Faster Combinatorial k-Clique Algorithms*

Amir Abboud, Nick Fischer, and Yarin Shechter

Weizmann Institute of Science
Rehovot, Israel
{amir.abboud,nick.fischer,yarin.shechter}@weizmann.ac.il

Abstract. Detecting if a graph contains a k-Clique is one of the most fundamental problems in computer science. The asymptotically fastest algorithm runs in time $O(n^{\omega k/3})$, where ω is the exponent of Boolean matrix multiplication. To date, this is the only technique capable of beating the trivial $O(n^k)$ bound by a polynomial factor. Due to this technique's various limitations, much effort has gone into designing "combinatorial" algorithms that improve over exhaustive search via other techniques.
The first contribution of this work is a faster combinatorial algorithm for k-Clique, improving Vassilevska's bound of $O(n^k/\log^{k-1} n)$ by two log factors. Technically, our main result is a new reduction from k-Clique to Triangle detection that exploits the same divide-and-conquer at the core of recent combinatorial algorithms by Chan (SODA'15) and Yu (ICALP'15).
Our second contribution is exploiting combinatorial techniques to improve the state-of-the-art (even of non-combinatorial algorithms) for generalizations of the k-Clique problem. In particular, we give the first $o(n^k)$ algorithm for k-clique in hypergraphs and an $O(n^3/\log^{2.25} n + t)$ algorithm for listing t triangles in a graph.

1 Introduction

One of the most fundamental problems in computer science is k-Clique: given an n-node graph, decide if there are k nodes that form a clique, i.e. that have all the $\binom{k}{2}$ edges between them. Our interest is in the case where $3 \leq k \ll n$ is a small constant. This is the "SAT of parameterized complexity" being the canonical problem of the W[1] class of "fixed parameter intractable" problems, and its basic nature makes it a core task in countless applications where we seek a small sub-structure defined by pairwise relations.

The naïve algorithm checks all subsets of k nodes and runs in $O(k^2\binom{n}{k})$ time, which is $\Theta(n^k)$ for constant k. Whether and how this bound can be beaten (in terms of worst-case asymptotic time complexity) is a quintessential form of the question: *can we beat exhaustive search?*

* This work is part of the project CONJEXITY that has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101078482). The first author is additionally supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.

The asymptotically fastest algorithms gain a speedup by exploiting fast matrix multiplication – one of the most powerful techniques for beating exhaustive search. In particular, for the important special case of $k = 3$, i.e. the *Triangle Detection* problem, the running time is $O(n^\omega)$ where $2 \leq \omega < 2.3719$ [35] is the exponent in the time complexity of multiplying two $n \times n$ binary matrices.[1] For larger $k > 3$, there is a reduction to the $k = 3$ case by Nešetřil and Poljak [48] that produces graphs of size $O(n^{\lceil k/3 \rceil})$.[2] The resulting time bound is $O(n^{\lceil \omega k/3 \rceil})$. Except for improvements for k that is not a multiple of 3 [36], and the developments in fast matrix multiplication algorithms reducing the value of ω over the years, this classical algorithm remains the state-of-the-art.

The one general technique underlying all fast matrix multiplication, starting with Strassen's algorithm [51], is to find some clever formula to exploit cancellations in order to replace multiplications with additions. To date, this is *the only* technique capable of beating exhaustive search by a polynomial n^ε factor for the k-Clique problem. All techniques have their limitations, and so does Strassen's. Consequently, much research has gone into finding "combinatorial algorithms" that beat exhaustive search by other techniques (see Section 1.2 below). Existing techniques have only led to polylogarithmic speedups, leading the community to the following conjectures that have become the basis for many conditional lower bounds.

Conjecture 1 (Combinatorial BMM). *Combinatorial algorithms cannot solve Triangle Detection in time $O(n^{3-\varepsilon})$ where $\varepsilon > 0$.*[3]

A reduction of Vassilevska and Williams [54] shows that this conjecture is *equivalent* to the classical conjecture that combinatorial algorithms cannot solve Boolean Matrix Multiplication (BMM) in truly subcubic time [44,50]. Following their reduction, many conditional lower bounds were based on this conjecture, e.g. [8,31,29,26] (we refer to the survey [53] for a longer list).

Conjecture 2 (Combinatorial k-Clique). *Combinatorial algorithms cannot solve k-Clique in time $O(n^{k-\varepsilon})$ for any $k \geq 3$ and $\varepsilon > 0$.*

The latter conjecture is stronger than the former, in the sense that faster algorithms for $k = 3$ imply faster algorithms for larger $k > 3$ but the converse is not known. The first use of this conjecture as a basis for conditional lower bounds was by Chan [27] to prove an $n^{k-o(1)}$ lower bound for a problem in computational geometry. Later, Abboud, Backurs, and Vassilevska Williams [2] used it to prove $n^{3-o(1)}$ lower bounds in P. Several other papers have used it since then, e.g. [30,23,46,24,1,21,33,5,20,17,40].

[1] Simply compute A^2 where A is the adjacency matrix of the graph and check if $A^2[i,j] > 0$ for any $\{i,j\}$ that are an edge.

[2] Each $k/3$-clique becomes a node and edges are defined in a natural way so that a triangle corresponds to a k-clique.

[3] Note the informality in these combinatorial conjectures stemming from the lack of precise definition for "combinatorial" in this context. See full paper for further discussion.

Previous Combinatorial Bounds. The previous bounds for Triangle detection ($k = 3$) fall under three conceptual techniques. We open with an overview of these techniques (see Section 2.1 for a more detailed review). Note that ($\log \log n$) factors are omitted in this paragraph.

1. The *Four-Russians technique* [12] from 1970 gives an $O(n^3/\log^2 n)$ bound, and is used in all later developments.
2. In 2010, Bansal and Williams [15] use *pseudoregular partitions* to shave off an additional $\log^{1/4} n$ factor.
3. In 2014, Chan [28] introduced a simple *divide-and-conquer technique* to get an $O(n^3/\log^3 n)$ bound, and a year later, Yu [56] optimized this technique to achieve a bound of $O(n^3/\log^4 n)$.

For $k > 3$ there are two options: (1) we either apply these algorithms inside the aforementioned reduction to Triangle, getting a bound of $O(n^k/\log^4 n)$, or (2) we apply these combinatorial techniques directly to k-Clique. An early work of Vassilevska [52] from 2009 applied the Four-Russians technique directly to get an $O(n^k/\log^{k-1})$ bound. Note that this generalizes the $\log^2$ shaving from the first bullet naturally to all k, and is favorable to the algorithms from option (1) for $k > 5$. Vassilevska's bound remains state-of-the-art, and in this work, we address the challenge of generalizing the other combinatorial techniques to k-Clique.

1.1 Our Results

The first result of this paper is a faster combinatorial algorithm for k-Clique for all $k > 3$ based on a generalization of the divide-and-conquer technique from Chan's and Yu's algorithms for $k = 3$. We use divide-and-conquer to design a *more efficient reduction* from k-Clique to the $k = 3$ case. The main feature of this reduction is that we get an additional log factor shaving each time k increases by one; this should be contrasted with the classical reduction from option(1) above, in which we gain nothing when k grows.

Theorem 1 (Reduction from k-Clique to Triangle). *Let $k \geq 3$, and let a, b be reals such that there is a combinatorial triangle detection algorithm running in time $O(n^3(\log n)^a(\log\log n)^b)$. Then there is a combinatorial k-clique detection algorithm in time $O(n^k(\log n)^{a-(k-3)}(\log\log n)^{b+k-3})$.*

Combining our reduction with Yu's state-of-the-art combinatorial algorithm for Triangle detection, we improve Vassilevska's bound by two log factors.

Corollary 1 (Faster Combinatorial k-Clique Detection). *There is a k-clique detection algorithm running in time $O(n^k(\log n)^{-(k+1)}(\log\log n)^{k+3})$.*

It may be interesting to note that our reduction can even be combined with the naïve $O(n^3)$ algorithm for Triangle detection, giving a $(\log n)^{k-3}$ shaving for k-Clique *without using the Four-Russians technique.*

Another interesting implication of our reduction is concerning the framework of Bansal and Williams' [15]. Their algorithm can be improved if better dependencies for regularity/triangle removal lemmas are achieved. The best known upper bound on $f(\varepsilon)$ in a triangle removal lemma is of the form $\frac{c}{(\log^*(1/\varepsilon))^\delta}$ for some constants $c > 1$ and $\delta > 0$.[4]. Due to this dependency, their first algorithm [15, Theorem 2.1] only shaves a $\log^*(n)$ factor from the running times achieved with the standard Four-Russians technique. However, it is not ruled out that much better dependencies can be achieved that would accelerate their algorithm to the point where, combined with our reduction, a k-clique algorithm with faster running times than Corollary 1 is obtained.[5]

As discussed in Section 1.2, a primary reason to seek combinatorial algorithms for k-Clique is that the techniques may generalize in ways fast matrix multiplication cannot (see full paper for detailed discussion). Our second set of results exhibits this phenomenon by shaving logarithmic factors over state-of-the-art for general (non-combinatorial) algorithms.

