Threesomes, Degenerates, and Love Triangles

ALLAN GRØNLUND, Aarhus University
SETH PETTIE, University of Michigan

The 3SUM problem is to decide, given a set of n real numbers, whether any three sum to zero. It is widely conjectured that a trivial $O(n^2)$-time algorithm is optimal on the Real RAM, and optimal even in the nonuniform linear decision tree model. Over the years the consequences of this conjecture have been revealed. This 3SUM conjecture implies $\Omega(n^2)$ lower bounds on numerous problems in computational geometry, and a variant of the conjecture for integer inputs implies strong lower bounds on triangle enumeration, dynamic graph algorithms, and string matching data structures.

In this article, we refute the conjecture that 3SUM requires $\Omega(n^2)$ in the Real RAM and refute more forcefully the conjecture that its complexity is $\Omega(n^2)$ in the linear decision tree model. In particular, we prove that the decision tree complexity of 3SUM is $O(n^{3/2}\sqrt{\log n})$ and give two subquadratic 3SUM algorithms, a deterministic one running in $O(n^2/(\log n/\log\log n)^{2/3})$ time and a randomized one running in $O(n^2(\log\log n)^2/\log n)$ time with high probability. Our results lead directly to improved bounds on the decision tree complexity of k-variate linear degeneracy testing for all odd $k \geq 3$.

Finally, we give a subcubic algorithm for a generalization of the $(\min,+)$-product over real-valued matrices and apply it to the problem of finding zero-weight triangles in edge-weighted graphs. We give a depth-$O(n^{5/2}\sqrt{\log n})$ decision tree for this problem, as well as a deterministic algorithm running in time $O(n^3(\log\log n)^2/\log n)$.

CCS Concepts: • **Mathematics of computing → Combinatorial algorithms**; • **Theory of computation → Oracles and decision trees**;

Additional Key Words and Phrases: 3SUM, linear degeneracy testing, general position testing, triangle enumeration

ACM Reference format:
Allan Grønlund and Seth Pettie. 2018. Threesomes, Degenerates, and Love Triangles. *J. ACM* 65, 4, Article 22 (April 2018), 25 pages.
https://doi.org/10.1145/3185378

1 INTRODUCTION

Much of theoretical computer science is devoted to classifying problems according to the computational resources (time, space, etc.) needed to solve them. Whereas *upper bounding* the time complexity of problems has been quite successful (via the design and analysis of efficient algorithms), there are currently no unconditional superlinear time lower bounds in a general model of

An extended abstract of this work was presented at the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS), October 2014, in Philadelphia, PA.

This work is supported by the Danish National Research Foundation under grant DNRF84 and National Science Foundation grants CCF-1217338, CCF-1514383, and CCF-1637546.

Authors' addresses: A. Grønlund, Aarhus University, Abogade 34, Aarhus N, 8200, Denmark; email: jallan@cs.au.dk; S. Pettie, University of Michigan, 2260 Hayward St, Ann Arbor, MI, 48109; email: pettie@umich.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 ACM 0004-5411/2018/04-ART22 $15.00
https://doi.org/10.1145/3185378

computation for natural problems in **NP**.[1] As a general rule, existing lower bounds are either for restricted models of computation[2] or for general models of computation, but are *conditioned on the veracity of a plausible conjecture*. These hardness conjectures include broad complexity class separations (e.g., $\mathbf{P} \neq \mathbf{NP}$) and fine-grained conjectures on the hardness of archetypical problems such as all-pairs shortest paths on dense graphs (APSP), CNF-SAT, k-CLIQUE, or 3SUM. Let us review a few of the more popular conjectures used to prove conditional lower bounds.

3SUM Conjecture. The 3SUM problem is to decide, given a set $A \subset \mathbb{R}$ with size n, whether there exist $a, b, c \in A$ such that $a + b + c = 0$. It is conjectured that 3SUM requires $\Omega(n^2)$ time on the Real RAM and $\Omega(n^2)$ numerical operations [39, 44]. In the Integer3SUM problem, $A \subset \{-U, \ldots, U\} \subset \mathbb{Z}$, where U is the *universe* size. It is conjectured [4, 56, 64] that even when $U = n^3$, there is no Integer3SUM algorithm running in $O(n^{2-\epsilon})$ time on the $O(\log n)$-bit word RAM, for any $\epsilon > 0$.

APSP Conjecture. The all-pairs shortest path problem (APSP) on a dense n-vertex graph is equivalent [6] to computing the (min, +) product of two $n \times n$ matrices. Given $A, B \in (\mathbb{R} \cup \{\infty\})^{n \times n}$, the (min, +) product C is defined as

$$C(i, j) = \min_{0 \le k < n} \{A(i, k) + B(k, j)\}.$$

The APSP Conjecture is that there is no (min, +)-matrix multiplication algorithm taking $O(n^{3-\epsilon})$ time. It is conjectured [4, 71] to hold even on a $O(\log n)$-bit word RAM when $A, B \in (\{-n^{O(1)}, \ldots, n^{O(1)}\} \cup \{\infty\})^{n \times n}$.

Strong Exponential Time Hypothesis. This is a hypothesis about the complexity of k-CNF-SAT, which takes a CNF formula on n variables and m clauses, each containing at most k literals. Let c_k be the infimum of all c such that k-CNF-SAT can be solved in $\text{poly}(m)c^n$ time. The Exponential Time Hypothesis (ETH) states [52] that $c_3 > 1$, that is, that k-CNF-SAT requires $2^{\Omega(n)}$ time for all $k \ge 3$. The Strong ETH (SETH) states that $\lim_{k \to \infty} c_k = 2$; that is, the $O(m2^n)$-time exhaustive search for CNF-SAT is essentially optimal, for k sufficiently large.

These conjectures form the foundation upon which *numerous* conditional lower bounds are built, but exactly how solid is this foundation? Although the formal APSP and (S)ETH conjectures have been stated relatively recently [4, 26, 51, 52, 70], the APSP and CNF-SAT *problems* have been studied for decades, thereby confirming at the very least that no *obvious* refutation of the APSP or SETH conjecture exists. The 3SUM problem is different from the others in that fast 3SUM algorithms have no practical applications per se.[3] This partly explains why the 3SUM problem itself has attracted less than its fair share of attention since Gajentaan and Overmars [44] proposed it 20 years ago.

In this article, we study the 3SUM problem and a number of closely related problems such as *linear degeneracy testing* (LDT) and detecting zero-weight triangles in weighted graphs. Let us briefly review these problems before surveying their connections and the lively area of conditional lower bounds.

[1]The problems defined by Hartmanis and Stearns [47] in the proof of the *time hierarchy* theorem do have unconditional lower bounds. However, it is arguable whether these problems are "natural."

[2]For example, lower bounds for low-depth circuits [48], monotone circuits [9, 66], data structural problems in the cell-probe model [42, 57, 62], restricted decision trees [8, 39], black-box query models [1, 17], or sublinear-space-bounded computation [16].

[3]More formally, there are many reductions $\underline{\text{from}}$ 3SUM $\underline{\text{to}}$ practical data structuring problems [4, 56, 64], but only one published reduction in the reverse direction. Jafargholi and Viola [53] showed that near-linear 3SUM implies slight improvements to triangle enumeration.

k-LDT and k-SUM: Fix a k-variate linear function $\phi(x_1, \ldots, x_k) = \alpha_0 + \sum_{i=1}^{k} \alpha_i x_i$, where $\alpha_0, \ldots, \alpha_k \in \mathbb{R}$. Given a set $A \subset \mathbb{R}$, determine if $\phi(\mathbf{x}) = 0$ for any $\mathbf{x} \in A^k$. When ϕ is $\sum_{i=1}^{k} x_i$, the problem is called k-SUM.

ZeroTriangle: Given a weighted undirected graph $G = (V, E, w)$, where $w : E \to \mathbb{R}$, determine if there exists a triangle $(a, b, c) \in V^3$ for which $w(a, b) + w(b, c) + w(c, a) = 0$. (From the definition of *love : a score of zero*, one could also call this the LoveTriangle problem.)

Real RAM Model. In this article, we assume a simplified Real RAM model. Real numbers are subject to only two unit-time operations: addition and comparison. In all other respects, the machine behaves like a $w = O(\log n)$-bit word RAM with the standard repertoire of unit-time AC^0 operations: bitwise Boolean operations, left and right shifts, integer addition, and integer comparison. No multiplication operation is assumed.

1.1 Implications of 3SUM, APSP, and (S)ETH Conjectures

Gajentaan and Overmars [44] gave reductions from 3SUM to numerous problems in computational geometry. These reductions did not alter the size of the input (asymptotically), so they implied $\Omega(n^2)$ lower bounds on both the algorithmic complexity and decision tree complexity of computational geometry problems, conditioned, of course, on the validity of the 3SUM Conjecture. This matched the $O(n^2)$ time upper bounds known for several of the problems. Over the years others [7, 15, 44, 69] established more reductions from 3SUM to problems in geometry. The following is a representative list of problems reducible from 3SUM or one of its equivalent variants.[4]

— Given an n-point set in $\mathbb{R}^2$, determine whether it contains three collinear points [44].
— Given two n-edge convex polygons in $\mathbb{R}^2$, determine whether one can be placed inside the other via rotation and translation [15].
— Given a polygonal robot in $\mathbb{R}^2$ and a set of polygonal obstacles, determine if the robot can be maneuvered to a target position using rotation and translation [44].
— Given n closed triangles in $\mathbb{R}^2$, determine whether their union contains a hole, or determine the area of their union [44].
— Given a polygonal chain in $\mathbb{R}^3$, determine for each edge (u, v) whether performing a *dihedral rotation* of the suffix of the chain attached to v perpendicular to (u, v) causes a self-intersection [69].
— Given n opaque triangles in $\mathbb{R}^3$, determine whether a specific triangle is visible from a specific vantage point [44].

All the known reductions cited above are algebraic, meaning they map a 3SUM instance to a set of geometric objects, manipulating numbers using only addition and multiplication. For example, a set $A \subset \mathbb{R}$ contains a 3SUM witness if and only if the point set $P = \{(x, x^3) \mid x \in A\}$ contains three collinear points [44]. Pătraşcu [64] gave a series of intrinsically nonalgebraic[5] reductions from Integer 3SUM to the offline set disjointness problem, and from offline set disjointness to triangle enumeration and various problems in dynamic graphs. The *set disjointness problem* is, given a list of sets (S_i) over some universe and a list of disjointness queries of the form $S_i \cap S_j = \emptyset$?, to answer all queries. In more recent work, Kopelowitz, Pettie, and Porat [56] gave an improved reduction from Integer3SUM to set disjointness and new reductions from Integer 3SUM to set *intersection*,

[4]For example, the input to 3SUM is sometimes defined to be three sets A, B, $C \subset \mathbb{R}$ and the problem is to determine if there exists $a \in A$, $b \in B$, $c \in C$ such that $a + b + c = 0$. Even if there is only one set, there is sometimes an additional constraint that a, b, and c be *distinct* elements. These variants are all equivalent [44].
[5]They apply specific hash functions to the input numbers and, in general, freely inspect the representation of the integers.

where the queries are to enumerate the contents of $S_i \cap S_j$. The following lower bounds follow from the Integer 3SUM Conjecture, using reductions via set disjointness/intersection.

—Given an undirected m-edge graph, enumerating up to m triangles (3-cycles) requires at least $m^{4/3-o(1)}$ time [64]. (Björklund et al. [18] recently proved that the exponent 4/3 is optimal if the matrix multiplication exponent ω is 2.) For any $\beta \in (0, 1/2]$, given an undirected m-edge graph with arboricity[6] $\alpha = m^\beta$, enumerating up to m triangles requires $m\alpha^{1-o(1)}$ time [56]. This essentially matches the best algorithms enumerating t triangles in low arboricity graphs, which run in $O(m\alpha)$ time deterministically [34] or in $O(m + m\alpha \log\log n/\log n + t)$ time with high probability [55].
—Given a sequence of m updates to a dynamic graph (edge insertions and deletions) and two specified vertices s, t, determining whether t is reachable from s after each update requires at least $m^{4/3-o(1)}$ time in total [4].
—Given a dynamic undirected graph subject to m edge insertions and deletions, maintaining the size of the maximum cardinality matching after each update takes $m^{1/3-o(1)}$ amortized time per update [4]. If the graph is incremental (subject only to vertex and edge additions), the amortized time per update is $\max\{m^{1/3-o(1)}, n^{0.3903-o(1)}\}$ [56].
—The ZeroTriangle problem requires at least $n^{3-o(1)}$ time [71].
—A d-failure connectivity oracle [35, 36] preprocesses an undirected graph $G = (V, E)$ to handle *failure updates*, in which a set $F \subset V$ with $|F| \leq d$ is deleted, and *connectivity queries*, which determine if two given vertices are connected in the subgraph induced by $V \backslash F$. If the preprocessing time is subquadratic and deletion time is $\tilde{O}(\text{poly}(d))$, then the time per query is $d^{1/2-o(1)}$ (see also [49]), which is close to the $O(d)$ query time of [35, 36].

