Towards Optimal Output-Sensitive Clique Listing or: Listing Cliques from Smaller Cliques*

Mina Dalirrooyfard
minad@mit.edu
Morgan Stanley Research
Calgary, AB, Canada

Surya Mathialagan
smathi@mit.edu
MIT
Cambridge, MA, USA

Virginia Vassilevska Williams
virgi@mit.edu
MIT
Cambridge, MA, USA

Yinzhan Xu
xyzhan@mit.edu
MIT
Cambridge, MA, USA

ABSTRACT

We study the problem of finding and listing k-cliques in an m-edge, n-vertex graph, for constant $k \geq 3$. This is a fundamental problem of both theoretical and practical importance.

Our first contribution is an algorithmic framework for finding k-cliques that gives the first improvement in 19 years over the old runtimes for 4 and 5-clique finding, as a function of m [Eisenbrand and Grandoni, TCS'04]. With the current bounds on matrix multiplication, our algorithms run in $O(m^{1.66})$ and $O(m^{2.06})$ time, respectively, for 4-clique and 5-clique finding.

Our main contribution is an output-sensitive algorithm for listing k-cliques, for any constant $k \geq 3$. We complement the algorithm with tight lower bounds based on standard fine-grained assumptions. Previously, the only known conditionally optimal output-sensitive algorithms were for the case of 3-cliques given by Björklund, Pagh, Vassilevska W. and Zwick [ICALP'14]. If the matrix multiplication exponent ω is 2, and if the number of k-cliques t is large enough, the running time of our algorithms is

$$\tilde{O}\left(\min\{m^{\frac{1}{k-2}}t^{1-\frac{2}{k(k-2)}}, n^{\frac{2}{k-1}}t^{1-\frac{2}{k(k-1)}}\}\right),$$

and this is *tight* under the Exact-k-Clique Hypothesis. This running time naturally extends the running time obtained by Björklund, Pagh, Vassilevska W. and Zwick for $k = 3$.

Our framework is very general in that it gives k-clique listing algorithms whose running times can be measured in terms of the number of ℓ-cliques Δ_ℓ in the graph for any $1 \leq \ell < k$. This generalizes the typical parameterization in terms of n (the number of 1-cliques) and m (the number of 2-cliques).

If ω is 2, and if the size of the output, Δ_k, is sufficiently large, then for every $\ell < k$, the running time of our algorithm for listing k-cliques is

$$\tilde{O}\left(\Delta_\ell^{\frac{2}{\ell(k-\ell)}}\Delta_k^{1-\frac{2}{k(k-\ell)}}\right).$$

We also show that this runtime is *optimal* for all $1 \leq \ell < k$ under the Exact k-Clique hypothesis.

*Full version of this paper is available at https://arxiv.org/abs/2307.15871.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649663

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**.

KEYWORDS

cliques, graph algorithms, fine-grained complexity

ACM Reference Format:
Mina Dalirrooyfard, Surya Mathialagan, Virginia Vassilevska Williams, and Yinzhan Xu. 2024. Towards Optimal Output-Sensitive Clique Listing or: Listing Cliques from Smaller Cliques. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649663

1 INTRODUCTION

Finding, counting and listing cliques in graphs are fundamental tasks with numerous applications. In any type of network (social, biological, financial, web, maps, etc.) clique listing is used to find patterns such as communities, spam-link farms, motifs, correlated genes and more (see [17, 32] and the many citations within).

As finding a clique of maximum size has long been known to be NP-hard [23], the focus in numerous practical works (see [14, 15, 17, 25, 30–34]) is on listing cliques of small size such as triangles.

More generally, in an n-node m-edge graph, for a constant $k \geq 3$ (independent of n and m), we want to find, count or list the k-cliques in G. Chiba and Nishizeki [14] presented an algorithm that for any constant $k \geq 3$ can list all k-cliques in a graph in $O(m\alpha^{k-2})$ time, where $\alpha \leq O(\sqrt{m})$ is the *arboricity* of the given graph. This algorithm is among the most efficient clique-listing approaches in practice (see e.g. [17] and the references within).

Purely in terms of m, Chiba and Nishizeki's algorithm runs in $O(m^{k/2})$ time. Since $O(m^{k/2})$ is also the maximum number of k-cliques in an m-edge graph, this algorithm is optimal, as long as the graph has $\Theta(m^{k/2})$ cliques (e.g. when the graph itself is a clique). However, when the graph has t k-cliques, where t is $o(m^{k/2})$, the optimality argument no longer works. In fact, it has been known for almost 40 years [28] that when $t = 1$, a much faster runtime is possible using fast matrix multiplication.

This motivates the study of **output-sensitive** algorithms for k-clique listing: algorithms whose running time depends on the

number of k-cliques in the output. An even more desirable version of an output-sensitive algorithm is one that can also take as input some parameter t, and can list up to t k-cliques in the graph. When t is much smaller than the number of k-cliques in the graph, such an algorithm could potentially be more efficient. These two versions are actually runtime-equivalent up to logarithmic factors for most natural running times (we provide a proof in Section 2 for completeness). We thus use these two notions interchangeably.

Björklund, Pagh, Vassilevska W. and Zwick [9] designed such output-sensitive algorithms for triangle listing with runtime $\tilde{O}(n^\omega + n^{\frac{3(\omega-1)}{5-\omega}} t^{\frac{2(3-\omega)}{5-\omega}})$ and $\tilde{O}(m^{\frac{2\omega}{\omega+1}} + m^{\frac{3(\omega-1)}{\omega+1}} t^{\frac{3-\omega}{\omega+1}})$[1], where $\omega < 2.372$ [18, 38] is the exponent of matrix multiplication and t is the number of triangles listed. If $\omega = 2$, the runtimes simplify to $\tilde{O}(n^2 + nt^{2/3})$ and $\tilde{O}(m^{4/3} + mt^{1/3})$, and these are shown to be conditionally optimal for any $t = \Omega(n^{1.5})$ and $t = \Omega(m)$ respectively under the popular 3SUM hypothesis [24, 29] and the even more believable Exact Triangle hypothesis [37]. There have also been many recent works focusing on output-sensitive cycle-listing algorithms. The works of [4, 22] show $O(\min\{n^2+t, m^{4/3}+t\})$ algorithms for listing t 4-cycles, and the work of [21] shows $\tilde{O}(n^2 + t)$ algorithm for listing t 6-cycles. Moreover, matching conditional lower bounds for 4-cycle listing were shown under the 3SUM hypothesis [3, 22], which was subsequently strengthened to hold under the Exact Triangle hypothesis [13].

While the output-sensitive questions for triangle listing and 4-cycle listing are is well-understood by now, no similar conditionally optimal results are known for k-clique listing when $k \geq 4$.