One limitation of the $O(n^\omega)$ algorithm for Triangle detection is that it does not solve the *Triangle listing* problem: we cannot specify a parameter t and get all triangles in the graph in time $O(n^\omega + t)$ assuming their number is up to t. Listing triangles in an input graph is not only a natural problem, but it is also connected to the fundamental 3SUM problem (given n numbers, decide if there are three that sum to zero). A reduction from 3SUM [49,41] shows that in order to beat the longstanding $O(n^2/\log^2 n)$ bound over integers [16] it is enough to shave a $\log^{6+\varepsilon} n$ factor for Triangle listing – i.e., achieve a running time of $O(\frac{n^3}{\log^{6+\varepsilon} n} + t)$ for some $\varepsilon > 0$. Although research has seen some results on triangle listing [19], we are not aware of any previous $o(n^3) + O(t)$ time bound for this problem (even with non-combinatorial techniques). Our second result produces such a time bound, showing that the other combinatorial techniques (namely Four-Russians and regularity lemmas) can be exploited. We shave a $\log^{2.25} n$ factor for this problem, generalizing the Bansal-Williams bound for BMM. Note we use the non-standard notation $\widetilde{O}(n) = n(\log\log n)^{O(1)}$ to suppress polyloglog factors.

Theorem 2 (Faster Triangle Listing). *There is a randomized combinatorial algorithm that lists up to t triangles in a given graph in time $\widetilde{O}(\frac{n^3}{(\log n)^{2.25}} + t)$, and succeeds with probability $1 - n^{-100}$.*

Another well-known limitation of Strassen-like techniques is that they are ineffective for detecting hypergraph cliques. They fail to give any speedup even for the first generalization in this direction: detecting a 4-clique in a 3-uniform hypergraph (i.e. a hypergraph where each hyper-edge is a set of three nodes). We are not aware of any non-trivial $o(n^4)$ algorithm for this problem (even with non-combinatorial techniques). The conjecture that $O(n^{4-\varepsilon})$ time cannot be achieved

[4] Fox achieved some improved dependencies with a new proof of the removal lemma [37], however, it is not clear whether it can be implemented efficiently.

[5] Note that the same cannot be said about their second algorithm [15, Theorem 2.1]; see the lower bound for pseudoregular partitions due to Lovasz and Szegedy [47]).

has been used to prove conditional lower bounds, e.g. [46,25]. Our third result is a $\log^{1.5} n$ factor shaving for this problem. The following theorem provides our general bound and strengthens the result for listing (detection can be obtained by setting $t = 1$).

Theorem 3 (Faster k-Hyperclique Listing). *There is an algorithm for listing up to t k-hypercliques in an r-uniform hypergraph in time*

$$O\left(\frac{n^k}{(\log n)^{\frac{k-1}{r-1}}} + t\right)$$

(assuming a word RAM model with word size $w = \Omega(\log n)$).

Subsequent Work. Shortly after this work, Abboud, Fischer, Kelly, Lovett, and Meka announced a combinatorial algorithm for BMM with $O(\frac{n^3}{2^{(\log n)^{\varepsilon}}})$ running time [4]. This implies an improvement for k-Clique as well that is stronger than any poly-log speedup and thus improves over Corollary 1 (by using pseudo-regularity techniques rather than divide-and-conquer). Moreover, building on our proof of Theorem 2 the authors present a speedup for triangle listing as well. However, our result for hypergraphs in Theorem 3 remains unbeaten.

1.2 On Combinatorial Algorithms

While a single satisfying definition of "combinatorial algorithms" remains elusive, we believe the search for such algorithms to be of great importance and that the research entertaining this loosely defined notion has already been invaluable. To explain this, let us review the limitations of the Strassen-like technique that motivate us to seek other techniques. The first two motivations in this list are most commonly mentioned in works on the subject; however, we believe that the latter two are no less important.

One could ask for a specific definition of a "combinatorial" algorithm satisfying each motivation. Indeed, each such definition would be of some interest, and in fact, it has already been accomplished to some extent (as we discuss below). However, satisfying one definition does not guarantee satisfying the others; an algorithm that generalizes in one setting may not generalize in another or may not be practical or elegant. At this point of history, in which we are very much interested in *any* new technique breaking Conjectures 1 and 2 and satisfying *any* of the motivations, it is natural that the community prefers to work with the most inclusive definition ("anything but Strassen-like techniques" or "cancellation-free") that is inevitably loose.

1. *Practical efficiency:* Strassen's algorithm, and especially its successors, are not as efficient in practice as their asymptotic complexity suggests. This has two reasons: (1) the hidden factors are large, and (2) it is not cache-friendly since it frequently needs to access remote parts of the matrices. It is hoped

that achieving comparable running times (and not just log shavings) with combinatorial techniques will lead to gains in the real world. This concern has motivated some of the classical works on combinatorial algorithms such as [44,10]. A definition for this specific notion may have to be empirical, i.e. "does it work well in practice?"

2. *Elegance:* The matrix multiplication algorithms may be considered inelegant and uninterpretable since it is very hard to grasp the relationship between the intermediate values computed throughout the execution with the very simple problem at hand (e.g. detecting a triangle). This concern is probably at the origin of the name "combinatorial" where one imagines algorithms that only deal with natural combinatorial structures of the input graph such as the neighborhoods of nodes. Formal models of computations that aim to capture this definition have been suggested and analyzed [11,34] (strong lower bounds were obtained). They are unfortunately not satisfying *yet* because they do not capture the recent algorithms based on divide-and-conquer or regularity partitions that are widely acknowledged as "combinatorial" in the same sense they aim to address.

3. *Generalizability:* Fine-grained complexity has isolated the Triangle detection problem as the most basic form of hundreds of problems; several of its generalizations are the subject of conjectures that form the basis of conditional lower bounds for dozens of problems. Despite decades of efforts, Strassen-like techniques have failed to achieve speedups for some of these generalizations, and one may hope that a different technique breaking Conjecture 1 will also break the corresponding conjectures (that have nothing to do with "combinatorial algorithms"). For example:
   - An algorithm that generalizes to *weighted* graphs (where we ask for the minimum weight triangle) would break the *All-Pairs Shortest-Paths Conjecture* (APSP) [54]. A corresponding conjecture for $k > 3$ is also popular (e.g. [9,13,14,21]).
   - An algorithm that generalizes to hypergraphs (e.g. to find a 4-clique in a hypergraph) would break the *Hyper-Clique Conjecture* [46,3].
   - An algorithm that lets us output witnesses efficiently (e.g. to listing all triangles) would break both the *3SUM Conjecture* [49,42] and the APSP Conjecture [55]. Similar connections exist for $k > 3$ [32].
   - An algorithm that generalizes to the online setting (where the nodes are revealed one at a time) would break the *Online Matrix Vector Conjecture* [39,43]. Jin and Xu have recently introduced a corresponding conjecture for $k > 3$ [40].

   Note that a precise definition of "combinatorial" for each specific motivation can readily be made by asking for an algorithm that refutes the problem in parenthesis (and the other conjectures have essentially done this).

4. *Being the right technique:* It is plausible that Triangle detection can be solved in $O(n^{2+o(1)})$ time (in particular, if $\omega = 2$) and that k-Clique can be solved

in $n^{2k/3}$ or $n^{k/2}$ or perhaps even faster. Achieving that with the Strassen-like technique has been tedious. One may hope that if we get a small speedup such as $n^{2.9}$ with the *right* technique, we would quickly be able to optimize it and reach the final goal of $O(n^2)$. Anecdotally, one could say that such a discovery was recently made for the All-Pairs Max-Flow problem where a 60-year-old bound was broken by an $O(n^{2.8334})$ combinatorial algorithm [7] (for unweighted graphs) and a complete resolution with an $n^{2+o(1)}$ time bound quickly followed [6] (even for weighted graphs).

We emphasize that a "combinatorial algorithm" is expected to satisfy at least one of these motivations, but not all of them. Indeed, some of the contributions of this work is to show that some of the combinatorial techniques do generalize to some settings. Specifically, Theorem 3 shows that the Four-Russians technique applies in *hyper-graphs*, and Theorem 2 shows that regularity partitions can be used to give a *listing* algorithm.

On Combinatorial Lower Bounds. Finally, we would like to remark on the value of the related work on "*combinatorial conditional lower bounds*" that are based on Conjectures 1 and 2, proved by designing "combinatorial reductions".

Imagine you are an algorithm researcher working on a problem A (e.g. RNA Folding). You wake up in the morning and try technique T_1 on A. It does not work, so you try T_2, and then T_3 and so on up to T_{100} where 100 represents the number of relevant techniques. Each time, you may spend days or weeks trying the technique on A, searching for a way to fit into the right framework. Perhaps you hire new students to work on this fruitlessly each time. Now, suppose someone comes and tells you, because of a reduction from problem B (k-Clique) to problem A, that it is useless to try any of $T_1, ..., T_{100}$ on A, because all the experts have already tried them on B, without any success, and your problem is at least as hard. The next morning, you wake up knowing that your choices are either (1) solve A with fast matrix multiplication (the only technique found to be effective against B), or (2) invent new techniques but try them on k-Clique first (since its bare-bones nature compared to A makes it a better test-bed).

