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.

J. A. Soto and A. Wiese (Eds.): LATIN 2024, LNCS 14578, pp. 193–206, 2024.
https://doi.org/10.1007/978-3-031-55598-5_13

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 $2 \leq \omega < 2.3719$ [23] 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 [31] 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 [24], 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 [34], 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; we defer a detailed discussion on this to the full paper due to space constraints. Consequently, much research has gone into finding "combinatorial algorithms" that beat exhaustive search by other techniques. 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 [37] shows that this conjecture is *equivalent* to the classical conjecture that combinatorial algorithms cannot solve Boolean Matrix Multiplication (BMM) in truly subcubic time [28,33]. Following their reduction, many conditional lower bounds were based on this conjecture, e.g. [5,16,19,21] (we refer to the survey [36] 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 [17] 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. [1,4,9,11–14,20,22,26,29].

[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 (see full paper for more details). We will omit $(\log\log n)$ factors in this paragraph.

1. The *Four-Russians technique* [6] from 1970 gives an $O(n^3/\log^2 n)$ bound, and is used in all later developments.
2. In 2010, Bansal and Williams [7] use *pseudoregular partitions* to shave off an additional $\log^{1/4} n$ factor.
3. In 2014, Chan [18] introduced a simple *divide-and-conquer technique* to get an $O(n^3/\log^3 n)$ bound, and a year later, Yu [38] 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 [35] 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' [7]. 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 [7, 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]

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 [27,32] shows that in order to beat the longstanding $O(n^2/\log^2 n)$ bound over integers [8] 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 [10], 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 $\tilde{\tilde{O}}(n) = n(\log\log n)^{O(1)}$ 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 $\tilde{\tilde{O}}(\frac{n^3}{(\log n)^{2.25}} + t)$, 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 [25], however, it is not clear whether it can be implemented efficiently.

[5] Note that the same cannot be said about their second algorithm [7, Theorem 2.1]; see the lower bound for pseudoregular partitions due to Lovasz and Szegedy [30]).

has been used to prove conditional lower bounds, e.g. [15,29]. 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 [3]. 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 Outline

We start with some preliminaries in Sect. 2. In Sect. 3 we provide our improved combinatorial k-Clique algorithm. In Sects. 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 $[n] := \{1, \dots, n\}$. 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 $(V_1, \dots, V_k, E)$ and the goal is to determine whether there exist k vertices $v_1 \in V_1, \dots, v_k \in V_k$ 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 $V_1, \dots, V_k$ 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 $(V_1,\dots,V_k,E)$ there are vertices $v_1 \in V_1,\dots,v_k \in V_k$ such that all hyperedges on $\{v_1,\dots,v_k\}$ 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.

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 [18] 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*

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

Proof. The algorithm is simple: For each vertex $v \in V_1$, we construct the subgraph $G_v = G[N_2(v) \cup \cdots \cup N_k(v)]$ consisting of all neighbors of v and test whether G_v contains a $(k-1)$-clique. Let $n_v = d_2(v) + \cdots + d_k(v)$ 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

[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.

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 $(V_1, \dots, V_k, E)$. Let D and α be parameters to be determined later and let d be initialized to 0.

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

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 $d_2(v) \cdot \dots \cdot d_k(v) \geq \alpha \cdot |V_2| \cdot \dots \cdot |V_k|$.
   If such a vertex exists:
   a. Test whether the subgraph $G_v$ induced by $N_2(v) \cup \dots \cup N_k(v)$ 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 $2 \le i \le k$, 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 $d_2(v) \cdot \ldots \cdot d_k(v) \ge \alpha \cdot |V_2| \cdot \ldots \cdot |V_k|$ 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. We 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

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

and thus

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

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

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

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

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

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 quickly outline our triangle listing algorithm which is based on Bansal and Williams' BMM algorithm [7]. Our contribution is in reformulating and reanalyzing their algorithm for the purpose of triangle listing achieving Theorem 2. Note that we cannot achieve the running time stated in the theorem by applying state-of-the-art black-box reductions from triangle listing to Boolean matrix multiplication [37].

