, in an amortized sense. Next, we look for barriers for Max-Flow computations in static graphs.

New lower bounds for variants of Max-Flow. Equipped with Conjecture 1 and its realization in the simple Triangle-Collection problem, we try to obtain reductions to Max-Flow, in hopes of proving under a weak assumption that certain tasks will not be solvable in near-linear time.

Breakthrough algorithms for s,t-Max-Flow were found in recent years using the powerful tools of Laplacian systems solvers, and interior-point methods [66, 47, 54, 51]. It also seems that these algorithms take near-linear time in practice, and the bottleneck for improving the upper bounds might be in the limitations of our current analysis. Thus, attempting to prove super-linear lower bounds for Max-Flow under a conjecture we believe to hold might be ill-advised. Instead, we consider two other versions of Max-Flow, in which we have multiple pairs of sources and sinks, and for which the potential of these new powerful tools is still unexplored.

Definition 1.3 (Single-Source-Max-Flow)**.** *Given a directed edge-capacitated graph G and source vertex $s \in V$, output, for every $t \in V$, the maximum flow that can be transferred in G from s to t.*

Definition 1.4 (ST-Max-Flow)**.** *Given a directed edge-capacitated graph G and two subsets of vertices $S, T \subseteq V(G)$, output, for every pair of nodes $s \in S, t \in T$, the maximum flow that can be transferred in G from s to t.*

Let $T(n,m)$ be the time complexity of Max-Flow. The current bound is $T(n,m) = \tilde{O}(m\sqrt{n})$ by Sidford and Lee [51]. Obviously, Single-Source-Max-Flow can be solved in $O(n \cdot T(n,m))$ time, and ST-Max-Flow can be solved in $O(|S||T|T(n,m))$ time. In the unit-capacity case, Cheung et al. [22] solve the all-pairs version, i.e. ST-Max-Flow with $S = T = V(G)$ in $O(m^{\omega})$ time, and Single-Source-Max-Flow in $O(n^{\omega-1}m)$ time. In general graphs, Hao and Orlin [36] show that the maximum flow between $O(n)$ st-pairs can be found in the time it takes for a single Max-Flow computation, however these pairs cannot be specified in advance. Łącki et al. [50] obtained a near-linear time algorithm for Single-Source-Max-Flow in planar digraphs. Note that in undirected graphs, all-pairs-max-flow can be read from the Gomory-Hu tree of the graph, which can be computed in $\tilde{O}(mn)$ time [37].

First, we devise a simple reduction from Triangle-Collection to ST-Max-Flow and prove that a near-linear-time algorithm for it would shatter our conjectures.

Theorem 1.3. *Conjecture 1 implies that ST-Max-Flow on a network with n nodes and $O(n)$ edges requires $n^{1.5-o(1)}$ time, even when $|S| = |T| = \sqrt{n}$.*

Although this lower bound does not match the currently known $n \cdot T(n)$ upper bounds, where $T(n) = \tilde{O}(n^{1.5})$ is the time it takes to solve Max-Flow on sparse graphs, it gives the first connection between a Max-Flow-like problem and our popular conjectures. Moreover, this result implies that under Conjecture 1, any $O(m^{1.5-\varepsilon})$ time algorithm for Max-Flow, cannot also output the maximum flow between n st-pairs of our choice.

This new connection to Max-Flow allows us to obtain perhaps more currently relevant conditional lower bounds for Single-Source-Max-Flow.

Theorem 1.4. *If for some $\varepsilon > 0$, Single-Source-Max-Flow on a graph with n nodes, $\tilde{O}(n)$ edges with capacities in $[n]$, can be solved in $O(n^{2-\varepsilon})$ time, then MAX-CNF-SAT on n variables and* poly (n) *clauses can be solved in $2^{(1-\delta)n}$*poly (n) *time, for some $\delta > 0$, and SETH is false.*

Note that the current best upper bound is $n \cdot T(m)$, and this lower bound would be tight if Max-Flow is in $T(m) = m^{1+o(1)}$ time. A first interesting consequence of this result is that, under SETH, unlike for shortest paths where the s, t version and the single-source versions have roughly the same complexity, the single-source version of Max-Flow (requires $n^{2-o(1)}$) is much harder than the s, t version (is in $O(n^{1.5})$ time), at least on sparse graphs. Another interesting consequence is that, under SETH, either Max-Flow requires $m^{1+\delta-o(1)}$ time, for some $\delta > 0$, or the following counter-intuitive thing is true: it is not possible to compute n single-source flows in a network faster than by calling an st-flow algorithm n times.[2] In the full version of the paper we give other reductions from Triangle-Detection and APSP to single-source Max-Flow and Min-Cost-Flow, obtaining other interesting consequences.

Towards a better understanding of P. The time hierarchy theorem promises the existence of problems with complexity $\Theta(n^c)$ for any constant $c > 1$, and is proven by constructing a diagonalizing Turing-Machine, but are there *natural* problems with complexity $n^{2.1}, n^{2.7}$, or $n^{2.9}$? And what would such problems look like? Obviously, an unconditional answer to this question will require concrete polynomial lower bounds, and we are satisfied with a conditional answer. For integers $k > 2$, we have a good sense of what an n^k hierarchy might look like: the k-Dominating-Set problem has complexity $n^{k \pm o(1)}$ under SETH [59, 29], and it is quite intuitive that the complexity of this problem increases from n^5 to n^6 as k increases from 5 to 6. But what about a hierarchy of problems with complexity $\Theta(n^c)$ for $c \in (2, 3)$? Even under one of the conjectures, it is not clear how to find such problems. [3]

It turns out that the Δ-Matching-Triangles problem, which asks if there is a triple of colors containing at least Δ triangles, allows us to find such hierarchy of problems, even under our very weak Conjecture 1! Recall that when $\Delta = 1$ there is an $O(n^\omega)$ upper bound, and when $\omega(1) < \Delta < n^{o(1)}$ we have a an $n^{3-o(1)}$ lower bound based on the very weak Conjecture 1. In Section 3, we obtain a *truly subcubic* algorithm for Δ-Matching-Triangles for any fixed integer $\Delta \geq 1$.

Theorem 1.5. *The Δ-Matching-Triangles problem on an n-node graph G can be solved in $\tilde{O}\left(n^{3-c_\Delta}\right)$ time for $c_\Delta = \frac{2(3-\omega)^2}{(5-\omega)\Delta+1-\omega} > 0$.*

Moreover, Theorem 1.1 also proves a *truly-super-quadratic* lower bound that approaches $n^{3-o(1)}$ for Δ-Matching-Triangles, for a large enough constant Δ, assuming Conjecture 1.

Corollary 1.1. *Conjecture 1 implies that for any $\delta < 1$, there is an integer $\Delta \geq 1$ such that Δ-Matching-Triangles requires $n^{2+\delta-o(1)}$ time.*

Combining the lower and the upper bounds, we conclude that there is some constant D such that for every integer $\Delta > D$, Δ-Matching-Triangles has time complexity $\tilde{\Theta}(n^{c_\Delta})$ for some $c_\Delta \in (2, 3)$. We also remark

[2]Łącki et al. [50] conjecture that computing all n^2 st-flows in a general graph can be done faster than by calling a Max-Flow algorithm n^2 time.

[3]An uninteresting way to find such problems is by padding the input to APSP for example, so that all nodes but the first $n^{c/3}$ are ignored, however we would not consider such a problem natural as it would not contribute to our understanding of what makes the computational complexity of a problem increase from $n^{2.7}$ to $n^{2.8}$.

that when Δ increases beyond $n^{o(1)}$, the complexity of Δ-Matching-Triangles decreases to truly subcubic yet again.

Finally, in order to obtain a better understanding of the complexity of Δ-Matching-Triangles for smaller constants Δ, like $\Delta = 3$, we consider the following conjecture.

Conjecture 2. *At least one of the* 3*-SUM conjecture or the APSP-conjecture holds.*

We are able to show a much better lower bound from Conjceture 2, which is $n^{3-9/(\Delta+3)-o(1)}$. For example, this bound is $n^{2.1-o(1)}$ when $\Delta = 7$, and is $n^{2.9-o(1)}$ when $\Delta = 87$. Examining the reductions, we notice that this lower bound applies for a restricted version of the problem which we call Δ-Matching-Triangles* (defined in Section 2), which turns out to have a matching upper bound, allowing us to prove the following hierarchy theorem.