Success stories of such lower bounds leading to the discovery of breakthroughs exploiting fast matrix multiplication include the first $n^{3-\varepsilon}$ algorithm for RNA Folding [22], the first $n^{k-\varepsilon}$ algorithm for minimum k-cut [45], and new techniques in pattern matching [18].

This work contributes to this line of work by providing an algorithmic attack on Conjecture 2.

1.3 Outline

We start with some preliminaries in Section 2. In Section 3 we provide our improved combinatorial k-Clique algorithm. In Sections 4 and 5 we provide the high-level ideas of our improvements for Triangle Listing and k-Hyperclique Detection; due to space constraints we are forced to defer the technical details to the the full paper.

2 Preliminaries

Let $[n] := \{1, \ldots, n\}$. We write $\widetilde{O}(n) = n(\log n)^{O(1)}$ to suppress polylogarithmic factors and use the non-standard notation $\widetilde{\widetilde{O}}(n) = n(\log \log n)^{O(1)}$.

Throughout we consider undirected, unweighted graphs. In the k-clique problem, we are given a k-partite graph $(V_1, \ldots, V_k, E)$ and the goal is to determine whether there exist k vertices $v_1 \in V_1, \ldots, v_k \in V_k$ such that there is an edge $(v_i, v_j) \in E$ for every pair $i \neq j$. Note that the assumption that the input graphs are k-partite is without loss of generality, and can be achieved by a trivial transformation of any non-k-partite graph $G = (V, E)$: We create k copies $V_1, \ldots, V_k$ of the vertex set and for every $(u, v) \in E$ we add the edges (u_i, v_j) for every $i \neq j$. Another typical relaxation is that we only design an algorithm that *detect* the presence of k-cliques (without actually returning one). It is easy to transform a detection algorithm into a finding algorithm using binary search without asymptotic overhead.[6]

We additionally define the following notation for a k-partite graph as before: For a vertex v, let $N_i(v) = \{u \in V_i : (v, u) \in E\}$ denote the neighbourhood of v in V_i and $d_i(v) = |N_i(v)|$ denote the degree of v in V_i. Moreover, for a vertex set $V' \subseteq V$ we let $G[V']$ denote the subgraph of G induced by the vertex set V'. Throughout we further let $n = |V_1| + \cdots + |V_k|$ denote the total number of vertices in the graph.

An *r-uniform hypergraph* is a pair (V, E), where V is a vertex set and $E \subseteq \binom{V}{r}$ is a set of *hyperedges*. In the *r-uniform k-hyperclique* problem we need to decide whether in a k-partite hypergraph $(V_1, \ldots, V_k, E)$ there are vertices $v_1 \in V_1, \ldots, v_k \in V_k$ such that all hyperedges on $\{v_1, \ldots, v_k\}$ are present. Similarly, the assumption that the hypergraph is a k-partite is without loss of generality.

We are using the standard word RAM model with word size $w \in \Theta \log(n)$. In this model a random-access machine can perform arithmetic and bitwise operations on w-bit words in constant time.

2.1 Overview of Previous Combinatorial Algorithms

The Four-Russians Method We start with presenting the most basic form of the Four-Russians method [12], and then move on to a specific variant which we will use throughout this paper. In the Four-Russians method, we start by precomputing the solutions for triangle detection on tiny instances. In particular, we precompute and store the solution to triangle detection on every tri-partite graph with vertex sets of size $s = 0.01\sqrt{\log(n)}$, say, and store the answers in

[6] More specifically, any detection algorithm can be transformed into a finding algorithm with constant running time overhead by using binary search as follows: Arbitrarily split each of the k vertex parts into two halves. Then for each subgraph induced by one of the 2^k combination of halves whether it contains a k-clique. If the detection algorithm succeeds on some combination, we continue on this combination recursively. For any natural running time the recursive overhead becomes a geometric sum and thus is constant.

a lookup table; note that there are only $n^{0.02}$ such graphs. Then, we use this precomputation to accelerate triangle detection on the input graph: We partition every vertex set in the input graph into blocks of size s. Then, for each triplet of blocks we query the lookup table with the subgraph induced by the triplet to obtain an answer to whether there exists a triangle among these blocks. Assuming a word-RAM model with $w \geq \Omega(\log n)$ (as is standard), and some additional preprocessing which allows to determine subgraphs on combinations of blocks efficiently, the running time is dominated by the number of queries to the lookup table which is $O(\frac{n^3}{(\log n)^{1.5}})$.

This basic idea can be slightly improved: Given a tri-partite graph $G = ((V_1, V_2, V_3), E)$, we begin by preprocessing the graph: We divide V_2 and V_3 into blocks of size $\varepsilon \cdot \log n$ for some small $\varepsilon > 0$. Now, for every pair (S, S') where S is a subset of a block in V_2 and S' is a subset of a block in V_3, we check whether there exists an edge between the two sets and store the answer in a lookup table. Again this preprocessing is affordable as each block has $O(n^{\varepsilon})$ subsets and the number of blocks in each vertex set is at most $O(\frac{n}{\log n})$. The total number of pairs (S, S') is thus $O(\frac{n^{2+2\varepsilon}}{(\log n)^2})$. To detect a triangle, we can now, for each $v \in V_1$, partition the neighborhoods of v in V_2 and V_3 into subsets of blocks, and test whether these subsets are connected by an edge using one query to the lookup table. In the positive case we have detected a triangle. For each vertex we perform $O(\frac{n^2}{(\log n)^2})$ queries and therefore the total running time is $O(\frac{n^3}{(\log n)^2})$. Interestingly, this technique can also be used to *list* all triangles in the graph: Instead of storing a flag in the lookup table, we explicitly store a list of all edges between the sets S and S'.

Vassilevska [52] demonstrated how the Four-Russians technique can be used for k-clique in a similar fashion: Partition $V_2, \ldots, V_k$ into blocks and for every $(k-1)$-tuple of blocks we precompute whether there is a $(k-1)$-clique among these blocks. This results in time $O(\frac{n^k}{(\log n)^{k-1}})$ for k-clique detection.

Other Approaches A line of work published in recent years obtained improvements beyond the Four-Russians bound. In the first such result, Bansal and Williams [15] combined regularity lemmas with the Four-Russians method. The basic intuition is that when looking at large (and therefore time-consuming) sets of neighbors of some vertex that lie within dense and random-like sets, there must exist an edge between the sets which reveals a triangle. They combined this intuition with the important observation that the Four-Russians method can be altered in a way that compounds sparseness in the graph (see Lemma 1), and were able to achieve a running time of $\widetilde{O}(\frac{n^3}{(\log(n))^{2.25}})$.

Chan [28] improved this further using an entirely different approach. His main observation was that if there is a high-degree node, we can check its neighbors for the existence of an edge (such an edge exists if and only if the vertex is involved in a triangle), and recurse smartly on significantly smaller subproblems. This divide-and-conquer scheme effectively reduces Triangle Detection in dense graphs to Triangle Detection in sparse graphs, and leads to a running time of $\widetilde{O}(\frac{n^3}{(\log n)^3})$. Yu [56] later refined this result and achieved a running time of $\widetilde{O}(\frac{n^3}{(\log(n))^4})$.

3 Combinatorial Log-Shaves for k-Clique

In this section we provide our improved algorithmic reduction from k-clique to triangle detection (see Theorem 1). In our core we follow a divide-and-conquer approach for k-clique reminiscent to Chan's algorithm for triangle detection [28] with a simple analysis. We start with the following observation:

Observation 1 (Trivial Reduction from k-Clique to $(k-1)$-Clique). *Let $k \geq 4$, let $f(n)$ be a nondecreasing function, and assume that there is a combinatorial $(k-1)$-clique detection algorithm running in time $O(n^{k-1}/f(n))$. Then there is a combinatorial k-clique detection algorithm running in time*

$$O\left(\sum_{v \in V_1} \frac{d_2(v) \cdot \dots \cdot d_k(v)}{f(\min\{d_2(v), \dots, d_k(v)\})}\right).$$

Proof. The algorithm is simple: For each vertex $v \in V_1$, we construct the subgraph $G_v = G[N_2(v) \cup \dots \cup N_k(v)]$ consisting of all neighbors of v and test whether G_v contains a $(k-1)$-clique. Let $n_v = d_2(v) + \dots + d_k(v)$ denote the number of vertices in G_v. Our intention is to use the efficient $(k-1)$-clique algorithm—however, simply running the algorithm in time $O(n_v^k/f(n_v))$ is possibly too slow. Instead, we partition each of the $k-1$ vertex parts in G_v into blocks of size $d_v := \min\{d_2(v), \dots, d_k(v)\}$ (plus one final block of smaller size, respectively). Then, for each combination of $k-1$ blocks, we use the efficient $(k-1)$-clique detection algorithm. It is clear that the algorithm is correct, since we exhaustively test every tuple $(v_1, v_2, \dots, v_k)$. For the running time, note that testing whether G_v contains a k-clique takes time

$$\left\lceil \frac{d_2(v)}{d_v} \right\rceil \cdot \dots \cdot \left\lceil \frac{d_k(v)}{d_v} \right\rceil \cdot O\left(\frac{(d_v)^{k-1}}{f(d_v)}\right) = O\left(\frac{d_2(v) \cdot \dots \cdot d_k(v)}{f(\min\{d_2(v), \dots, d_k(v)\})}\right),$$

and thus the total running time is indeed

$$O\left(\sum_{v \in V_1} \frac{d_2(v) \cdot \dots \cdot d_k(v)}{f(\min\{d_2(v), \dots, d_k(v)\})}\right)$$

(possibly after preprocessing the graph in time $O(n^2)$ to allow for constant-time edge queries. Note that this also covers the cost of constructing G_v for every $v \in V_1$). □

Before moving to the formal proof of Theorem 1, let us give a simplified high-level description of this algorithmic reduction in the specific case of 4-clique. For a given 4-partite graph (V_1, V_2, V_3, V_4), the core idea is the following: If the degrees in V_1 tend to be small, i.e. if for every $v \in V_1$ we have $d_2(v) \cdot d_3(v) \cdot d_4(v) \leq \alpha \cdot |V_2| \cdot |V_3| \cdot |V_4|$ for some fraction $\alpha \approx \frac{1}{\log n}$, then we can apply Observation 1. Otherwise, there is a *heavy* vertex $v \in V_1$ with $d_2(v) \cdot d_3(v) \cdot d_4(v) > \alpha \cdot |V_2| \cdot |V_3| \cdot |V_4|$. In this case, we will check every triplet of the form $(u, w, z) \in N_2(v) \times N_3(v) \times N_4(v)$.

