SIAM J. COMPUT.
Vol. 47, No. 3, pp. 1098–1122

MATCHING TRIANGLES AND BASING HARDNESS ON AN EXTREMELY POPULAR CONJECTURE*

AMIR ABBOUD†, VIRGINIA VASSILEVSKA WILLIAMS‡, AND HUACHENG YU§

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 all pairs shortest paths (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; and a hierarchy of natural graph problems on n nodes with complexity n^c for $c \in (2,3)$. Only slightly nontrivial 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.

Key words. SETH, APSP, 3-SUM, triangle finding, conditional hardness, hardness in P

AMS subject classifications. 68Q17, 68W05

DOI. 10.1137/15M1050987

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 superlinear" 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 satisfiabil-

*Received by the editors December 2, 2015; accepted for publication (in revised form) March 21, 2018; published electronically June 26, 2018. A preliminary version of this paper [9] has appeared in STOC 2015.

http://www.siam.org/journals/sicomp/47-3/M105098.html

Funding: The first and second authors were supported by NSF grants CCF-1417238, CCF-1528078, and CCF-1514339 and BSF grant BSF:2012338. The third author was supported by an Alfred P. Sloan Fellowship and NSF grants CCF-1212372 and CCF-1552651 (CAREER).

†IBM Almaden Research Center, Stanford, CA 94305 (amir.abboud@gmail.com).

‡EECS Department, CSAIL, MIT, Cambridge, MA 02139 (virgi@mit.edu).

§John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02139 (yuhch123@gmail.com).

ity. 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* $\{-n^c, \ldots, n^c\}$ *for some large*[1] *constant* $c \geq 1$*, in* $O(n^{3-\varepsilon})$ *time for any* $\varepsilon > 0$.

Strong Exponential Time Hypothesis (SETH). For every $\varepsilon > 0$ *there is an integer* $k \geq 3$ *such that* k*-SAT on* n *variables cannot be solved in* $O(2^{(1-\varepsilon)n})$ *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 [68]: if k-SUM can be solved in $n^{o(k)}$ time, then the weaker conjecture ETH[2] 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 [80] and interior-point methods [57, 67] 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 [66, 78, 63, 58, 31]. 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.

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 [70], and Vassilevska Williams and Williams [82] 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, both the 3-SUM and APSP conjectures would be refuted. More recently, Abboud, Vassilevska Williams, and Weimann [11] show that an $O(n^{2-\varepsilon})$ algorithm for the local alignment problem from bioinformatics refutes

[1] To refute this conjecture, one has to find an algorithm that, for all $c \geq 1$, solves APSP in $O(n^{3-\varepsilon})$ whenever the weights are in $\{-n^c, \ldots, n^c\}$. A bolder but still plausible conjecture is that truly subcubic algorithms do not exist even when $c = 1$.

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

both the 3-SUM conjecture and SETH. No nontrivial 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 as follows: 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 [87, 44]. However, when Δ increases to $\omega(1)$ it must require $n^{3-o(1)}$ time under Conjecture 1. It is natural to wonder what the complexity of the problem is 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 interesting results 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. 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 [81] and a near-matching $\Omega(\log n)$ unconditional, cell-probe, lower bound [71]. 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) [46, 17, 74, 61, 72, 49], the number of nodes reachable from a fixed source node (#SSR) [76, 36, 40, 54] 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) [41, 26, 27, 37]. 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.

With the goal of improving our understanding of the complexity of these problems, Pătraşcu [70] proposed to prove lower bounds conditioned on the 3-SUM conjecture. After a sequence of reductions from 3-SUM by Pătraşcu [70] that was later optimized by Abboud and Vassilevska Williams [7] and by Kopelowitz, Pettie, and Porat [59], we can conclude that the above problems require an $n^{2/3-o(1)}$ amortized update if the 3-SUM conjecture holds. This lower bound does not match the known upper bounds, and, in fact, Abboud and Vassilevska Williams [7] 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.

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 at 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 the 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 [78, 58, 66, 63]. It also seems that these algorithms take near-linear time in practice, and the bottleneck for improving the upper bounds might be a weakness in the current algorithm analysis. Thus, attempting to prove superlinear 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 Lee and Sidford [63].[3] 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, Lau, and Leung [30] 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 [47] show that the maximum flow between $\Omega(n)$ st-pairs can be found in the time it takes for a single Max-Flow computation; however, these pairs cannot be specified in advance; i.e. the algorithm is free to choose which $\Omega(n)$ pairs to output. Łącki et al. [62] 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 for unit-capacity graphs [48].

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 counterintuitive thing is true: it is not possible to compute n single-source flows in

[3] Throughout this paper, $\tilde{O}(f(n))$ stands for $O(f(n) \cdot \log^{O(1)} n)$.

a network faster than by calling an *st*-flow algorithm n times.[4] In section 4 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 [68, 38], 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.[5]

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 a 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 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 superquadratic* 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 $\Omega(n^{2+\delta})$ 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 that is both truly subcubic and truly superquadratic (that is, $\Omega(n^{2+\varepsilon})$ for some $\varepsilon > 0$). Note that for smaller $\Delta \leq D$ the time complexity could be $O(n^{2+o(1)})$ as far as we know. We also remark that when Δ reaches n^ε for some constant ε, the complexity of Δ-Matching-Triangles decreases to truly subcubic yet again.[6]

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 Conjecture 2, which is

[4] Łącki et al. [62] 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.

[5] 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 $n^{2.8}$.

[6] If $\Delta = \Omega(n^\varepsilon)$, then at least one of the three colors in the solution has to have $\Omega(n^{\varepsilon/3})$ nodes. For each such "heavy" color A with n_A nodes, we can use an $n \times n_A \times n$ matrix product to count, for each pair of other colors B, C, the number of triangles with colors A, B, C. This product takes time $O(n^2 \cdot n_A^{(1-\varepsilon')})$ for some $\varepsilon' > 0$, since $n_A = \Omega(n^{\varepsilon/3})$, and the total time is truly subcubic.