Theorem 1.6. *Conjecture 2 implies that for any $\Delta > 6$, the complexity of Δ-Matching-Triangles* is exactly $n^{3-9/(\Delta+3)\pm o(1)}$.*

2 Reductions to Matching Triangles

Recall the definitions of Δ-Matching-Triangles and Triangle-Collection problem in the introduction. In this section, we are going to reduce the three hard problems to Δ-Matching-Triangles and Triangle-Collection.

First, we show 3-SUM and APSP-hardness using EW-Triangle as an intermediate problem, since it requires $n^{3-o(1)}$ time unless both conjectures are false.

Definition 2.1 (EW-Triangle)**.** *Given a graph $G = (V, E)$ with integer edge weights $w : E \to [-n^c, n^c]$, determine if there is a triangle (a, b, c) of total weight $w(x, y) + w(y, z) + w(x, z) = 0$.*

Our main ingredient in the reductions from EW-Triangle is a set of $n^{o(1)}$ mappings from integers in $[-n^c, n^c]$ to vectors in $[-p, p]^d$ where $(p/3)^d > n^c$ so that three numbers sum to 0, if and only if in at least one of the mappings, the three corresponding vectors will sum to a certain target vector $\mathbf{t}$. The basic idea is to group the bits of a number into blocks of size $\log p$ and guess all the carries. The following mapping was suggested by Abboud, Lewi, and Williams [3] as a step towards reducing k-SUM to k-Clique. Besides using this lemma, our reductions are very different from theirs.

Lemma 2.1 ([3])**.** *For any integers $n, c, d, p \geq 1$ such that $p \geq 3n^{c/d}$, there is a set of $s = 2^{O(d)}$ mappings $f_1, \dots, f_s : [-n^c, n^c] \to [-p/3, p/3]^d$ and s target vectors $\mathbf{t_1}, \dots, \mathbf{t_s} \in [-p, p]^d$ such that for any three numbers $x, y, z \in [-n^c, n^c]$: $x + y + z = 0$ if and only if for some $i \in [s]$, $f_i(x) + f_i(y) + f_i(z) = \mathbf{t_i}$.*

EW-Triangle to Matching-Triangles We are now ready to prove the new reduction from EW-Triangle to Δ-Matching-Triangles. This is perhaps the most novel reduction in this work. After reducing the edge-weights to vectors with small values in each coordinate, we remove the numbers completely and simulate them using pointers. The summation of numbers is simulated by a path that walks along these pointers. A path on three edges that starts and ends at the same node (a triangle) will correspond to a sum of three numbers being zero.

Lemma 2.2. *An instance of EW-Triangle on n nodes, m edges, and edge weights in $[-n^c, n^c]$ can be reduced to $s = 2^{O(\Delta)}$ instances of Δ-Matching-Triangles on $O(n \cdot n^{c/\Delta} \cdot \Delta)$ nodes and $O(mn^{c/\Delta}\Delta)$ edges in linear time.*

Proof. Given $G = (V, E), V = A \cup B \cup C, w : E \to [-n^c, n^c]$ as input to EW-Triangle, we construct an unweighted graph $G'_i = (V'_i, E'_i)$ on $O(n \cdot n^{c/\Delta} \cdot \Delta)$ nodes with node colors $\chi : V'_i \to [n]$ as follows.

First, apply Lemma 2.1 with $d = \Delta$ and $p = O(n^{c/\Delta})$ to construct $s = 2^{O(\Delta)}$ mappings from integers to vectors and apply them to each of the edge weights in G. For each $i \in [s]$, we use the mapping f_i and the target vector $\mathbf{t_i}$ to construct a graph G'_i with nodes $V'_i = A'_i \cup B'_i \cup C'_i$. For each node $a \in A$ we add d nodes $a_1, \dots, a_d$ to A'_i and set their color to a (we abuse notation and assume that each node in A is a number in $[n]$). The node a_i will help us simulate the addition in the i^{th} dimension of the vectors. For nodes $b \in B, c \in C$ we add $d \cdot 2p$ nodes $b_{j,x}$ and $c_{j,x}$ to B'_i and C'_i, where $j \in [d]$ and $x \in [-p,p]$. Intuitively, the index j corresponds to the dimension and the index x corresponds to the value in that dimensions. Let the color of every $b_{j,x}$ node be b and the color of every $c_{j,x}$ node be c (we abuse notation again and assume that every node in $B \cup C$ has a unique number in $[n+1, 3n]$). We now define the edges of G'_i.

- (A to B) For each edge (a,b) in G where $a \in A, b \in B$, we map the weight of the edge using f_i to get a vector $f_i(w(a,b)) \in [-p,p]^d$ and we add d edges to G'_i: for each dimension $j \in [d]$ we add an edge from a_j to $b_{j,x}$ where $x = f_i(w(a,b))[j]$ is the value in the j^{th} dimension of the vector corresponding to the weight.

- (B to C) For each edge (b,c) in G where $b \in B, c \in C$, we map the weight of the edge using f_i to get a vector $f_i(w(b,c)) \in [-p,p]^d$ and for each dimension $j \in [d]$ we add up to $2p$ edges to G'_i: for each value $x \in [-p,p]$ we let $y = x + f_i(w(b,c))[j]$ and if $y \in [-p,p]$ we add an edge from $b_{j,x}$ to $c_{j,y}$. That is, for each dimension $j \in [d]$ the edges we add simulate an increase of $f_i(w(b,c))[j]$ in the value at the j^{th} dimension.

- (C to A) Finally, for each edge (c,a) in G where $c \in C, a \in A$, we map the weight of the edge using f_i to get a vector $f_i(w(c,a)) \in [-p,p]^d$ and we add d edges to G'_i: for each dimension $j \in [d]$ we add an edge from $c_{j,x}$ to a_j where $x = t_i[j] - f_i(w(c,a))[j]$.

The number of nodes in G'_i is $n\Delta + n\Delta 2p + n\Delta 2p = O(n^{1+c/\Delta}\Delta)$, while the number of edges is $m\Delta + m\Delta n^{c/\Delta} = O(mn^{c/\Delta}\Delta)$. The number of colors is $|A| + |B| + |C| = 3n$. If one of the $s = 2^{O(\Delta)}$ instance of Δ-Matching-Triangles is a YES instance, we say that G contains a triangle of weight 0. The following claim shows the correctness of our reduction.

Claim 1. *There is a triangle $(a,b,c) \in A \times B \times C$ in G of weight 0 iff for some $i \in [s]$, there are at least Δ triangles in G'_i with colors a,b,c.*

Proof. For the first direction, assume $(a,b,c) \in A \times B \times C$ is a triangle in G and $w(a,b)+w(b,c)+w(c,a) = 0$. By Lemma 2.1, we know that for some $i \in [s]$, the vectors sum to $f_i(w(a,b)) + f_i(w(b,c)) + f_i(w(c,a)) = \mathbf{t_i}$. Therefore, for every $j \in [d]$ we have that $(a_j, b_{j,x}), (b_{j,x}, c_{j,y}), (c_{j,y}, a_j) \in E(G'_i)$ where $x = f_i(w(a,b))[j]$ and $y = x + f_i(w(b,c))[j] = f_i(w(a,b))[j] + f_i(w(b,c))[j]$, since under our assumption $y = t_i[j] - f_i(w(c,a))[j]$. By our assignment of colors to nodes, we get $\Delta(= d)$ triangles with colors a,b,c in G'_i.