If any of these triplets form a triangle, we have detected a 4-clique. Otherwise, we have learned that no triplet in $N_2(v) \times N_3(v) \times N_4(v)$ is part of a 4-clique. We will therefore recurse in such a way that ensures we never test these triplets again and thereby make sufficient progress.

Proof. Assume that there is a combinatorial triangle detection algorithm which runs in time $O(n^3(\log n)^a(\log\log n)^b)$. We prove the claim by induction on k. The base case ($k = 3$) is immediate by the assumption there exists a triangle detection algorithm running in time $O(n^3(\log n)^a(\log\log n)^b)$.

For the inductive step, consider the following recursive algorithm to detect a k-clique in a given k-partite graph $(V_1, \ldots, V_k, E)$. Let D and α be parameters to be determined later and let d be initialized to 0.

KCliqueRec$(G = (V_1, \ldots, V_k, E), d)$:

1. If $d = D$, meaning depth D in the recursion is reached, perform exhaustive search. Return YES if a k-clique was detected, otherwise NO.
2. Test whether there is some $v \in V_1$ with $d_2(v) \cdot \ldots \cdot d_k(v) \geq \alpha \cdot |V_2| \cdot \ldots \cdot |V_k|$. If such a vertex exists:
   a. Test whether the subgraph G_v induced by $N_2(v) \cup \cdots \cup N_k(v)$ contains a $(k-1)$-clique by exhaustive search. If it does return YES since this means we've found a k-clique involving v.
   b. For $2 \leq i \leq k$, partition V_i into $V_{i,0} = V_i \setminus N_i(v)$ and $V_{i,1} = V_i \cap N_i(v)$. Recursively solve the $2^{k-1} - 1$ subproblems on $(V_1, V_{2,i_2}, \ldots, V_{k,i_k})$ for $(i_2, \ldots, i_k) \in \{0,1\}^{k-1} \setminus \{1^{k-1}\}$, while incrementing the depth.
   In other words, for each $(i_2, \ldots, i_k) \in \{0,1\}^{k-1} \setminus \{1^{k-1}\}$, call KCliqueRec$(G[V_1 \cup V_{2,i_2} \cup \cdots \cup V_{k,i_k}], d+1)$.
   c. If any of the calls returned YES, return YES. Otherwise, return NO.
3. Solve the instance using Observation 1.

Correctness. As soon as the algorithm reaches recursion depth D, the algorithm will correctly detect a k-clique in step 1. In earlier levels of the recursion, the algorithm first attempts to find a vertex v with $d_2(v) \cdot \ldots \cdot d_k(v) \geq \alpha \cdot |V_2| \cdot \ldots \cdot |V_k|$ in step 2. If this succeeds, we test whether v is involved in a k-clique (and terminate in this case). Otherwise, we recurse on $(V_1, V_{2,i_2}, \ldots, V_{k,i_k})$ for all combinations $(i_2, \ldots, i_k) \in \{0,1\}^{k-1} \setminus \{1^{k-1}\}$. Note that we can indeed ignore the instance $(V_1, V_{2,1}, \ldots, V_{k,1})$ knowing that $(V_{2,1}, \ldots, V_{k,1})$ does not contain a $(k-1)$-clique. If the condition in step 2 is not satisfied, we instead correctly solve the instance by means of Observation 1 (which reduces the problem to an instance of $(k-1)$-clique).

Running Time. Imagine a recursion tree in which every node corresponds to an execution of the algorithm; the root corresponds to the initial call and child nodes correspond to recursive calls. Thus, every node in the tree is either a leaf (indicating that this execution does not spawn recursive calls), or an internal node with fan-out exactly $2^{k-1} - 1$. The *time* at a node is the running time of the respective call of the algorithm (ignoring the cost of further recursive calls).

In other words, the *time* at a node is the amount of local work performed in the corresponding call. To bound the total running time of the algorithm, we bound the total time across all nodes in the recursion tree.

We analyze the contributions of all steps individually. Let us introduce some notation first: At a node x in the recursion tree, let $(V_1^x, \ldots, V_k^x)$ denote the instance associated to the respective invocation. Similarly write $d_2^x(v), \ldots, d_k^x(v)$.

Cost of Step 1. Note that at any node x at depth D in the recursion tree, the time is $O(|V_1^x| \cdot \ldots \cdot |V_k^x|)$ since we solve the instance by exhaustive search. Next, observe that for any internal node x in the recursion tree, we have that

$$\begin{aligned}|V_1^x| \cdot \ldots \cdot |V_k^x| &= |V_1^x| \cdot \sum_{i_2,\ldots,i_k \in \{0,1\}^{k-1}} |V_{2,i_2}^x| \cdot \ldots \cdot |V_{k,i_k}^x| \\ &\geq |V_1^x| \cdot d_2^x(v) \cdot \ldots \cdot d_k^x(v) + \sum_{y \text{ child of } x} |V_1^y| \cdot \ldots \cdot |V_k^y| \\ &\geq \alpha \cdot |V_1^x| \cdot \ldots \cdot |V_k^x| + \sum_{y \text{ child of } x} |V_1^y| \cdot \ldots \cdot |V_k^y|,\end{aligned}$$

and thus

$$\sum_{y \text{ child of } x} |V_1^y| \cdot \ldots \cdot |V_k^y| \leq (1-\alpha) \cdot |V_1^x| \cdot \ldots \cdot |V_k^x|.$$

It follows by induction that at any depth $d \leq D$ in the recursion tree, we have that

$$\sum_{x \text{ at depth } d} |V_1^x| \cdot \ldots \cdot |V_k^x| \leq (1-\alpha)^d n^k.$$

In particular, the total time of all nodes at depth D is bounded by $O((1-\alpha)^D n^k)$.

Cost of Step 2. Note that the number of nodes in our recursion tree is at most 2^{kD} since the recursion tree has degree $\leq 2^k$ and the recursion depth is capped at D. At each node, the time of step 2a is bounded by $O(n^{k-1})$ and the cost of step 2b is bounded by $O(n^2)$. Therefore, the total time of step 2 across all nodes is bounded by $O(2^{kD} n^{k-1})$.

Cost of Step 3. By induction we have obtained a $(k-1)$-clique algorithm in time $O(n^{k-1}/f(n))$, where $f(n) = (\log n)^{-a+k-4}(\log\log n)^{-b-(k-4)}$. Therefore, by Observation 1 the total time of step 3 across all nodes x in the recursion tree is

$$O\left(\sum_{x \text{ leaf}} \sum_{v \in V_1^x} \frac{d_2^x(v) \cdot \ldots \cdot d_k^x(v)}{f(\min\{d_2^x(v), \ldots, d_k^x(v)\})}\right).$$

To bound this quantity, we distinguish two subcases: A pair (x, v) (where x is a leaf in the recursion tree and $v \in V_1^u$) is called *relevant* if $d_2^x(v), \ldots, d_k^x(v) \geq \sqrt{n}$

(where n is the initial number of nodes). On the one hand, it is easy to bound the total cost of all irrelevant pairs by

$$O\left(\sum_{(x,\,v)\text{ irrelevant}} \frac{d_2^x(v)\cdot\ldots\cdot d_k^x(v)}{f(\min\{d_2^x(v),\ldots,d_k^x(v)\})}\right) \leq O(2^{kD}n^{k-1/2}),$$

since there are at most 2^{kD} nodes in the recursion tree. On the other hand, for any relevant pair (x,v), we have $\min\{d_2^x(v),\ldots,d_k^x(v)\} \geq \sqrt{n}$. Moreover, since we reach step 3 of the algorithm we further know that $d_2^x(v)\cdot\ldots\cdot d_k^x(v) \leq \alpha|V_2^x|\cdot\ldots\cdot|V_k^x|$ (as otherwise the condition in step 2 had triggered). It follows that

$$\begin{aligned}
&O\left(\sum_{(x,\,v)\text{ relevant}} \frac{d_2^x(v)\cdot\ldots\cdot d_k^x(v)}{f(\min\{d_2^x(v),\ldots,d_k^x(v)\})}\right)\\
&\quad\leq O\left(\sum_{(x,\,v)\text{ relevant}} \frac{\alpha|V_2^x|\cdot\ldots\cdot|V_k^x|}{f(\sqrt{n})}\right)\\
&\quad\leq O\left(n^k\cdot\frac{\alpha}{f(\sqrt{n})}\right).
\end{aligned}$$