$n^{3-9/(\Delta+3)-o(1)}$. 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 the 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 ExactWeight-Triangle (EW-Triangle) as an intermediate problem, since it requires $n^{3-o(1)}$ time unless both conjectures are false [6].

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* (x, y, z) *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 iff 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 [6] as a step toward reducing k-SUM to k-Clique. Besides using this lemma, our reductions are very different from theirs.

LEMMA 2.1 ([6]). *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* $f_1, \ldots, f_s : [-n^c, n^c] \to [-p/3, p/3]^d$ *and* s *target vectors* $\mathbf{t_1}, \ldots, \mathbf{t_s} \in [-p, p]^d$ *such that for any three numbers* $x, y, z \in [-n^c, n^c]$*,* $x + y + z = 0$ *iff for some* $i \in [s]$*,* $f_i(x) + f_i(y) + f_i(z) = \mathbf{t_i}$.

EW-Triangle to Δ*-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 edges. The summation of numbers is simulated by a path that walks along these edges. 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 $a_1, \ldots, a_d$ 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 ith 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 dimension. 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 jth 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 jth 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 $s = 2^{O(\Delta)}$ instances 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.*

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 an 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.*

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.

EW-Triangle to Triangle-Collection. We first define a "restricted" version of Triangle-Collection, called Triangle-Collection*. Then we reduce the 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 that are in parts A, B, C respectively, there is a triangle (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, except we take the complement of one side of the edges to make the *absence* of an $(a_j, b_{j,\star}, c_{j,\star})$ triangle correspond to the sum being zero on the jth 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 complement all the edges between B'_i and C'_i (a pair is now connected by an edge iff it was not an edge originally) 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 iff there are no triangles of colors (a, b, c) in G''_i for some i. It is not hard to verify that all G''_is 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.*

Proof. 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*, we construct a new colored graph G' with vertex sets A', B', C'. For each vertex $a \in A$ of G, we create vertices a, a_B, and a_C in A', which all have the same color as vertex a in G. Similarly, for each $b \in B$ and $c \in C$, we create b, b_C, b_A and c, c_A, c_B. For every edge in G, we add the same edge to G'. In addition, 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 $(b_A, b'_A), (b_C, b'_C), (c_A, c'_A), (c_B, c'_B)$. 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). G' preserves every triangle in G and 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. This completes the proof of the claim. ∎

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. ∎

Setting $\Delta = 2^{\Theta(\sqrt{\log n})}$ in Lemma 2.4 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.*

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 [6]. 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),*

determine whether there is 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 after preprocessing the graph in $O(N^2)$ time. Therefore, the problem can be solved in time

$$O((\#\text{colors})^3 + (\#\text{nodes})^2) = O(n^3 + N^2) = O(N^{3\Delta/(\Delta+3)}) = O(N^{3-9/(\Delta+3)}),$$

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 $O((\Delta 2^{n/3+m/3\Delta})^{c\Delta})$ 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 $v_{\alpha_{i+1},k,s_{i+1}} \in V_{i+1}$ 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 $i = 3$).[7] 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 $v_{\alpha_1,k,*}, v_{\alpha_2,k,*}, v_{\alpha_3,k,*}$ iff they satisfy all clauses in $C_{3k+1}, C_{3k+2}, C_{3k+3}$. If there is a 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 of 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 the 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 iff there is a triple of partial assignments that satisfies enough clauses in every group. Using the algorithm for Δ-Matching-Triangles, we can solve CNF-SAT in $O((\Delta 2^{n/3+m/3\Delta})^{c\Delta})$ time as we stated. ∎

The above reduction together with the sparsification lemma [23] gives us the following corollary.

COROLLARY 2.3. *If there is an algorithm solving Δ-Matching-Triangles on an 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 $O\left(2^{n(1-\epsilon/6+o(1))}\right)$ time for every $k \geq 3$, refuting SETH.*

Proof. Given a k-SAT instance, we first apply the sparsification lemma [23] to generate $2^{\epsilon n/6}$ 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 $O\left(2^{n(1-\epsilon/3+o(1))}\right)$ time. The total running time will be $O\left(2^{n(1-\epsilon/6+o(1))}\right)$. ∎

[7] For simplicity of notation, let $\alpha_4 = \alpha_1, s_4 = s_1, V_4 = V_1$.

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. Let $\alpha^{(i)}$ be the ith partial assignment to the variables in U_1; we add m nodes $\alpha^i_1, \ldots, \alpha^i_m$ to the set A and set their color to i. Let $\beta^{(i)}$ be the ith partial assignment to the variables in U_2; we add $2m$ nodes $\beta^i_{1,T}, \ldots, \beta^i_{m,T}$ and $\beta^i_{1,F}, \ldots, \beta^i_{m,F}$ to B and set their color to $N + i$.[8] Finally, let $\gamma^{(i)}$ be the ith partial assignment to the variables in U_3; then add nodes $\gamma^i_1, \ldots, \gamma^i_m$ 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 the jth clause C_j in our formula F, and then do the following:

- We add an edge between α_j and $\beta_{j,T}$ if α or β satisfies C_j, 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_j.
- 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 satisfies all clauses. To see this, note that a triangle must be of the form $\alpha_j \to \beta_{j,F} \to \gamma_j$. Such a triangle exists iff clause C_j is not satisfied by assignment (α, β, γ); i.e., all triples of colors contain a triangle iff any assignment does not satisfy the formula.

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

Corollary 2.1 only proves hardness of Δ-Matching-Triangles when Δ is sublogarithm in n. However, the following lemma shows the hardness does not decrease (omitting $n^{o(1)}$ factors) when Δ increases, as long as it is subpolynomial 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 $O\left(((\Delta - \Delta')n)^{c\Delta}\right)$ 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 colors. Then take the ith newly added nodes in all colors and make them a complete graph. It adds exactly $\Delta - \Delta'$ triangles to every triple of colors. Then run the Δ-Matching-Triangles algorithm on the new graph. The running time is $O(((\Delta - \Delta')n)^{c\Delta})$. ∎

Now we are ready to prove Theorem 1.1.