Question 1. *What is the best output-sensitive algorithm for k-clique listing for $k > 3$?*

When analyzing algorithms, researchers look at a variety of **parameters** to understand performance: the size of the input (typically n and m for graph problems), the size of the output (the number of k-cliques), and other natural parameters of the input (e.g. the arboricity, as in [14]). In this work, we study clique-listing algorithms parameterized by Δ_ℓ, the number of ℓ-cliques in the graph for $\ell < k$.

To motivate this, let us consider the first non-trivial algorithm for k-clique finding by Nešetril and Poljak [28]. For simplicity, assume that k is divisible by 3. First, the algorithm enumerates all $k/3$-cliques in the input graph G, and forms a new graph H whose nodes represent the $k/3$-cliques of G and whose edges connect two $k/3$-cliques that together form a $2k/3$-clique. The triangles of H correspond to k-cliques in G, and so Nešetril and Poljak reduce k-clique finding, counting and listing in G to finding, counting and listing (respectively) of *triangles* in H[2]. As there are $O(n^{k/3})$ $k/3$-cliques in G, and since triangle finding or counting in N-node graphs can be done in $O(N^\omega)$ time [20], [28] gave an $O(n^{\omega k/3})$ time algorithm for k-clique finding or counting in n-node graphs. Eisenbrand and Grandoni [19] extended Nešetril and Poljak's reduction to obtain a k-clique runtime of $O(n^{\beta(k)})$ where $\beta(k) = \omega(\lceil k/3 \rceil, \lceil (k-1)/3 \rceil, \lfloor k/3 \rfloor)$, and $\omega(a, b, c)$ is the exponent of multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix. As the runtime of k-clique detection has remained unchallenged for several decades, the hypothesis that these algorithms are optimal has been used to provide conditional lower bounds in several works (e.g. [1, 8, 12]). Throughout the paper, we consider the word-RAM model of computation with $O(\log n)$ bit words.

Hypothesis 1.1 (k-Clique Hypothesis). *On a word-RAM model with $O(\log n)$ bit words, detecting a k-clique in an n-node graph requires $n^{\beta(k)-o(1)}$ time , where $\beta(k) = \omega(\lceil k/3 \rceil, \lceil (k-1)/3 \rceil, \lfloor k/3 \rfloor)$.*

Now, suppose G has a small number q of $k/3$-cliques and suppose we can list these $k/3$-cliques quickly, then Nešetril and Poljak's algorithm would run in only $O(q^\omega)$ additional time which can be much faster than $O(n^{\omega k/3})$.

More generally, if a graph has a small number Δ_ℓ of ℓ-cliques for $\ell < k$, a simple generalization of Nešetril and Poljak's reduction would reduce k-clique to k/ℓ-clique in a graph with Δ_ℓ nodes (assuming k is divisible by ℓ for simplicity). If one can list the ℓ-cliques fast, then k-clique finding, listing and detection can all be done faster in graphs with small Δ_ℓ.

In other words, for k-clique problems, the number of ℓ-cliques Δ_ℓ, where $\ell < k$ is arguably the most natural parameter. The usual input parameters n and m can be viewed as the special cases Δ_1 and Δ_2. We are not the first to suggest this natural parameterization of the input. In fact, small Δ_ℓ values have been exploited to obtain faster k-clique algorithms in experimental algorithmics: e.g., [30] and [16] count k-cliques faster in graphs with a small number of triangles. Motivated by these practical results, we are the first to consider the following question within theoretical computer science:

Question 2. *Can we get a general conditionally optimal algorithms for output-sensitive k-clique listing in terms of the number Δ_ℓ of ℓ-cliques for any $\ell < k$?*

1.1 Our Contributions

We present a systematic study of clique finding and listing, and provide answers to both Questions 1 and 2. We give the first output-sensitive algorithms for listing k-cliques for $k \geq 4$. We also give the first general algorithms for detecting and listing k-cliques in terms of the number of ℓ-cliques, and the first fine-grained lower bounds for the listing problem for general k. Our lower bounds show that our algorithms are tight for a non-trivial range of the number of k-cliques to output. We summarize our contributions in Table 1. (k, ℓ)-Clique-Detection and (k, ℓ)-Clique-Listing refer to *detecting* and *listing* k-cliques respectively given a list of all ℓ-cliques. Here, t is the number of k-cliques we are asked to list.

Improved 4 and 5-clique detection in sparse graphs. We provide a general algorithmic framework for detecting cliques. As special cases of the framework, we give the first improvement over the the runtime of Eisenbrand and Grandoni [19] for 4 and 5-clique detection in sparse graphs (we show this in Examples 3.3 and 3.4 in Section 3.2).

Theorem 1.2. *There is an $O(m^{1.657})$ time algorithm for 4-clique detection and an $O(m^{2.057})$ time algorithm for 5-clique detection in m-edge graphs.*

In comparison, the runtimes of the algorithms of [19] in terms of the current bounds for square and rectangular matrix multiplication

[1] We use $\tilde{O}$ to hide polylog factors.

[2] Note the reduction also works for counting and listing because every k-clique is represented by exactly $\binom{k}{k/3,k/3,k/3}$ triangles.

Table 1: Summary of our contributions. (k,ℓ)-Clique-Detection **and** (k,ℓ)-Clique-Listing **refer to *detecting* and *listing* k-cliques respectively given a list of all ℓ-cliques. Here, t is the number of k-cliques we are asked to list.**

	Results	References
Detection	New (k,ℓ)-Clique-Detection framework	Section 3
	Improved $(4,2)$-Clique-Detection and $(5,2)$-Clique-Detection	Theorem 1.2
Lower bounds	Conditional lower bounds for (k,ℓ)-Clique-Listing	Theorems 1.4, 1.10
Listing	Optimal algorithms for $(4,1)$ and $(5,1)$-Clique-Listing	Theorems 1.5, 1.6
	Nearly-everywhere optimal algorithms for $(4,\ell)$, $(5,\ell)$-Clique-Listing	Theorems 1.7, 1.8
	Optimal (k,ℓ)-Clique-Listing algorithms for large t	Theorems 1.9, 1.11
	Generalized (k,ℓ)-Clique-Listing algorithm for all t	Section 5
	Refined analysis for $(6,1)$-Clique-Listing	full version

[38] were $O(m^{1.668})$ and $O(m^{2.096})$ for 4 and 5-clique detection respectively.

Lower bounds for k-clique listing. Prior works [24, 29, 37] give fine-grained lower bounds for listing triangles in an n-node, m-edge graph: triangle-listing requires $n^{1-o(1)}t^{2/3}$ time in n-node graphs, and requires $m^{1-o(1)}t^{1/3}$ in m-edge graphs time, under standard fine-grained hypotheses. The lower bounds imply tightness of the known algorithms [9] if t is large enough: $t=\Omega(n^{1.5})$ or $t=\Omega(m)$ respectively.