Choosing the Parameters. Summing over all contributions computed before, the total running time is bounded by

$$O\left(n^k\cdot(1-\alpha)^D + n^k\cdot\frac{\alpha}{f(\sqrt{n})} + n^{k-1/2}\cdot 2^{kD}\right).$$

We pick $D = \log n/(4k)$ such that the latter term becomes $n^{k-1/4}$. Next, we pick $\alpha = \log((-a+k)\log n)/D = \Theta((\log n)^{-1}\log\log n)$ such that the first term becomes

$$n^k\cdot(1-\alpha)^D \leq n^k\cdot 2^{-\alpha D} \leq n^k(\log n)^{a-k}.$$

All in all, the total running time is dominated by the second term

$$\begin{aligned}
n^k\cdot\frac{\alpha}{f(\sqrt{n})} &\leq O(n^k\cdot\alpha\cdot(\log n)^{a-(k-4)}(\log\log n)^{b+k-4})\\
&\leq O(n^k(\log n)^{a-(k-3)}(\log\log n)^{b+k-3}),
\end{aligned}$$

which is as claimed. □

4 Combinatorial Log-Shaves for Triangle Listing by Weak Regularity

In this section we reformulate and reanalyze Bansal and Williams' BMM algorithm [15] to support triangle *listing*. Specifically, they provide a combinatorial algorithm for Boolean matrix multiplication in time $\widetilde{\widetilde{O}}(\frac{n^3}{(\log n)^{2.25}})$, and our adaptation is as follows:

Theorem 2 (Faster Triangle Listing). *There is a randomized combinatorial algorithm that lists up to t triangles in a given graph in time $\widetilde{\widetilde{O}}(\frac{n^3}{(\log n)^{2.25}} + t)$, and succeeds with probability $1 - n^{-100}$.*

Note that we cannot achieve this running time by applying state-of-the-art black-box reductions from triangle listing to Boolean matrix multiplication ([54]). Before we can present the algorithm we will need to define the notion of pseudoregular partitions in graphs and present a lemma by Frieze and Kannan [38] known as the weak regularity lemma.

4.1 Pseudoregularity

Let $G = (V, E)$ be a graph and let $S, T \subseteq V$ be disjoint subsets of vertices. We define the *density* of (S, T) as

$$\delta(S,T) = \frac{e(S,T)}{|S| \cdot |T|},$$

where

$$e(S,T) = |\{(u,v) \in E : u \in S, v \in T\}|.$$

Fix a partition $\mathcal{P} = (V_1, \ldots, V_k)$ of the vertices V; we call the parts $V_1, \ldots, V_k$ the *pieces* of $\mathcal{P}$. For the sake convenience, we often employ the following shorthand notation: For a subset $S \subseteq V$ and $i \in [k]$, write $S_i := S \cap V_i$. Moreover, we set $\delta_{i,j} := \delta(V_i, V_j)$. The partition $\mathcal{P}$ is called *ε-pseudoregular* if for every pair of disjoint subsets $S, T \subseteq V$ we have

$$\left| e(S,T) - \sum_{i,j \in [k]} \delta_{i,j} \cdot |S_i| \cdot |T_j| \right| \leq \varepsilon n^2.$$

We further call the partition $\mathcal{P}$ *equitable* if the sizes of the vertex parts differ by at most 1 (i.e., $| \, |V_i| - |V_j| \, | \leq 1$ for all $i, j \in [k]$).

We can think about pseudoregularity in the following way: We compare the density between the pieces to the density of subsets of these pieces. In a random graph these densities are expected to be equal. If this was indeed the case, then $e(S,T) = \sum_{i,j \in [k]} \delta_{i,j} |S_i| \cdot |T_j|$. Pseudoregularity requires that these quantities are only approximately equal, allowing an error term of εn^2.

The following weak regularity lemma by Frieze and Kannan [38] shows that we can always find a pseudoregular partition with relatively few parts, and that such a partition can be computed efficiently:

Theorem 4 ([38, Theorem 2 and Section 5.1]). *Let $\varepsilon, \delta > 0$. Given an n-vertex graph, we can construct a ε-pseudoregular partition with at most $2^{O(1/\varepsilon)}$ pieces in time $2^{O(1/\varepsilon^2)} \cdot n^2 \cdot \varepsilon^{-2} \cdot \delta^{-3}$ by a randomized algorithm that succeeds with probability at least $1 - \delta$.*

4.2 Triangle Listing

We now turn to the triangle listing algorithm à la Bansal-Williams [15], starting with an informal overview. For a given tripartite graph $G = (V_1, V_2, V_3, E)$, we first compute an ε-pseudoregular partition of the bipartite graph $G[V_2 \cup V_3]$. We then distinguish between two types of pieces—pieces with low density (less than $\sqrt{\varepsilon}$) and pieces with high density. Based on this we divide the instance into two triangle listing instances—G_L which only includes edges connecting low density parts between V_2 and V_3 in G, and its complement G_H consisting of edges connecting the high-density parts between V_2 and V_3. In the former case we can exploit the sparsity (by construction the total number of edges G_L is at most $\sqrt{\varepsilon}n^2$). In the latter case, due to the pseudoregularity, there must be many triangles in G_H. We can thus charge the extra cost of computing with G_H towards the output-size.

The key ingredient is the following Lemma 1 that applies the Four-Russians method to *sparse* graphs. Recall that the standard Four-Russians technique we presented in the introduction leads to a triangle listing algorithm in time $O(\frac{n^3}{\log^2(n)} + t)$.

Lemma 1 (Sparse Four-Russians). *There is an algorithm which lists up to t triangles in a given graph (V_1, V_2, V_3, E) (with $n = \min\{|V_1|, |V_2|, |V_3|\}$) in time*

$$\widetilde{\widetilde{O}}\left(\frac{|V_1| \cdot |V_2| \cdot |V_3|}{(\log n)^{100}} + \sum_{v \in V_1} \frac{d_2(v) \cdot d_3(v)}{(\log n)^2} + t\right).$$

Proof. The idea is similar to the standard Four-Russians method: Let s, Δ be parameters to be determined. We partition V_2 into $g = \lceil |V_2|/s \rceil$ blocks $V_{2,1}, \ldots, V_{2,g}$ of size at most s, and similarly partition V_3 into $h = \lceil |V_3|/s \rceil$ blocks $V_{3,1}, \ldots, V_{3,h}$. Then consider the following three steps:

1. We precompute, for each pair $i \in [g], j \in [h]$ and for each pair of subsets $S, T \subseteq [s]$ with size at most $|S|, |T| \leq \Delta$, the list of all edges in the induced graph on the vertex sets $V_{2,i} \cap S$ and $V_{3,j} \cap T$. Here, to represent a set $S \subseteq [s]$ of size Δ we fix a bit-representation consisting of at most $\lceil \log \binom{s}{\Delta} \rceil$ bits. We will later pick s and Δ in such a way that $\binom{s}{\Delta} \leq n^{0.1}$; note that any such set S can be represented by $O(1)$ machine words.
2. For each vertex $v \in V_1$, we write $N_{2,i} = N_2(v) \cap V_{2,i}$ and $N_{3,j} = N_3(v) \cap V_{3,j}$. Let us further set $d_{2,i} = |N_{2,i}|$ and $d_{3,j} = |N_{3,j}|$. We arbitrarily partition the set $N_{2,i}(v)$ into $\lceil \frac{d_{2,i}}{\Delta} \rceil$ subsets of size at most Δ; denote this partition by $\mathcal{S}_i(v)$. We also compute a partition $\mathcal{T}_j(v)$ of $N_{3,j}(v)$ with analogous properties. Here we represent each set $S \in \mathcal{S}(v), T \in \mathcal{T}(v)$ in its bit-representation.
3. For each $v \in V_1$, for each $i \in [g], j \in [h]$, and for each $S \in \mathcal{S}_i(v), T \in \mathcal{T}_j(v)$, list all edges $(v_2, v_3) \in S \times T$ (as precomputed in the first step). For each such edge, we report (v, v_2, v_3) as a triangle. (If at some point we have listed t triangles, we stop the algorithm.)

The correctness is straightforward: For each triangle (v, v_2, v_3), we eventually test the indices $i \in [g], j \in [h]$ and sets $S \in S_i(v), T \in \mathcal{T}_j(v)$ where $v_2 \in S$ and $v_3 \in T$.