[8] We added the nodes $\beta^i_{j,T}$ merely to fit the definition of Triangle-Collection*. The reduction works without them, but having this definition of Triangle-Collection* will simplify later reductions to other problems.

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 $\omega(1) < \Delta < n^{o(1)}$ that Δ-Matching-Triangles cannot be solved in $O(n^{3-\varepsilon})$ time, for any $\varepsilon > 0$ under Conjecture 1. Then by Corollary 2.2, 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* $\Omega(n^{2+\delta})$ *time.*

Proof. Assume for contradiction that there is an $\epsilon > 0$ such that Δ-Matching-Triangles can be solved 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 solved in $a_\Delta n^{3-\epsilon}$ steps. Let $\Delta(n) = \max\{\Delta : a_\Delta \leq n^{\epsilon/2} a_1, \Delta \leq n^{1/\log\log n}\}$. We claim that with this parameter $\Delta(n)$, our assumption contradicts with Theorem 1.1.

When $n \geq \max\{k^{2\log\log k}, (a_k/a_1)^{2/\epsilon}\}$, $\Delta(n) \geq k$, since $a_k \leq n^{\varepsilon/2} a_1$ and $k \leq n^{1/\log\log n}$. This shows $\Delta(n) > \omega(1)$. Also, by definition, $\Delta(n) < n^{o(1)}$. But $\Delta(n)$-Matching-Triangles can be solved in $a_{\Delta(n)} n^{3-\epsilon} \leq a_1 n^{3-\epsilon/2} = O(n^{3-\epsilon/2})$ time. It contradicts with the fact that such $\Delta(n)$-Matching-Triangles cannot be solved 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. *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* $c_\Delta = \frac{2(3-\omega)^2}{(5-\omega)\Delta+1-\omega} > 0$.

Proof. Without loss of generality, we may assume that the graph G is tripartite. Since for a general graph G, we may create a new tripartite graph with three copies of the vertex set V_1, V_2, V_3 and three copies o_1, o_2, o_3 for each color o, such that for each vertex $v \in V$ with color o, the corresponding copy $v_i \in V_i$ has color o_i. For each edge (u, v) with different colors, we add edges (u_i, v_j) for $i \neq j$. It is not hard to verify that the new tripartite graph has a triple of colors with Δ triangles iff the original graph does.

We use two different approaches to detect if there are Δ triangles with the same triple of colors based on the number of nodes in the color classes. The *size* of a color c from now on will refer to the number of nodes with color c. Also, without loss of generality, we can renumber the colors so that each color appears only in one of the partitions of G; picking the color identifies the partition.

Suppose that there is a triple of colors with Δ triangles. Let C_1, C_2, C_3 be the sizes of these three 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.

First, for each color c, we arbitrarily order the vertices of color c and associate the ith vertex in this order with the number i. Now, we will go over the possible choices of $\Delta - 1$ triples of indices (i_1, j_1, k_1), $(i_2, j_2, k_2), \ldots, (i_{\Delta-1}, j_{\Delta-1}, k_{\Delta-1})$ where

$i_r \in \{1, \ldots, C_1\}$, $j_r \in \{1, \ldots, C_2\}$, $k_r \in \{1, \ldots, C_3\}$ for all $r \in \{1, \ldots, \Delta - 1\}$. These indices are supposed to represent the nodes in the first $\Delta - 1$ of the Δ matching triangles. There are $\binom{C_1 C_2 C_3}{\Delta - 1}$ possibilities. Notice that we do not know which colors the vertices of the matching triangles are in yet, but if someone tells us the color triple of the triangles, we know exactly which vertices in the triple form the first $\Delta - 1$ triangles.

Now, we will build a graph in which we will search for the last triangle from the Δ matching triangles. For any two colors c, c' from different parts 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, the two colors c, c' cannot have those $\Delta - 1$ triangles whose indices we picked. Thus we delete *all edges* between nodes of colors c, c' 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. At this point, if we guessed the indices of the first $\Delta - 1$ matching triangles correctly, then these triangles are still in G', and if we have three colors c_1, c_2, c_3 such that there is at least one edge between each pair of colors, then this triple of colors must contain $\Delta - 1$ triangles, namely those whose indices are exactly $(i_1, j_1, k_1), (i_2, j_2, k_2), \ldots, (i_{\Delta-1}, j_{\Delta-1}, k_{\Delta-1})$, the current triples that we are attempting.

Using G', we want to search for the Δth triangle T. This last triangle T will be different from all of the first $\Delta - 1$. That is, for each of the first $\Delta - 1$ triangles, there is at least one edge that does not appear in T. We enumerate all $3^{\Delta-1}$ possibilities of which edge can be omitted from each of the $\Delta - 1$ triangles so that the graph would still contain T. For each such choice, we remove the corresponding edges from each possible pair of colors in G'. For instance, if our choice says that the first edge from the second triangle can be omitted, for every color c in the first partition and every color c' in the second partition of G', we remove the edge between the i_2th node of c and the j_2th node of c'.

If our choice of the $\Delta-1$ edges to remove from the $\Delta-1$ triangles is correct, the last triangle T would still be in G'. Moreover, if G' contains any triangle, say (p, q, r) for p with color c, q with color c', and r with color c'', then by our earlier observation, since there are edges between c, c', c'', this triple must contain (in G) the $\Delta-1$ triangles with indices (in the color classes c, c', c'') $(i_1, j_1, k_1), (i_2, j_2, k_2), \ldots, (i_{\Delta-1}, j_{\Delta-1}, k_{\Delta-1})$. Now, since (p, q, r) is a brand new triangle that differs from all of these in at least one edge, the color triple c, c', c'' must contain Δ matching triangles. This statement is an iff statement—if a color triple does contain Δ matching triangles, for some choice of edges to remove and some choice of the indices of the first $\Delta - 1$ triangles, we will have the last triangle remaining in G'.