The Integer3SUM Conjecture has been used to get lower bounds on numerous dynamic graph problems (see [4, 56, 64]) and, more recently, on a variety of pattern matching [5, 10, 11, 25, 33] and document retrieval [56] problems.

Vassilevska Williams and Williams [70] proved that several problems are *equivalent* to APSP or (min, +)-matrix multiplication, inasmuch as a truly subcubic ($n^{3-\epsilon}$) algorithm for one would imply truly subcubic algorithms for all the others. Some of the problems include determining whether a dense graph contains a negative-weight triangle, determining whether an $n \times n$ matrix defines a metric, and computing replacement paths. More recently, it has been shown that several graph centrality measures are subcubicly equivalent to APSP [3] and that computing the *most likely parsing* of a string in a probabilistic CFG is subcubicly equivalent to APSP [68].

The (Strong) ETH of Impagliazzo, Paturi, and Zane [52] has been shown to imply lower bounds on many diverse algorithmic problems. Pătraşcu and Williams [65] proved that conditioned on ETH, no $n^{o(k)}$ algorithm exists for k-SUM, and conditioned on SETH, no $n^{k-\epsilon}$ algorithm exists for the k-dominating set. Roditty and Vassilevska Williams [67] (see also [32]) proved that no $n^{2-\epsilon}$-time algorithm exists for $(3/2 - \epsilon)$-approximating the diameter of an undirected graph with $O(n)$ edges. More recently, a number of problems with trivial quadratic-time dynamic programming algorithms were shown to be insoluble in $n^{2-\epsilon}$ time, conditioned on SETH, such as computing the Fréchet distance between two polygonal curves [22], computing the edit distance between two strings [12] even over the $\{0, 1\}$ alphabet [23], and computing the longest common subsequence of two strings [2, 23]. Abboud, Backurs, and Vassilevska Williams [2] also proved that the LCS of k strings cannot be computed in $n^{k-\epsilon}$ time. Braverman, Ko, and Weinstein [20] showed that

[6]The *arboricity* of a graph $G = (V, E)$ is the least number of forests that partition E. Nash-Williams [61] proved that the arboricity is precisely $\max_{S \subseteq V : |S| \geq 2} |E(S)|/(|S| - 1)$. The arboricity is, up to constant factors, equivalent to other measures of graph sparsity, such as the *degeneracy*, defined to be $\max_{S \subseteq V} \min_{v \in S} \deg_S(v)$.

conditioned on the ETH, no $n^{o(\log n)}$ algorithm exists for computing the best Nash equilibrium of a two-player game.

1.2 The Status of the 3SUM Conjecture

The strongest evidence in favor of the 3SUM and Integer3SUM conjectures comes from the lower bounds of Erickson [39] and a subsequent generalization due to Ailon and Chazelle [8]. They proved that any k-linear decision tree for solving k-LDT must have depth $\Omega(n^{k/2})$ when k is even and $\Omega(n^{(k+1)/2})$ when k is odd. In particular, any 3-linear decision tree for 3SUM has depth $\Omega(n^2)$. (An s-linear decision tree is one where each internal node asks for the sign of a linear expression in s input elements.) The Integer3SUM problem is obviously not *harder* than 3SUM, but no other relationship between these two problems is known. Indeed, the assumption that elements are integers opens the door to a variety of algorithmic techniques that cannot be modeled as decision trees. Using the fast Fourier transform, it is possible to solve Integer3SUM in $O(n + U \log U)$ time, which is subquadratic even for a rather large universe size U.[7] Baran, Demaine, and Pătraşcu [13] showed that Integer3SUM can be solved in $O(n^2/(\log n/\log\log n)^2)$ time (with high probability) on the word RAM, where $U = 2^w$ and $w > \log n$ is the machine word size. The algorithm uses a mixture of randomized universe reduction (via hashing), word packing, and table lookups.

The Baran et al. [13] algorithm illustrates the power of word-level parallelism, and the FFT-based algorithm highlights the importance of the universe size. Neither casts much doubt on the Integer3SUM conjecture. However, very recently Chan and Lewenstein [31] gave *truly* subquadratic algorithms for both online variants and structured variants of Integer3SUM that make use of theorems from additive combinatorics. For example, Integer3SUM on three sets A, B, C in $[O(n)]^d$ that are each monotone in all dimensions can be solved in $O(n^{2-\epsilon_d})$ time, for some $\epsilon_d = \Theta(1/d)$. Arbitrary sets $A, B, C \subset \mathbb{Z}$ can be preprocessed in $\tilde{O}(n^2)$ time such that Integer3SUM on any instance (A', B', C') with $A' \subseteq A, B' \subseteq B$, and $C' \subseteq C$ can be solved in $\tilde{O}(n^{13/7})$ time, or in $\tilde{O}(n^{1.9})$ time if only A and B are available at preprocessing time.

1.3 New Results

We give the first subquadratic bounds on both the decision tree complexity of 3SUM and the algorithmic complexity of 3SUM, which also gives the first *deterministic* subquadratic algorithm for Integer3SUM. Our method leads to similar improvements to the decision tree complexity of k-LDT when $k \geq 3$ is odd. Refer to Figure 1 for a summary of prior work and our results.

Theorem 1.1. *There is a 4-linear decision tree for 3SUM with depth $O(n^{3/2}\sqrt{\log n})$. Furthermore, 3SUM can be solved deterministically in $O(n^2/(\log n/\log\log n)^{2/3})$ time and, using randomization, in $O(n^2(\log\log n)^2/\log n)$ time with high probability.*

Theorem 1.2. *When $k \geq 3$ is odd, there is a $(2k-2)$-linear decision tree for k-LDT with depth $O(n^{k/2}\sqrt{\log n})$,*

Theorem 1.1 refutes the 3SUM conjecture and casts serious doubts on the optimality of many $O(n^2)$-time Real RAM algorithms for problems in computational geometry [44]; see Section 1.1. Theorem 1.1 also answers a question of Erickson [39] and Ailon and Chazelle [8] about whether $(k+1)$-linear decision trees are more powerful than k-linear decision trees in solving k-LDT problems. In the case of $k = 3$, they are.

We define a new product of three real-valued matrices called *target-min-plus*, which is trivially computable in $O(n^3)$ time. We observe that ZeroTriangle is reducible to a target-min-plus product,

[7]Erickson [39] credits R. Seidel with this 3SUM algorithm.

Table 1. A Summary of the New Results

3SUM		
trivial	n^2	
new	$n^{3/2}\sqrt{\log n}$	DEC. TREE
	$n^2/\left(\frac{\log n}{\log\log n}\right)^{2/3}$	
	$n^2/\frac{\log n}{(\log\log n)^2}$	RAND.

Integer3SUM		
trivial	n^2	
Seidel 1997	$n + U\log U$	
Baran, Demaine,	$n^2/\left(\frac{\log n}{\log\log n}\right)^2$	RAND.
Pătraşcu 2005	$n^2/\frac{w}{\log^2 w}$	RAND.
new	$n^2/\left(\frac{\log n}{\log\log n}\right)^{2/3}$	

Convolution3SUM		
trivial	n^2	
new	$n^{3/2}\sqrt{\log n}$	DEC. TREE
	$n^{3/2}$	RAND., DEC. TREE
	$n^2/\frac{\log n}{(\log\log n)^2}$	
	$n^2/\frac{\log n}{\log\log n}$	RAND.

ZeroTriangle		
trivial	n^3	
new	$n^{5/2}\sqrt{\log n}$	DEC. TREE
	$n^{5/2}$	RAND., DEC. TREE
	$n^3/\frac{\log n}{(\log\log n)^2}$	
	$n^3/\frac{\log n}{\log\log n}$	RAND.
	$m^{5/4}\sqrt{\log m}$	DEC. TREE
	$m^{5/4}$	RAND., DEC. TREE
	$m^{3/2}/\left(\frac{\log m}{(\log\log m)^2}\right)^{1/4}$	
	$m^{3/2}/\left(\frac{\log m}{\log\log m}\right)^{1/4}$	RAND.

Results in the decision tree model are Indicated by DEC. TREE; results using randomization are indicated by RAND. In ZeroTriangle, n and m are the number of vertices and edges, whereas in all other problems n is the length of the input. In Integer3SUM, $w = \Omega(\log n)$ is the machine word size and $U \leq 2^w$ the size of the universe.

then give subcubic bounds on the decision tree and algorithmic complexity of target-min-plus. Theorem 1.3 is an immediate consequence.

THEOREM 1.3. *The decision tree complexity of ZeroTriangle is $O(n^{5/2}\sqrt{\log n})$ on an n-vertex graph and its randomized decision tree complexity is $O(n^{5/2})$ with high probability. There is a deterministic ZeroTriangle algorithm running in $O(n^3(\log\log n)^2/\log n)$ time and a randomized algorithm running in $O(n^3\log\log n/\log n)$ time with high probability.*

Any m-edge graph contains $O(m^{3/2})$ triangles that can be enumerated in $O(m^{3/2})$ time, so ZeroTriangle can clearly be solved in $O(m^{3/2})$ time as well. We improve this bound for all m.

THEOREM 1.4. *The decision tree complexity of ZeroTriangle on an m-edge graph is $O(m^{5/4}\sqrt{\log m})$ and, using randomization, $O(m^{5/4})$ with high probability. The ZeroTriangle problem can be solved in $O(m^{3/2}((\log\log m)^2/\log m)^{1/4})$ time deterministically or $O(m^{3/2}(\log\log m/\log m)^{1/4})$ with high probability.*

Pătraşcu [64] introduced a problem called Convolution3SUM and used it as a stepping stone in his reductions from Integer3SUM to triangle enumeration and dynamic graph problems. Subsequent reductions [10, 71] work directly from Convolution3SUM. Here we distinguish the versions of this problem over reals and integers.

Convolution3SUM: Given a vector $A \in \mathbb{R}^n$, determine if there exist i, j for which $A(i) + A(j) = A(i + j)$.

IntegerConv3SUM: The same as Convolution3SUM, except that $A \in \{0, \ldots, U - 1\}^n$ and $U \leq 2^w$, where $w = \Omega(\log n)$ is the machine word size.

By invoking the Vassilevska Williams and Williams reduction [71] from Convolution3SUM to ZeroTriangle, Theorem 1.3 implies subquadratic bounds on the complexity of Convolution3SUM. By designing Convolution3SUM algorithms from scratch, we can obtain speedups comparable to those of Theorem 1.3.

Theorem 1.5. *The decision tree complexity of Convolution3SUM is $O(n^{3/2}\sqrt{\log n})$, and its randomized decision tree complexity is $O(n^{3/2})$ with high probability. The Convolution3SUM problem can be solved in $O(n^2(\log\log n)^2/\log n)$ time deterministically or in $O(n^2 \log\log n/\log n)$ time with high probability.*

1.4 Recent Developments

After the initial publication of this work [46], there have been many interesting developments on 3SUM and related problems.

- Freund [43] and Gold and Sharir [45] independently improved our algorithms to run in $O(n^2 \log\log n/\log n)$ deterministically. Using a different approach, Chan [30] developed a deterministic 3SUM algorithm running in time $O(n^2(\log\log n)^{O(1)}/\log^2 n)$.
- Kane, Lovett, and Moran [54] proved that the decision tree complexity of 3SUM is $O(n \log^2 n)$ and gave near-optimal decision trees for other problems such as sorting $X + Y$ and ZeroTriangle.
- Lincoln et al. [58] proved that existing 3SUM algorithms (ours as well as [30, 43, 45]) can be implemented to use just $\tilde{O}(\sqrt{n})$ read/write memory.
- Nondeterministically, positive 3SUM instances can be trivially confirmed in $O(1)$ time. Carmosino et al. [27] proved that *negative* Integer3SUM instances can be confirmed in $\tilde{O}(n^{3/2})$ time. They proved that this result precludes the possibility that SETH could be shown to imply $n^{2-o(1)}$-hardness of Integer3SUM.
- Barba et al. [14] considered a polynomial variant of 3SUM, in which the criterion $x + y + z = 0$ is replaced by $f(x, y, z) = 0$ for some constant-degree polynomial f. They proved that this problem has an $O(n^{12/7+\epsilon})$-depth algebraic decision tree and that it can be solved in the Real RAM in $O(n^2/\text{polylog}(n))$ time. They also showed that general position testing (GPT) of n points in $\mathbb{R}^2$ (testing whether any three points lie on a line) can be solved in $O(n^2/\text{polylog}(n))$ time, assuming the n input points lie on at most $(\log n)^{1/6-\epsilon}$ constant-degree polynomial curves.
- Improving on [64], Kopelowitz, Pettie, and Porat [56] showed that any $\Omega(n^{3/2+\epsilon})$-lower bound on Integer3SUM implies polynomial lower bounds on various problems such as triangle enumeration and offline set disjointness.