The lower bounds of [24, 29] are under the 3SUM hypothesis. Extending these to lower bounds for k-clique listing seems difficult. Instead we focus on the approach of [37] who showed hardness under the Exact-Triangle hypothesis which states that finding a triangle of weight sum 0 in an n-node edge-weighted graph requires $n^{3-o(1)}$ time in the word-RAM model. The Exact-Triangle hypothesis is one of the most believable hypotheses in fine-grained complexity, as it is implied by both the 3SUM hypothesis and the APSP hypothesis (see [36]).

A natural generalization of the Exact-Triangle hypothesis is the Exact-k-Clique hypothesis (which coincides with the Exact-Triangle hypothesis for $k=3$):

Hypothesis 1.3 (Exact-k-Clique hypothesis). *For a constant $k\geq 3$, let* Exact-k-Clique *be the problem that given an n-node graph with edge weights in $\{-n^{100k},\ldots,n^{100k}\}$, asks to determine whether the graph contains a k-clique whose edges sum to 0. Then,* Exact-k-Clique *requires $n^{k-o(1)}$ time, on the word-RAM model of computation with $O(\log n)$ bit words.*

The Exact-k-Clique hypothesis is among the popular hardness hypotheses in fine-grained complexity. Most recently, it has been used to give hardness for the Orthogonal Vectors problem in moderate dimensions [2] and join queries in databases [10]. Moreover, due to known reductions (see e.g. [36]), the Exact-k-Clique hypothesis is at least as believable as the Max-Weight-k-Clique hypothesis which is used in many previous papers (e.g. [5, 7, 8, 11, 27]).

Under the Exact-k-Clique hypothesis we prove lower bounds for k-clique listing for all $k\geq 3$. These are the first lower bounds for output-sensitive clique listing for $k\geq 4$.

Theorem 1.4. *For any $k\geq 3$, and $\gamma\in[0,k]$, listing t k-cliques in a graph with n vertices, and in a graph with m nodes requires*

$$\left(n^{\frac{2}{k-1}}t^{1-\frac{2}{k(k-1)}}\right)^{1-o(1)} \quad \text{and} \quad \left(m^{\frac{1}{k-2}}t^{1-\frac{2}{k(k-2)}}\right)^{1-o(1)}$$

time respectively under the Exact-k-Clique hypothesis.

For $k=3$ this is the same lower bound as previously proven [24, 29, 37]. Shortly, we will present algorithms that match our lower bound for all k,m,n and for large t if $\omega=2$, implying that our lower bound is tight. This is in fact **the first output-sensitive lower bound** for k-clique listing problems for $k\geq 4$, and the first such lower bound for *any* graph pattern of size at least 5.

Optimal algorithms for 4 and 5-clique listing. For the special cases of $k=4,5$, we give algorithms parametrized by the number of vertices n and number of k-cliques t which are conditionally **optimal** if $\omega=2$ in the full version.

Similar to [9], we state our runtimes in terms of ω. In our analysis, we compute rectangular matrix multiplication by truncating it to multiple instances of square matrix multiplication. If one is interested in better numerical values, one could instead use the best upper bound on rectangular matrix multiplication [38] in these steps.

Theorem 1.5. *Given a graph on n nodes, one can list t 4-cliques in*

$$\tilde{O}\left(n^{\omega+1}+n^{\frac{4(\omega-1)(2\omega-3)}{\omega^2-5\omega+12}}t^{1-\frac{(\omega-1)(2\omega-3)}{\omega^2-5\omega+12}}\right)$$

time. If $\omega=2$, the runtime is $\tilde{O}(n^3+n^{2/3}t^{5/6})$.

Recall that the 4-Clique hypothesis, which is a special case of Hypothesis 1.1 when $k=4$, gives a lower bound of $n^{3-o(1)}$ if $\omega=2$. Moreover, Theorem 1.4 gives a lower bound of $(n^{2/3}t^{5/6})^{1-o(1)}$. Therefore, this 4-clique listing algorithm is indeed conditionally optimal.

Theorem 1.6. *Given a graph on n nodes, one can list t 5-cliques in*

$$\tilde{O}\left(n^{\omega+2}+n^{\frac{5(\omega-1)(2\omega-3)(3\omega-5)}{48-47\omega+16\omega^2-\omega^3}}t^{1-\frac{(\omega-1)(2\omega-3)(3\omega-5)}{48-47\omega+16\omega^2-\omega^3}}\right)$$

time. If $\omega=2$, the runtime is $\tilde{O}(n^4+n^{1/2}t^{9/10})$.

Recall that the 5-Clique hypothesis from Hypothesis 1.1 gives us a lower bound of $n^{4-o(1)}$ if $\omega=2$. Moreover, Theorem 1.4 gives

a lower bound of $(n^{1/2}t^{9/10})^{1-o(1)}$. Therefore, this 5-clique listing algorithm is also conditionally optimal.

Nearly-everywhere optimal algorithms for 4 and 5-clique listing in sparse graphs. In the case of sparse graphs, we obtain conditionally optimal runtimes for 4 and 5-clique listing for almost all values of t if $\omega = 2$. The runtimes are stated in the following theorems and are pictorially depicted in Figure 1.

Theorem 1.7. *If $\omega = 2$, one can list t 4-cliques in a graph with m edges in time*

$$\begin{cases} \tilde{O}(m^{3/2}) & \text{if } t \le m^{5/4}, \\ \tilde{O}(mt^{2/5}) & \text{if } m^{5/4} \le t \le m^{10/7}, \\ \tilde{O}(m^{1/2}t^{3/4}) & \text{if } t \ge m^{10/7}. \end{cases}$$

This algorithm matches the lower bound in Hypothesis 1.1 when $t \le m^{5/4}$, and it matches our lower bound of Theorem 1.4 when $t \ge m^{10/7}$.

Theorem 1.8. *If $\omega = 2$, one can list t 5-cliques in a graph with m edges in time*

$$\begin{cases} \tilde{O}(m^2) & \text{if } t \le m^{19/10}, \\ \tilde{O}(m^{17/18}t^{10/18}) & \text{if } m^{19/10} \le t \le m^{55/28}, \\ \tilde{O}(m^{1/3}t^{13/15}) & \text{if } t \ge m^{55/28}. \end{cases}$$

This algorithm matches the runtime of the lower bound in Hypothesis 1.1 when $t \le m^{19/10}$, and it matches our lower bound from Theorem 1.4 when $t \ge m^{55/28}$.

We prove Theorem 1.7 and Theorem 1.8 are proved in the full version of the paper.