To analyze the running time, we choose the parameters $s = \lfloor (\log n)^{100} \rfloor$ and $\Delta = \lfloor \frac{\log n}{1000 \log\log n} \rfloor$. Note that for this choice we indeed have

$$\binom{s}{\Delta} \leq s^{\Delta} = (\log n)^{\frac{100 \log n}{1000 \log\log n}} = n^{0.1}.$$

Moreover, the precomputation produces a table of size $O(g^2 \cdot \binom{s}{\Delta}^2) = O(g^2 n^{0.2})$ and can be performed in time $O(g^2 \cdot \binom{s}{\Delta}^2 \cdot s^2) = O(n^{2.2})$. The second step takes time $O(n^2 s) = \widetilde{O}(n^2)$. Finally, in the third step we spend time

$$\begin{aligned}
&O\left(\sum_{v \in V_1} \sum_{i \in [g]} \sum_{j \in [h]} |\mathcal{S}_i(v)| \cdot |\mathcal{T}_j(v)| + t\right) \\
&\quad = O\left(\sum_{v \in V_1} \left(\sum_{i \in [g]} \left\lceil \frac{d_{2,i}(v)}{\Delta} \right\rceil\right) \cdot \left(\sum_{j \in [h]} \left\lceil \frac{d_{3,j}(v)}{\Delta} \right\rceil\right) + t\right) \\
&\quad = O\left(|V_1| \cdot |V_2| \cdot h + |V_1| \cdot g \cdot |V_3| + \sum_{v \in V_1} \frac{d_2(v) \cdot d_3(v)}{\Delta^2} + t\right) \\
&\quad = O\left(\frac{|V_1| \cdot |V_2| \cdot |V_3|}{(\log n)^{100}} + \sum_{v \in V_1} \frac{d_2(v) \cdot d_3(v)}{\Delta^2} + t\right),
\end{aligned}$$

which is as claimed. □

Corollary 2 (Sparse Four-Russians). *There is an algorithm which lists up to t triangles in a given graph (V_1, V_2, V_3, E) (with $n = \min\{|V_1|, |V_2|, |V_3|\}$) in time*

$$\widetilde{\widetilde{O}}\left(\frac{|V_1| \cdot |V_2| \cdot |V_3|}{(\log n)^{100}} + \frac{n \cdot e(V_2, V_3)}{(\log n)^2} + t\right).$$

Proof. Apply the previous lemma to the graph (V_2, V_1, V_3, E). Since we can bound

$$\sum_{v \in V_2} \frac{d_1(v) \cdot d_3(v)}{(\log n)^2} \leq \frac{n e(V_2, V_3)}{(\log n)^2},$$

the claim follows. □

Proof of Theorem 2. As a first step, we show how to list *all* triangles in the given graph in time $\widetilde{\widetilde{O}}(\frac{n^3}{(\log n)^{2.25}} + t_G)$, where t_G is the total number of triangles in G. We will later remove this restriction.

Let $U = V_2 \cup V_3$ and consider the bipartite graph $G[U]$. Let $\varepsilon > 0$ be a parameter to be determined later. We apply Theorem 4 to $G[U]$ with parameters ε

and $\delta = n^{-0.1}$ to compute an ε-pseudoregular partition $\mathcal{P} = (U_1, \dots, U_k)$ into at most $k = 2^{O(1/\varepsilon^2)}$ pieces. Then, for each pair $i, j \in [k]$, construct the tripartite induced subgraph on V_1, $V_{2,i} := V_2 \cap U_i$ and $V_{3,j} := V_3 \cap U_j$, and use Lemma 1 or Corollary 2 to list all triangles in that graph (whichever is faster).

The correctness is immediate (and we will analyze the success probability later). It remains to argue that this algorithm runs in the claimed time budget. We need some notation: For a node $v \in V_1$ and $i \in [k]$, let $N_{2,i}(v) = N_1(v) \cap U_i$ and let $d_{2,i} = |N_{2,i}(v)|$ (and similarly for $N_{3,i}, d_{3,i}$). Moreover, as before, let $\delta_{i,j} := \delta(U_i, U_j)$ denote the density between the pieces U_i and U_j. Let us ignore the preprocessing for now. Then, writing

$$T_{i,j} = \min\left\{ \sum_{v \in V_1} \frac{d_{2,i}(v) \cdot d_{3,j}(v)}{(\log n)^2}, \frac{n \cdot e(V_{2,i}, V_{3,j})}{(\log n)^2} \right\},$$

the running time is bounded by

$$\begin{aligned} &\widetilde{\widetilde{O}}\left(\sum_{i,j \in [k]} \left(n^2 + \frac{|V_1| \cdot |V_{2,i}| \cdot |V_{3,j}|}{(\log n)^{100}} + T_{i,j} \right) + t_G \right) \\ &\quad = \widetilde{\widetilde{O}}\left(k^2 n^2 + \frac{n^3}{(\log n)^{100}} + \sum_{i,j \in [k]} T_{i,j} + t_G \right). \end{aligned}$$

By setting $\varepsilon = \Theta(\frac{1}{\sqrt{\log n}})$ for an appropriately small hidden constant, this becomes

$$= \widetilde{\widetilde{O}}\left(n^{2.5} + \frac{n^3}{(\log n)^{100}} + \sum_{i,j \in [k]} T_{i,j} + t_G \right).$$

It remains to bound the sum $\sum_{i,j} T_{i,j}$. To this end we distinguish two cases: On the one hand, we have that

$$\sum_{\substack{i,j \in [k] \\ \delta_{i,j} \le \sqrt{\varepsilon}}} T_{i,j} \le \sum_{\substack{i,j \in [k] \\ \delta_{i,j} \le \sqrt{\varepsilon}}} \frac{n \cdot e(V_{2,i}, V_{3,j})}{(\log n)^2} \le \sum_{\substack{i,j \in [k] \\ \delta_{i,j} \le \sqrt{\varepsilon}}} \frac{\sqrt{\varepsilon} n \cdot |V_{2,i}| \cdot |V_{3,j}|}{(\log n)^2} \le \frac{\sqrt{\varepsilon} n^3}{(\log n)^2}.$$

On the other hand,

$$\begin{aligned} \sum_{\substack{i,j \in [k] \\ \delta_{i,j} > \sqrt{\varepsilon}}} T_{i,j} &\le \sum_{\substack{i,j \in [k] \\ \delta_{i,j} > \sqrt{\varepsilon}}} \sum_{v \in V_1} \frac{d_{2,i}(v) \cdot d_{3,j}(v)}{(\log n)^2} \\ &\le \frac{1}{\sqrt{\varepsilon}} \sum_{i,j \in [k]} \delta_{i,j} \sum_{v \in V_1} \frac{d_{2,i}(v) \cdot d_{3,j}(v)}{(\log n)^2} \\ &\le \frac{1}{\sqrt{\varepsilon}} \sum_{v \in V_1} \sum_{i,j \in [k]} \frac{\delta_{i,j} \cdot d_{2,i}(v) \cdot d_{3,j}(v)}{(\log n)^2} \end{aligned}$$

This is where finally the pseudoregularity becomes useful. Let $v \in V_1$ be arbitrary, let $S = N_2(v)$ and $T = N_3(v)$ (and write $S_i = N_{2,i}(v)$ and $T_j = N_{3,j}(v)$ as before). By the ε-pseudoregularity we have $\sum_{i,j\in[k]} \delta_{i,j}|S_i| \cdot |T_i| \leq e(S,T) + \varepsilon n^2$, and thus

$$\leq \frac{1}{\sqrt{\varepsilon}} \sum_{v\in V_1} \frac{\varepsilon n^2 + e(N_2(v), N_3(v))}{(\log n)^2}$$

Since clearly $\sum_{v\in V_1} e(N_2(v), N_3(v)) = t_G$, we finally obtain:

$$\leq \frac{\sqrt{\varepsilon} n^3}{(\log n)^2} + \frac{t_G}{\sqrt{\varepsilon}(\log n)^2}.$$

All in all, the remaining term in the running time can indeed be bounded by

$$\sum_{i,j\in[k]} T_{i,j} = O\left(\frac{\varepsilon n^3}{(\log n)^2} + \frac{t_G}{\sqrt{\varepsilon}(\log n)^2}\right) = O\left(\frac{n^3}{(\log n)^{2.25}} + t_G\right).$$

Finally, recall that the preprocessing (to compute the pseudoregular partition) runs in time $2^{O(1/\varepsilon^2)} \cdot n^2 \cdot \varepsilon^{-2} \cdot \delta^{-3} = 2^{O(1/\varepsilon^2)} \cdot \widetilde{O}(n^{2.3})$ (Theorem 4) which by our choice of ε becomes $O(n^{2.5})$, say.

Concerning the success probability, note that the only source of error in the algorithm is the computation of the pseudoregular partition which fails with probability at most $\delta = n^{-0.1}$. But even if computing this partition fails, we will nevertheless correctly report all triangles, and only the running time of our algorithm is affected. We can thus run our algorithm 1000 times in parallel and stop as soon as the first copy reports an answer. The probability that all calls exceed their time budget is at most $(n^{-0.1})^{1000} = n^{-100}$ as claimed.