1.5 An Overview

All of our algorithms borrow liberally from Fredman's 1976 articles on the decision tree complexity of (min, +)-matrix multiplication [41] and the complexity of sorting $X + Y$ [40]. Throughout the article, we shall refer to the ingenious observation that $a + b < c + d$ if and only if $a - c < d - b$ as *Fredman's trick*. In order to shave off poly$(\log n)$ factors in runtime, we apply the bichromatic domination technique invented by Chan [28] and developed further by Bremner, Chan, Demaine, Erickson, Hurtado, Iacono, Langerman, Pătraşcu, and Taslakian [21].

In Section 2, we review a number of useful lemmas due to Fredman [40], Buck [24], and Chan [28] about sorting with partial information, the complexity of hyperplane arrangements, and the complexity of dominance reporting in $\mathbb{R}^d$. In Section 3, we review a standard $O(n^2)$-time 3SUM algorithm, and in Section 4, we present an $\tilde{O}(n^{3/2})$-depth decision tree for 3SUM. Subquadratic algorithms for 3SUM are presented in Section 5. Section 6 presents new bounds on the decision tree complexity of k-LDT for odd $k \geq 3$. Section 7 presents new bounds on the decision tree and algorithmic complexity of ZeroTriangle and Convolution3SUM. Section 8 concludes with some open problems.

2 USEFUL LEMMAS

Fredman [40] considered the problem of sorting a list of n numbers known to be arranged in one of $\Pi \leq n!$ permutations. When Π is sufficiently small, the list can be sorted using a linear number of comparisons.

Lemma 2.1 (Fredman [40]). *A list of n numbers whose sorted order is one of Π permutations can be sorted with $2n + \log \Pi$ pairwise comparisons.*

Throughout the article, $[N]$ denotes the first $\lceil N \rceil$ natural numbers $\{0, \ldots, \lceil N \rceil - 1\}$, where N may or may not be an integer. We apply Lemma 2.1 to the problem of sorting *Cartesian sums*. Given lists $A = (a_i)_{i \in [n]}$ and $B = (b_i)_{i \in [n]}$ of distinct numbers, define $A + B = \{a_i + b_j \mid i, j \in [n]\}$. We often regard $A + B$ as an $|A| \times |B|$ matrix (which may contain multiple copies of the same number) or as the point $(a_1, \ldots, a_n, b_1, \ldots, b_n)$ in the space $\mathbb{R}^{2n}$. Let the coordinates of $\mathbb{R}^{2n}$ be called $x_1, \ldots, x_n, y_1, \ldots, y_n$. The points in $\mathbb{R}^{2n}$ that agree with a fixed permutation of $A + B$ form a convex cone bounded by the $\binom{n^2}{2}$ hyperplanes $H = \{x_i + y_j - x_k - y_l \mid i, j, k, l \in [n] \text{ and } (i, j) \neq (k, l)\}$. Each hyperplane corresponds to a comparison between two elements. The sorted order of $A + B$ is encoded as a sign vector $\{-1, 0, 1\}^{\binom{n^2}{2}}$ depending on whether (A, B) lies on, above, or below a particular hyperplane in H. Therefore, the number of possible sorted orders of $A + B$ is exactly the number of regions (of all dimensions) defined by the arrangement H. (Regions of dimension less than $2n$ correspond to instances in which some numbers appear multiple times.)

Lemma 2.2 (Buck [24]). *Consider the partition of space defined by an arrangement of m hyperplanes in $\mathbb{R}^d$. The number of regions of dimension $k \leq d$ is at most*

$$\binom{m}{d-k}\left(\binom{m-d+k}{0} + \binom{m-d+k}{1} + \cdots + \binom{m-d+k}{k}\right),$$

and the number of regions of all dimensions is $O(m^d)$.

In one of our algorithms, we will construct the hyperplane arrangement explicitly. Edelsbrunner, O'Rourke, and Seidel [38] proved that the natural incremental algorithm takes $O(m^d)$ time (linear in the size of the arrangement), but any trivial $m^{O(d)}$-time algorithm suffices in our application. The hyperplane arrangements we use correspond to fragments of the Cartesian sum $A + B$. Lemma 2.3 is a direct consequence of Lemmas 2.1 and 2.2.

Lemma 2.3. *Let $A = (a_i)_{i \in [n]}$ and $B = (b_i)_{i \in [n]}$ be two lists of numbers and let $F \subseteq [n]^2$ be a set of positions in the $n \times n$ grid. The number of realizable orders of $(A + B)_{|F} \overset{\text{def}}{=} \{a_i + b_j \mid (i, j) \in F\}$ is $O(\binom{|F|}{2}^{2n})$ and therefore $(A + B)_{|F}$ can be sorted with at most $2|F| + 4n \log |F| + O(1)$ comparisons.*

Proof. Let $H_{|F}$ be the set of $m = \binom{|F|}{2}$ hyperplanes corresponding to comparisons between elements at positions in F. The number of realizable orders of $(A+B)_{|F}$ is the number of regions of $\mathbb{R}^{2n}$ defined by $H_{|F}$, which, by Lemma 2.2, is $\Pi = O(m^{2n})$. By Lemma 2.1, $(A+B)_{|F}$ can be sorted with $2|F| + \log \Pi = 2|F| + 2n \log m + O(1) = 2|F| + 4n \log |F| + O(1)$ comparisons. □

It is sometimes convenient to assume that the elements of a Cartesian sum are distinct (and therefore have exactly one sorted order), even though numbers may appear multiple times. Lemma 2.4 illustrates one way to break ties consistently. The proof is straightforward.

Lemma 2.4. *Let $A = (a_i)$ and $B = (b_i)$ be two lists of numbers. Define $a'_i = (a_i, i, 0)$ and $b'_j = (b_j, 0, j)$. The Cartesian sum $A' + B'$ is totally ordered and is a linear extension of the partially ordered $A + B$. (Addition over tuples is pointwise addition; tuples are ordered lexicographically. The tuple (u, v, w) can be regarded as a representation of a real number $u + \epsilon_1 v + \epsilon_2 w$, where $\epsilon_1 \gg \epsilon_2$ are sufficiently small so as not to invert strictly ordered elements of $A + B$.)*

Given a list $P = (p_0, \ldots, p_{n-1})$ of points in $\mathbb{R}^d$, each colored red or blue, the *bichromatic dominating pairs* problem is to enumerate every pair $(i, j) \in [n]^2$ such that p_i is red, p_j is blue, and p_i is greater than p_j at each of the d coordinates.[8] A natural divide-and-conquer algorithm [63, p. 366] runs in time linear in the output size and $O(n \log^d n)$. Chan [28] provided an improved analysis when d is logarithmic in n. For the sake of completeness, we give a short proof of Lemma 2.5 in Appendix A.

Lemma 2.5 (Bichromatic Dominance Reporting [28]). *Given a list P of n red and blue points in $\mathbb{R}^d$, it is possible to return the indices of all bichromatic dominating pairs in time linear in the output size and $c_\epsilon^d |P|^{1+\epsilon}$. Here $\epsilon \in (0, 1)$ is arbitrary and $c_\epsilon = 2^\epsilon/(2^\epsilon - 1)$.*

We typically invoke Lemma 2.5 with $\epsilon = 1/2, c_\epsilon \approx 3.42$, and $d = \delta \log n$, where $\delta > 0$ is sufficiently small to make the running time subquadratic, excluding the time allotted to reporting the output. Chan [29] (see also [50]) recently proved that bichromatic dominating pairs could be solved in subquadratic time for dimensions up to $d = O(\log^2 n/(\log \log n)^3)$. Unfortunately, due to other bottlenecks in our algorithm, Chan's [29] new algorithm does not improve our results.

3 THE QUADRATIC 3SUM ALGORITHM

We shall review a standard $O(n^2)$ algorithm for the three-set version of 3SUM and introduce some terminology used in Sections 4 and 5. We are given sets $A, B, C \subset \mathbb{R}$ and must determine if there exists $a \in A, b \in B, c \in C$ such that $a + b + c = 0$. For each $c \in C$, the algorithm searches for $-c$ in the Cartesian sum $A + B$. Each search takes $O(|A| + |B|)$ time, for a total of $O(|C|(|A| + |B|))$. We view $A + B$ as being a matrix whose rows correspond to A and columns correspond to B, both listed in increasing order.

```
1       Sort $A$ and $B$ in increasing order as $A(0), \ldots, A(|A|-1)$ and $B(0), \ldots, B(|B|-1)$.
2       For each $c \in C$,
2.1       Initialize lo ← 0 and hi ← $|B| - 1$.
2.2       Repeat:
2.2.1       If $-c = A(\text{lo}) + B(\text{hi})$, report witness "$(A(\text{lo}), B(\text{hi}), c)$"
2.2.2       If $-c < A(\text{lo}) + B(\text{hi})$, then decrement hi, otherwise increment lo.
2.3       Until lo = $|A|$ or hi = $-1$.
3       If no witnesses were found report "no witness."
```

[8]It is crucial that the output be a list of index pairs (i, j) rather than point pairs (p_i, p_j) because we cannot afford the latter. In general, $d = \omega(1)$ is not constant, but some slowly growing function of n.

Note that when a witness is discovered in Step 2.2.1, the algorithm continues to search for more witnesses involving c. Since the elements in each row and each column of the $A+B$ matrix are distinct, it does not matter whether we increment lo or decrement hi after finding a witness. We choose to increment lo in such situations; this choice is reflected in Lemma 3.1 and its applications in Sections 5.2.1 and 5.2.2.

Define the *contour* of x, $\textsc{contour}(x, A+B)$, to be the sequence of positions $(\mathrm{lo}, \mathrm{hi})$ encountered while searching for x in $A+B$. When $A+B$ is understood from context, we will write it as $\textsc{contour}(x)$. If $A+B$ is viewed as a topographic map, with the lowest point in the NW corner $(0,0)$ and highest point in the SE corner $(|A|-1, |B|-1)$, $\textsc{contour}(x)$ represents the path taken by an agent attempting to stay as close to altitude x as possible, starting in the NE corner $(0, |B|-1)$ and ending when it falls off the western or southern side of the map. Lemma 3.1 is straightforward.

Lemma 3.1. *Every occurrence of x in $A+B$ lies on $\textsc{contour}(x)$. Let $y = (A+B)(i,j)$ be any element of $A+B$. Then $y > x$ if and only if either (i,j) lies strictly below $\textsc{contour}(x)$ or both (i,j) and $(i, j-1)$ lie on $\textsc{contour}(x)$. Similarly, $y \le x$ if and only if either (i,j) lies strictly above $\textsc{contour}(x)$ or both (i,j) and $(i+1, j)$ lie on $\textsc{contour}(x)$.*

4 A SUBQUADRATIC 3SUM DECISION TREE

Recall that we are given a set $A \subset \mathbb{R}$ of reals and must determine if there exist $a, b, c \in A$ summing to zero. We first state the algorithm, then establish its correctness and efficiency.

```
1       Sort $A$ in increasing order as $A(0), \ldots, A(n-1)$. Partition $A$ into $\lceil n/g \rceil$ groups
        $A_0, \ldots, A_{\lceil n/g \rceil - 1}$ of size at most $g$, where $A_i \overset{\text{def}}{=} \{A(ig), \ldots, A((i+1)g-1)\}$ and $A_{\lceil n/g \rceil - 1}$
        may be smaller. The first and last elements of $A_i$ are $\min(A_i) = A(ig)$ and $\max(A_i) =$
        $A((i+1)g-1)$.
2       Sort $D \overset{\text{def}}{=} \bigcup_{i \in [n/g]} (A_i - A_i) = \{a - a' \mid a, a' \in A_i \text{ for some } i\}$.
3       For all $i, j \in [n/g]$, sort $A_{i,j} \overset{\text{def}}{=} A_i + A_j = \{a + b \mid a \in A_i \text{ and } b \in A_j\}$.
4       For $k$ from $0$ to $n-1$,
4.1       Initialize lo $\leftarrow 0$ and hi $\leftarrow \lfloor k/g \rfloor$ to be the group index of $A(k)$.
4.2       Repeat:
4.2.1       If $-A(k) \in A_{\mathrm{lo},\mathrm{hi}}$, report "solution found" and halt.
4.2.2       If $\max(A_{\mathrm{lo}}) + \min(A_{\mathrm{hi}}) > -A(k)$, then decrement hi; otherwise, increment lo.
4.3       Until hi $<$ lo.
5       Report "no solution" and halt.
```

With appropriate modifications, this algorithm also solves the three-set version of 3SUM, where the input is $A, B, C \subset \mathbb{R}$.