At last we check if the remaining graph has a triangle (without color restriction) using matrix multiplication in $O(n^\omega)$ time.

The first approach correctly finds Δ matching triangles (if they exist), for all small-sized colors, in time

$$\tilde{O}\left(\binom{C_1 C_2 C_3}{\Delta - 1} 3^\Delta n^\omega\right).$$

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 by fast matrix multiplication, there must be Δ

matching triangles. We can report YES immediately. Otherwise, we can list all the triangles in G' efficiently. Björklund et al. [18] 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. We have given two approaches—one that works very well when all colors in the color triple are small, and one that works very well when all colors are large. In the following, we show how to combine these two approaches to get an efficient algorithm for a general graph 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 approach to detect if there are Δ matching triangles among Groups u, v, w. There are $\log^3 n$ such triples of groups; the algorithm runs in

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

time. The maximum value of the running time is achieved when two terms are equal (when $2^{u+v+w} = \left(n^{3-\omega}\Delta^c 3^{-\Delta}(\Delta-1)!\right)^{1/(\Delta-1+c)}$):

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

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* with parameters n, Δ, p 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 $N = O(n\Delta p)$ nodes.*

Proof. We have slightly different reductions for each one of the problems in the lemma. We start with the main construction which will be common to all of them, and then describe how to modify it in each case.

Let G be the input to Triangle-Collection* with parameters n, Δ, p as in the definition. 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 add a source node s which will dynamically represent the different colors of A. We also add a target node t_c, for every color c of C. Denote the number of nodes in H by $N = O(n\Delta p)$.

Then, 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 for each c we have a substage which we will refer to as the (a, c) substage: We 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. Note that after we complete this stage, H is encoding all the information about the edges in G between the current color a and all the nodes in C.

In the second stage, we go over all colors b of B, and for each b we perform some updates and one query as follows. This will be referred to as the (a, b) substage. First, we add 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 a query to one of our problems and figure out whether, for all colors c of C, s can reach the node t_c. Then, we undo what we did in the (a, b) substage, by removing the edges we added, and move on to the next b. After going through all colors b of B, we terminate this stage and move on to the next color a of A.

If, at some point in this process, in some substage (a, b) the query tells us that in the current graph, s does not reach all nodes t_c, then we conclude that there is a triple (a, b, c) without any triangles, and we output "yes." Otherwise, if we finish all the stages and in all queries s could reach all t_c nodes, then we output "no." Note that we have performed a total of $O(N^2)$ updates and queries in the reduction.

To see the correctness of the reduction, observe that if s cannot reach t_c in the graph corresponding to a pair (a, b), then the triple (a, b, c) contains no triangles in G. This is because any triangle $(a_j, b_{j,x}, c_{j,y})$ corresponds to a path $s \to b_{j,x} \to c_{j,y} \to t_c$.

Next, we show how to modify this graph H so that the queries that each of our problems supports is able to tell us whether there is some node t_c that s cannot reach.

(#SSR) In this problem, the query tells us the number of nodes that s can reach, while we are only interested in the number of nodes t_c that s can reach. Therefore, we make the following change to the above construction. For all colors c of C, we add edges from the node t_c back to all the nodes $c_{j,x}$, for any $j \in [\Delta], x \in [p]$. This makes it so that if s can reach t_c, then it can also reach all $\Delta \cdot p$ nodes of the form $c_{j,x}$. Then, when we ask a query, we get that s can reach exactly Δ nodes in B, plus exactly $n \cdot \Delta \cdot p$ nodes in C (that is, all of C), plus all n nodes t_c, *only if* s could reach all nodes t_c in the above construction. Otherwise, s cannot reach more than $\Delta + n \cdot (\Delta \cdot p + 1)n - 1$ nodes.

(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. The query to Max-Flow will tell us how much flow we can push from s to t. This flow will be n iff s can reach all nodes t_c for all the n colors c of C.

(#SS-Subgraph-Connectivity) In this problem, the graph is undirected, and an update can turn a node "on" or "off." A query tells us whether the graph induced on the "on" nodes is connected. Thus, we remove the direction from the edges in the above construction and replace the addition of the $c_{j,x} \to t_c$ edges by "turning on" updates on the $c_{j,x}$ nodes. All other nodes in C will be turned "off." We add permanent edges between t_c and all nodes $c_{j,x}$ for all colors c of C, and the nodes t_c are permanently turned "on." Similarly, replace the addition of $s \to b_{j,x}$ edges with updates that "turn on" the $b_{j,x}$ nodes. All other nodes in B will be turned "off."

Observe that the only way s and t_c can be in the same connected component is if we have turned "on" two nodes $b_{j,x}$ and $c_{j,y}$ that have an edge between them, and a_j is connected to both of them in G. That is, if the triple (a, b, c) contains a triangle. Thus, the query allows us to learn whether s can "reach" every t_c node.

(#Strongly Connected Components) Consider the above reduction 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 that x_B and x_C are in their own connected components, while 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 component which does not include s. ∎

Reductions to flow problems. 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* with parameters n, Δ, p and on $N = O(n\Delta p)$ nodes can be reduced in $O(N^2)$ time 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 follows.

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 $c_{j,x} \in C$ 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 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 on formulas with n variables and $m = \text{poly}(n)$ clauses cannot be solved in $O(2^{(1-\varepsilon)n})$ time—a problem for which even $2^n/\text{poly}(n)$ algorithms are not known (as opposed to the usual CNF-SAT).

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 satisfies 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 create 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 N 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 α.