It remains to remove the assumption that we list all triangles, and instead only list triangles up to a given threshold t. To this end, we apply the following preprocessing to a given tripartite graph (V_1, V_2, V_3, E): Split V_1 into $g := \Theta(\sqrt{n})$ blocks $V_{1,1}, \ldots, V_{1,g}$ of size $\Theta(\sqrt{n})$, say, and similarly for V_2 and V_3. For each triple $i, j, k \in [g]$ we list all triangles in the induced graph $G_{i,j,k} := G[V_{1,i} \cup V_{2,j} \cup V_{3,k}]$ in time

$$\widetilde{\widetilde{O}}\left(\frac{n^{3/2}}{(\log n)^{2.25}} + t_{G_{i,j,k}}\right)$$

We stop as soon as we have listed t triangles in total. Note that the total running time is thus bounded by $\widetilde{\widetilde{O}}(\frac{n^3}{(\log n)^{2.25}} + t)$ plus $\widetilde{\widetilde{O}}(\max_{i,j,k} t_{G_{i,j,k}}) = \widetilde{\widetilde{O}}(n^{3/2})$ for listing the surplus triangles. This overhead is negligible and the running time is as claimed. □

Note that from Theorem 2 it easily follows that we can list *all* triangles in time $O(n^3/(\log n)^{2.25} + t_G)$, even in a black-box way. We simply apply Theorem 2 with $t = n^3/(\log n)^{2.25}$ and list up to t triangles. As long as the graph contains at least t triangles we double t and repeat. The total running time is a geometric sum and thus bounded by $O(n^3/(\log n)^{2.25} + t_G)$ as claimed.

5 Combinatorial Log-Shaves for k-Hyperclique

We first provide an intuitive description of the algorithm in the simplest case $k = 4, r = 3$ (detecting a 4-clique in a 3-uniform hypergraph in faster than $O(n^4)$ time), for complete and general specification refer to the full paper. We are given a 4-partite 3-uniform graph $G = (V_1, V_2, V_3, V_4, E)$ with vertex sets of size n. For each $v \in V_1$, we can define a tri-partite graph $G_v = (V_2, V_3, V_4, E')$ in which we draw an edge between two vertices if and only if they share a hyperedge with v in G. It is easy to check that there is a 4-hyperclique in G if and only if there are vertices v_2, v_3, v_4 that form a triangle in G_v *and* in G (meaning they are a hyperedge in G). The naive search for such a triplet would take $O(n^3)$, and we present an algorithm that accelerates this search:

1. Let $s = \sqrt{c \log n}$ for some small constant $c > 0$, and partition V_2, V_3 and V_4 each into $g = \lceil n/s \rceil$ blocks of size at most s. We let $V_{i,j}$ denote the j'th block in V_i.
2. For every combination $j_2, j_3, j_4 \in [g]$:
   a. Create a lookup table T_{j_2,j_3,j_4} with an entry for every possible tripartite graph on the vertex sets $V_{2,j_2}, V_{3,j_3}, V_{4,j_4}$ (there are $2^{s^2} = n^c$ such graphs).
   b. For every entry corresponding to a graph G' store whether G' has a triangle that is a hyperedge in G.

Note that this preprocessing is fast: We construct $\frac{n^3}{s^3}$ tables, each consisting of n^c entries, and each entry takes $O(s^3)$ time to determine. So, the total preprocessing time is $O(n^{3+c})$. Given these tables we can now search for a 4-clique more efficiently: For each $v \in V_1$ we break G_v into triples of blocks as before, and query T_{j_2,j_3,j_4} for the graphs $G_v[V_{2,j_2} \cup V_{3,j_3} \cup V_{4,j_4}]$, for all j_2, j_3, j_4. If one the answers is positive we have found a hyperclique. Assuming every query is performed in constant time, the running time is determined by the number of queries which is

$$O\left(n \cdot \frac{n^3}{s^3}\right) = O\left(\frac{n^4}{(\log n)^{1.5}}\right).$$

All that is left now is to justify the assumption that every query is performed in constant time. The main question is given $v \in V_1$ and a combination of blocks $V_{2,j_2}, V_{3,j_3}, V_{4,j_4}$, how can we determine the key corresponding to $G_v[V_{2,j_2}, V_{3,j_3}, V_{4,j_4}]$ in T_{j_2,j_3,j_4} in constant time? For this purpose, we define in the proof a *compact representation* of tripartite graphs (on vertex sets of size s) used to index the tables T_{j_2,j_3,j_4}. This compact representation is chosen in such a way which allows to efficiently precompute the compact representations of all such graphs $G_v[V_{2,j_2}, V_{3,j_3}, V_{4,j_4}]$.

Complete description. We now turn to describe the algorithm in full generality and detail. We will in fact extend it to *list* up to t hypercliques in output-sensitive time (see Theorem 3). Throughout, $G = (V_1, \ldots, V_k, E)$ is a k-partite

r-uniform hypergraph. For a vertex $v \in V_1$, we define the *adjacency subgraph of* v, denoted by G_v, to be the $(k-1)$-partite $(r-1)$-uniform hypergraph with vertex sets $V_2, \ldots, V_k$ which has a hyperedge $\{u_1, \ldots, u_{r-1}\}$ if and only there is a hyperedge $\{v, u_1, \ldots, u_{r-1}\} \in E$ in G.

Observation 2. *Let $3 \leq r < k$, and let $(V_1, \ldots, V_k, E)$ be a k-partite r-uniform hypergraph. Then, for every tuple $v_1 \in V_1, \ldots, v_k \in V_k$, $(v_1, \ldots, v_k)$ is a k-hyperclique in G if and only if $(v_2, \ldots, v_k)$ is a $(k-1)$-hyperclique in G_{v_1} and is also a $(k-1)$-hyperclique (or a hyperedge if $r = k-1$) in G.*

Let us briefly reflect on this observation. To find a k-clique in a graph we can search through the neighborhoods of each vertex for a $(k-1)$-clique. The Four-Russians method involves preprocessing the graph in order to perform this search more efficiently. Observation 2 gives rise to an analogous process to find k-hypercliques in hypergraphs: Searching through the induced adjacency graph of each vertex for a $(k-1)$-hyperclique *that is also a clique in* G. We present an adaptation of the Four-Russians method to perform this analogous search more efficiently.

Theorem 3 (Faster k-Hyperclique Listing). *There is an algorithm for listing up to t k-hypercliques in an r-uniform hypergraph in time*

$$O\left(\frac{n^k}{(\log n)^{\frac{k-1}{r-1}}} + t\right)$$

(assuming a word RAM model with word size $w = \Omega(\log n)$).

Proof. Let s be a parameter to be determined. As a first step, we partition the vertex sets $V_2, \ldots, V_k$ into $g := \lceil n/s \rceil$ *blocks* $V_i = V_{i,1} \cup \cdots \cup V_{i,g}$ of size at most s. For $j = (j_2, \ldots, j_k) \in [g]^{k-1}$, we let

$$V^j = V_{2,j_2} \cup \cdots \cup V_{k,j_k},$$

meaning V^j denotes some combination of blocks across the vertex sets, and we let G^j denote the $(k-1)$-partite subgraph of G induced by V^j. Similarly, for $v \in V_1$, we let G_v^j denote the subgraph of G_v induced by V^j. It follows from the preceding discussion that there is a k-hyperclique in G if and only if there is some $v \in V_1$ and some $j \in [g]^{k-1}$ such that G^j and G_v^j share a common $(k-1)$-hyperclique. To detect whether G^j and G_v^j share a common hyperclique, we now present an efficient algorithm.

Compressed Representation. A critical feature of the Four-Russians technique is that we need to *compactly represent* small graphs. Specifically, let $j = (j_2, \ldots, j_k)$ and consider a $(k-1)$-partite $(r-1)$-regular hypergraph (V^j, H) on the vertex set V^j. We will fix a description of this graph as a sequence of bits. First, partition the hyperedges H into parts H_I for every set $I = \{i_1 < \cdots < i_{r-1}\} \subseteq \{2, \ldots, k\}$, such that the part H_I contains exactly the hyperedges involving the vertex parts $V_{i_1,j_{i_1}}, \ldots, V_{i_{r-1},j_{i_{r-1}}}$. We define the compact representation of H_I as the

bit-string of length s^{r-1} obtained by listing (in an arbitrary but fixed order) all tuples $(v_{i_1}, \ldots, v_{i_{r-1}}) \in V_{i_1,j_{i_1}} \times \cdots \times V_{i_{r-1},j_{i_{r-1}}}$, where we indicate present edges by 1 and missing edges by 0. Finally, the compact representation of H is defined as the concatenation of the compact representations of the parts H_I (in an arbitrary but consistent order).

Note that this compact representation has length exactly $L = \binom{k-1}{r-1} \cdot s^{r-1}$. By choosing

$$s := \left(\frac{\log n}{2^{\binom{k-1}{r-1}}}\right)^{\frac{1}{r-1}} = \Theta((\log n)^{\frac{1}{r-1}}),$$

the length becomes at most $L \leq \frac{1}{2}\log n$. In particular, we can store the compact representation of any graph H as before in $O(1)$ machine words.

In the next two claims we show that we can preprocess *all* graphs H.

Claim 1. In time $O(n^{k-1}2^L s^k)$ we can compute a data structure that, given $j \in [g]^{k-1}$ and an $(r-1)$-uniform hypergraph on V^j (represented compactly), tests in $O(1)$ time whether there are vertices $v_2, \ldots, v_k \in V^j$ such that

- *$(v_2, \ldots, v_k)$ is a $(k-1)$-hyperclique in G^j, and*
- *$(v_2, \ldots, v_k)$ is a $(k-1)$-hyperclique in H.*

Moreover, we can enumerate all such tuples $(v_2, \ldots, v_k)$ with constant delay.