Efficiency of the Algorithm. Step 1 requires $n \log n$ comparisons. By Lemma 2.3, Step 2 requires $O(n \log n + |D|) = O(n \log n + gn)$ comparisons to sort D. Using Fredman's trick, Step 3 requires no comparisons at all, given the sorted order on D. Note that if $a, a' \in A_i$ and $b, b' \in A_j$, $a + b < a' + b'$ holds if and only if $a - a' < b' - b$. For each iteration of the outer loop (Step 4), there are at most $\lceil n/g \rceil$ iterations of the inner loop (Step 4.2) since each iteration ends by either incrementing lo or decrementing hi. In Step 4.2.1, we can determine whether $-A(k)$ is in $A_{\mathrm{lo},\mathrm{hi}}$ with a binary search, in $\log |A_{\mathrm{lo},\mathrm{hi}}| = \log(g^2)$ comparisons. In total, the number of comparisons is on the order of $n \log n + gn + (n^2 \log g)/g$, which is $O(n^{3/2}\sqrt{\log n})$ when $g = \sqrt{n \log n}$.

Correctness of the Algorithm. The purpose of the outer loop (Step 4) is to find $a, b \in A$, for which $a, b \leq A(k)$ and $a + b + A(k) = 0$. This is tantamount to finding indices lo, hi for which $a \in A_{\text{lo}}, b \in A_{\text{hi}}$, and $-A(k) \in A_{\text{lo,hi}}$. We maintain the loop invariant that if there exist a, b for which $a + b + A(k) = 0$, then both of a and b lie in $A_{\text{lo}}, A_{\text{lo}+1}, \ldots, A_{\text{hi}}$. Suppose the algorithm has not halted in Step 4.2.1; that is, there are no solutions with $a \in A_{\text{lo}}, b \in A_{\text{hi}}$. If $\max(A_{\text{lo}}) + \min(A_{\text{hi}}) > -A(k)$, then there can clearly be no solutions with $b \in A_{\text{hi}}$ since $b \geq \min(A_{\text{hi}})$, so decrementing hi preserves the invariant. Similarly, if $\max(A_{\text{lo}}) + \min(A_{\text{hi}}) < -A(k)$, then there can be no solutions with $a \in A_{\text{lo}}$ since $a \leq \max(A_{\text{lo}})$, so incrementing lo preserves the invariant. If it is ever the case that hi < lo, then, by the invariant, no solutions involving $A(k)$ exist.

Algorithmic Implementation. This 3SUM algorithm can be implemented to run in $O(n^2 \log n)$ time while performing only $O(n^{3/2} \log n)$ comparisons. Using any optimal sorting algorithm, Steps 1–3 can be executed in $O(gn \log(gn) + (n/g)^2 \cdot g^2 \log g) = O(n^2 \log n)$ time while using $O(gn \log(gn))$ comparisons. (Whenever the sorting algorithm wants to test whether $a + b < a' + b'$, we determine the outcome by consulting the positions of $a - a'$ and $b' - b$ in the sorted order on D.) Now the boxes $\{A_{i,j}\}$ have been explicitly sorted, so the binary searches in Step 4.2.1 can be executed in $O(\log g)$ time per search. The total running time is $O(n^2 \log n)$ and the number of comparisons is now minimized when $g = \sqrt{n}$, for a total of $O(n^{3/2} \log n)$ comparisons. We do not know of any polynomial-time 3SUM algorithm that performs $O(n^{3/2}\sqrt{\log n})$ comparisons.

5 SOME SUBQUADRATIC 3SUM ALGORITHMS

In our 3SUM decision tree, sorting D (Step 2) is a comparison-efficient way to accomplish Step 3, but it only lets us *deduce* the sorted order of the boxes $\{A_{i,j}\}$. It does not give us a useful representation of these sorted orders, namely, one that lets us implement each comparison of the binary search in Step 4.2.1 in $O(1)$ time. In this section, we present several methods for sorting the boxes based on bichromatic dominating pairs, as in Chan [28] and Bremner et al. [21]; see Lemma 2.5. The total time spent performing binary searches in Step 4.2.1 is $O((n^2 \log g)/g)$, so our goal is to make g as large as possible, provided that the cost of sorting the boxes is of a lesser order.

Overview. As a warmup we give, in Section 5.1, a relatively simple subquadratic 3SUM algorithm running in $O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$ time. In Section 5.2, we present a more sophisticated algorithm, some of whose parameters can be selected either deterministically or randomly. Sections 5.2.1 and 5.2.2 give two parameterizations of the algorithm, which lead to an $O(n^2(\log\log n/\log n)^{2/3})$-time deterministic 3SUM algorithm and $O(n^2(\log\log n)^2/\log n)$-time randomized 3SUM algorithm.

5.1 A Simple Subquadratic 3SUM Algorithm

First, let's see a fairly simple $O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$-time 3SUM algorithm that introduces the overall approach we use for all our Real RAM algorithms. Given a box size $g = \frac{1}{2}\sqrt{\log n/\log\log n}$, the algorithm enumerates *every* permutation $\pi : [g^2] \to [g]^2$, where $\pi = (\pi_{\text{r}}, \pi_{\text{c}})$ is decomposed into row and column functions $\pi_{\text{r}}, \pi_{\text{c}} : [g^2] \to [g]$. By definition, π is the correct sorting permutation of $A_{i,j}$ if and only if $A_{i,j}(\pi(t)) < A_{i,j}(\pi(t+1))$ for all $t \in [g^2 - 1]$.[9] Since $A_{i,j} = A_i + A_j$, this inequality can also be written $A_i(\pi_{\text{r}}(t)) + A_j(\pi_{\text{c}}(t)) < A_i(\pi_{\text{r}}(t+1)) + A_j(\pi_{\text{c}}(t+1))$. By Fredman's trick, this is equivalent to saying that the (red) point p_j dominates the

[9]Without loss of generality we can assume $A_{i,j}$ is totally ordered. See Lemma 2.4.

(blue) point q_i, where

$$p_j = \Big(A_j(\pi_c(1)) - A_j(\pi_c(0)), \ldots, A_j(\pi_c(g^2-1)) - A_j(\pi_c(g^2-2))\Big)$$
$$q_i = \Big(A_i(\pi_r(0)) - A_i(\pi_r(1)), \ldots, A_i(\pi_r(g^2-2)) - A_i(\pi_r(g^2-1))\Big).$$

We find all such dominating pairs. Each dominating pair found gives the sorting permutation of the corresponding $g \times g$ box and for each box we simply represent the permutation as the enumeration index. This index is bounded by $(g^2)!$, requires at most $\frac{1}{4}\log n$ bits, and fits in a machine word. By Lemma 2.5, the time to report red/blue dominating pairs, over all $(g^2)!$ invocations of the procedure, is $O((g^2)!c_\epsilon^{g^2-1}(2n/g)^{1+\epsilon} + (n/g)^2)$, the last term being the total size of the outputs. With $\epsilon = 1/2$ and $g = \frac{1}{2}\sqrt{\log n/\log\log n}$, the first term is negligible.

By the choice of g, $\log(g^2!) \le \frac{1}{4}\log n$ bits suffice to encode a permutation and $\log(g^2) \le \log\log n$ bits suffice to encode a rank $r \in [g^2]$. We build, in $o(n)$ time, a lookup table indexed by tuples (π, r) that contains the location in $[g]^2$ of the element with rank r in a $g \times g$ box, sorted according to π. Using this lookup table, it is straightforward to perform a binary search for x in $A_{i,j}$, in $O(\log(g^2)) = O(\log g)$ time.

The total running time is therefore $O((n/g)^2)$ for dominance reporting and $O((n^2\log g)/g) = O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$ for the binary searches in Step 4.2.1. Since there are at most g^{8g} realizable permutations of $A_{i,j}$, not $(g^2)!$ (see Lemma 2.3 and Fredman [40]), we could possibly shave off another $\sqrt{\log\log n}$ factor with a box size of $g = \Theta(\sqrt{\log n})$. However, with a bit more work, it is possible to save poly$(\log n)$ factors, as we now show.

5.2 Faster 3SUM Algorithms

To improve the running time of the simple algorithm, we must sort larger boxes. However, enumerating all permutations of g^2 elements is infeasible for $g = \omega(\sqrt{\log n})$. Instead, our approach is to partition each box into ordered layers and then sort each layer separately. So long as each layer has size $c\log n$ for a small enough constant c, the cost of applying red/blue dominance reporting for sorting will be negligible. The main difficulty we face is that the boundaries between the layers we create are input dependent and different for each box. First, we show how to construct and sort layers of all $g \times g$ boxes given a set of grid points to define the layers. These special grid points can be selected either randomly or deterministically. Section 5.2.1 analyzes a natural randomized method and Section 5.2.2 gives a deterministic variant. These algorithms establish the remaining claims of Theorem 1.1.

Let $P \subset [g]^2$ be an input set of p positions in the $g \times g$ grid that includes positions $(0,0)$ and $(g-1, g-1)$. For each $(l,m) \in P$, consider $\textsc{contour}(A_{i,j}(l,m), A_{i,j})$, that is, the path in $[g]^2$ taken by the standard 3SUM algorithm of Section 3 when searching for $A_{i,j}(l,m)$ inside $A_{i,j}$. Clearly $\textsc{contour}(A_{i,j}(l,m))$ goes through position (l,m). For any two $(l,m),(l',m') \in P$, $\textsc{contour}(A_{i,j}(l,m))$ and $\textsc{contour}(A_{i,j}(l',m'))$ may intersect in several places (see Figure 1), though they never cross. According to Lemma 3.1, the two contours define a *tripartition* (R,S,T) of the positions of $[g]^2$ into three regions, where

$$A_{i,j}(R) \subset (-\infty,\ A_{i,j}(l,m)]$$
$$A_{i,j}(S) \subset (A_{i,j}(l,m),\ A_{i,j}(l',m'))$$
$$A_{i,j}(T) \subset [A_{i,j}(l',m'),\ \infty).$$

250	272	362	368	372	385	416	546	549	606
289	311	401	407	411	424	455	585	588	645
299	321	411	417	421	434	465	595	598	655
311	333	423	429	433	**446**	477	607	610	667
325	347	437	443	447	460	491	621	624	681
331	353	443	449	453	466	497	627	630	687
363	385	475	481	485	498	529	659	662	719
384	406	496	502	506	519	550	680	683	740
412	434	524	530	534	547	**578**	708	711	768
415	437	527	533	537	550	581	711	714	771

Fig. 1. A tripartition of a block defined by two contours (search paths). The contours of the elements at positions (3,5) and (8,6) are blue and red, respectively, and their intersection is green.

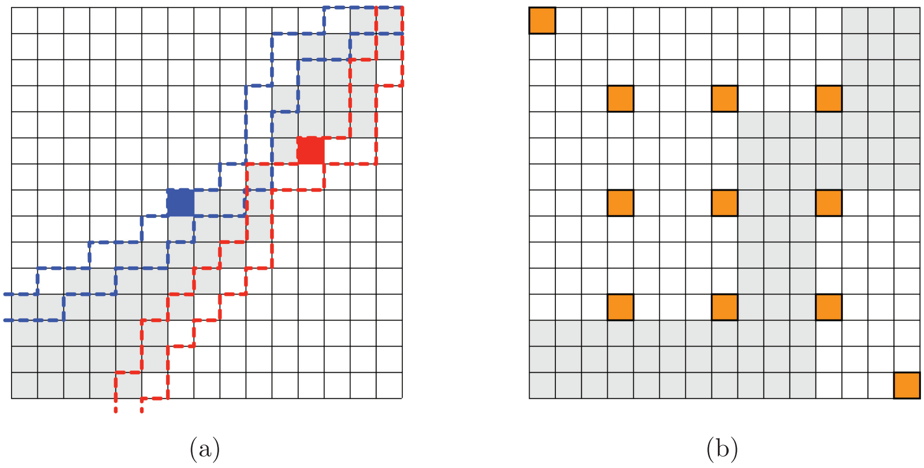

Fig. 2. Illustrations of tripartitions defined by two contours in a $[15] \times [15]$ grid. Left: the blue and red locations are in P. Two possible contours are indicated by blue and red paths. They define a tripartition (R, S, T) with S marked in gray. Right: P is chosen to include two corner locations and an evenly spaced $q \times q$ grid. Any tripartition (R, S, T) defined by a legal pair of contours has $P \cap S = \emptyset$, implying that $|S| \leq 2g^2/(q+1)$. An example of an S nearly achieving that size is marked in gray.

Here $A_{i,j}(X) = \{A_{i,j}(x) \mid x \in X\}$ for a subset $X \subseteq [g]^2$. Note that (R, S, T) is fully determined by the shapes of the contours, not the specific contents of $A_{i,j}$.[10] See Figure 1 for an example with actual numbers or Figure 2(a) for a schematic illustration. Layers are positions between contours, and the contours act as borders between layers that are ordered by construction. Now we show how to find the contours and sort the layers of each box using red/blue dominance reporting.