For the second direction, assume that there are Δ triangles in G'_i for some $i \in [s]$ using the same triple of colors. First note that a triangle cannot use two colors from a single partition A, B, or C, since the nodes of each partition form an independent set. Therefore, the triple of colors must be of the form a,b,c for some colors corresponding to nodes $a \in A, b \in B, c \in C$. By construction of our graphs G'_i, we know exactly which are the Δ triangles: for each $j \in [d]$ the node a_j has only one neighbor in B with color b and one neighbor in C with color c and therefore we can have at most one triangle, which is $(a_j, b_{j,x}, c_{j,y})$ in G'_i. For each $j \in [d]$, this triangle exists iff $f_i(w(a,b))[j] + f_i(w(b,c))[j] = t_i[j] - f_i(w(c,a))[j]$. Thus, there are Δ triangles iff $f_i(w(a,b))[j] + f_i(w(b,c))[j] + f_i(w(c,a))[j] = t_i[j]$ for every $j \in [d]$. By Lemma 2.1, this occurs only if $w(a,b) + w(b,c) + w(c,a) = 0$. □

□

Corollary 2.1. *If there is an algorithm which solves Δ-Matching-Triangles on n-node graph in $O(n^{3-\epsilon})$ time for any constant $\epsilon > 0$, any $\omega(1) < \Delta(n) < o(\log n)$, we can solve EW-Triangle in $O(n^{3-\epsilon+o(1)})$ time.*

EW-Triangle to Triangle-Collection Note that our lower bounds for Δ-Matching-Triangles increase as Δ grows from 1 to $O(\log n / \log\log n)$. In fact, the $\Delta = 1$ can be solved in truly subcubic $O(n^\omega)$ time. Interestingly, we are able to show the highest $n^{3-o(1)}$ lower bounds for the "$\Delta = 0$ case" as well, by reductions from APSP, 3-SUM, and CNF-SAT.

We first define a "restricted" version of Triangle-Collection, called Triangle-Collection*. Then we reduce EW-Triangle problem to the restricted version. Finally, we will reduce Triangle-Collection* to Triangle-Collection.

Definition 2.2 (Triangle-Collection*). *Given an undirected tripartite node colored graph G with partitions A, B, C of the following form:*

- *A contains $n\Delta$ nodes denoted a_j where $a \in [n]$ and $j \in [\Delta]$ so that a_j is colored with color a.*
- *B and C contain $n\Delta p$ nodes each, denoted $b_{j,x}$ and $c_{j,x}$ where $b, c \in [n], j \in [\Delta]$, and $x \in [p]$ so that $b_{j,x}$ ($c_{j,x}$) is colored b (c).*
- *For each node a_j in A and colors b, c, there is exactly one edge of the form $\{a_j, b_{j,x}\}$ and exactly one edge of the form $\{a_j, c_{j,y}\}$, for some $x, y \in [p]$.*
- *A node $b_{j,x}$ in B can only be connected to nodes of the form $c_{j,y}$ in C (no edges across different j's).*

Is it true that for all triples of distinct colors a, b, c there is a triangles (x, y, z) in G in which x has color a, y has color b, and z has color c?

Lemma 2.3. *An instance of EW-Triangle on n nodes and edge weights in $[-n^c, n^c]$ can be reduced to $s = 2^{O(\Delta)}$ instances of Triangle-Collection* on $O(n \cdot n^{c/\Delta} \cdot \Delta)$ nodes and $O(n^{2+2c/\Delta}\Delta)$ edges in linear time.*

Proof. The reduction is similar to the one in the proof of Lemma 2.2, flipping one side of the edges to make the *absence* of a $(a_j, b_{j,\star}, c_{j,\star})$ triangle correspond to the sum being zero on the j^{th} dimension.

Take the unweighted graphs $G_i' = (V_i', E_i')$ in the proof of Lemma 2.2, where $V_i' = A_i' \cup B_i' \cup C_i'$. We flip all the edges between B_i' and C_i', and get a collection of $2^{O(\Delta)}$ new graphs G_i''. According to the proof of Claim 1, there is a zero-weight triangle (a, b, c) in G if and only if there are no triangle of colors (a, b, c) in G_i'' for some i. It is not hard to verify that all G_i''s have the format of Triangle-Collection* input graphs. □

Lemma 2.4. *An instance of Triangle-Collection* on n-node graphs can be reduced to an instance of Triangle-Collection on $O(n)$ nodes.*

Setting $\Delta = 2^{\Theta(\sqrt{\log n})}$ gives us the following corollary.

Corollary 2.2. *If there is an algorithm solving Triangle-Collection on n-node graph in $O(n^{3-\epsilon})$ time for any constant $\epsilon > 0$, we can solve EW-Triangle in $O(n^{3-\epsilon+o(1)})$ time.*

Proof of Lemma 2.4. The only problem we need to worry about is that there is no restriction on the triple of colors in the general Triangle-Collection problem. That is, we do not want to find a triangle-free triple of colors such that they are not from sets A, B, C respectively. To solve this issue, given an input graph G for Triangle-Collection*, for every pair of nodes $a, a' \in A$ we add edges $(a_B, a_B'), (a_C, a_C')$ to G. Similarly, for nodes $b, b' \in B$ and $c, c' \in C$, we add edges $(b_A, b_A'), (b_C, b_C'), (c_A, c_A'), (c_B, c_B')$ to G. Then, for every pair $a \in A, b \in B$ we add an edge (a_B, b_A), for every pair $a \in A, c \in C$ we add an edge (a_C, c_A), and for every pair $b \in B, c \in C$ we add an edge (b_C, c_B) to G. Finally, we get a graph G'. It makes every triple of colors that is not in $A \times B \times C$ contain a triangle, but does not add new triangles for triples in $A \times B \times C$.

Claim 2. *If a triple of colors (x, y, z) is not in $A \times B \times C$, then that triple contains a triangle in G'.*

Proof. If all three colors come from the same partition, e.g. $x, y, z \in A$ (or B or C) then the nodes x_B, y_B, z_B form a triangle, by construction. If, however, two nodes come from the same partition, e.g. $x, y \in A$ but $z \in B$ (the other cases are similar), then the nodes x_B, y_B, z_A form a triangle in every G_i', by construction. □

The new nodes in G' add triangles to every "invalid" triple of colors, while for every "valid" triple, the existence of triangles does not change. This proves the correctness of our reduction. □

A hierarchy with exact bounds We note that by standard random hashing of the edge-weights by working modulo a random prime, EW-Triangle with $c = 3$ (weights in $[-n^c, n^c]$) is as hard as the more general case. Thus, Conjecture 2 implies an $n^{3-o(1)}$ lower bound for EW-Triangle even with $c = 3$, and therefore together with Lemma 2.2 it implies an $n^{3-9/(\Delta+3)-o(1)}$ lower bound for Δ-Matching-Triangles. Observe that the proof of Lemma 2.2 proves hardness even for input graphs of a restricted form, thus obtaining the same lower bound even for the following problem:

Definition 2.3 (Δ-Matching-Triangles*)**.** *Given an undirected tripartite node colored graph G on $N = O(n\Delta p)$ nodes, with partitions A, B, C of the following form, where $p = O(n^{3/\Delta})$:*

- *A contains $n\Delta$ nodes denoted a_j where $a \in [n]$ and $j \in [\Delta]$ so that a_j is colored with color a.*
- *B and C contain $n\Delta p$ nodes each, denoted $b_{j,x}$ and $c_{j,x}$ where $b, c \in [n], j \in [d]$, and $x \in [p]$ so that $b_{j,x}$ ($c_{j,x}$) is colored b (c).*
- *For each node a_j in A and colors b, c, there is exactly one edge of the form $\{a_j, b_{j,x}\}$ and exactly one edge of the form $\{a_j, c_{j,y}\}$, for some $x, y \in [p]$.*
- *A node $b_{j,x}$ in B can only be connected to nodes of the form $c_{j,y}$ in C (no edges across different j's).*

Is there a triple of distinct colors a, b, c such that there are at least Δ triangles (x, y, z) in G in which x has color a, y has color b, and z has color c?

Reminder of Theorem 1.6 *Conjecture 2 implies that for any $\Delta > 6$, the complexity of Δ-Matching-Triangles* is exactly $n^{3-9/(\Delta+3)\pm o(1)}$.*

In such restricted instances, one can check in $O(\Delta)$ time whether there are Δ triangles with a given triple of colors, and therefore the problem can be solved in $O((\#\text{colors})^3 + (\#\text{nodes})^2)$ which is $O(N^{3-9/(\Delta+3)})$ when $\Delta > 6$ is constant, matching our lower bound, and proving Theorem 1.6.