Optimal algorithms for listing many k-cliques. More generally, we consider the problem of listing k-cliques for $k \ge 3$. For instance, consider the problem of listing 6-cliques in sparse graphs with m edges. If we adapt the existing approach for k-clique detection [19, 28] and directly reduce it to triangle listing in a graph with m nodes and then use [9], we get an $\tilde{O}(m^2 + mt^{2/3})$ runtime when $\omega = 2$. In comparison, the lower bound from Theorem 1.10 is $(m^{1/4}t^{11/12})^{1-o(1)}$. When t is close to maximum (as $t \to O(m^3)$), the $\tilde{O}(m^2 + mt^{2/3})$ runtime is polynomially higher than the lower bound. Therefore, we cannot only rely on such reductions.

Nevertheless, we give a conditionally *tight* algorithm for graphs with many k-cliques, provided that $\omega = 2$ for sufficiently large number of cliques. In particular, the runtime of the algorithm in the theorem below matches the lower bound of Theorem 1.4.

Theorem 1.9 (Informal). *If $\omega = 2$, there is an algorithm for k-clique listing which runs in time*

$$\tilde{O}\left(\min\left\{n^{\frac{2}{k-1}}t^{1-\frac{2}{k(k-1)}}, m^{\frac{1}{k-2}}t^{1-\frac{2}{k(k-2)}}\right\}\right)$$

when t is large.

We give more explicit bounds on t and the runtimes in terms of ω in Sections 4.2 and 4.3. In other words, we have an algorithm which **match the lower bound** in Theorem 1.4 for graphs with many k-cliques.

General listing algorithm for all t. In Section 5, we give a general black-box approach (by non-trivially adapting previous reductions [19, 28]) that uses our (conditionally) optimal algorithm for a large number of k-cliques t to obtain a fast algorithm that works for *all* t. The main advantage of this approach is its simplicity and generality. In particular, we obtain an intuitive and simple analysis of the runtime for all k, t. In Section 5, we show a comparison of our lower bounds and the runtime of our general algorithm in some examples. We illustrate the runtime of the general algorithm for some specific cases in Figure 2.

Improved algorithm for 6-clique listing. We note that our generic algorithm trades simplicity for optimality, and it is not always the best algorithm one can obtain for fixed k. In the full version of the paper, we give a more refined algorithm for 6-clique listing in terms of n and t if $\omega = 2$ to illustrate how one might obtain a tighter runtime bound for specific k. In Figure 3, we compare our "general" bound, our best bound and our lower bounds to illustrate the improvement in the algorithm. However, since the number of terms and parameters in the runtime increases significantly with k, we do not do this refined analysis for all k.

Listing cliques from smaller cliques. In fact, our frameworks are much more general and it extends to the problems of finding and listing k-cliques given a list of all ℓ-cliques in the graph, for $\ell \ge 1$. We use the notation Δ_ℓ to denote the number of ℓ-cliques in the graph.

Let (k, ℓ)-Clique-Detection be the problem of detecting a k-clique in a graph G, given the list of all ℓ-cliques in the graph for some $\ell \in \{1, \dots, k-1\}$. Our algorithmic framework in fact applies to the general problem of (k, ℓ)-Clique-Detection for any $k \ge 3, 1 \le \ell < k$. We note that while we only mention k-clique detection, we can use well-known techniques to also find k-cliques in the same runtime up to a log factor (see Section 2.2). Moreover, our algorithm can also be used to count the number of cliques with the same runtime.

In Table 2, we present our runtimes for (k, ℓ)-Clique-Detection for small values of k and ℓ in terms of the current value of ω. For $\ell = 1$, we captures the best known k-clique detection algorithm and hence matches Hypothesis 1.1. Although our general framework is simple, it is actually quite powerful, and allows us to obtain the first improvement in almost 20 years over the runtime of Eisenbrand and Grandoni [19], as discussed in Theorem 1.2.

Let (k, ℓ)-Clique-Listing be the problem of listing all k-cliques in a graph G, given all the ℓ-cliques of G. Equivalently, it is the problem of listing t k-cliques in a graph given all the ℓ-cliques, where t is an input to the problem (see a proof in Section 2).

Under the Exact-k-Clique hypothesis we prove the following lower bound for (k, ℓ)-Clique-Listing for *all* $k \ge 3, 1 \le \ell < k$. In fact, Theorem 1.4 is a special case of this theorem.

Theorem 1.10. *For any $k \ge 3, 1 \le \ell < k$, and $\gamma \in [0, k/\ell]$, (k, ℓ)-Clique-Listing in a graph with Δ_ℓ given ℓ-cliques and $t = \tilde{\Theta}(\Delta_\ell^\gamma)$ k-cliques requires*

$$\left(\Delta_\ell^{\frac{2}{\ell(k-\ell)}} t^{1-\frac{2}{k(k-\ell)}}\right)^{1-o(1)}$$

time under the Exact-k-Clique hypothesis.

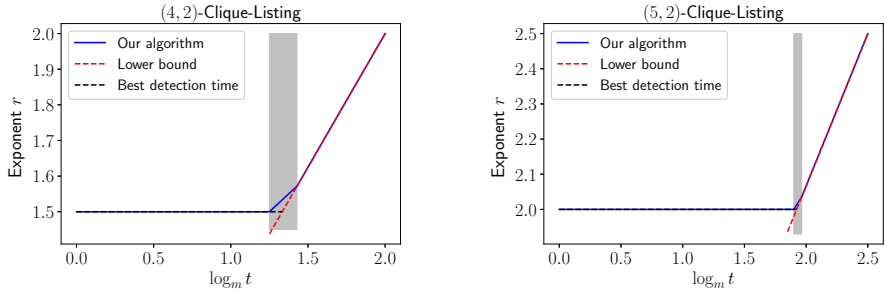

Figure 1: Upper and lower bounds for 4 and 5-cliques in graphs with m edges, if $\omega = 2$. Here, r is such that one can list t 4-cliques or 5-cliques respectively, in $\tilde{O}(m^r)$ time. The blue line corresponds to our upper bound from Theorems 1.7 and 1.8, the dashed red line denotes our lower bound from Theorem 1.10, and the dashed black line corresponds to the lower bound from Hypothesis 1.1. The shaded region highlights the portions of the algorithms which are not conditionally optimal.

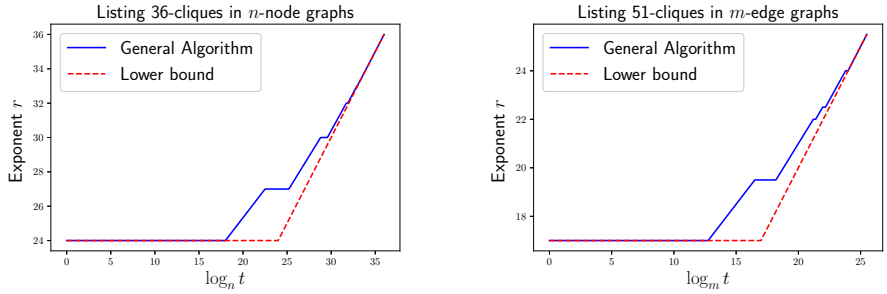

Figure 2: Upper and lower bounds listing 36-cliques in n-node graphs, and 51-cliques in m-edge graphs if $\omega = 2$. Here, the exponent r is such that one can list t 36-cliques or 51-cliques respectively, in $\tilde{O}(n^r)$ and $\tilde{O}(m^r)$ time respectively. The blue line corresponds to our upper bound from the general listing algorithm, and the dashed red line denotes the lower bounds from Hypothesis 1.1 and Theorem 1.4.