A contour is defined by at most $2g - 1$ comparisons between the element searched for and the elements of the box. Suppose that $\tau = (\tau_r, \tau_c)$ is purported to be $\textsc{contour}(A_{i,j}(l, m))$; that is, $\tau(0) = (\tau_r(0), \tau_c(0)) = (0, g-1)$ is the starting position of (lo, hi) and $\tau(t+1) \in \tau(t) + \{(1,0), (0,-1)\}$ depending on whether lo is incremented or hi is decremented after the $(t+1)$th comparison. The contour ends at the first $t^\star$, for which $\tau(t^\star) = (g, \cdot)$ or $(\cdot, -1)$ depending on whether the search for $A_{i,j}(l, m)$ falls off the southern or western boundary of $A_{i,j}$. Clearly τ is the correct contour if and

[10]We continue to assume that ties are broken to make $A_{i,j}$ totally ordered. Refer to Lemma 2.4.

only if

$$\begin{aligned} A_{i,j}(l,m) &< A_{i,j}(\tau(t)) && \text{when } \tau(t+1) = \tau(t) + (0,-1) \\ A_{i,j}(l,m) &> A_{i,j}(\tau(t)) && \text{when } \tau(t+1) = \tau(t) + (1,0) \end{aligned}$$

for every $t \in [t^\star]$, excluding the t for which $\tau(t) = (l,m)$ since in this case we have equality: $A_{i,j}(l,m) = A_{i,j}(l,m)$. Restating this, τ is the correct contour if the (red) point p_j dominates the (blue) point q_i, defined as

$$\begin{aligned} p_j &= \Big(\ldots,\ \sigma(t)\left(A_j(\tau_{\rm c}(t)) - A_j(m)\right), \ldots\Big) \\ q_i &= \left(\ldots,\ \sigma(t)\left(A_i(l) - A_i(\tau_{\rm r}(t))\right), \ldots\right), \end{aligned}$$

where $\sigma(t) \in \{1,-1\}$ is the proper sign:

$$\sigma(t) = \begin{cases} 1 & \text{when } \tau(t+1) = \tau(t) + (0,-1), \text{ and} \\ -1 & \text{when } \tau(t+1) = \tau(t) + (1,0). \end{cases}$$

The coordinate t for which $\tau(t) = (l,m)$ is, of course, omitted from p_j and q_i, so both vectors have length at most $2g-2$.

Call a pair (τ,τ') of contours *legal* if the following points apply:

(i) Whenever τ and τ' do not intersect, τ is above τ'.
(ii) There are two $(l,m), (l',m') \in P$ such that τ contains (l,m) and τ' contains (l',m').
(iii) Let (R,S,T) be the tripartition of $[g]^2$ defined by (τ,τ'), where S are those positions lying strictly between (l,m) and (l',m'). Then $P \cap S = \emptyset$ and $|S| \le s$, where s is a parameter to be determined.

Let us clarify criterion (iii). It states that if $A_{i,j}$ is any *specific* box for which (τ,τ') are correct contours of $A_{i,j}(l,m)$ and $A_{i,j}(l',m')$, then the number of positions $(l'',m'') \in [g]^2$ for which $A_{i,j}(l'',m'') \in (A_{i,j}(l,m), A_{i,j}(l',m'))$ is at most s, and no such position appears in P. This is to ensure that we never need to sort more than s elements.

Our algorithm enumerates every legal pair (τ,τ') of contours, at most 2^{4g} in total. Let $(l,m),(l',m') \in P$ be the points lying on τ,τ' and (R,S,T) be the tripartition of $[g]^2$ defined by (τ,τ'). For each (τ,τ'), the algorithm enumerates every realizable permutation $\pi : [|S|] \to S$ of the elements at positions in S. By Lemma 2.3, there are $O(\binom{s}{2}^{2g}) < 2^{4g\log s}$ such permutations, which can be enumerated in $O(2^{4g\log s})$ time. For each (τ,τ',π), we create red points $\{p_j\}_{j\in[n/g]}$ and blue points $\{q_i\}_{i\in[n/g]}$ in $\mathbb{R}^{4g+s-5}$ such that p_j dominates q_i if and only if $\tau = \textsc{contour}(A_{i,j}(l,m))$ and $\tau' = \textsc{contour}(A_{i,j}(l',m'))$ are the correct contours (w.r.t. $A_{i,j}$) and π is the correct sorting permutation of $A_{i,j}(S)$. The first $4g-4$ coordinates encode the correctness of τ and τ' and the last $s-1$ coordinates encode the correctness of π.

According to Lemma 2.5, the time to report all dominating pairs is $O(p(n/g)^2 + 2^{4g} \cdot 2^{4g\log s} \cdot (c_\epsilon)^{4g+s-5}(2n/g)^{1+\epsilon})$. The first term is the output size, since by criterion (iii) of the definition of *legal*, at most $p-1$ pairs are reported for each of the $(\lceil n/g\rceil)^2$ boxes. There are $2^{4g}2^{4g\log s}$ choices for (τ,τ',π) and the dimension of the point set is at most $4g+s-5$, but could be smaller if the contours happen to be short or $|S| < s$. Fixing $\epsilon = 1/2$, if $g\log s$ and $g+s$ are both $O(\log n)$ (with a sufficiently small leading constant), then the running time of the algorithm will be dominated by the time spent reporting the output.

Call a box $A_{i,j}$ *bad* if the output of the dominating pairs algorithm fails to determine its sorted order. The only way a box can be bad is if an otherwise legal (τ,τ') with tripartition (R,S,T) were correct for $A_{i,j}$ but failed to be legal because $|S| > s$, leaving the sorted order of $A_{i,j}(S)$ unknown.

If all boxes are not bad, we can search for $x \in \mathbb{R}$ in $A_{i,j}$ in $O(\log g)$ time using binary search, as follows. Each box $A_{i,j}$ was associated with a list of $p-1$ triples of the form (τ,τ',π) returned by the dominating pairs algorithm, one for each pair of successive elements in $A_{i,j}(P)$. The first step is to find the predecessor of x in $A_{i,j}(P)$, that is, to find the consecutive $(l,m),(l',m') \in P$ for which $A_{i,j}(l,m) \leq x < A_{i,j}(l',m')$. Let (τ,τ',π) be the triple associated with $(l,m),(l',m')$ and (R,S,T) be the tripartition of (τ,τ'). Each legal, realizable (τ,τ',π) is encoded as a bit string with length $4g(1+\log s)$, which must fit comfortably in $\log n$ bits. Before executing the algorithm proper, we build, in $o(n)$ time, a lookup table indexed by tuples (τ,τ',π,r) that contains the location in $[g]^2$ of the element with rank r in S, sorted according to π. Using this lookup table, it is straightforward to perform a binary search for x in $A_{i,j}(S)$, in $O(\log|S|) = O(\log g)$ time.

5.2.1 A Randomized Parameterization of the Algorithm. In this section, we give a randomized algorithm for selecting a point set that gives an expected $O(n^2(\log\log n)^2/\log n)$-time algorithm with high probability.

Let $\delta > 0$ be a sufficiently small constant. Set $g = s = \delta \ln n/\ln\ln n$ and $p = 2 + 3\delta\ln n$. The points $(0,0)$ and $(g-1,g-1)$ must be in P and the remaining $3\delta\ln n$ points are chosen uniformly at random. With these parameters, the probability of a box being *bad* is sufficiently low to keep the expected cost per search $O(\log g)$.

Lemma 5.1. *The probability a particular box is bad is at most* $1/g$.

Proof. Let π be the sorted order for some box $A_{i,j}$. The probability that $A_{i,j}$ is bad is precisely the probability that there are $s+1$ consecutive elements (according to π) that are not included in P. The probability that this occurs for a particular set of $s+1$ elements is less than $(1-(p-2)/g^2)^{s+1} < e^{-(p-2)(s+1)/g^2} < e^{-3\ln\ln n} < 1/g^3$. By a union bound over all $g^2-(s+1)$ sets of $s+1$ consecutive elements, the probability that $A_{i,j}$ is bad is at most $1/g$. □

The expected time per search is therefore $O(\log g) + 1/g \cdot O(g) = O(\log g)$. By linearity of expectation, the expected total running time is $O(p(n/g)^2 + n^2(\log g)/g) = O(n^2\frac{(\log\log n)^2}{\log n})$.

Remark 1. We could have set the parameters differently and achieved the same running time. For example, setting $g = p = \Theta(\log n/\log\log n)$ and $s = \Theta(\log n)$ with appropriately chosen constants would also work. The advantage of keeping $s = O(\log n/\log\log n)$ is simplicity: we can afford to enumerate all $s!$ permutations of $S \subset [g]^2$ rather than explicitly construct a hyperplane arrangement in order to enumerate only those *realizable* permutations of S.

High Probability Bounds. The running time of the algorithm may deviate from its expectation with nonnegligible probability since the *badness* events for the boxes $\{A_{i,j}\}$ can be strongly positively correlated. The easiest way to obtain high probability bounds is simply to choose $L = c\log n$ random point sets $\{P_l\}_{l\in[L]}$, estimate the cost of the algorithm under each point set, then execute the algorithm under the point set with the best estimated cost. The first step is to run a truncated version of the algorithm in order to determine which queries will be asked in Step 4.2.1. Rather than answer the query $-A(k) \in A_{i,j}$ (where (i,j) is the current value of $(\mathrm{lo},\mathrm{hi})$), we simply record the triple (k,i,j) in a list Q to be answered later. The running time of the algorithm under P_l is $O(n^2(\log g)/g)$ plus g times the number of bad triples in Q, that is, those (k,i,j) for which $A_{i,j}$ is bad according to P_l.

Let ϵ_l be the true fraction of bad triples in Q according to P_l and $\hat{\epsilon}_l$ be the estimate of ϵ_l obtained by the following procedure. Sample $M = cg^2\ln n$ elements of Q uniformly at random and for each, test whether the given block is bad according to P_l by sorting its elements, in $O(g^2\log g)$ time. If X is the number of blocks discovered to be bad, report the estimate $\hat{\epsilon}_l = X/M$. By a standard version of the Chernoff bound,[11] we have

$$\Pr(|\hat{\epsilon}_l - \epsilon_l| > 1/g) = \Pr(|X - \mathbb{E}(X)| > M/g) < 2e^{-2(M/g)^2/M} = 2n^{-2c}.$$

By Lemma 5.1, $\mathbb{E}(\epsilon_l) = 1/g$ for each l, so by Markov's inequality, $\Pr(\min_{l\in L}\{\epsilon_l\} \le 2/g) \ge 1 - 2^{-L} = 1 - n^{-c}$. With high probability, each $\hat{\epsilon}_l$ deviates from ϵ_l by at most $1/g$, so the running time of the algorithm will be within a constant factor of its expectation with probability $1 - O(n^{-c})$. The time to pick the best point set $P_{l^\star}$, $l^\star$ being $\operatorname{argmin}_{l\in[L]}\{\hat{\epsilon}_l\}$, is $O(LMg^2 \log g) = o(\log^6 n)$. We could set L and M as high as $n/\operatorname{polylog}(n)$, making the probability that the algorithm deviates from its expectation exponentially small, $\exp(-n/\operatorname{polylog}(n))$.

5.2.2 A Deterministic Parameterization of the Algorithm. We achieve an $O(n^2/(\frac{\log n}{\log\log n})^{2/3})$ worst-case 3SUM algorithm by choosing parameters g, s, p, and point set P such that no block can be bad. As above, let $\delta > 0$ be a sufficiently small constant. Fix $g = (\delta \log n)^{2/3}(\log\log n)^{1/3}$ and $p = 2 + q^2 < (\delta \log n)^{2/3}(\log\log n)^{4/3}$ for an integer q to be determined. Aside from the two obligatory points, P contains an evenly spaced $q \times q$ grid in $[g] \times [g]$. Setting $\Delta = \lceil \frac{g+1}{q+1} \rceil$, P is defined as

$$P = \{(0,0), (g-1, g-1)\} \cup \{(k\Delta - 1, l\Delta - 1) \mid \text{ where } 1 \le k, l \le q\}.$$

See Figure 2(b).

We now argue that no box can be bad if $s = 2g(\Delta - 1)$. For any legal pair of contours (τ, τ'), in the corresponding tripartition (R, S, T), no element of P can be contained in S; that is, in any row (or column) containing elements of P, the width (or height) of the band S is at most $\Delta - 1$. Since both τ and τ' are monotone paths in $[g]^2$ (nondecreasing by row, nonincreasing by column), we always have $|S| < 2g(\Delta - 1) < 2g^2/(q+1) < 2\delta \log n$. See Figure 2(b) for a worst-case example. For δ sufficiently small, the overhead for reporting dominating pairs will be negligible. The overall running time is therefore $O(n^2(\log g)/g) = O(n^2/(\log n/\log\log n)^{2/3})$.