The two key ingredients are pseudoregularity and the following lemma which applies four russians to sparse graphs (see full paper for discussion on pseudoregularity and proof of the lemma).

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*

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

Let us give an informal overview of the algorithm. 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 benefit from the sparseness (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. For complete specification refer to the full paper.

5 Combinatorial Log-Shaves for k-Hyperclique

In this section we give 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 $G_v[V_{2,j_2}, V_{3,j_3}, V_{4,j_4}]$.

References

1. Abboud, A., Backurs, A., Bringmann, K., Künnemann, M.: 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), pp. 192–203. IEEE (2017)

2. Abboud, A., Backurs, A., Williams, V.V.: If the current clique algorithms are optimal, so is valiant's parser. SIAM J. Comput. **47**(6), 2527–2555 (2018)
3. Abboud, A., Fischer, N., Kelley, Z., Lovett, S., Meka, R.: New graph decompositions and combinatorial boolean matrix multiplication algorithms. CoRR, abs/2311.09095 (2023). arxiv:2311.09095
4. Abboud, A., et al.: Faster algorithms for all-pairs bounded min-cuts. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S., editors, 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9–12, 2019, Patras, Greece, volume 132 of LIPIcs, pp. 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.7
5. Abboud, A., Williams, V.V.:. Popular conjectures imply strong lower bounds for dynamic problems. In: 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014), pp. 434–443. IEEE Computer Society (2014). https://doi.org/10.1109/FOCS.2014.53
6. Arlazarov, V.L.V., Dinitz, Y.A., Kronrod, M.A., Faradzhev, I.: On economical construction of the transitive closure of an oriented graph. In: Akademii Nauk, D., vol. 194, pp. 487–488. Russian Academy of Sciences (1970)
7. Bansal, N., Williams, R.: Regularity lemmas and combinatorial algorithms. Theor. Comput. **8**(1), 69–94 (2012). https://doi.org/10.4086/toc.2012.v008a004
8. Baran, I., Demaine, E.D., Patrascu, M.: Subquadratic algorithms for 3SUM. Algorithmica **50**(4), 584–596 (2008). https://doi.org/10.1007/s00453-007-9036-3
9. Bergamaschi, T., Henzinger, M., Gutenberg, M.P., Williams, V.V., Wein, N.: New techniques and fine-grained hardness for dynamic near-additive spanners. In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1836–1855. SIAM (2021)
10. Björklund, A., Pagh, R., Williams, V.V., Zwick, U.: Listing triangles. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 223–234. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_19
11. Bringmann, K., Fischer, N., Künnemann, M.: A fine-grained analogue of schaefer's theorem in p: Dichotomy of existsˆ k-forall-quantified first-order graph properties. In: 34th Computational Complexity Conference (CCC 2019). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2019)
12. Bringmann, K., Gawrychowski, P., Mozes, S., Weimann, O.: Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). ACM Trans. Algorithm. (TALG) **16**(4), 1–22 (2020)
13. Bringmann, K., Grønlund, A., Larsen, K.G.. A dichotomy for regular expression membership testing. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pp. 307–318. IEEE (2017)
14. Bringmann, K., Wellnitz, P.: Clique-based lower bounds for parsing tree-adjoining grammars. arXiv preprint arXiv:1803.00804 (2018)
15. Carmeli, N., Kröll, M.: 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, pp. 134–148. ACM (2019). https://doi.org/10.1145/3294052.3319700
16. Casel, K., Schmid, M.L.: Fine-grained complexity of regular path queries. arXiv preprint arXiv:2101.01945 (2021)
17. Chan, T.M.: A (slightly) faster algorithm for klee's measure problem. In: Proceedings of the Twenty-fourth Annual Symposium on Computational geometry, pp. 94–100 (2008)