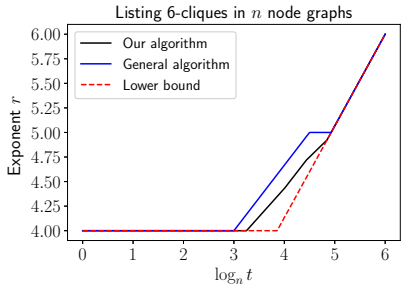

Figure 3: Upper and lower bounds for listing 6-cliques in n-node graphs if $\omega = 2$. Here, r is such that one can list t 6-cliques in $\tilde{O}(n^r)$ time. The blue line corresponds to the upper bound of our general listing algorithm, the black line corresponds to the upper bound of our refined algorithm (deferred to the full version of the paper), and the dashed red line denotes lower bound from Theorem 1.4 and Hypothesis 1.1.

We give an overview of the proof in 1.2, and defer the full proof to the full version of the paper.

Moreover, we give a conditionally *tight* algorithm for graphs with many k-cliques, provided that $\omega = 2$. In particular, the runtime of the algorithm in the theorem below matches the lower bound of Theorem 1.10.

Theorem 1.11 (Informal). *If $\omega = 2$, there exists an algorithm for (k, ℓ)-Clique-Listing which runs in time*

$$\tilde{O}\left(\Delta_\ell^{\frac{2}{\ell(k-\ell)}} \Delta_k^{1-\frac{2}{k(k-\ell)}}\right)$$

for $\Delta_k \geq \Delta_\ell^{\gamma_{k,\ell}}$ where $\gamma_{k,\ell} = \frac{k(k^2-2k-1)}{\ell(k^2-k-\ell-1)}$.

Theorem 1.9 is a special case of this theorem.

1.2 Our Techniques

In this section, we highlight our main techniques used in the algorithms and lower bounds.

Detection algorithms. The previous algorithms for k-clique detection in n-node graphs [19, 28] can be viewed as reductions to triangle detection, as mentioned earlier. Here is how they work when k is not necessarily divisible by 3. For some integers $a, b, c \in [1, k]$ where $a+b+c = k$, the algorithm creates a tripartite graph on node parts A, B, C with n^a, n^b, n^c nodes respectively, which represent tuples of a, b, c nodes respectively. It also suffices to keep only the tuples of nodes that form a clique in the original graph. For every node $(u_1, \ldots, u_a) \in A$ and every node $(v_1, \ldots, v_b) \in B$, the algorithm adds an edge between them if and only if $u_1, \ldots, u_a, v_1, \ldots, v_b$ form an $(a+b)$-clique in the original graph. It similarly adds edges between B, C and between A, C. It is not difficult to see that there is a triangle in the new graph if and only if there is a k-clique in the original graph, so we can simply detect triangles by multiplying an $|A| \times |B|$ matrix with a $|B| \times |C|$ matrix.

We generalize this approach to k-clique detection in terms of the number of ℓ-cliques for $\ell < k$.

Suppose we are given a list of all ℓ-cliques in the graph, and we want to find a k-clique. Let $a, b, c \in [1, k]$ be as before where $a + b + c = k$. Let A, B, and C, respectively, be the sets of a-, b- and c-cliques in the graph. We would like to bound their sizes in terms of Δ_ℓ. Let us focus on bounding $|A|$; bounding $|B|$, $|C|$ is done similarly.

For $a \geq \ell$, a (probably folklore) bound shows that $\Delta_a \leq O(\Delta_\ell^{a/\ell})$ (we also provide a proof for completeness in Section 2).

For $a < \ell$, we set a parameter Λ and consider two types of a-cliques: "low-degree" ones that are contained in $< \Lambda$ ℓ-cliques, and "high-degree" ones that are contained in $\geq \Lambda$ ℓ-cliques. There are at most $O(\Delta_\ell/\Lambda)$ high-degree a-cliques.

Consider a low-degree a-clique K and its neighborhood consisting of the nodes adjacent to all nodes of K. We can *recurse* on the neighborhood: find a $(k-a)$-clique, given the list of $(\ell-a)$-cliques formed by excluding K from all ℓ-cliques that contain K. We can bound the recursion runtime using the fact that K has low degree. Since we have handled all low-degree a-cliques, we can set A to be only the $O(\Delta_\ell/\Lambda)$ high-degree a-cliques. Similarly, we can get bounds on $|B|$ and $|C|$.

Finally, following previous k-clique detection algorithms [19, 28], we perform a rectangular matrix multiplication between an $|A| \times |B|$ matrix and a $|B| \times |C|$ matrix. By analyzing the recursive steps and setting parameters appropriately, we obtain our detection runtimes. As we show in Examples 3.3 and 3.4, our recursion and its analysis are more careful than in prior work, allowing us to obtain improved runtimes for 4 and 5-clique detection.

We give some explicit examples of this algorithm in Section 3.2. We also analyze the asymptotic efficiency of this algorithm in the full version.

Lower bounds for listing. We obtain our lower bound in Theorem 1.10 for listing from the Exact-k-Clique hypothesis. Our lower bound technique can be seen as a generalization of the reduction from Exact Triangle to triangle listing problems in [37].

We note that there is also a different generalization of the technique of [37] that shows a conditional lower bound for the k-Set-Intersection problem [10]. We briefly describe the problem. At a very high level, the lower bound of [10] applies to the following hypergraph problem: the nodes are partitioned into $k+1$ parts: $V_1, \ldots, V_k$ (these correspond to the sets) and U (this corresponds to the universe). There are hyperedges among the nodes in $V_1, \ldots, V_k$ (corresponding to k-set-intersection queries) and there are edges between U and V_i for $i \in [k]$ (corresponding to elements belonging to each set). Given this hypergraph, the problem asks for each hyperedge, whether its nodes share a common neighbor in U (i.e., whether the sets intersect). As the lower bound of [10] is for a problem in a hypergraph with hyperedges of cardinality > 2, it does not directly apply to our applications. Hypergraph problems are generally harder than their graph counterparts (see e.g. [27]), and there is no easy way to convert a hardness proof for hypergraphs into one for graphs without increasing the instance size significantly.