6 LINEAR DEGENERACY TESTING

Recall that we are given a set $S \subset \mathbb{R}$ and a function $\phi(x_1, \ldots, x_k) = \alpha_0 + \sum_{i=1}^k \alpha_i x_i$, for some real coefficients $\{\alpha_i\}$. The problem is to determine if there is a point $(x_1, \ldots, x_k) \in S^k$ where ϕ is zero. As we show below, our 3SUM decision tree can be generalized in a straightforward way to solve k-LDT with $O(n^{k/2}\sqrt{\log n})$ comparisons, when $k \ge 3$ is odd. Unfortunately, we do not see how to generalize our 3SUM *algorithms* to solve k-LDT in $O(n^{(k+1)/2}/\operatorname{polylog}(n))$ time, for any odd $k \ge 5$.

Proof (of Theorem 1.2). Define $\alpha \cdot S$ to be the set $\{\alpha \cdot a \mid a \in S\}$, where $\alpha \in \mathbb{R}$. Begin by sorting the sets

$$A = \{\alpha_0 + a_1 + a_2 + \cdots + a_{(k-1)/2} \mid a_i \in \alpha_i \cdot S, \text{ for each } i > 0\}$$

and

$$B = \{a_{(k+1)/2} + \cdots + a_{k-1} \mid a_i \in \alpha_i \cdot S, \text{ for each } i\}.$$

We have effectively reduced k-LDT to an unbalanced 3SUM problem. Letting C be the set $\alpha_k \cdot S$, the problem is to determine if there exist $a \in A, b \in B, c \in C$ such that $a + b + c = 0$. Note that $|A| = |B| = n^{(k-1)/2}$, whereas $|C| = n$. The standard 3SUM algorithm of Section 3 performs $|C| \cdot (|A| + |B|) = n^{(k+1)/2}$ comparisons. Generalizing the decision tree of Section 4 (from one list to three), by splitting $A + B$ into $g \times g$ boxes, sorting them, and searching for elements in C using binary search,

[11]If X is the number of successes in n independent Bernoulli trials, $\Pr(X > \mathbb{E}(X) + t)$ and $\Pr(X < \mathbb{E}(X) + t)$ are both upper bounded by $e^{-2t^2/n}$. See [37, Theorem 1.1].

we can solve unbalanced 3SUM using $O(g(|A| + |B|) + g^{-1}|C|(|A| + |B|)\log g)$ comparisons, which is $O(n^{k/2}\sqrt{\log n})$ when $g = \sqrt{n \log n}$. □

Our subquadratic 3SUM algorithms do not extend naturally to unbalanced instances. When $g = \text{polylog}(n)$, we can no longer afford to *explicitly* sort all $g \times g$ boxes in $A + B$ as this would require at least $\Omega(|A| \cdot |B|/g^2) = \Omega(n^{k-1}/\text{polylog}(n))$ time.[12]

7 ZERO TRIANGLES

We consider a matrix product called *target-min-plus* that subsumes the (min, +)-product (aka distance product) and the ZeroTriangle problem of [71]. Given real matrices $A \in (\mathbb{R} \cup \{\infty\})^{r\times s}$, $B \in (\mathbb{R} \cup \{\infty\})^{s\times t}$, and a target matrix $T \in (\mathbb{R} \cup \{-\infty, \infty\})^{r\times t}$, the goal is to compute $C = \circledcirc(A, B, T)$, where

$$C(i,j) = \min\{\, A(i,k) + B(k,j) \mid k \in [s] \text{ and } A(i,k) + B(k,j) \geq T(i,j)\}\,,$$

as well as the matrix of witnesses, that is, the k (if any) for which $C(i,j) = A(i,k) + B(k,j)$. This operation reverts to the (min, +)-product when $T(i,j) = -\infty$. It can also solve ZeroTriangle on a weighted graph $G = (V, E, w)$ by setting A, B, and T as follows. Let $A(i,j) = B(i,j) = w(i,j)$, where $w(i,j) \overset{\text{def}}{=} \infty$ if $(i,j) \notin E$, and let $T(i,j) = -w(i,j)$ if $(i,j) \in E$ and ∞ otherwise. If $C(i,j) = T(i,j)$, then there is a zero-weight triangle containing (i,j) and the witness matrix gives the third corner of the triangle.

The trivial target-min-plus algorithm runs in $O(rst)$ time and performs the same number of comparisons. We can compute the target-min-plus product using fewer comparisons using Fredman's trick.

Theorem 7.1. *The decision tree complexity of the target-min-plus product of three $n \times n$ matrices is $O(n^{5/2}\sqrt{\log n})$. This product can be computed in $O(n^3(\log\log n)^2/\log n)$ time.*

Proof. We first show that the target-min-plus product $\circledcirc(A, B, T)$ can be determined with $O((r + t)s^2 + rt \log s)$ comparisons, where A, B, and T are $r \times s$, $s \times t$, and $r \times t$ matrices, respectively. Begin by sorting the set

$$D = \{A(i,k) - A(i,k'),\ B(k',j) - B(k,j) \mid i \in [r], j \in [t],\ \text{and } k, k' \in [s]\}.$$

By Lemma 2.3, the number of comparisons required to sort D is $O(|D| + (r + t)s \log(rst)) = O((r + t)s^2 + (r + t)s \log(rst))$. We can now deduce the sorted order on

$$S(i,j) = \{A(i,k) + B(k,j) \mid k \in [s]\}$$

for any pair $(i,j) \in [r] \times [t]$, and can therefore find $C(i,j) = \min(S(i,j) \cap [T(i,j), \infty))$ with a binary search over $S(i,j)$ using $\log s$ additional comparisons. The total number of comparisons is $O((r + t)s^2 + rt \log s)$. Note that this provides no improvement when A and B are square, that is, when $r = s = t = n$. Following Fredman [41], we partition A and B into rectangular matrices and compute their target-min-plus products separately.

Choose a parameter g and partition A into $A_0, \ldots, A_{\lceil n/g\rceil -1}$ and B into $B_0, \ldots, B_{\lceil n/g\rceil -1}$, where A_ℓ contains columns $\ell g, \ldots, (\ell + 1)g - 1$ of A and B_ℓ contains the corresponding rows of B. For each $\ell \in [n/g]$, compute the target-min-plus product $C_\ell = \circledcirc(A_\ell, B_\ell, T)$ and set $C(i,j) = \min_{\ell \in [n/g]}(C_\ell(i,j))$. This algorithm performs $O((n/g) \cdot (ng^2 + n^2 \log n))$ comparisons to compute

[12]Note that there are only $O(|C|(|A| + |B|)/g)$ boxes of interest. The dominating-pairs approach does not let us sort an arbitrary selection of boxes in constant time per box, but it is possible to accomplish this task in a more powerful model of computation. On a souped-up word RAM with $O(g^2 \log n)$-bit words and a couple of nonstandard unit-time operations, any $g \times g$ box can be sorted in $O(1)$ time. Simulating such a unit-time operation on the traditional word RAM is a challenging problem.

$\{C_\ell\}_{\ell\in[n/g]}$ and $n^2(n/g)$ comparisons to compute C. When $g = \sqrt{n\log n}$, the number of comparisons is $O(n^{5/2}\sqrt{\log n})$.

To compute the product efficiently, we use the geometric dominance approach of Chan [28] and Bremner et al. [21]. Let $g = \frac{1}{2}\log n/\log\log n$, and partition A into $n \times g$ matrices $\{A_\ell\}$ and B into $g \times n$ matrices $\{B_\ell\}$. For each $\ell \in [n/g]$ and permutation $\pi : [g] \to [g]$, we will find those pairs $(i, j) \in [n]^2$ for which π is the sorted order on $\{A_\ell(i,k) + B_\ell(k,j) \mid k \in [g]\}$.[13] Such a triple satisfies the inequality $A_\ell(i,\pi(k)) + B_\ell(\pi(k),j) < A_\ell(i,\pi(k+1)) + B_\ell(\pi(k+1),j)$, for all $k \in [g-1]$. By Fredman's trick, this is equivalent to saying that the (red) point

$$(\ldots, A_\ell(i,\pi(k+1)) - A_\ell(i,\pi(k)), \ldots)$$

dominates the (blue) point

$$(\ldots, B_\ell(\pi(k),j) - B_\ell(\pi(k+1),j), \ldots)$$

in each of the $g-1$ coordinates. By Lemma 2.5, the total time for all $\lceil n/g\rceil g!$ invocations of the dominance algorithm is $O((n/g)\cdot g!\cdot c_\epsilon^{g-1}\cdot(2n/g)^{1+\epsilon})$ plus the output size, which is precisely $n^2\lceil n/g\rceil$. For $\epsilon = 1/2$, the running time is $O(n^3/g)$. We can encode one of $g!$ sorted orders π using $\frac{1}{2}\log n$ bits. Using a precomputed lookup table with size $o(n)$, we can query in $O(1)$ time the element with rank k according to π.

We can now compute the target-min-plus product $C_\ell = \circledcirc(A_\ell, B_\ell, T)$ in $O(n^2\log g)$ time by iterating over all $(i, j) \in [n]^2$ and performing a binary search to find the minimum element in $\{A_\ell(i,k) + B_\ell(k,j) \mid k \in [g]\} \cap [T(i,j),\infty)$. Since $C = \circledcirc(A,B,T)$ contains the pointwise minima of $\{C_\ell\}$, the total time to compute the target-min-plus product is $O(n^3(\log g)/g) = O(n^3(\log\log n)^2/\log n)$. □

The $\sqrt{\log n}$ factor in the decision tree complexity of target-min-plus arises from the binary searches, the n/g searches per pair $(i, j) \in [n]^2$. If the searches were sufficiently correlated (either for fixed (i, j) or fixed ℓ), then there would be some hope that we could evade the information theoretic lower bound of $\Omega(\log g)$ per search. Using random sampling, we form a hierarchy of rectangular target-min-plus products such that the solutions at one level give a hint for the solutions at the next lower level. The cost of finding the solution, given the hint from the previous level, is $O(1)$ in expectation. The same approach lets us shave another $\log\log n$ factor off the algorithmic complexity of target-min-plus.

Theorem 7.2. *The randomized decision tree complexity of the target-min-plus product of three $n \times n$ matrices is $O(n^{5/2})$. It can be computed in $O(n^3\log\log n/\log n)$ time with high probability.*

Proof. As usual let A, B, and T be $n \times n$ matrices and g be a parameter. We will eventually set $g = \lceil\sqrt{n}\rceil$. We partition the indices $[n]$ at $\log\log n$ levels. Define $I_{l,p} = [pg2^l, (p+1)g2^l)$ to be the pth interval at level l. In other words, level-l intervals have width $g2^l$ and a level-$(l+1)$ interval is the union of two level-l intervals. Form a series of nested index sets $[n] = J_0 \supset J_1 \supset \cdots \supset J_{\log\log n-1}$, such that $J_l \cap I_{l,p}$ is a uniformly random subset of $I_{l,p}$ of size g. In other words, each element of J_{l-1} is promoted to J_l with probability 1/2, but in such a way that $|J_l \cap I_{l,p}|$ is precisely its expectation g.

After generating the sets $\{J_l\}$, the algorithm sorts D with $O(n^2\log n + |D|) = O(n^{5/2})$ comparisons (see Lemma 2.3), where

$$D = \left\{ A(i,k) - A(i,k'),\quad B(k',i) - B(k,i) \;\middle|\; \begin{array}{l} i \in [n] \text{ and } k, k' \in J_l \cap I_{l,p}, \\ \text{for some level } l \text{ and index } p \end{array} \right\}.$$

[13]Break ties in any consistent fashion so that the sorted order is unique.

Fix $i, j \in [n]$. We proceed to compute $C(i,j)$ with $O(n/g)$ comparisons with high probability. If $K \subset [n]$ is a set of indices, define $\kappa(K)$ to be the witness of the target-min-plus product restricted to K, that is,

$$\kappa(K) = \underset{\substack{k \in K \text{ such that} \\ A(i,k)+B(k,j) \geq T(i,j)}}{\operatorname{argmin}} (A(i,k) + B(k,j)).$$

There may, in fact, be no such witness, in which case $\kappa(K) = \perp$. Let $\kappa_{l,p}$ be short for $\kappa(J_l \cap I_{l,p})$. Notice that by Fredman's trick, we can deduce the sorted order on

$$S_{l,p} = \{A(i,k) + B(k,j) \mid k \in J_l \cap I_{l,p}\}$$

without additional comparisons, for any l and p. We can therefore compute the top-level witnesses $\{\kappa_{\log\log n-1,p}\}_{p\in[n/(g2^{\log\log n-1})]}$ with $O(\frac{n}{g2^{\log\log n}} \cdot \log n) = O(\sqrt{n})$ comparisons via binary search. Our goal now is to compute the witnesses at all lower-level intervals with $O(\sqrt{n})$ comparisons. Suppose we have computed the level-$(l+1)$ witness $\kappa_{l+1,p}$ and wish to compute the level-l witnesses of the constituent sequences, namely, $\kappa_{l,2p}$ and $\kappa_{l,2p+1}$. Define $\kappa'_{l,2p} = \kappa(J_{l+1} \cap I_{l,2p})$. Note that $\kappa'_{l,2p}$ is determined by $\kappa_{l+1,p}$ and the sorted order on $S_{l+1,p}$. The distance between $\kappa_{l,2p}$ and $\kappa'_{l,2p}$ (according to the sorted order on $S_{l,2p}$) is stochastically dominated by a geometric random variable with mean 1.[14] The expected number of comparisons needed to determine $\kappa_{l,p}$ using linear search is therefore $O(1)$. These geometric random variables are independent due to the independence of the samples, so we can apply standard Chernoff-type concentration bounds [37]. The probability that the sum of these independent geometric random variables exceeds twice its expectation μ is $\exp(-\Omega(\mu)) = \exp(-\Omega(\sqrt{n}))$.