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 $N \cdot (m - K + 1)$, 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 satisfies at least $m - (m - K + 1) + 1 = K$ clauses. ∎

Finally, we present reductions from Triangle-Detection to Single-Source-Max-Flow. A standard trick is to take an instance $G = (V, E)$ of Triangle-Detection and create three copies of the node set $A = B = C = V$ and create three copies $(u_A, v_B), (u_B, v_C), (u_C, v_A)$ for each edge (u, v) in E, so that it is enough to solve the following version of Triangle-Detection: given a tripartite graph (A, B, C), determine if there is a triple of nodes, one from each partition, that forms a clique. It is easy to see that the answer to the latter problem is "yes" iff there is a triangle in G. Thus, from now on, we assume that the input to Triangle-Detection is a tripartite graph of this form.

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 $T(N, M)$ time, then Triangle-Detection on a graph with n nodes and m edges can be solved in $T(O(n), O(m + n \log n))$ time.*

Proof. Given a Triangle-Detection instance (A, B, C) on $3n$ nodes and m edges, where $|A| = n$, 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 $X = \{x_1, \ldots, x_{\lceil \log n \rceil}\} \cup \{x'_1, \ldots, x'_{\lceil \log n \rceil}\}$. Connect $a \in A$ to every node $x_i \in X$ for which the ith bit in the integer $a \in [n]$ is 1, and to x'_i if the ith bit is 0. Connect $x_i \in X$ to every node $a' \in A'$ for which the ith bit in the integer $a \in [n]$ is 0, and connect x'_i to a' if the ith bit is 1. These nodes X make it so that there is always a path from a_1 to a'_2 for all $a_1 \in A$ and $a'_2 \in A'$ where $a_1 \neq a_2$, i.e., they do not correspond to the same node in the original graph, while if $a_1 = a_2$, then the only way a_1 can reach a'_2 is by going through the $B \cup C$ nodes. We could have achieved the same functionality by adding direct edges from a_1 to a'_2 for all $a_1 \neq a_2$, but this would cost us $\Omega(n^2)$ edges instead of $O(n \log n)$. Observe that a path from a in A to its copy in A' that goes through $B \cup C$ exists iff the node a is in a triangle in G. Set all of 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 $O(n^2)$ 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 $T(N, M)$ time, then Triangle-Detection on a graph on n nodes can be solved in $T(O(n), O(n^2))$ time.*

Proof. In the previous proof, remove the X layer, and instead add 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 [82], 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 [82]). 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.

PROPOSITION 3. *If Single-Source-Min-Cost-Max-Flow with* unit capacities *on a directed graph with N nodes can be solved in $O(N^{3-\varepsilon})$ time, for some $\varepsilon > 0$, then APSP on n node graphs can be solved in $O(n^{3-\delta})$ time, for some $\delta > 0$.*

Subsequent work. Since the conference version of this paper [9], there have been many exciting developments in this rapidly growing field of *Hardness in P* within the context of *Fine-Grained Complexity*. While in Appendix A we tried to survey most *previous* lower bounds under these popular conjectures, an exposition of all the lower bounds that were proven since then would require many more pages and is left for dedicated survey articles. Here, we mention some of the most related subsequent results.

Dahlgaard [33] proves that our Conjecture 1 implies interesting lower bounds for the Diameter problem. The first results are for dynamically maintaining the diameter of a graph under edge insertions or deletions. In particular, unless Conjecture 1 fails, every incremental (or decremental) algorithm that can maintain a $4/3-\varepsilon$-approximation, for $\varepsilon > 0$, for the diameter of a graph needs update time $n^{1/2-o(1)}$; a stronger lower bound of $n^{0.681-o(1)}$ holds for node insertions or deletions. Dahlgaard [33] also proves that under Conjecture 1, any (static) algorithm that can compute a $4/3-\varepsilon$-approximation to the diameter for $\varepsilon > 0$ requires $n^{1-o(1)}\sqrt{m}$ time, for every function m of n. Compare this to the prior result of Roditty and Vassilevska Williams [73]. From a more believable conjecture (Conjecture 1 versus SETH), Dahlgaard shows that a better approximation factor ($4/3$ versus $3/2$) cannot be beaten in worse time ($n^{1-\varepsilon}\sqrt{m}$ vs $n^{2-\varepsilon}$) but for all sparsities as opposed to only for $m = \tilde{O}(n)$.

Carmosino et al. [25] prove barriers for reducing SETH to APSP and 3SUM; such reductions would refute a version of SETH (NSETH), stating that SETH cannot be refuted even by co-nondeterministic algorithms. In particular, their work implies that for some problems, such as 3SUM, it might be difficult to prove a lower bound under Conjecture 1. Abboud et al. [4] take a different approach for proving lower bounds under weaker assumptions. The authors show that many SETH-based lower bounds (such as [14, 1, 22]) can instead be based on the assumption that SAT on arbitrary Boolean formulas (not necessarily CNFs) of size $2^{o(n)}$ cannot be solved in $(2-\varepsilon)^n$ time, for some $\varepsilon > 0$. Meanwhile, several works have gone in the other direction and have proved lower bounds under new and stronger assumptions related to problems such as k-Clique [2], Online Matrix Vector Multiplication [50], and Hitting Set [8]. Most recently, Krauthgamer and Trabelsi [60] proved higher lower bounds for ST-Max-Flow and All-Pairs Max-Flow that are based on the hardness of Max-SAT rather than on Conjecture 1. For ST-Max-Flow the conditional lower bound is $|S||T|m^{1-o(1)}$ and for

All-Pairs Max-Flow it is $n^{2-o(1)}m$ for $m = O(n)$.

Appendix 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 [15] and more recently by Jørgensen 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 [42, 43], there have been many papers proving the hardness of computational geometry problems, based on the 3-SUM conjecture, e.g., [35, 65, 39, 13, 29, 16]. More recently, the 3-SUM Conjecture has been used in surprising ways to show polynomial lower bounds for combinatorial problems in dynamic algorithms [70, 7, 59], graph algorithms [70, 55, 83, 5], and pattern-matching [28, 11, 12].