SETH to Matching-Triangles Next, we prove a new SETH lower bound. Our reduction uses the same split-and-list technique that is used in all of the SETH-based lower bounds, yet unlike most previous reductions, ours splits the variables into three sets, not two. Moreover, since our reduction incurs an overhead of 2^M where M is proportional to the number of clauses, we introduce new tricks to reduce the dependence on the number of clauses.

Lemma 2.5. *If Δ-Matching-Triangles on N-node graphs can be solved in $O(N^{c_\Delta})$ time, then CNF-SAT on n variables and m clauses can be solved in $O\left(\left(\Delta 2^{n/3+m/3\Delta}\right)^{c_\Delta}\right)$ time.*

Proof. Given a CNF formula F on n variables and m clauses as input of CNF-SAT. First we split the variables into three sets U_1, U_2, U_3 of size $n/3$ each and enumerate over all the $N = 2^{n/3}$ partial assignments to each set. Also we arbitrarily divide m clauses into 3Δ groups $C_1, \ldots, C_{3\Delta}$, each of which contains $m/3\Delta$ clauses.

Then we construct a graph G on $O(N\Delta 2^{m/3\Delta})$ nodes $V_1 \cup V_2 \cup V_3$, containing $O(\Delta 2^{m/3\Delta})$ nodes for each partial assignment. Let α_i be a partial assignment to variables in U_i. For each group C_{3k+i}, partial assignment α_i and bit string $s_i \in \{0,1\}^{m/3\Delta}$, we build a vertex $v_{\alpha_i,k,s_i} \in V_i$. The bit string s_i will correspond to some subset of clauses of group C_{3k+i}. Then for every partial assignment, we assign a different color. Thus we have $3N$ colors in total. Finally, we need to describe the edges in G. We add an edge between $v_{\alpha_i,k,s_i} \in V_i$ and $v_{\alpha_{i+1},k,s_{i+1}} \in V_{i+1}$[4], if α_{i+1} satisfies exactly the subset s_i of group C_{3k+i} and α_i, α_{i+1} together with the subset s_{i+1} satisfy all clauses in C_{3k+i+1} (C_{3k+1} if $i = 3$). Basically, s_i corresponds to the subset which α_{i+1} satisfies. When considering a pair of partial assignments, together with the information carried about the third part, we can decide whether we have satisfied enough clauses in one group.

We claim that for any triple of partial assignments $\alpha_1, \alpha_2, \alpha_3$ and $k \in [\Delta]$, there is a triangle among vertices $v_{\alpha_1,k,*}, v_{\alpha_2,k,*}, v_{\alpha_3,k,*}$ if and only if they satisfy all clauses in $C_{3k+1}, C_{3k+2}, C_{3k+3}$. If there is a

[4]For simplicity of notations, let $\alpha_4 = \alpha_1, s_4 = s_1, V_4 = V_1$

triangle $(v_{\alpha_1,k,s_1}, v_{\alpha_2,k,s_2}, v_{\alpha_3,k,s_3})$, there are edges between any two of them. This means α_2 satisfies exactly the subset s_1 of C_{3k+1}, and α_1, α_3 together with s_1 satisfy all clauses of C_{3k+1}. Therefore, they satisfy clauses in all these three groups due to symmetry. On the other hand, if they satisfy clauses in the three groups, let s_i be the subset of clauses α_{i+1} satisfies, there can only be edges between v_{α_i,k,s_i} and every $v_{\alpha_{i+1},k,*}$. Since $\alpha_1, \alpha_2, \alpha_3$ satisfy all clauses in C_{3k+i+1} (C_{3k+1} if $i = 3$), v_{α_i,k,s_i} and $v_{\alpha_{i+1},k,s_{i+1}}$ will be connected by an edge. They form the only triangle between these vertices.

Based on above claim and the way we assign colors, it is not hard to see that there are Δ triangles of the same triple of colors if and only if there is a triple of partial assignments satisfies enough clauses in every group. Using the algorithm for Δ-Matching-Triangles, we can solve CNF-SAT in $O\left(\left(\Delta 2^{n/3+m/3\Delta}\right)^{c_\Delta}\right)$ time as we stated. $\square$

Above reduction together with the sparsification lemma [16], give us the following corollary.

Corollary 2.3. *If there is an algorithm solving Δ-Matching-Triangles on N-node graph in $O(N^{3-\epsilon})$ time for any constant $\epsilon > 0$, any $\omega(1) < \Delta(N) < N^{o(1)}$, we can solve k-SAT in $O\left(2^{n(1-\epsilon/6)}\right)$ time for every $k \geq 3$, refuting SETH.*

Proof. Given a k-SAT instance, we first apply the sparsification lemma [16] to get $2^{\epsilon/6}$ "sparse" k-SAT instances with n variables and cn clauses, where $c \leq (6k/\epsilon)^{O(k)}$. By Lemma 2.5, each instance runs in $O\left(2^{n(1-\epsilon/3)}\right)$ time. The total running time will be $O\left(2^{n(1-\epsilon/6)}\right)$. $\square$

SETH to Triangle-Collection Now we reduce CNF-SAT to Triangle-Collection. By Lemma 2.4, it is sufficient to reduce it to the restricted version.

Lemma 2.6. *If there is an algorithm which solves Triangle-Collection* on N-node graphs in $O(N^{c_1}\Delta^{c_2})$ time, then CNF-SAT on n variables and $m = n^{O(1)}$ clauses can be solved in $2^{nc_1/3}n^{O(1)}$ time.*

Proof. Given a CNF formula F on n variables and $m = n^c$ clauses, we split the variables into three sets U_1, U_2, U_3 of size $n/3$ each and enumerate over all the $N = 2^{n/3}$ partial assignments to each set.

We will construct a graph G on $O(Nm)$ nodes $A \cup B \cup C$, containing $O(m)$ nodes for each partial assignment. Suppose α is the i^{th} partial assignment to the variables in U_1, then add m nodes $\alpha_1, \ldots, \alpha_m$ to the set A and set their color to i. Suppose β is the i^{th} partial assignment to the variables in U_2, then add $2m$ nodes $\beta_{1,T}, \ldots, \beta_{m,T}$ and $\beta_{1,F}, \ldots, \beta_{m,F}$ to B and set their color to $N + i$. Finally, suppose γ is the i^{th} partial assignment to the variables in U_3, then add a node γ to the set C with color $2N + i$. Note that every color corresponds to a partial assignment.

The edges of G are defined according to the satisfiability relations between partial assignments and clauses. We say that a partial assignment ρ satisfies a clause C iff ρ sets one of the literals of C to true. For each triple of partial assignments α, β, γ to U_1, U_2, U_3 (respectively), we define the following edges. For each $j \in [m]$, we check whether α, β, γ satisfy C - the j^{th} clause in our formula F, and then:

- We add an edge between α_j and $\beta_{j,T}$ if α or β satisfy C, and we add an edge between α_j and $\beta_{j,F}$ otherwise.
- We add an edge between $\beta_{j,F}$ and γ_j if γ does not satisfy C.
- We add an edge between α_j and γ_j.

By the construction of G, it fits the input graph of Triangle-Collection*. We claim that a triple of colors will have no triangles in G iff the corresponding triple of partial assignments satisfy all clauses. To see this, note that a triangle must be of the form $\alpha_j \to \beta_{j,x} \to \gamma_j$ where $x \in \{T, F\}$, and that each α_j can participate in at most one such triangle. Moreover, this triangle exists in G iff the partial assignment obtained by combining (α, β, γ) does not satisfy the j^{th} clause in F, by construction.

By the assumption on Triangle-Collection* algorithm, we can solve CNF-SAT in $2^{nc_1/3}n^{O(1)}$ time.

$\square$

Corollary 2.1 only proves hardness of Δ-Matching-Triangles when Δ is sub-logarithm in n. However, the following lemma shows the hardness does not decrease when Δ increases, as long as it is sub-polynomial in n.

Lemma 2.7. *If we can solve Δ-Matching-Triangles on n-node graph G in $O(n^{c_\Delta})$ time, then we can solve Δ'-Matching-Triangles in $O\left(((\Delta-\Delta')n)^c\right)$ time for $\Delta' < \Delta$.*

Proof. Given an instance of Δ'-Matching-Triangles G on n nodes, we add $\Delta' - \Delta$ nodes to each of the color. Then take the i-th newly added node in all colors, make them a complete graph. It adds exactly $\Delta' - \Delta$ triangles to every triple of colors. Then run Δ-Matching-Triangles algorithm on the new graph. The running time is $O\left(((\Delta-\Delta')n)^c\right)$. □

Now we are ready to prove Theorem 1.1.