Once we have computed all the witnesses for level-0, $\{\kappa_{0,p}\}_{p\in[n/g]}$, we simply have to choose the best among them, so $C(i,j) = \min\{A(i,\kappa_{0,p}) + B(\kappa_{0,p},j) \mid p \in [n/g] \text{ and } \kappa_{0,p} \neq \perp\}$. The total number of witnesses computed for fixed i,j is $\sum_{l\geq 0} n/(g2^l) < 2n/g$. The total number of comparisons is therefore $O(n^2 g)$ to sort D and $O(n^3/g)$ to compute all the witnesses and C, which is $O(n^{5/2})$ when $g = \sqrt{n}$.

To improve the $O(n^3(\log\log n)^2/\log n)$ algorithm, we apply the ideas above with different parameters. Let $g = \frac{1}{2}\log n/\log\log n$. We consider the same partitions $\{I_{l,p}\}_{l,p}$ and nested index sets $\{J_l\}_l$, but only use the first $\log\log\log n$ levels, not $\log\log n$ as before. For each level $l \in [\log\log\log n]$, index $p \in [n/(g2^l)]$, and permutation $\pi : [g] \to [g]$, we compute those pairs (i,j) for which π is the sorting permutation on the elements of $J_l \cap I_{l,p}$. This can be done in time linear in the output size $O(n^3/g)$ and $\sum_{l\in[\log\log\log n]} \frac{n}{g2^l} g! c_\epsilon n^{1+\epsilon}$. When $\epsilon = 1/2$ and $g = O(\log n/\log\log n)$, the time spent computing dominating pairs is $O(n^3/g)$. Since $g = \frac{1}{2}\log n/\log\log n$, we can encode the sorting permutation of each $J_l \cap I_{l,p}$ comfortably in less than $\frac{1}{2}\log n$ bits. We build lookup tables to answer the following two types of queries: (1) given a permutation $\pi : [g] \to [g]$ and rank $k \in [g]$, return the index $\pi^{-1}(k)$ of the element with rank k, and (2) given a sorting permutation π for some $J_{l+1} \cap I_{l+1,p}$, the number of elements in $J_{l+1} \cap I_{l,2p}$[15], and the position of $\kappa_{l+1,p}$ within $J_{l+1} \cap I_{l+1,p}$, return the positions of $\kappa'_{l,2p}, \kappa'_{l,2p+1}$ within $J_l \cap I_{l,2p}$ and $J_l \cap I_{l,2p+1}$, respectively.

Fix a pair $(i,j) \in [n]^2$. When finding the top-level witnesses $\{\kappa_{\log\log\log n-1,p}\}_{p\in[2n/(g\log\log n)]}$, we can implement each step of the binary searches in $O(1)$ time using queries of type (1) (see above), for a total of $O(n/g)$ time. Suppose we know the position of $\kappa_{l+1,p}$. With a type (2) query, we can

[14]With probability 1/2, $\kappa_{l,2p} = \kappa'_{l,2p}$; with probability less than 1/4, $\kappa_{l,2p}$ is one less than $\kappa'_{l,2p}$ according to the sorted order on $S_{l,2p}$; and so on.

[15](so we know which elements came from $I_{l,2p}$ and which came from $I_{l,2p+1}$).

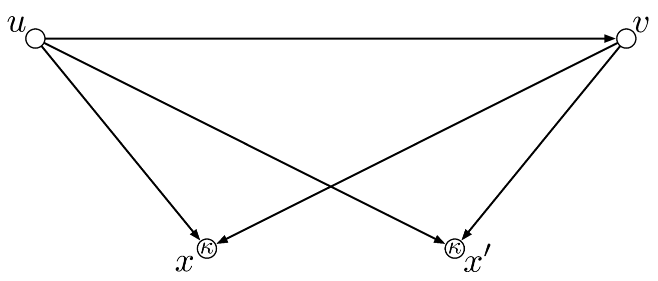

Fig. 3. In this example, $\textsc{color}(x) = \textsc{color}(x') = \kappa$ and both triangles on $\{u, v, x\}$ and $\{u, v, x'\}$ are of the same type.

learn the positions of $\kappa'_{l,2p}, \kappa'_{l,2p+1}$. Suppose element $\kappa'_{l,2p}$ has rank k w.r.t. the sorting permutation π of $J_l \cap I_{l,2p}$. We can conduct the linear search for $\kappa_{l,2p}$ using type (1) lookups to the elements of π with rank $k, k-1, \ldots$ until it is discovered. The search for $\kappa_{l,2p+1}$ from $\kappa'_{l,2p+1}$ is identical. Over all $(i,j) \in [n]^2$, the total number of comparisons is $O(n^3/g) = O(n^3 \log\log n / \log n)$ with high probability. □

The trivial time to solve ZeroTriangle on sparse m-edge graphs is $O(m^{3/2})$. Such graphs contain at most $O(m^{3/2})$ triangles, which can be enumerated in $O(m^{3/2})$ time. We now restate and prove Theorem 1.4.

Theorem 1.4. *The decision tree complexity of ZeroTriangle on an m-edge graph is $O(m^{5/4}\sqrt{\log m})$ and, using randomization, $O(m^{5/4})$ with high probability. The ZeroTriangle problem can be solved in $O(m^{3/2}((\log\log m)^2/\log m)^{1/4})$ time deterministically or $O(m^{3/2}(\log\log m/\log m)^{1/4})$ with high probability.*

Proof. We begin by greedily finding an acyclic orientation of the graph $G = (V, E, w)$. Iteratively choose the vertex v with the fewest number of still unoriented edges and direct them all away from v. Since every m-edge graph contains a vertex of degree less than $\Delta = \sqrt{2m}$, the maximum outdegree in this orientation is less than Δ. We now use $\vec{E}$ instead of E to emphasize that the set is oriented.

Select a random mapping $\textsc{color} : V \to [K]$, where K will be fixed soon. The expected number of pairs of oriented edges $\{(u,x),(u,x')\} \subset \vec{E}$ having $\textsc{color}(x) = \textsc{color}(x')$ is less than $m\Delta/K$. Any coloring that does not exceed this expected value suffices; we do not need to choose $\textsc{color}$ at random. We now sort the set D with $O(m\log m + |D|) = O(m\log m + m\Delta/K)$ comparisons [40], where

$$D = \{w(u,x) - w(u,x') \mid u \in V \text{ and } (u,x),(u,x') \in \vec{E} \text{ and } \textsc{color}(x) = \textsc{color}(x')\}.$$

Call a triangle on $\{u,v,x\}$ *type*-$((u,v),\kappa)$ if the orientation of the edges is $(u,v),(u,x),(v,x)$ and $\textsc{color}(x) = \kappa$. Clearly every triangle is of one type and there are mK types. A type-$((u,v),\kappa)$ zero-weight triangle exists if and only if $-w(u,v)$ appears in the set $\{w(u,x) + w(v,x) \mid x \in V \text{ and } \textsc{color}(x) = \kappa\}$. By Fredman's trick, the sorted order of this set is determined by the sorted order of D, since $w(u,x) + w(v,x) < w(u,x') + w(v,x')$ if and only if $w(u,x) - w(u,x') < w(v,x') - w(v,x)$. See Figure 3. We can therefore determine if there exists a zero-weight triangle of a particular type with $O(\log\Delta)$ comparisons via binary search. The total number of comparisons is $O(m\log m + m\Delta/K + mK\log\Delta)$, which is $O(m^{5/4}\sqrt{\log m})$ when $K = \sqrt{\Delta/\log\Delta} = O(m^{1/4}/\sqrt{\log m})$. The $\sqrt{\log m}$ factor can be shaved off using randomization, exactly as in Theorem 7.2. We form $\log\log n$ levels of colorings, where color class p at the $(l+1)$th level is the union of classes $2p$ and $2p+1$ at the lth level. After the searches are conducted at level $l+1$, the expected cost per search at level l is $O(1)$.

To solve ZeroTriangle efficiently, we greedily orient the graph as before, stopping when all remaining vertices have degree at least Δ, where Δ is a parameter to be fixed shortly. (The unoriented subgraph remaining is called the Δ-*core*.) For each vertex u and each pair of outgoing edges $(u,v),(u,x) \in \vec{E}$, we check whether (u,v,x) is a triangle and, if so, whether it has zero weight. (Note that the edge (v,x), if it exists, may be in the Δ-core and therefore not have an orientation.) This takes $O(m\Delta)$ time. It remains to check triangles contained entirely in the Δ-core. Since the Δ-core has at most $2m/\Delta$ vertices, we can solve ZeroTriangle on it in $O((m/\Delta)^3(\log\log m)^2/\log m)$ time or $O((m/\Delta)^3 \log\log m/\log m)$ time with high probability. The total cost is balanced when $\Delta = \sqrt{m}((\log\log m)^2/\log m))^{1/4}$ or $\Delta = \sqrt{m}(\log\log m/\log m)^{1/4}$ depending on whether we use the randomized or deterministic ZeroTriangle algorithm. □

The Convolution3SUM problem reduces to 3SUM, so our $O(n^{3/2}\sqrt{\log n})$ and $O(n^2/\operatorname{polylog}(n))$ bounds for 3SUM extend directly to Convolution3SUM. However, Convolution3SUM has additional structure, which makes it amenable to the same random sampling techniques used in Theorem 7.2. We give only a sketch of the proof of Theorem 1.5 as the analysis is essentially the same as that found in Theorem 7.2.

Theorem 1.5. *The decision tree complexity of Convolution3SUM is* $O(n^{3/2}\sqrt{\log n})$, *and its randomized decision tree complexity is* $O(n^{3/2})$ *with high probability. The Convolution3SUM problem can be solved in* $O(n^2(\log\log n)^2/\log n)$ *time deterministically or in* $O(n^2\log\log n/\log n)$ *time with high probability.*

Proof (sketch). In the Convolution3SUM problem, we must determine if there is a $k \in [n]$ such that $A(k)$ occurs on the kth antidiagonal of the matrix $A+A$. In contrast to 3SUM, the rows and columns of $A+A$ are not sorted. On the other hand, we do not need to look for $A(k)$ in the whole matrix, just those locations along an antidiagonal.

The $O(n^{3/2}\sqrt{\log n})$ decision tree bound is proved as in Section 4, by partitioning the matrix into $g \times g$ blocks and, for each k, conducting binary searches for $A(k)$ in the appropriate antidiagonals of at most $2n/g$ boxes. In order to shave off the $\sqrt{\log n}$ factor, we use the same random sampling approach of Theorem 7.2. We partition $A+A$ at $\log\log n$ levels, where level-l boxes have size $g2^l \times g2^l$ and are the union of four level-$(l-1)$ boxes. The rows and columns are sampled at $\log\log n$ levels, where a row or column at level $l-1$ is promoted to level l with probability 1/2. Note that an element of $A+A$ appears at level-l if and only if both its row and column are in the level-l sample. Since elements along any antidiagonal share no rows or columns, the events in which they appear at level-l are entirely independent. This independence property allows us to search for $A(k)$ in level-l sampled boxes in $O(1)$ expected time, given the predecessors of $A(k)$ in the level-$(l+1)$ sampled boxes. Algorithms running in $O(n^2(\log\log n)^2/\log n)$ (deterministically) or $O(n^2\log\log n/\log n)$ (with high probability) are obtained using the methods applied in Theorems 7.1 and 7.2. Alternatively, we could apply the Vassilevska Williams and Williams [71] reduction from Convolution3SUM to ZeroTriangle and then invoke the algorithms of Theorems 7.1 and 7.2 as black boxes. □

8 CONCLUSION AND OPEN PROBLEMS

The most pressing open problem is to break the n^2 barrier for general position testing (GPT) of point-sets in $\mathbb{R}^2$, and to resolve other 3SUM-hard problems in computational geometry. Unfortunately, none of these problems appear to be directly reducible to 3SUM. Nonetheless, the existence of subquadratic decision trees for 3SUM inspires some confidence that there also exist subquadratic counterparts for GPT and related problems; see [14, 30].