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 subpolynomial 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 [85]. 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 compute all pairs shortest paths (APSP) on n node graphs with edge weights in $\{-n^c, \ldots, n^c\}$, for some large constant $c \geq 1$, in $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$.

Many 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 [82, 20, 3]. In addition, many conditional lower bounds have been shown under the APSP conjecture [83, 5, 75, 7].

As remarked in the introduction, we are considering a conservative version of the conjecture (which is enough for all the above lower bounds), which states that no truly subcubic algorithm can solve APSP with weights in $\{-n^c, \ldots, n^c\}$ for all $c \geq 1$. A bolder but still plausible conjecture is that truly subcubic algorithms do not exist even when $c = 1$. We do not find the conjecture plausible for $c < 1$ since it would be false if fast enough matrix multiplication algorithms exist. If $\omega = 2 + o(1)$, then Shoshan and Zwick's algorithm for undirected graphs runs in time $O(n^{2+c+o(1)})$ [79] and Zwick's algorithm for directed graphs runs in time $O(n^{2.5+c/2+o(1)})$ [88].

The third example concerns the exact complexity of k-SAT: given a k-CNF formula on n variables and m clauses, is it satisfiable? The best upper bounds remain of the form $2^{n-o(n)}\text{poly}(m)$ when k is superconstant (e.g., [51, 69, 77, 10]). SETH of Impagliazzo, Paturi, and Zane [52, 53] states that better algorithms do not exist.

The Strong Exponential Time Hypothesis (SETH). For every $\varepsilon > 0$ there is an integer $k \geq 3$ such that k-SAT on n variables cannot be solved in $O(2^{(1-\varepsilon)n})$ time.[9]

[9] In fact, it suffices to consider the case where $m = O(n)$ [23].

Recently, many surprising SETH-based lower bounds have been shown in several different areas like graph algorithms [68, 73, 19, 3, 7], pattern matching [11, 86, 84], computational geometry [21], and exact algorithms [24, 34, 64, 32]. Moreover, it is known that refuting SETH implies new circuit lower bounds [56].

Acknowledgments. We thank Søren Dahlgaard and Ryan Williams for their helpful comments.

REFERENCES

[1] A. Abboud, A. Backurs, and V. Vassilevska Williams, *Tight hardness results for LCS and other sequence similarity measures*, in Proc. of the 56th FOCS, IEEE, Washington, DC, 2015, pp. 59–78.

[2] A. Abboud, A. Backurs, and V. V. Williams, *If the current clique algorithms are optimal, so is Valiant's parser*, in Proc. of the 56th FOCS, IEEE, Washington, DC, 2015, pp. 98–117.

[3] A. Abboud, F. Grandoni, and V. Vassilevska Williams, *Subcubic equivalences between graph centrality problems, APSP and diameter*, in Proc. of the 26th SODA, ACM, New York, SIAM, Philadelphia, 2015, pp. 1681–1697, https://doi.org/10.1137/1.9781611973730.112.

[4] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams, *Simulating branching programs with edit distance and friends: or: A polylog shaved is a lower bound made*, in Proc. of the 48th STOC, ACM, New York, 2016, pp. 375–388.

[5] A. Abboud and K. Lewi, *Exact weight subgraphs and the k-SUM conjecture*, in Proc. of the 40th ICALP, Part I, 2013, pp. 1–12.

[6] A. Abboud, K. Lewi, and R. Williams, *Losing weight by gaining edges*, in Proc. of the 22nd ESA, 2014, pp. 1–12.

[7] A. Abboud and V. Vassilevska Williams, *Popular conjectures imply strong lower bounds for dynamic problems*, in Proc. of the 55th FOCS, IEEE, Washington, DC, 2014, pp. 434–443.

[8] A. Abboud, V. Vassilevska Williams, and J. R. Wang, *Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs*, in Proc. of the 27th SODA, ACM, New York, SIAM, Philadelphia, 2016, pp. 377–391, https://doi.org/10.1137/1.9781611974331.ch28.

[9] A. Abboud, V. Vassilevska Williams, and H. Yu, *Matching triangles and basing hardness on an extremely popular conjecture*, in Proc. of the 47th STOC, ACM, New York, 2015, pp. 41–50.

[10] A. Abboud, R. Williams, and H. Yu, *More applications of the polynomial method to algorithm design*, in Proc. of the 26th SODA, ACM, New York, SIAM, Philadelphia, 2015, pp. 218–230, https://doi.org/10.1137/1.9781611973730.17.

[11] A. Abboud, V. V. Williams, and O. Weimann, *Consequences of faster alignment of sequences*, in Proc. of the 41st ICALP, Part I, 2014, pp. 39–51.

[12] A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein, *On hardness of jumbled indexing*, in Proc. of the 41st ICALP, Part I, 2014, pp. 114–125.

[13] B. Aronov and S. Har-Peled, *On approximating the depth and related problems*, SIAM J. Comput., 38 (2008), pp. 899–921, https://doi.org/10.1137/060669474.

[14] A. Backurs and P. Indyk, *Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)*, in Proc. of the 47th STOC, ACM, New York, 2015, pp. 51–58.

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

[16] G. Barequet and S. Har-Peled, *Polygon containment and translational min-Hausdorff-distance between segment sets are* 3*SUM-hard*, Internat. J. Comput. Geom. Appl., 11 (2001), pp. 465–474.

[17] M. A. Bender, J. T. Fineman, S. Gilbert, and R. E. Tarjan, *A new approach to incremental cycle detection and related problems*, ACM Trans. Algorithms, 12 (2016), 14.

[18] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick, *Listing triangles*, in Proc. of the 41st ICALP, Part I, 2014, pp. 223–234.

[19] M. Borassi, P. Crescenzi, and M. Habib, *Into the square: On the complexity of some quadratic-time solvable problems*, Electron. Notes Theoret. Comput. Sci., 322 (2016), pp. 51–67.