Now, we describe the high-level ideas of our reduction. Without loss of generality, we can assume the input instance of Exact-k-Clique is a k-partite graph on nodes $V_1 \sqcup \cdots \sqcup V_k$, where each V_i contains n nodes. At a high level, we first hash the edge weights so that they behave random enough. For simplicity, we assume all edge weights are independently uniformly at random from $[-n^k, n^k]$ in this overview (we deal with the randomness properly in our proof). Then we split $[-n^k, n^k]$ equally into s contiguous intervals, each of size $O(n^k/s)$ for some parameter s. We then enumerate combinations of intervals $(L_{i,j})_{1 \leq i < j \leq k}$, and consider the subgraph where we only keep edges between V_i and V_j whose weight is in $L_{i,j}$. Note that a subgraph cannot contain a k-clique of weight 0 if $0 \notin \sum_{1 \leq i < j \leq k} L_{i,j}$ (we denote the sum of two intervals as the sumset of them). Therefore, we only need to consider combinations of intervals where $0 \in \sum_{1 \leq i < j \leq k} L_{i,j}$. If we choose the first $\binom{k}{2} - 1$ intervals $(L_{i,j})_{1 \leq i < j \leq k, (i,j) \neq (k-1,k)}$, the final interval must intersect $-\sum_{1 \leq i < j \leq k, (i,j) \neq (k-1,k)} L_{i,j}$, which has size $O(\frac{n^k}{s})$. Therefore, there are only $O(1)$ choices for the final interval, and the total number of combinations of intervals we need to consider is $O(s^{\binom{k}{2}-1})$.

For each combination of intervals, we form the subgraph only containing edges with weights in the intervals, and we list all the k-cliques in this subgraph. The expected number of ℓ-cliques in the subgraph is $O(n^\ell/s^{\binom{\ell}{2}})$ and the expected number of k-cliques is $O(n^k/s^{\binom{k}{2}})$. For simplicity, we assume these upper bounds always hold in this overview (instead of only holding in expectation). Also, we can list all the ℓ-cliques in the subgraphs efficiently, i.e., in nearly linear time in their number, which is faster than n^k when s is small enough.

Then suppose we have an $O\left(\left(\Delta_\ell^{\frac{2}{\ell(k-\ell)}} t^{1-\frac{2}{k(k-\ell)}}\right)^{1-\varepsilon}\right)$ time algorithm for listing all k-cliques in a graph with t k-cliques and with a given list of Δ_ℓ ℓ-cliques. We can list all k-cliques in all the subgraphs in time

$$\tilde{O}\left(s^{\binom{k}{2}-1}\left(\left(n^\ell/s^{\binom{\ell}{2}}\right)^{\frac{2}{\ell(k-\ell)}}\left(n^k/s^{\binom{k}{2}}\right)^{1-\frac{2}{k(k-\ell)}}\right)^{1-\varepsilon}\right)$$
$$= \tilde{O}\left(n^{k-k\varepsilon}\left(s^{\binom{k}{2}-1}\right)^{\varepsilon}\right),$$

which is $\tilde{O}(n^{k-\varepsilon'})$ time for $\varepsilon' > 0$ for sufficiently small s, and violates the Exact-k-Clique hypothesis.

Listing algorithms for graphs with a large number t of k-cliques. We describe our optimal algorithm for (k,ℓ)-Clique-Listing in Theorem 1.11, for all $\ell < k$ and large enough t. We give the full algorithm in Section 4. The framework works for all values of t, but the runtime is conditionally optimal only for large t. We will later explain how to improve upon the framework for small t.

As a first step, we obtain output-sensitive algorithms for k-clique listing in terms of n ($\ell = 1$). We then use these algorithms in a black-box way for $\ell \geq 2$.

Björklund, Pagh, Vassilevska W. and Zwick [9] gave an algorithm for triangle listing using a *dense-sparse* paradigm. We generalize this algorithm to $k \geq 4$. Let t be the number of k-cliques in the graph which we want to list.

- **Dense algorithm:** When the input graph has many edges, we use sampling and rectangular matrix multiplication to find all the edges that occur in at most λ k-cliques, for some parameter λ. We then list all k-cliques incident to such edges, and can then delete these edges to obtain a graph with at most $O(t/\lambda)$ edges. We then call the algorithm for sparse graphs.
- **Sparse algorithm:** When the input graph has few edges, we list all k-cliques incident to nodes with degree at most x by *listing* $(k-1)$-cliques in their neighborhoods, for some parameter x. We are then left with a graph with at most $O(m/x)$ nodes, at which point we call the dense algorithm.

The key change from the framework of [9] is in the sparse algorithm. There, [9] uses brute-force to list triangles through low-degree nodes. We on the other hand, recursively use $(k-1,1)$-Clique-Listing algorithms to list the $(k-1)$-cliques in the neighborhoods of low-degree nodes. This makes our algorithm efficient, but also complicates the analysis significantly.

For $\ell \geq 2$, we exploit recursion even more: we recursively use algorithms for both k-clique listing in terms of nodes, and $(k-1)$-clique listing in terms of $(\ell-1)$-cliques. At a high level, we first find all nodes that are contained in at most y ℓ-cliques, for some parameter y. Then, in the neighborhoods of such nodes, we can find all $(k-1)$-cliques based on the list of all $(\ell-1)$-cliques in the neighborhood. We can then delete all the low-degree nodes. The resulting graph now only has $O(\Delta_\ell/y)$ nodes. Now, we can call the k-clique listing algorithm in terms of n.

Because of the extra recursion, the analysis gets more complicated, but we are able to keep the algorithms relatively simple. Thus we get the best of both worlds: simplicity and optimality (at least for large t).

The reason why our $(k,1)$-Clique-Listing algorithm is only optimal for large t is that our dense algorithm has an inherent cost of $\Omega(n^{k-1})$ due to the rectangular matrix multiplication that we use. This bottleneck extends to (k,ℓ)-Clique-Listing for all ℓ as well since all of these algorithms call $(k,1)$-Clique-Listing.

Generalizing the listing algorithm to all values of t. In Section 5, we explain how to improve upon our listing framework above when t is smaller. While our general runtime analysis for arbitrary k, t and ℓ quickly gets complicated, here we will focus on a small example, to give intuition.

Let us consider the example of 6-clique listing in an n-node graph G assuming $\omega = 2$. The algorithm in Theorem 1.11 has runtime $\tilde{O}(n^{\frac{2}{5}}t^{\frac{14}{15}})$ only when $t \geq n^{4+\frac{13}{14}}$, and otherwise runs in $\tilde{O}(n^5)$ time[3] which is worse than the 6-clique detection runtime $\tilde{O}(n^4)$.