It is known that the randomized complexities of Integer3SUM and IntegerConv3SUM are within an $O(\log n)$ factor [56]. Can this reduction be generalized to connect 3SUM and Convolution3SUM over the reals? Can it be derandomized?

APPENDIX

A BICHROMATIC DOMINATING PAIRS

For the sake of completeness, we shall review a standard divide-and-conquer dominating-pairs algorithm of Preparata and Shamos [63, p. 366] and give a short proof of Lemma 2.5 due to Chan [28].

A.1 The Divide-and-Conquer Algorithm. We are given a list of n red and blue points $P = (p_1, \ldots, p_n)$, each $p_i \subset \mathbb{R}^d$, and wish to report all pairs (i, j) for which p_i is red, p_j is blue, and $p_i(k) \geq p_j(k)$ for each $k \in [d]$. When $d = 0$, the algorithm simply reports every pair of red/blue points, so assume $d \geq 1$. Find the median h on the last coordinate in $O(n)$ time [19] and partition P into disjoint sets P_L, P_R of size at most $\lceil n/2 \rceil$, where

$$P_L \subset \{p \in P \mid p(d-1) \leq h\}$$
$$P_R \subset \{p \in P \mid p(d-1) \geq h\}.$$

Furthermore, there cannot be a red $p_i \in P_L$ and blue $p_j \in P_R$ such that $p_i(d-1) = p_j(d-1) = h$.[16] At this point all dominating pairs are in P_L, or P_R, or have one point in each, in which case the blue point is necessarily in P_L and the red in P_R. We make three recursive calls to find dominating pairs of each variety. The first two calls are on $\lceil n/2 \rceil$ points in $\mathbb{R}^d$. The third recursive call is on all blue points in P_L and all red points in P_R; after stripping their last coordinate, they lie in $\mathbb{R}^{d-1}$.

Excluding the cost of reporting the output, the running time of this algorithm is bounded by $T_d(n)$, defined inductively as

$$T_0(n) = T_d(1) = 0$$
$$T_d(n) = 2T_d(n/2) + T_{d-1}(n) + n.$$

We prove by induction that $T_d(n) \leq c_\epsilon^d n^{1+\epsilon} - n$, a bound that holds in all base cases. Assuming the claim holds for all smaller values of d and n,

$$\begin{aligned} T_d(n) &\leq 2\left(c_\epsilon^d (n/2)^{1+\epsilon} - n/2\right) + \left(c_\epsilon^{d-1} n^{1+\epsilon} - n\right) + n \\ &= \left(c_\epsilon^d / 2^\epsilon + c_\epsilon^{d-1}\right) n^{1+\epsilon} - n \\ &= (1/2^\epsilon + 1/c_\epsilon) \cdot c_\epsilon^d n^{1+\epsilon} - n \\ &= c_\epsilon^d n^{1+\epsilon} - n && \text{By definition of } c_\epsilon = 2^\epsilon/(2^\epsilon - 1). \end{aligned}$$

ACKNOWLEDGMENTS

We would like to thank an anonymous reviewer for suggesting several improvements to the presentation of this work.

REFERENCES

[1] S. Aaronson and Y. Shi. 2004. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM* 51, 4 (2004), 595–605.

[16] In other words, among points with the same last coordinate, blue points precede red points. If the domination criterion were *strict*, that is, if (p_i, p_j) were a dominating pair only if $p_i(k) > p_j(k)$ for all $k \in [d]$, then we would break ties the other way, letting red points precede blue points.

[2] A. Abboud, A. Backurs, and V. Vassilevska Williams. 2015. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*. 59–78.

[3] A. Abboud, F. Grandoni, and V. Vassilevska Williams. 2015. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 1681–1697.

[4] A. Abboud and V. V. Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. 434–443. Retrieved from https://doi.org/10.1109/FOCS.2014.53.

[5] A. Abboud, V. Vassilevska Williams, and O. Weimann. 2014. Consequences of faster sequence alignment. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*. 39–51.

[6] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. 1975. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

[7] O. Aichholzer, F. Aurenhammer, E. D. Demaine, F. Hurtado, P. Ramos, and J. Urrutia. 2012. On k-convex polygons. *Comput. Geom.* 45, 3 (2012), 73–87.

[8] N. Ailon and B. Chazelle. 2005. Lower bounds for linear degeneracy testing. *J. ACM* 52, 2 (2005), 157–171.

[9] N. Alon and R. B. Boppana. 1987. The monotone circuit complexity of boolean functions. *Combinatorica* 7, 1 (1987), 1–22.

[10] A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein. 2014. Consequences of faster sequence alignment. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*. 114–125.

[11] A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom. 2016. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC'16)*. 12:1–12:12. Retrieved from https://doi.org/10.4230/LIPIcs.ISAAC.2016.12

[12] A. Backurs and P. Indyk. 2015. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC'15)*. 51–58.

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

[14] L. Barba, J. Cardinal, J. Iacono, S. Langerman, A. Ooms, and N. Solomon. 2017. Subquadratic algorithms for algebraic generalizations of 3SUM. In *Proceedings of the 33rd International Symposium on Computational Geometry (SoCG'17)*. 13:1–13:15. Retrieved from https://doi.org/10.4230/LIPIcs.SoCG.2017.13.

[15] G. Barequet and S. Har-Peled. 2001. Polygon containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. *Int. J. Comput. Geometry Appl.* 11, 4 (2001), 465–474.

[16] P. Beame. 1991. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.* 20, 2 (1991), 270–277. Retrieved from https://doi.org/10.1137/0220017.

[17] C. H. Bennett, E. Bernstein, G. Brassard, and U. V. Vazirani. 1997. Strengths and weaknesses of quantum computing. *SIAM J. Comput.* 26, 5 (1997), 1510–1523.

[18] A. Bjorklund, R. Pagh, V. Vassilevska Williams, and U. Zwick. 2014. Listing triangles. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*. 223–234.

[19] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461.

[20] M. Braverman, Y. K. Ko, and O. Weinstein. 2015. Approximating the best nash equilibrium in $n^{o(\log n)}$-time breaks the exponential time hypothesis. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 970–982.

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

[22] K. Bringmann. 2014. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. 661–670.

[23] K. Bringmann and M. Künnemann. 2015. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*. 79–97.

[24] R. C. Buck. 1943. Partition of space. *Amer. Math. Monthly* 50 (1943), 541–544.

[25] A. Butman, P. Clifford, R. Clifford, M. Jalsenius, N. Lewenstein, B. Porat, E. Porat, and B. Sach. 2013. Pattern matching under polynomial transformation. *SIAM J. Comput.* 42, 2 (2013), 611–633.

[26] C. Calabro, R. Impagliazzo, and R. Paturi. 2009. The complexity of satisfiability of small depth circuits. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC'09)*. 75–85. Retrieved from https://doi.org/10.1007/978-3-642-11269-0_6

[27] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. 2016. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM*

Conference on Innovations in Theoretical Computer Science (ITCS'16). 261–270. Retrieved from https://doi.org/10.1145/2840728.2840746.

[28] T. M. Chan. 2008. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica* 50, 2 (2008), 236–243.

[29] T. M. Chan. 2015. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15).* 212–217.

[30] T. M. Chan. 2018. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18).* 881–897. Retrieved from https://doi.org/10.1137/1.9781611975031.57

[31] T. M. Chan and M. Lewenstein. 2015. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC'15).* 31–40.

[32] S. Chechik, D. Larkin, L. Roditty, G. Schoenebeck, R. E. Tarjan, and V. Vassilevska Williams. 2014. Better approximation algorithms for the graph diameter. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14).* 1041–1052.

[33] K.-Y. Chen, P.-H. Hsu, and K.-M. Chao. 2009. Approximate matching for run-length encoded strings is 3SUM-hard. In *Combinatorial Pattern Matching*. Lecture Notes in Computer Science, Vol. 5577. Springer, 168–179.

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

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

[36] R. Duan and S. Pettie. 2017. Connectivity oracles for graphs subject to vertex failures. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA'17).* 490–509.

[37] D. P. Dubhashi and A. Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press.

[38] H. Edelsbrunner, J. O'Rourke, and R. Seidel. 1986. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.* 15, 2 (1986), 341–363.

[39] J. Erickson. 1999. Bounds for linear satisfiability problems. *Chicago J. Theor. Comput. Sci.* 1999, 8 (1999).

[40] M. L. Fredman. 1976. How good is the information theory bound in sorting? *Theor. Comput. Sci.* 1, 4 (1976), 355–361.

[41] M. L. Fredman. 1976. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.* 5, 1 (1976), 83–89.

[42] M. L. Fredman and M. Saks. 1989. The cell probe complexity of dynamic data structures. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC'89).* 345–354.

[43] A. Freund. 2017. Improved subquadratic 3SUM. *Algorithmica* 77, 2 (2017), 440–458. Retrieved from https://doi.org/10.1007/s00453-015-0079-6.

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

[45] O. Gold and M. Sharir. 2017. Improved bounds for 3SUM, k-SUM, and linear degeneracy. In *Proceedings of the 25th Annual European Symposium on Algorithms (ESA'17) (Leibniz International Proceedings in Informatics (LIPIcs))*, Kirk Pruhs and Christian Sohler (Eds.), Vol. 87. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 42:1–42:13. Retrieved from https://doi.org/10.4230/LIPIcs.ESA.2017.42.

[46] A. Grønlund and S. Pettie. 2014. Threesomes, degenerates, and love triangles. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS'14).* 621–630.

[47] J. Hartmanis and R. E. Stearns. 1965. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.* 117 (1965), 285–306.

[48] J. Håstad. 1986. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC'86).* 6–20.

[49] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC'15).* 21–30.

[50] R. Impagliazzo, S. Lovett, R. Paturi, and S. Schneider. 2014. 0-1 integer linear programming with a linear number of constraints. *Electron. Colloq. Comput. Complex. (ECCC)* 21 (2014), 24.

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

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

[53] Z. Jafargholi and E. Viola. 2016. 3SUM, 3XOR, triangles. *Algorithmica* 74, 1 (2016), 326–343. Retrieved from https://doi.org/10.1007/s00453-014-9946-9.

[54] D. Kane, S. Lovett, and S. Moran. 2018. Near-optimal linear decision trees for k-SUM and related problems. In *Proceedings of the 50th ACM Symposium on Theory of Computing (STOC'18).*

[55] T. Kopelowitz, S. Pettie, and E. Porat. 2015. Dynamic set intersection. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS'15).* 470–481.

[56] T. Kopelowitz, S. Pettie, and E. Porat. 2016. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*. 1272–1287. Retrieved from https://doi.org/10.1137/1.9781611974331.ch89.

[57] K. G. Larsen. 2012. The cell probe complexity of dynamic range counting. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC'12)*. 85–94.

[58] A. Lincoln, V. Vassilevska Williams, J. R. Wang, and R. R. Williams. 2016. Deterministic time-space trade-offs for k-SUM. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP'16)*. 58:1–58:14. Retrieved from https://doi.org/10.4230/LIPIcs.ICALP.2016.58.

[59] S. Meiser. 1993. Point location in arrangements of hyperplanes. *Inf. Comput.* 106, 2 (1993), 286–303.

[60] F. Meyer auf der Heide. 1984. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. ACM* 31, 3 (1984), 668–676.

[61] C. St. J. A. Nash-Williams. 1964. Decomposition of finite graphs into forests. *J. London Math. Soc.* 39 (1964), 12.

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

[63] F. P. Preparata and M. I. Shamos. 1985. *Computational Geometry*. Springer, New York.

[64] M. Pătraşcu. 2010. Towards polynomial lower bounds for dynamic problems. In *Proceedings 42nd ACM Symposium on Theory of Computing (STOC'10)*. 603–610.

[65] M. Pătraşcu and R. Williams. 2010. On the possibility of faster SAT algorithms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. 1065–1075.

[66] A. A. Razborov. 1985. Lower bounds for the monotone complexity of some boolean functions. *Dokl. Ak. Nauk. USSR* 281 (1985), 798–801(in Russian). English translation in *Sov. Math. Dokl.* 31 (1985): 354–357.

[67] L. Roditty and V. Vassilevska Williams. 2013. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC'13)*. 515–524.

[68] B. Saha. 2015. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*. 118–135. Retrieved from https://doi.org/10.1109/FOCS.2015.17.

[69] M. A. Soss, J. Erickson, and M. H. Overmars. 2003. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Comput. Geom.* 26, 3 (2003), 235–246.

[70] V. Vassilevska Williams and R. Williams. 2010. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*. 645–654.

[71] V. Vassilevska Williams and R. Williams. 2013. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.* 42, 3 (2013), 831–854.

Received August 2015; revised September 2017; accepted February 2018

Copyright of Journal of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.