[20] D. Bremner, T. M. Chan, E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, M. Patrascu, and P. Taslakian, *Necklaces, convolutions, and X+Y*, Algorithmica, 69 (2014), pp. 294–314.

[21] K. Bringmann, *Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails*, in Proc. of the 55th FOCS, IEEE, Washington, DC, 2014, pp. 661–670.

[22] K. Bringmann and M. Kunnemann, *Quadratic conditional lower bounds for string problems and dynamic time warping*, in Proc. of the 56th FOCS, IEEE, Washington, DC, 2015, pp. 79–97.

[23] C. Calabro, R. Impagliazzo, and R. Paturi, *A duality between clause width and clause density for SAT*, in Proc. of the 21st CCC, 2006, pp. 252–260.

[24] C. Calabro, R. Impagliazzo, and R. Paturi, *The complexity of satisfiability of small depth circuits*, in Parameterized and Exact Computation, Springer, New York, 2009, pp. 75–85.

[25] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider, *Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility*, in Proc. of the 7th ITCS, ACM, New York, 2016, pp. 261–270.

[26] T. M. Chan, *Dynamic subgraph connectivity with geometric applications*, SIAM J. Comput., 36 (2006), pp. 681–694, https://doi.org/10.1137/S009753970343912X.

[27] T. M. Chan, M. Pătraşcu, and L. Roditty, *Dynamic connectivity: Connecting to networks and geometry*, in Proc. of the 49th FOCS, IEEE, Washington, DC, 2008, pp. 95–104.

[28] K. Chen, P. Hsu, and K. Chao, *Approximate matching for run-length encoded strings is 3SUM-hard*, in Proc. of the 20th CPM, Springer, New York, 2009, pp. 168–179.

[29] O. Cheong, A. Efrat, and S. Har-Peled, *Finding a guard that sees most and a shop that sells most*, Discrete Comput. Geom., 37 (2007), pp. 545–563.

[30] H. Y. Cheung, L. C. Lau, and K. M. Leung, *Graph connectivities, network coding, and expander graphs*, SIAM J. Comput., 42 (2013), pp. 733–751, https://doi.org/10.1137/110844970.

[31] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S. Teng, *Electrical flows, Laplacian systems, and faster approximation of Maximum Flow in undirected graphs*, in Proc. of the 43rd STOC, ACM, New York, 2011, pp. 273–282.

[32] M. Cygan, S. Kratsch, and J. Nederlof, *Fast Hamiltonicity checking via bases of perfect matchings*, in Proc. of the 45th STOC, ACM, New York, 2013, pp. 301–310.

[33] S. Dahlgaard, *On the hardness of partially dynamic graph problems and connections to diameter*, in Proc. of the 43rd ICALP, 2016, 48.

[34] E. Dantsin and A. Wolpert, *On moderately exponential time for SAT*, in Proc. of the 13th International Conference on Theory and Applications of Satisfiability Testing, 2010, pp. 313–325.

[35] M. de Berg, M. de Groot, and M. H. Overmars, *Perfect binary space partitions*, Comput. Geom. Theory Appl., 7 (1997), pp. 81–91.

[36] C. Demetrescu and G. F. Italiano, *Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier*, in Proc. of the 41st FOCS, IEEE, Washington, DC, 2000, pp. 381–389.

[37] R. Duan, *New data structures for subgraph connectivity*, in Proc. of the 37th ICALP, Part I, 2010, pp. 201–212.

[38] F. Eisenbrand and F. Grandoni, *On the complexity of fixed parameter clique and dominating set*, Theoret. Comput. Sci., 326 (2004), pp. 57–67.

[39] J. Erickson, *New lower bounds for convex hull problems in odd dimensions*, SIAM J. Comput., 28 (1999), pp. 1198–1214, https://doi.org/10.1137/S0097539797315410.

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

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

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

[43] A. Gajentaan and M. H. Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Comput. Geom., 45 (2012), pp. 140–152.

[44] F. L. Gall, *Powers of tensors and fast matrix multiplication*, in Proc. of the 39th ISSAC, 2014, pp. 296–303.

[45] A. Grønlund and S. Pettie, *Threesomes, degenerates, and love triangles*, J. ACM, 65 (2018), 22.