Reminder of Theorem 1.1 *Conjecture 1 implies that Triangle-Collection and Δ-Matching-Triangles, with $\omega(1) < \Delta(n) < n^{o(1)}$, on graphs with n nodes cannot be solved in $O(n^{3-\varepsilon})$ time, for any $\varepsilon > 0$.*

Proof. By Corollary 2.1, Corollary 2.3 and Lemma 2.7, we can get for $\omega(1) < \Delta < n^{o(1)}$, Δ-Matching-Triangles cannot be solve in $O(n^{3-\varepsilon})$ time, for any $\varepsilon > 0$ under Conjecture 1. Then by Corollary 2.2 and Lemma 2.6 and Lemma 2.4, we can prove the hardness for Triangle-Collection under Conjecture 1. □

From the theorem, we have the following corollary stating a "Hierarchy" between n^2 and n^3.

Reminder of Corollary 1.1 *Conjecture 1 implies that for any $\delta < 1$, there is an integer $\Delta \geq 1$ such that Δ-Matching-Triangles requires $n^{2+\delta-o(1)}$ time.*

Proof. Assume for contradiction that there is an $\epsilon > 0$ such that Δ-Matching-Triangles can be solve in $O(n^{3-\epsilon})$ for all constant $\Delta \geq 1$. Therefore, there is a sequence $\{a_\Delta\}_{\Delta\geq 1}$ of positive numbers such that Δ-Matching-Triangles on n-node graphs can be solve in $a_\Delta n^{3-\epsilon}$ steps. Let $\Delta(n) = \max\{\Delta : a_\Delta \leq n^{\epsilon/2}a_1, \Delta \leq \sqrt{n}\}$. We claim that with this parameter $\Delta(n)$, it contradicts with Theorem 1.1.

When $n \geq \max\{k^2, (a_k/a_1)^{2/\epsilon}\}$, $\Delta(n) \geq k$. This shows $\Delta(n) > \omega(1)$. Also by definition, $\Delta(n) < n^{o(1)}$. But $\Delta(n)$-Matching-Triangles can be solve in $a_{\Delta(n)}n^{3-\epsilon} \leq a_1 n^{3-\epsilon/2} = O(n^{3-\epsilon/2})$ time. It contracts with the fact that such $\Delta(n)$-Matching-Triangles cannot be solve in any truly subcubic time. □

3 Algorithm for Matching Triangles

In this section, we show how to solve Δ-Matching-Triangles efficiently when Δ is small.

Reminder of Theorem 1.5 *Δ-Matching-Triangles problem on an n-node graph G can be solved in $\tilde{O}\left(n^{3-c_\Delta}\right)$ time for $c_\Delta = \frac{2(3-\omega)^2}{(5-\omega)\Delta+1-\omega} > 0$.*

Proof. Without loss of generality, we may assume that the graph G is tripartite. We use two different approaches to detect if there are Δ triangles with the same triple of colors based on the number of nodes of the colors.[5] Let C_1, C_2, C_3 be the sizes of the colors.

First approach: In this case, we check if there are Δ triangles with the same triple of colors such that the sizes of the three colors are at most C_1, C_2, C_3 respectively. Let us focus on the colors with at most C_1, C_2, C_3 vertices in each part of the graph. We first arbitrarily assign an index for every vertex of a color from one to the size of the color. We enumerate the indices of vertices of $\Delta - 1$ triangles. There are $\binom{C_1C_2C_3}{\Delta-1}$ possibilities. For any two colors from different part in G, we check if all indices and corresponding edges of the $\Delta - 1$ triangles exist in these two colors. If some of them are missing, it means together with any third color, these two colors cannot have those $\Delta - 1$ triangles currently enumerated to. Thus we delete all edges between them from G. After we have checked for every pair of colors, we get a new graph G' depending on the indices of $\Delta - 1$ triangles. The last triangle will be different from first $\Delta - 1$. That is, for each of the first

[5]We may also call it the size of the color in the rest of the proof.

$\Delta - 1$ triangles, there is one edge that is different from the last one. We enumerate all 3^{Δ} possibilities, and remove the corresponding edges from each possible pair of colors in G'. At last we check if the remaining graph H has a triangle (without color restriction) using matrix multiplication in $O(n^{\omega})$ time.

We claim that the graph G has Δ triangles with the same triple of colors if and only if we found an H with a triangle in the algorithm. If G has such Δ triangles, at some point of the algorithm, we will enumerate to the correct indices of first $\Delta - 1$ triangles. G' will still contain the last triangle. Then eventually we will enumerate to a right guess of different edges from the first $\Delta - 1$ triangles. In this case, the resulting H still has the last triangle unremoved. H has a triangle. On the other hand, if at some point of the algorithm, we get an H with some triangle T in it. Look at the triple of colors which T has, their edges are not all deleted, which means they have all the edges of first $\Delta - 1$ currently enumerated to. These three colors contain the current first $\Delta - 1$ triangles, and T is different from all of them, since there is one edge removed from each of them.

The first approach correctly checks for all small-sized colors in

$$\tilde{O}\left(\binom{C_1C_2C_3}{\Delta - 1}3^{\Delta}n^{\omega}\right)$$

time.

Second approach: We check if there are Δ triangles with the same triple of colors with size at least C_1, C_2, C_3 respectively. First we delete all colors of small size from G, and get a graph G'. Note that in each part of G', there are at most $n/C_1, n/C_2, n/C_3$ different colors respectively. If there are more than $(\Delta - 1)n^3/(C_1C_2C_3) + 1$ triangles, which can be detected in $\tilde{O}(n^{\omega})$ time, there must be Δ such triangles. We can report YES immediately. Otherwise, we can list all the triangles in G' efficiently. [12] proposed an algorithm that can list t triangles in an n-node graph in $\tilde{O}\left(n^{\omega} + n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)}\right)$ time. Let $c = 2(3-\omega)/(5-\omega)$. Applying this algorithm to G', in

$$\begin{aligned}&\tilde{O}\left(n^{\omega} + n^{3(\omega-1)/(5-\omega)}(\Delta n^3/(C_1C_2C_3))^{2(3-\omega)/(5-\omega)}\right)\\ &= \tilde{O}\left(n^{\omega} + n^3(\Delta/(C_1C_2C_3))^c\right)\end{aligned}$$

time, we can list all triangles and check if there are Δ of them with the same triple of colors using a table of size $n^3/(C_1C_2C_3)$.

Main algorithm: In the following, we show how to combine these two approaches to get an efficient algorithm for general G. We divide the colors into $\log n$ groups based on their size. The colors in Group i will have between 2^i and 2^{i+1} nodes. We go over all triples of groups. For Groups u, v, w, the first approach runs in

$$\tilde{O}\left(\binom{2^{u+v+w}}{\Delta - 1}3^{\Delta}n^{\omega}\right)$$

time, while the second approach runs in

$$\tilde{O}\left(n^{\omega} + n^3(\Delta/2^{u+v+w})^c\right)$$

time. The algorithm first computes these two values, and then picks the faster one to detect if there are Δ such triangles among Groups u, v, w. There are $\log^3 n$ such triples of groups, the algorithm runs in

$$\tilde{O}\left(\max_{0\leq u,v,w\leq \log n} \min\left\{\binom{2^{u+v+w}}{\Delta - 1}3^{\Delta}n^{\omega}, n^{\omega} + n^3(\Delta/2^{u+v+w})^c\right\}\right)$$

time. The maximum value is achieved when two terms are equal:

$$\begin{aligned}
&\tilde{O}\left(\max_{0\le u,v,w\le \log n}\min\left\{\binom{2^{u+v+w}}{\Delta-1}3^{\Delta}n^{\omega}, n^{\omega}+n^3(\Delta/2^{u+v+w})^c\right\}\right)\\
\le&\tilde{O}\left(\max_{0\le s\le 3\log n}\min\left\{2^{s(\Delta-1)}3^{\Delta}n^{\omega}/(\Delta-1)!, n^3(\Delta/2^s)^c\right\}\right)\\
\le&\tilde{O}\left(\left(n^{3-\omega}\Delta^c 3^{-\Delta}(\Delta-1)!\right)^{(\Delta-1)/(\Delta-1+c)}3^{\Delta}n^{\omega}/(\Delta-1)!\right)\\
\le&\tilde{O}\left(\left(n^3\Delta^c\right)^{(\Delta-1)/(\Delta-1+c)}\left(3^{\Delta}n^{\omega}/(\Delta-1)!\right)^{c/(\Delta-1+c)}\right)\\
\le&\tilde{O}\left(n^{3-(3-\omega)c/(\Delta-1+c)}\right)\\
=&\tilde{O}\left(n^{3-c_{\Delta}}\right)
\end{aligned}$$