Proof of Claim 1. The data structure consists of an array of length $g^{k-1} \cdot 2^L$. Each position is associated to a pair (j, H), where $j \in [g]^{k-1}$ and where H is a $(r-1)$-uniform hypergraph on V^j (that can be represented compactly in L bits). To fill the array, we enumerate all positions (j, H) and test by exhaustive search whether there is a tuple $(v_2, \ldots, v_k) \in V^j$ that forms a hyperclique in G^j and in H. Each such test requires time $O(s^k)$, therefore the total running time is indeed $O(n^{k-1}2^L s^k)$.

Each query testing whether there is a common hyperclique in G^j and H indeed runs in constant time (implemented by one lookup operation). For the enumeration, we separately store for each entry (j, H) in the array a list of all common hypercliques $(v_2, \ldots, v_k)$ of G^j and H. □

Claim 2. In time $O(n^k/s^{k-1})$ we can compute for all $j \in [g]^{k-1}$ and all $v \in V_1$ the graphs G_v^j in compressed representation.

Proof of Claim 2. Recall that the compact representation of G_v^j is the concatenation of the compact representations of $(G_v^j)_I$ for all sets $I = \{i_1 < \ldots < i_{r-1}\} \subseteq \{2, \ldots, k\}$. Thus, in a first step we will precompute the compact representations of $(G_v^j)_I$. Note that this representation only depends on $j_I := (j_{i_1}, \ldots, j_{i_{r-1}})$ (and not on the remaining j-indices). We can precompute all compressed representations determined by $v \in V_1$ and $j_I \in [g]^{r-1}$ in time $O(n \cdot g^{r-1}) = O(n^r)$. After this precomputation, we can assemble the compressed representation of any graph G_v^j in constant time $O(\binom{k-1}{r-1}) = O(1)$. Therefore, the total time is $O(n^r + n \cdot g^{k-1} = n^k/s^{k-1})$. □

Given the previous Claims 1 and 2, the proof of the theorem is almost complete. We first use Claim 2 to prepare the compressed representations of all graphs G_v^j (for $v \in V_1$ and $j \in [g]^{k-1}$). Using Claim 1 we prepare a data structure that decide in constant time for each G_v^j whether it shares a $(k-1)$-hyperclique with G^j. Whenever this test succeeds, we enumerate all common hypercliques $(v_2, \ldots, v_k)$ in G^j and G_v^j with constant delay, and report $(v, v_2, \ldots, v_k)$. If at some point during the execution we have listed t k-hypercliques, we interrupt the algorithm. As mentioned before, the correctness follows by Observation 2. The running time is

$$O(n^{k-1}2^L s^k) + O\left(\frac{n^k}{s^{k-1}}\right) + O(t) = \widetilde{O}(n^{k-1/2}) + O\left(\frac{n^k}{s^{k-1}} + t\right) = O\left(\frac{n^k}{(\log n)^{\frac{k-1}{r-1}}} + t\right)$$

as claimed. □

Corollary 3 (Faster k-Hyperclique Detection). *There is an algorithm for detecting k-hypercliques in r-uniform hypergraphs running in time*

$$O\left(\frac{n^k}{(\log n)^{\frac{k-1}{r-1}}}\right)$$

(assuming a word RAM model with word size $w = \Omega(\log n)$).

Proof. Call Theorem 3 with $t = 1$. □

Acknowledgements

We would like to thank Oded Goldreich and Nathan Wallheimer for discussions on combinatorial BMM algorithms.

References

1. Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 192–203. IEEE, 2017.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018.
3. Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying seth and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–266, 2018.

4. Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New graph decompositions and combinatorial boolean matrix multiplication algorithms. *CoRR*, abs/2311.09095, 2023. URL: https://doi.org/10.48550/arXiv.2311.09095, arXiv:2311.09095, doi:10.48550/ARXIV.2311.09095.
5. Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.7.
6. Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 884–895. IEEE, 2022. doi:10.1109/FOCS54457.2022.00088.
7. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory-hu tree in unweighted graphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1725–1737. ACM, 2021. doi:10.1145/3406325.3451073.
8. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
9. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pages 39–51. Springer, 2014.
10. Donald Aingworth, Chandra Chekuri, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). In Éva Tardos, editor, *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 28-30 January 1996, Atlanta, Georgia, USA*, pages 547–553. ACM/SIAM, 1996. URL: http://dl.acm.org/citation.cfm?id=313852.314117.
11. Dana Angluin. The four russians' algorithm for boolean matrix multiplication is optimal in its class. *ACM SIGACT News*, 8(1):29–33, 1976.
12. Vladimir L'vovich Arlazarov, Yefim A Dinitz, MA Kronrod, and IgorAleksandrovich Faradzhev. On economical construction of the transitive closure of an oriented graph. *Doklady Akademii Nauk*, 194(3):487–488, 1970.
13. Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 81:1–81:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.81.
14. Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Pro-*

ceedings of Machine Learning Research, pages 311–321. PMLR, 2017. URL: http://proceedings.mlr.press/v70/backurs17a.html.
15. Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory Comput.*, 8(1):69–94, 2012. doi:10.4086/toc.2012.v008a004.
16. Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. doi:10.1007/s00453-007-9036-3.
17. Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1836–1855. SIAM, 2021.
18. Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P Pissis, Giovanna Rosone, et al. Even faster elastic-degenerate string matching via fast matrix multiplication. *LEIBNIZ INTERNATIONAL PROCEEDINGS IN INFORMATICS*, 132:1–15, 2019.
19. Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014. doi:10.1007/978-3-662-43948-7_19.
20. Karl Bringmann, Nick Fischer, and Marvin Künnemann. A fine-grained analogue of schaefer's theorem in p: Dichotomy of exists^ k-forall-quantified first-order graph properties. In *34th Computational Complexity Conference (CCC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.
21. Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). *ACM Transactions on Algorithms (TALG)*, 16(4):1–22, 2020.
22. Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and rna folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019.
23. Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.
24. Karl Bringmann and Philip Wellnitz. Clique-based lower bounds for parsing tree-adjoining grammars. *arXiv preprint arXiv:1803.00804*, 2018.
25. Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 134–148. ACM, 2019. doi:10.1145/3294052.3319700.
26. Katrin Casel and Markus L Schmid. Fine-grained complexity of regular path queries. *arXiv preprint arXiv:2101.01945*, 2021.
27. Timothy M Chan. A (slightly) faster algorithm for klee's measure problem. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 94–100, 2008.
28. Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 212–217. SIAM, 2015. doi:10.1137/1.9781611973730.16.

29. Timothy M Chan, Saladi Rahul, and Jie Xue. Range closest-pair search in higher dimensions. *Computational Geometry*, 91:101669, 2020.
30. Yi-Jun Chang. Hardness of RNA folding problem with four symbols. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPIcs*, pages 13:1–13:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.13.
31. Raphael Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya. Upper and lower bounds for dynamic data structures on strings. *arXiv preprint arXiv:1802.06545*, 2018.
32. Mina Dalirrooyfard, Surya Mathialagan, Virginia Vassilevska Williams, and Yinzhan Xu. Listing cliques from smaller cliques. *CoRR*, abs/2307.15871, 2023. arXiv:2307.15871, doi:10.48550/arXiv.2307.15871.
33. Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster non-induced cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1167–1178, 2019.
34. Debarati Das, Michal Koucký, and Michael E. Saks. Lower bounds for combinatorial algorithms for boolean matrix multiplication. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.23.
35. Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *64th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2023)*. IEEE Computer Society, 2023. To appear. URL: https://doi.org/10.48550/arXiv.2210.10173.
36. Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.
37. Jacob Fox. A new proof of the graph removal lemma. *CoRR*, abs/1006.1300, 2010. URL: http://arxiv.org/abs/1006.1300, arXiv:1006.1300.
38. Alan M. Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Comb.*, 19(2):175–220, 1999. doi:10.1007/s004930050052.
39. Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
40. Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized bmm and omv. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1515–1528, 2022.
41. Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.ch89.
42. Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern hamming distance. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OASIcs*, pages 10:1–10:5. Schloss

Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.10.

43. Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142.
44. Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM (JACM)*, 49(1):1–15, 2002.
45. Jason Li. Faster minimum k-cut of a simple graph. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1056–1077. IEEE, 2019.
46. Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. SIAM, 2018.
47. László Miklós Lovász and Balázs Szegedy. Szemerédi's lemma for the analyst. *GAFA Geometric And Functional Analysis*, 17:252–270, 2007. URL: https://api.semanticscholar.org/CorpusID:15201345.
48. Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
49. Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *42nd Annual ACM Symposium on Theory of Computing (STOC 2010)*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.
50. Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In Susanne Albers and Tomasz Radzik, editors, *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2004. doi:10.1007/978-3-540-30140-0_52.
51. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
52. Virginia Vassilevska. Efficient algorithms for clique problems. *Inf. Process. Lett.*, 109(4):254–257, 2009. doi:10.1016/j.ipl.2008.10.014.
53. Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3447–3487, 2018. doi:10.1142/9789813272880_0188.
54. Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.
55. Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2020)*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.
56. Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Inf. Comput.*, 261:240–247, 2018. doi:10.1016/j.ic.2018.02.006.