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

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

[48] R. Hariharan, T. Kavitha, D. Panigrahi, and A. Bhalgat, *An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs*, in Proc. of the 39th STOC, ACM, New York, 2007, pp. 605–614.
[49] M. Henzinger, S. Krinninger, and D. Nanongkai, *Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs*, in Proc. of the 46th STOC, ACM, New York, 2014, pp. 674–683.
[50] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, *Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture*, in Proc. of the 47th STOC, ACM, New York, 2015, pp. 21–30.
[51] E. A. Hirsch, *Two new upper bounds for SAT*, in Proc. of the 9th SODA, ACM, New York, SIAM, Philadelphia, 1998, pp. 521–530.
[52] R. Impagliazzo and R. Paturi, *On the complexity of k-SAT*, J. Comput. System Sci., 62 (2001), pp. 367–375.
[53] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity?*, J. Comput. System Sci., 63 (2001), pp. 512–530.
[54] G. F. Italiano, *Finding paths and deleting edges in directed acyclic graphs*, Inform. Process. Lett., 28 (1988), pp. 5–11.
[55] Z. Jafargholi and E. Viola, *3SUM, 3XOR, triangles*, Algorithmica, 74 (2016), pp. 326–343.
[56] H. Jahanjou, E. Miles, and E. Viola, *Local reductions*, in Proc. of the 42nd ICALP, Part I, 2015, pp. 749–760.
[57] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, in Proc. of the 16th STOC, ACM, New York, 1984, pp. 302–311.
[58] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, *An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations*, in Proc. of the 25th SODA, ACM, New York, SIAM, Philadelphia, 2014, pp. 217–226, https://doi.org/10.1137/1.9781611973402.16.
[59] T. Kopelowitz, S. Pettie, and E. Porat, *Higher lower bounds from the 3SUM conjecture*, in Proc. of the 26th SODA, ACM, New York, SIAM, Philadelphia, 2016, pp. 1272–1287.
[60] R. Krauthgamer and O. Trabelsi, *Conditional Lower Bounds for All-Pairs Max-Flow*, preprint, https://arxiv.org/abs/1702.05805, 2017.
[61] J. Łącki, *Improved deterministic algorithms for decremental reachability and strongly connected components*, ACM Trans. Algorithms, 9 (2013), 27.
[62] J. Łącki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, *Single source—all sinks Max Flows in planar digraphs*, in Proc. of the 53rd FOCS, IEEE, Washington, DC, 2012, pp. 599–608.
[63] Y. T. Lee and A. Sidford, *Path finding methods for linear programming: Solving linear programs in $\tilde{O}(vrank)$ iterations and faster algorithms for Maximum Flow*, in Proc. of the 55th FOCS, IEEE, Washington, DC, 2014, pp. 424–433.
[64] D. Lokshtanov, D. Marx, and S. Saurabh, *Known algorithms on graphs on bounded treewidth are probably optimal*, in Proc. of the 22nd SODA, ACM, New York, SIAM, Philadelphia, 2011, pp. 777–789, https://doi.org/10.1137/1.9781611973082.61.
[65] J. E. M. Soss and M. H. Overmars, *Preprocessing chains for fast dihedral rotations is hard or even impossible*, Comput. Geom. Theory Appl., 26 (2002), pp. 235–246.
[66] A. Madry, *Navigating central path with electrical flows: From flows to matchings, and back*, in Proc. of the 54th FOCS, IEEE, Washington, DC, 2013, pp. 253–262.
[67] Y. Nesterov and A. S. Nemirovskii, *An interior-point method for generalized linear-fractional programming*, Math. Program., 69 (1995), pp. 177–204, https://doi.org/10.1007/BF01585557.
[68] M. Pătraşcu and R. Williams, *On the possibility of faster SAT algorithms*, in Proc. of the 21st SODA, ACM, New York, SIAM, Philadelphia, 2010, pp. 1065–1075, https://doi.org/10.1137/1.9781611973075.86.
[69] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane, *An improved exponential-time algorithm for k-SAT*, J. ACM, 52 (2005), pp. 337–364.
[70] M. Pătraşcu, *Towards polynomial lower bounds for dynamic problems*, in Proc. of the 42nd STOC, ACM, New York, 2010, pp. 603–610.
[71] M. Pătraşcu and E. D. Demaine, *Logarithmic lower bounds in the cell-probe model*, SIAM J. Comput., 35 (2006), pp. 932–963, https://doi.org/10.1137/S0097539705447256.
[72] L. Roditty, *Decremental maintenance of strongly connected components*, in Proc. of the 24th SODA, ACM, New York, SIAM, Philadelphia, 2013, pp. 1143–1150.
[73] L. Roditty and V. Vassilevska Williams, *Fast approximation algorithms for the diameter and radius of sparse graphs*, in Proc. of the 45th STOC, ACM, New York, 2013, pp. 515–524.

[74] L. Roditty and U. Zwick, *Improved dynamic reachability algorithms for directed graphs*, in Proc. of the 43rd FOCS, IEEE, Washington, DC, 2002, pp. 679–689.
[75] L. Roditty and U. Zwick, *On dynamic shortest paths problems*, in Proc. of the 12th ESA, 2004, pp. 580–591.
[76] P. Sankowski, *Dynamic transitive closure via dynamic matrix inverse*, in Proc. of the 45th FOCS, IEEE, Washington, DC, 2004, pp. 509–517.
[77] U. Schöning, *A probabilistic algorithm for k-SAT and constraint satisfaction problems*, in Proc. of the 40th FOCS, IEEE, Washington, DC, 1999, pp. 410–414.
[78] J. Sherman, *Nearly maximum flows in nearly linear time*, in Proc. of the 54th FOCS, IEEE, Washington, DC, 2013, pp. 263–269.
[79] A. Shoshan and U. Zwick, *All pairs shortest paths in undirected graphs with integer weights*, in Proc. of the 40th FOCS, IEEE, Washington, DC, 1999, pp. 605–614.
[80] D. A. Spielman and S. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proc. of the 36th STOC, ACM, New York, 2004, pp. 81–90.
[81] M. Thorup, *Near-optimal fully-dynamic graph connectivity*, in Proc. of the 32nd STOC, ACM, New York, 2000, pp. 343–350.
[82] V. Vassilevska Williams and R. Williams, *Subcubic equivalences between path, matrix and triangle problems*, in Proc. of the 51st FOCS, IEEE, Washington, DC, 2010, pp. 645–654.
[83] V. Vassilevska Williams and R. Williams, *Finding, minimizing, and counting weighted subgraphs*, SIAM J. Comput., 42 (2013), pp. 831–854, https://doi.org/10.1137/09076619X.
[84] R. Williams, *A new algorithm for optimal constraint satisfaction and its implications*, in Proc. of ICALP, 2004, pp. 1227–1237.
[85] R. Williams, *Faster all-pairs shortest paths via circuit complexity*, in Proc. of the 46th STOC, ACM, New York, 2014, pp. 664–673.
[86] R. Williams and H. Yu, *Finding orthogonal vectors in discrete structures*, in Proc. of the 25th SODA, ACM, New York, SIAM, Philadelphia, 2014, pp. 1867–1877, https://doi.org/10.1137/1.9781611973402.135.
[87] V. V. Williams, *Multiplying matrices faster than Coppersmith-Winograd*, in Proc. of the 44th STOC, ACM, New York, 2012, pp. 887–898.
[88] U. Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, J. ACM, 49 (2002), pp. 289–317.