We have the running time as we stated. □

4 Reductions to Other Problems

Recall the definition of Triangle-Collection* from Section 2. Lemmas 2.6 and 2.3 prove that even this restricted version of Triangle-Collection has an $n^{3-o(1)}$ lower bound under Conjecture 1. In this section we reduce this version to well-studied problems to prove our new lower bounds.

Triangle-Collection* to Dynamic Problems. The following reductions to dynamic problems prove Theorem 1.2.

Lemma 4.1. *Triangle-Collection* can be reduced to $\tilde{O}(n^2)$ updates and queries of #SSR, #SCC, #SS-Sub-Conn, or Max-Flow on a dynamic graph on $O(n)$ nodes.*

Proof. (#SSR) Let G be the input to Triangle-Collection*, we construct a graph H by directing the edges of G from B to C and removing part A from the graph (it will be implicitly simulated). We also add a source node s which will be dynamically represent the different colors of A. We also add a target node t_c, for every color c of C, and we add a path of length Δ^2 starting at t_c so the nodes on this path are reachable from s if and only if t_c is reachable from s. We have a phase for each color a of A, in which we perform two stages. In the first stage, we go over all colors c of C and add an edge from $c_{j,x}$ to t_c for every $j\in[\Delta]$, where x is so that $\{a_j, c_{j,x}\}$ is an edge in G. In the second stage, we go over all colors b of B and edges $s\to b_{j,x}$ for every $j\in[\Delta]$, where x is so that $\{a_j, b_{j,x}\}$ is an edge in G, then we ask the query and check if s can reach at least $(\Delta^2+2\Delta)n$ nodes, before removing the added edges and moving on to the next b.

Observe that the answer to the query in one of the (a,b) stage is "no" (s reaches less than $(\Delta^2+2\Delta)n$ nodes), iff there is a triple of colors a,b,c without any triangles. To see this, note that this happens iff s cannot reach some node t_c when the query is asked.

(Max-Flow) In the above reduction, add a target node t and connect every t_c node to it with an edge of capacity 1. The other edges of the graph will have capacity n. Observe that the answer to the query is again determined by whether s can real t_c for all colors c.

(#SS-Subgraph-Connectivity) In the reduction to #SSR, replace the addition of the $c_{j,x}\to t_c$ edges by "turning on" updates on the $c_{j,x}$ nodes. Similarly, replace the addition of $s\to b_{j,x}$ edges with "turning on" the $b_{j,x}$ node updates. Again, the query allows us to learn whether s can "reach" evert t_c node.

(#Strongly Connected Components) Consider the reduction to #SSR again. Add two new nodes x_B and x_C. Connect every node in B bidirectionally to x_B, similarly from C and x_c. Add edges from the t_c nodes to s. At the (a,c) substage, consider the $c_{j,x}$ nodes (the neighbors of a_j) and remove their edges to and from x_C, instead, bi-connect them to t_c. At the (a,b) substage, consider the $b_{j,x}$ nodes (the neighbors of a_j) and remove their edges to and from x_B, instead, bi-connect them to s. The claim now is that the number of strongly connected components is 3 iff s can reach all the t_c nodes. The first direction is clear: if

s can reach every t_c then we have the x_B and x_C components and the rest of the graph is one big component containing all the neighbors of a and the t_c nodes. The second is also simple: if some t_c cannot be reached from s through B, then there is no way it can be reached at all, and it will be in a fourth components. □

Reductions to flow. Finally, we present our reduction to ST-Max-Flow on a static graph, proving Theorem 1.3. The reduction shows how to use flow to count the number of different groups of nodes through which there is a path from the source to the target.

Lemma 4.2. *Triangle-Collection* on N nodes can be reduced to ST-Max-Flow on a graph with $O(N^2)$ nodes and edges, and $|S| = |T| = O(N)$.*

Proof. Given a tripartite graph G as input to Triangle-Collection*, we construct a flow network H as input to ST-Max-Flow as follow.

The nodes of H will be composed of five partitions: S, A', B', C', T. For each color a of A (in G) we create a node s_a in S. For each pair of colors a of A and b of B we create a node a_b in A'. For each node $b_{j,x} \in B$ we create a node $b'_{j,x}$ in B'. For each node $c_{j,x} \in B$ we create a node $c'_{j,x}$ in C'. For each color c of C we create a node t_c in T.

Next, we define the edges of H. Add edges of capacity 1 from s_a to the a_b nodes for every color b of B. For each edge $\{a_j, b_{j,x}\}$ in G, add an edge $a_b \to b'_{j,x}$ to H with capacity 1. For each edge $\{b_{j,x}, c_{j,y}\}$ in G, add an edge $b'_{j,x} \to c'_{j,y}$ to H with capacity 1. For each node $c'_{j,x} \in C'$, add an edge of capacity n from $c'_{j,x}$ to t_c to H. Finally, for every color a of A and node $c_{j,x} \in C$ such that $\{a_j, c_{j,x}\}$ is *not* an edge in G, we add an edge $s_a \to c'_{j,x}$ of capacity n to H.

Note that the number of nodes and edges in H is $O(|A||B|) = O(N^2)$, and that $|S| = |T| = n$. The next claim proves the correctness of our reduction.

Claim 3. *For a pair of nodes $s_a \in S, t_c \in T$, the maximum flow from s_a to t_c in H is $nd(p-1) + n$ if for every color b of B, the subgraph of G induced by the colors a, b, c contains a triangle, and is smaller otherwise.*

For the first direction, assume that for every color b of B, the triple a, b, c contains a triangle. Note that s_a can push n units of flow along each of the edges $s_a \to c'_{j,x}$ and then along the edges $c'_{j,x} \to t_c$, and by our assumptions on G, there are exactly $d(p-1)$ such edges in H, resulting in a total of $nd(p-1)$ units of flow. We will call this "base flow". We claim that every color b can contribute another unit of flow if the triple a, b, c contains a triangle in G. Indeed, s_a can push one unit of flow to a_b for every color b and then find the $j \in [d]$ for which $(a_j, b_{j,x}, c_{j,y})$ is a triangle in G and push a unit of flow along the edges $a_b \to b'_{j,x} \to c'_{j,y} \to t_c$. Note that since $\{a_j, c_{j,y}\}$ is an edge in G, we have not pushed any flow along the edge $c'_{j,y} \to t_c$ in our "base flow". Moreover, since we are adding up to n additional units of flow, we will not violate any of the capacity constraints. Thus, after these additions we end up with $nd(p-1) + n$ units of flow.