We improve the runtime for t smaller than the threshold of $n^{4+\frac{13}{14}}$ by instead following the techniques of [19, 28]. We create a new graph G' whose nodes correspond to the pairs of nodes of the original graph G, i.e. the new graph has n^2 nodes. We then add an edge between two nodes (a,b) and (c,d) if (a,b,c,d) forms a 4-clique in the original graph. Now, we run the triangle listing algorithm (in [9] or Theorem 1.11) in the new graph. This has runtime $\tilde{O}(n^2t^{2/3})$ when $t \geq (n^2)^{1.5} = n^3$. This also allows us to obtain an algorithm for all $t \leq n^3$, running in time $\tilde{O}(n^4)$, the 6-clique detection runtime, which is tight under Hypothesis 1.1.

The corresponding runtime is depicted in blue in Figure 3.

More generally, for larger k, we create a new graph where the nodes represent ℓ'-cliques in the original graph. Then, we list $\lceil k/\ell' \rceil$-cliques in the new graph. The best ℓ' varies for different t, and this gives us the trade-offs as seen in Figure 2.

Roughly speaking, the algorithm can be viewed as using different dimensions of rectangular matrix multiplication depending on the value of t. For example, in the case of $k = 6$, the algorithm for large $t \geq n^{4+\frac{13}{14}}$ uses $\tilde{O}(\lambda)$ matrix multiplications of size roughly $n \times n^4/\lambda$ by $n^4/\lambda \times n$ for some parameter $\lambda \geq 1$, and this requires at least $\Omega(n^5)$ time. For $n^3 \leq t \leq n^{4+\frac{13}{14}}$, the algorithm uses $\tilde{O}(\rho)$ matrix multiplications of size $n^2 \times n^2/\rho$ by $n^2/\rho \times n^2$ for some parameter $\rho \geq 1$, which requires at least $\Omega(n^4)$ time.

1.3 Organization

In Section 2, we give necessary definitions and standard algorithms. In Section 3, we show our framework for detecting cliques. In Section 4, we show our optimal algorithm for clique listing in graphs with many k-cliques, and we extend this algorithm to graphs with fewer k-cliques in Section 5. We defer the proof of our lower bound in Theorem 1.10, as well as our more efficient algorithm for 6-clique listing in Figure 3 to the full version of the paper.

2 PRELIMINARIES

Notation. Throughout this paper, we denote the number of nodes in a graph by n, the number of edges by m, and the number of ℓ-cliques by Δ_ℓ. For an ℓ'-clique K for some $1 \leq \ell' \leq \ell$, we use $\Delta_\ell(K)$ to denote the number of ℓ-cliques containing K. For the special case of $\ell = 2$, we use $\deg(v) := \Delta_2(v)$. For integer k, we use K_k to denote a k-clique.

For a nonnegative integer n, we use $[n]$ to denote $\{1, 2, \ldots, n\}$.

Matrix multiplication. We use $\omega < 2.372$ to denote the matrix multiplication exponent [18, 38]. For any constants $a, b, c \geq 0$, we use $\omega(a,b,c)$ to denote the exponent of multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix. The current best bounds for rectangular matrix multiplication are given by [38].

[3] Clearly, when t is smaller, the runtime can only be smaller or equal, so for any $t < n^{4+\frac{13}{14}}$, the runtime of this algorithm is $\tilde{O}(n^{\frac{2}{5}}(n^{4+\frac{13}{14}})^{\frac{14}{15}}) = \tilde{O}(n^5)$ when $\omega = 2$.

We denote by $\mathsf{MM}(A,B,C)$ the runtime of multiplying an $A\times B$ by a $B\times C$ matrix. If $A\le B\le C$, we can loosely bound $\mathsf{MM}(A,B,C)$ in terms of ω as follows:

$$\mathsf{MM}(A,B,C)\le O\left(A^{\omega}\cdot\frac{BC}{A^2}\right)=O(A^{\omega-2}BC).$$

This bound is obtained by splitting the matrix multiplication into $\frac{B}{A}\cdot\frac{C}{A}$ instances of square matrix multiplication of size A, and it is in general weaker than the bound in [26].

2.1 Problem Definitions

Now, we define the main clique problems that we consider in this paper.

Definition 2.1 ((k,ℓ)-Clique-Detection). *Given a graph $G=(V,E)$ and the list L of all ℓ-cliques in G, decide whether G contains a k-clique.*

Definition 2.2 ((k,ℓ)-Clique-Listing). *Given a graph $G=(V,E)$ and the list L of all ℓ-cliques in G, list all k-cliques in G.*

In (k,ℓ)-Clique-Listing, we use t to denote the total number of k-cliques in the graph. However, as we will show in Section 2.2, we can equivalently (up to $\tilde{O}(1)$ factor) use t to denote the number of k-cliques we wish to list.

2.2 Basic Clique Listing Algorithms

Next, we give some standard algorithms and reductions. We defer the proofs from this section to the full version of the paper.

Lemma 2.3. *Suppose (k,ℓ)-Clique-Detection can be solved in time $D(\Delta_\ell)$. Then, given the list of all ℓ-cliques in a graph, one can find a k-clique in $\tilde{O}(D(\Delta_\ell))$ time.*

Lemma 2.4. *(k,ℓ)-Clique-Listing can be solved in time $\tilde{O}(\Delta_\ell^{k/\ell})$.*

The proof of Lemma 2.4 also implies that the number of k-cliques in a graph with Δ_ℓ ℓ-cliques is $O(\Delta_\ell^{k/\ell})$.

Lemma 2.5. *Fix $1\le\ell<k$. Suppose there is a $T(\Delta_\ell,x)$ time algorithm for (k,ℓ)-Clique-Listing where the total number of k-cliques is $\Theta(\Delta_\ell^x)$. Then for any $x'<x$, (k,ℓ)-Clique-Listing on graphs where the total number of k-cliques is $\Theta(\Delta_\ell^{x'})$ can be solved in $O(T(\Delta_\ell,x))$ time.*

Let $f(\Delta_\ell,t)$ be the runtime of (k,ℓ)-Clique-Listing when the graph has (an unknown number of) t cliques in total, and let $g(\Delta_\ell,t)$ be the runtime of listing $\min\{\Delta_k,t\}$ distinct k-cliques, given the list of all ℓ-cliques in the graph and a specified t as input. We assume $f(\tilde{O}(\Delta_\ell),\tilde{O}(t))=\tilde{O}(f(\Delta_\ell,t))$ and $g(\tilde{O}(\Delta_\ell),\tilde{O}(t))=\tilde{O}(g(\Delta_\ell,t))$. This is true for all of our algorithms as well as any algorithm that has at most a polynomial dependence on Δ_ℓ and t.

The following lemma shows that $f(\Delta_\ell,t)=\tilde{\Theta}(g(\Delta_\ell,t))$. Therefore, we use both of these two notions interchangeably for the definition of (k,ℓ)-Clique-Listing. In particular, given an instance of (k,ℓ)-Clique-Listing with an unknown number of k-cliques, the proof of Lemma 2.6 allows us to assume that we know an 2-approximation of Δ_k, with only $\tilde{O}(1)$ loss in the running time.