18. Chan, T.M.: 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, pp. 212–217. SIAM (2015). https://doi.org/10.1137/1.9781611973730.16
19. Chan, T.M., Rahul, S., Xue, J.: Range closest-pair search in higher dimensions. Comput. Geometry **91** 101669 (2020)
20. Chang, Y.J.: Hardness of RNA folding problem with four symbols. In: Grossi, R., Lewenstein, M., editors, 27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27–29, 2016, Tel Aviv, Israel, volume 54 of LIPIcs, pp. 13:1–13:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.CPM.2016.13
21. Clifford, R., Grønlund, A., Larsen, K.G., Starikovskaya, T.: Upper and lower bounds for dynamic data structures on strings. arXiv preprint arXiv:1802.06545 (2018)
22. Dalirrooyfard, M., Vuong, T.D., Williams, V.V.: 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, pp. 1167–1178 (2019)
23. Duan, R., Wu, H., Zhou, R.: Faster matrix multiplication via asymmetric hashing. In: 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2023). IEEE Computer Society, 2023. To appear. https://doi.org/10.48550/arXiv.2210.10173
24. Eisenbrand, F., Grandoni, F.: On the complexity of fixed parameter clique and dominating set. Theor. Comput. Sci. **326**(1–3), 57–67 (2004). https://doi.org/10.1016/j.tcs.2004.05.009
25. Fox, J.: A new proof of the graph removal lemma. CoRR, abs/1006.1300 (2010). arxiv:1006.1300
26. Jin, C., Xu, Y.: Tight dynamic problem lower bounds from generalized bmm and omv. In: Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, pp. 1515–1528 (2022)
27. Kopelowitz, T., Pettie, S., Porat, E.: Higher lower bounds from the 3SUM conjecture. In: Krauthgamer, R., editor, 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), pp. 1272–1287. SIAM (2016). https://doi.org/10.1137/1.9781611974331.ch89
28. Lee, L.: Fast context-free grammar parsing requires fast boolean matrix multiplication. J. ACM (JACM) **49**(1), 1–15 (2002)
29. Lincoln, A., Williams, V.V., Williams, R.: Tight hardness for shortest cycles and paths in sparse graphs. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1236–1252. SIAM (2018)
30. lóVász, L., Lovász, M., Szegedy, B.: Szemerédi's lemma for the analyst. GAFA Geometric Funct. Anal. **17** 252–270 (2007). https://api.semanticscholar.org/CorpusID:15201345
31. Nešetřil, J., Poljak, S.: On the complexity of the subgraph problem. Comment. Math. Univ. Carol. **26**(2), 415–419 (1985)
32. Pătraşcu, M.: Towards polynomial lower bounds for dynamic problems. In: Schulman, L.J., editor, 42nd Annual ACM Symposium on Theory of Computing (STOC 2010), pp. 603–610. ACM (2010). https://doi.org/10.1145/1806689.1806772
33. Roditty, L., Zwick, U.: On dynamic shortest paths problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 580–591. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30140-0_52

34. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**, 354–356 (1969)
35. Vassilevska, V.: Efficient algorithms for clique problems. Inf. Process. Lett. **109**(4), 254–257 (2009). https://doi.org/10.1016/j.ipl.2008.10.014
36. Williams, V.V.: On some fine-grained questions in Algorithms and Complexity, pp. 3447–3487 (2018). https://doi.org/10.1142/9789813272880_0188
37. Williams, V.V., Williams, R.R.: Subcubic equivalences between path, matrix, and triangle problems. J. ACM **65**(5), 27:1–27:38 (2018). https://doi.org/10.1145/3186893
38. Huacheng, Yu.: An improved combinatorial algorithm for boolean matrix multiplication. Inf. Comput. **261**, 240–247 (2018). https://doi.org/10.1016/j.ic.2018.02.006