For the other direction, assume that for some color b, the triple a, b, c does not contain a triangle. This implies that for every $j \in [d]$, the edge $b'_{j,x} \to c'_{j,y}$ is not in H, where x, y are such that the edges $\{a_j, b_{j,x}\}$ and $\{a_j, c_{j,y}\}$ are in G. We will show that that at least one of the edges leaving s_a cannot be saturated in a legal flow in H, thus implying that the maximum flow is less than the sum of capacities on the $(\{s_a\}, H \setminus \{s_a\})$ cut, which is $nd(p-1)+n$. If there is no flow on the edge $s_a \to a_b$, we are done. Otherwise, one unit of flow is pushed along the path $s_a \to a_b \to b'_{j,x} \to c'_{j,z} \to t_c$, for some $j \in [d]$ and x, z such that $\{a_j, b_{j,x}\}, \{b_{j,x}, c_{j,z}\}$ are in G. But by the above, we know that $\{a_j, c_{j,z}\}$ is not an edge in G, and therefore the edge $s_a \to c'_{j,z}$ is in H. Only n flow can leave $c'_{j,z}$, while there is one unit coming from part B', which implies that only $n-1$ units of flow can come from the edge $s_a \to c'_{j,z}$, and we are done again, since we found an edge leaving s_a that is not saturated. □

5 Reductions to Single-Source Max-Flow

To prove Theorem 1.4 we give a simple reduction from MAX-CNF-SAT to Single-Source-Max-Flow. Note that our lower bound is not only based on SETH, but on the weaker assumption that MAX-CNF-SAT cannot be solved in $2^{(1-\varepsilon)n}\text{poly}\ (m)$ time - a problem for which even $2^n/\text{poly}\ (n)$ algorithms are not known.

Lemma 5.1. *MAX-CNF-SAT on n variables and m clauses can be reduced to $O(m)$ instances of Single-Source-Max-Flow on graphs with $N = 2^{n/2}$ nodes and $O(2^{n/2}m)$ edges with capacities in $[N]$.*

Proof. Let F be the input CNF formula on n variables and m clauses. As usual, we split the variables into two parts of size $n/2$ and enumerate all $N = 2^{n/2}$ partial assignments for each part. Our goal is to find the pair of partial assignments α, β that satisfy the maximum number of clauses. We have an instance of Single-Source-Max-Flow for each value $K \in [m]$ in which we check if there is a pair α, β that satisfies at least K clauses, using a single call to Single-Source-Max-Flow on a graph defined as follows.

Create a layer A containing a node v_α for each partial assignment α to the first set of variables, and a layer B containing a node v_β for each partial assignment β to the second set of variables. Add a layer C in the middle, containing a node c_j for each clause C_j in our CNF formula. Add edges $v_\alpha \to c_j$ of capacity 1 for each pair of α, C_j such that α does not satisfy C_j (does not set any of the literals to true), and add edges $c_j \to v_\beta$ of capacity 1 for each pair β, C_j such that β does not satisfy C_j. Finally, add a source node s, and connect it with edges $s \to v_\alpha$ of capacity $(m - K + 1)$, for each β.

Observe that the number of paths from a node v_α to a node v_β is exactly the number of clauses that are not satisfied by the (α, β) assignment. Therefore, the maximum flow from s to a node v_β would be $n(m - K + 1)$, unless for some α, there do not exist $(m - K + 1)$ paths from v_α to v_β, and thus we have a pair α, β that satisfy at least $m - (m - K + 1) + 1 = K$ clauses. ☐

Finally, we present reductions from Triangle detection to Single-Source-Max-Flow. An equivalent formulation of Triangle detection is: given a tripartite graph (A, B, C), determine if there is a triple of nodes, one from each partition, that form a clique.

Proposition 1. *If Single-Source-Max-Flow with capacities in $\{1, n\}$ on a directed graph with n nodes and m edges can be solved in $T(m, n)$ time, then Triangle detection on a graph with m edges can be solved in $T(O(m + n \log n), O(n))$ time.*

Proof. Given a Triangle detection instance (A, B, C), construct the following flow instance. First, create another copy of the nodes of A, call it A'. For every edge between parts A and B, add a directed edge from A to B, and similarly direct the edges from B to C. Then, replace the edges between A and C with corresponding edges from C to A'. Add a layer X on $O(\log n)$ nodes. Connect $a \in A$ to every node $x_i \in X$ for which the i^{th} bit in the integer $a \in [n]$ is 1. Connect $x_i \in X$ to every node $a' \in A'$ for which the i^{th} bit in the integer $a \in [n]$ is 0. Set all the above capacities to n. Add a source node s and connect it to all the nodes in A with capacity 1. The correctness of the reduction follows from the following simple claim: for a node $t = a' \in A'$, the max s,t-flow is n if a is in a triangle, and $n - 1$ otherwise. To see this, note that the max flow from s to a' is n iff there is a path from a to a', and such path must go through $B \cup C$, and every such path corresponds to a triangle. ☐

By incurring an overhead of n^2 extra edges, we can get a reduction to the unit capacity case.

Proposition 2. *If Single-Source-Max-Flow with* unit capacities *on a directed graph with n nodes and m edges can be solved in $T(m, n)$ time, then Triangle detection on a graph with m edges can be solved in $T(O(n^2), O(n))$ time.*

Proof. In the previous proof, remove the X layer, and instead add an a directed edge from a_1 to a'_2 for any $a_1 \neq a_2$. Set the capacity of all the edges to 1 and the same claim still holds. ☐

One interesting corollary of the above reductions is that a *combinatorial* algorithm for Single-Source-Max-Flow on dense unit capacity networks that runs in truly subcubic time would imply, via [73], a *combinatorial* truly subcubic Boolean Matrix Multiplication algorithm.

A final simple modification to the above reductions is to augment the triangle detection instance with edge weights, in order to solve the Minimum Weight Triangle problem (a problem equivalent to APSP under subcubic reductions [73]). In the above reductions, we would add the weights as costs to the flow instance and ask for the Single-Source *min-cost-max-flow.* This gives an APSP based lower bound for the min-cost version.

Acknowledgement. We thank Søren Dahlgaard for his helpful comments.

References

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. *SODA*, 2015.

[2] Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *ICALP (1)*, pages 1–12, 2013.

[3] Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 1–12, 2014.

[4] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. *SODA*, 2015.

[5] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. *FOCS*, 2014.

[6] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP (1)*, pages 39–51, 2014.

[7] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 114–125, 2014.

[8] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. In *Proc. SODA*, 2005.

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

[10] Gill Barequet and Sariel Har-Peled. Some variants of polygonal containment and minimum hausdorff distance undertranslation are 3SUM-hard. In *Proc. SODA*, pages 862–863, 1999.

[11] M. A. Bender, J. T. Fineman, S. Gilbert, and R. E. Tarjan. A new approach to incremental cycle detection and related problems. *CoRR*, abs/1112.0784, 2011.

[12] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 223–234, 2014.

[13] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square - on the complexity of quadratic-time solvable problems. *CoRR*, abs/1407.4972, 2014.

[14] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.

[15] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. *FOCS*, 2014.

[16] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260, 2006.

[17] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

[18] T. M. Chan. Dynamic subgraph connectivity with geometric applications. *SIAM J. Comput.*, 36(3):681–694, 2006.

[19] T. M. Chan, M. Pătraşcu, and L. Roditty. Dynamic connectivity: Connecting to networks and geometry. In *FOCS*, pages 95–104, 2008.

[20] K. Chen, P. Hsu, and K. Chao. Approximate matching for run-length encoded strings is 3sum-hard. In *CPM*, pages 168–179. Springer, 2009.

[21] Otfried Cheong, Alon Efrat, and Sariel Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. SODA*, pages 1098–1107, 2004.

[22] Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. Graph connectivities, network coding, and expander graphs. *SIAM Journal on Computing*, 42(3):733–751, 2013.

[23] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 273–282, 2011.

[24] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *STOC*, pages 301–310, 2013.

[25] Evgeny Dantsin and Alexander Wolpert. On moderately exponential time for SAT. In *Proc. 13th International Conference on Theory and Applications of Satisfiability Testing*, pages 313–325, 2010.

[26] Mark de Berg, Marko de Groot, and Mark H. Overmars. Perfect binary space partitions. *Computational Geometry: Theory and Applications*, 7(81):81–91, 1997.

[27] C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: Breaking through the $o(n^2)$ barrier. In *Proc. FOCS*, volume 41, pages 381–389, 2000.

[28] R. Duan. New data structures for subgraph connectivity. In *ICALP (1)*, pages 201–212, 2010.

[29] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comp. Sci.*, 326(1-3):57–67, 2004.

[30] J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal on Computing*, 28(4):1198–1214, 1999.

[31] S. Even and Y. Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981.

[32] D. Frigioni and G. F. Italiano. Dynamically switching vertices in planar graphs. *Algorithmica*, 28(1):76–103, 2000.

[33] A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.

[34] François Le Gall. Powers of tensors and fast matrix multiplication. *CoRR*, abs/1401.7714, 2014.

[35] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. E. Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Transactions on Algorithms*, 8(1):3, 2012.

[36] Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994.

[37] Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An õ(mn) gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 605–614, 2007.

[38] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 674–683, 2014.

[39] E. A. Hirsch. Two new upper bounds for SAT. In *Proc. SODA*, pages 521–530, 1998.

[40] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[41] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[42] G. F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Inf. Process. Lett.*, 28(1):5–11, 1988.

[43] Z. Jafargholi and E. Viola. 3sum, 3xor, triangles. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:9, 2013.

[44] Hamidreza Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. *CoRR*, abs/1311.3171, 2013.

[45] Allan Grønlund Jørgensen and Seth Pettie. Threesomes, degenerates, and love triangles. *CoRR*, abs/1404.0799, 2014.

[46] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 302–311, 1984.

[47] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

[48] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 3sum hardness in (dynamic) data structures. *CoRR*, abs/1407.6756, 2014.

[49] J. Lacki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Transactions on Algorithms*, 9(3):27, 2013.

[50] Jakub Lacki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 599–608, 2012.

[51] Yin Tat Lee and Aaron Sidford. Following the path of least resistance : An õ(m sqrt(n)) algorithm for the minimum cost flow problem. *CoRR*, abs/1312.6713, 2013.

[52] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA*, pages 777–789, 2011.

[53] J. Erickson M. Soss and M. H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry: Theory and Applications*, 26(3):235–246, 2002.

[54] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262, 2013.

[55] Yurii Nesterov and A. S. Nemirovskii. An interior-point method for generalized linear-fractional programming. *Math. Program.*, 69:177–204, 1995.

[56] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[57] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610, 2010.

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

[59] M. Pătraşcu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. SODA*, pages 1065–1075, 2010.

[60] L. Roditty. Decremental maintenance of strongly connected components. In *SODA*, pages 1143–1150, 2013.

[61] L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 515–524, New York, NY, USA, 2013. ACM.

[62] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. In *FOCS*, pages 679–689, 2002.

[63] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *ESA*, pages 580–591, 2004.

[64] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. FOCS*, volume 45, pages 509–517, 2004.

[65] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proc. FOCS*, pages 410–414, 1999.

[66] Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269, 2013.

[67] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90, 2004.

[68] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC*, pages 343–350, 2000.

[69] V. Vassilevska and R. Williams. Finding, minimizing, and counting weighted subgraphs. In *Proc. STOC*, pages 455–464, 2009.

[70] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Proc. ICALP*, pages 1227–1237, 2004.

[71] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014.

[72] Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *SODA*, pages 1867–1877, 2014.

[73] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.

[74] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

A Background

Conditional lower bounds. Here, we give a brief survey of the three conjectures and the known lower bounds.

The first, and most prominent example of this approach concerns the 3-SUM problem: given n integers in $\{-O(n^3), \ldots, O(n^3)\}$, do three of them sum to zero? A simple algorithm solves the problem in $O(n^2)$ time, and only logarithmic improvements are known, by Baran, Demaine, and Pătraşcu [9] and more recently by Grønlund and Pettie [45] for the more general problem on real numbers. The following widely-believed conjecture states that no n^ε factor improvements are possible for 3-SUM.

The 3-SUM Conjecture: *There is no algorithm that can solve 3-SUM on n numbers in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$.*

Since the seminal work of Gajentaan and Overmars [33], there have been many papers proving the hardness of computational geometry problems, based on the 3SUM conjecture, e.g. [26, 53, 30, 8, 21, 10]. More recently, the 3-SUM Conjecture has been used in surprising ways to show polynomial lower bounds for combinatorial problems in dynamic algorithms [57, 5, 48], Graph algorithms [57, 43, 69, 2], and pattern-matching [20, 6, 7].

The second example of this approach is the work on subcubic-equivalences with the All-Pairs-Shortest-Paths (APSP) problem: given an n node graph with edge weights in $\{-n^c, \ldots, n^c\}$, compute the distances between all pairs of nodes. Despite many attempts, only sub-polynomial improvements are known over the classic $O(n^3)$ algorithms for the problem. The current best is the recent $n^3/2^{\Omega(\sqrt{\log n})}$ of Williams [71]. A widely-believed conjecture in graph algorithms states that $n^{3-o(1)}$ time is required to solve APSP.

The APSP Conjecture: *There is no algorithm that can solve APSP on n node graphs in $O(n^{3-\varepsilon})$ time for some $\varepsilon > 0$.*

A long list of problems are known to be subcubic-equivalent to APSP in the sense that if any of them can be solved in $O(n^{3-\varepsilon})$ time, then all of them can [73, 14, 1]. In addition, many conditional lower bounds have been shown under the APSP conjecture [69, 2, 63, 5].

The third example concerns the exact complexity of CNF-SAT: given a CNF formula on n variables and m clauses, is it satisfiable? The best upper bounds remain of the form $2^{n-o(n)}\text{poly }(m)$ (e.g. [39, 56, 65, 4]). SETH of Impagliazzo, Paturi and Zane [40, 41] states that better algorithms do not exist.

The Strong Exponential Time Hypothesis (SETH): *There is no algorithm that can solve CNF-SAT on n variables and m clauses in $2^{(1-\varepsilon)n}\text{poly }(m)$ time for some $\varepsilon > 0$.*

Recently, many surprising SETH-based lower bounds have been shown in several different areas like graph algorithms [59, 61, 13, 1, 5], pattern matching [6, 72, 70], computational geometry [15], and exact algorithms [17, 25, 52, 24]. Moreover, it is known that refuting SETH implies new circuit lower bounds [44].

Dynamic algorithms. A very active area of research concerns finding efficient algorithms that can maintain certain properties of a dynamic graph - i.e. a graph that undergoes a sequence of insertions and deletions of nodes or edges. Algorithms with low amortized update and query times are desirable. The classic connectivity problem in undirected graphs has an algorithm with $O(\log n \log^3 \log n)$ amortized update time [68], and a near-matching $\Omega(\log n)$ unconditional, cell-probe, lower bound [58]. For many other classic problems, the best known algorithms require an $O(n^c)$ amortized update time for some $c > 0$, while no unconditional lower bounds beyond $\Omega(\log n)$ are known. Some examples include maintaining the number of strongly connected components (#SCC) [35, 11, 62, 49, 60, 38], the number of nodes reachable from a fixed source node (#SSR) [64, 27, 31, 42] in a directed graph under edge updates. Another example is to maintain the number of nodes connected to a fixed source in an undirected graph under node updates (#SS-Subgraph-Connectivity) [32, 18, 19, 28]. Trivial $O(m + n)$ update time algorithms for these problems recompute the answer after every update, and many faster algorithms have been proposed in recent years.

In search of better understanding of the complexity of these problems, Pătraşcu [57] proposed to prove lower bounds conditioned on the 3-SUM conjecture. After a sequence of reductions from 3-SUM by Pătraşcu [57] that was later optimized by Abboud and Vassilevska [5] and by Kopelowitz, Pettie and Porat [48], we can conclude that the above problems require $n^{2/3-o(1)}$ amortized update if the 3-SUM conjecture holds. This lower bound does not match the known upper bounds, and in fact, Abboud and Vassilevska [5] show that there is a higher $n^{1-o(1)}$ lower bound under SETH. However, as explained by the later works, obtaining a higher lower bound from the 3-SUM conjecture using Pătraşcu's approach seems impossible, due to certain inefficiencies in some of the steps in the reduction, and obtaining a higher lower bound from 3-SUM has remained an open question. No lower bound for these problems was known under the APSP conjecture.

In Section 4, we give simple reductions from the Triangle-Collection to these classic dynamic problems to obtain linear $n^{1-o(1)}$ lower bounds on the amortized update times, under our very weak Conjecture 1. The tightness of our reduction from 3-SUM to the purely-combinatorial Triangle-Collection problem allows us to overcome the $n^{2/3-o(1)}$ barrier for lower bounds under the 3-SUM conjecture.

We also add the dynamic Max-Flow problem to the list: what is the maximum flow from a source s to a target t in an n-node directed graph with capacities in $[n]$ that undergoes edge insertions and deletions.