Lemma 2.6. *$f(\Delta_\ell,t)=\tilde{\Theta}(g(\Delta_\ell,t))$.*

[9] gave similar reductions from listing a specified number of t triangles to listing all Δ_3 triangles in n-node or m-edge graphs. Their reduction is more efficient than ours when t is much smaller than Δ_3. However, their reduction requires an algorithm for *counting* the number of triangles. We instead provide a black box reduction that does not rely on counting, that works for arbitrary k,ℓ, and is more self-contained and efficient enough for our purpose.

3 DETECTING CLIQUES

In this section, we first give our (k,ℓ)-Clique-Detection algorithm, and then analyze its running time in some interesting cases.

We use $g(k,\ell)$ to denote our algorithm's running time exponent on the number of ℓ-cliques of (k,ℓ)-Clique-Detection, i.e., our algorithm for (k,ℓ)-Clique-Detection runs in $\tilde{O}(\Delta_\ell^{g(k,\ell)})$ time.

3.1 General Detection Framework

Now we describe a generic algorithm for (k,ℓ)-Clique-Detection for $k\ge 3$ (for $k=2$, we trivially list all edges in the graph, so $g(2,1)=2$) in Algorithm 1.

The correctness of this algorithm is immediate. We also remark that the algorithm can be used to count the number of k-cliques, by replacing all the recursive calls with the counting version of the algorithm, using the matrix multiplication to count the number of k-cliques in the remaining graph, and properly summing up and scaling the numbers. Clearly, the counting version of the algorithm will have the same running time.

3.2 Examples

Let us give some explicit examples to illustrate the algorithm.

$(k,1)$-Clique-Detection. The simplest example of our algorithm is $(k,1)$-Clique-Detection for $k\ge 3$. Let $\lfloor k/3\rfloor\le c\le b\le a\le\lceil k/3\rceil$ be integers such that $a+b+c=k$, which is one of the possible choices of a,b,c for the algorithm. Note that $c=\lfloor k/3\rfloor$, $b=\lceil(k-1)/3\rceil$, $a=\lceil k/3\rceil$. Since $a,b,c\ge\ell=1$, the algorithm would choose to use Lemma 2.4 to bound the number of cliques of sizes a,b,c as n^a,n^b,n^c respectively. Thus, the running time of the algorithm is $\tilde{O}(n^{\omega(a,b,c)})=\tilde{O}(n^{\beta(k)})$, matching the previous running time [19].

(k,ℓ)-Clique-Detection *for $\ell\le\lfloor k/3\rfloor$*. Similar as above, let $c=\lfloor k/3\rfloor$, $b=\lceil(k-1)/3\rceil$, $a=\lceil k/3\rceil$ and the algorithm would choose to use Lemma 2.4 to bound the number of cliques of sizes a,b,c. Thus, the running time of the algorithm is $\tilde{O}(\Delta_\ell^{\omega(a/\ell,b/\ell,c/\ell)})\le\tilde{O}(\Delta_\ell^{\omega(\lceil k/3\rceil,\lceil(k-1)/3\rceil,\lfloor k/3\rfloor)/\ell})$. This running time is optimal barring improvements for $(k,1)$-Clique-Detection:

Proposition 3.1. *Fix any positive integers $k\ge 3$ and $\ell\le\lfloor k/3\rfloor$, and let*

$$\beta(k)=\omega(\lceil k/3\rceil,\lceil(k-1)/3\rceil,\lfloor k/3\rfloor).$$

If $(k,1)$-Clique-Detection requires $n^{\beta(k)-o(1)}$ time, then we have that (k,ℓ)-Clique-Detection requires $\Delta_\ell^{\beta(k)/\ell-o(1)}$ time.

Proof. Suppose for contradiction that (k,ℓ)-Clique-Detection has an $O(\Delta_\ell^{\beta(k)/\ell-\varepsilon})$ time algorithm $\mathcal{A}$ for some $\varepsilon>0$. Then given a $(k,1)$-Clique-Detection instance, we can first use Lemma 2.4 to list all ℓ-cliques in $O(n^\ell)$ time, and the number of ℓ-cliques is bounded

Algorithm 1 Generic (k,ℓ)-Clique-Detection algorithm.

Input: Graph $G=(V,E)$ and the list L of all ℓ-cliques.
Output: Output YES if G contains a k-cliques, and NO otherwise.
The Algorithm:

- Let integers $k \geq a \geq b \geq c \geq 1$ be such that $k = a+b+c$ (the algorithm chooses a,b,c optimally). Then goal is then to bound the number of d-cliques for $d \in \{a,b,c\}$.
  - If $d \geq \ell$, we can use Lemma 2.4 to upper bound the number of d-cliques with $S_d = \tilde{\Theta}(\Delta_\ell^{d/\ell})$, and add these d-cliques to a list L_d in the same time.
  - If $d < \ell$, for every d-clique K with $\Delta_\ell(K) \leq \Delta_\ell^{x_d}$ (for some parameter $x_d \in [0,1]$ to be chosen), we check if K is in a k-clique by recursively running $(k-d,\ell-d)$-Clique-Detection in its neighbourhood. Then, let L_d denote the set of remaining d-cliques. Then, $S_d := |L_d| = \Theta(\Delta_\ell^{1-x_d})$. The running time of this step is
$$\tilde{O}\left(\sum_{\substack{K:d\text{-clique}\\ \Delta_\ell(K)\leq\Delta_\ell^{x_d}}} \Delta_\ell(K)^{g(k-d,\ell-d)}\right) \leq \tilde{O}\left(\sum_{\substack{K:d\text{-clique}\\ \Delta_\ell(K)\leq\Delta_\ell^{x_d}}} \Delta_\ell(K)\cdot\Delta_\ell^{x_d(g(k-d,\ell-d)-1)}\right) \leq \tilde{O}\left(\Delta_\ell^{1+x_d(g(k-d,\ell-d)-1)}\right).$$
- Finally, we conduct a usual matrix multiplication of dimensions S_a, S_b, S_c in time $\mathsf{MM}(S_a,S_b,S_c)$ as follows. If we find a k-clique, output YES, otherwise we output NO.
  - Create a matrix X whose rows are indexed by a-cliques in L_a and columns are indexed by b-cliques in L_b. Set $A[K_a,K_b]=1$ if the nodes of K_a and K_b form an $(a+b)$-clique, and 0 otherwise.
  - Create a matrix Y whose rows are indexed by b-cliques in L_b and columns are indexed by c-cliques in L_c, and set the entries similarly.
  - Compute $Z=XY$. For each pair of remaining a-clique K_a and c-clique K_c that form an $(a+c)$-clique, check if $Z[K_a,K_c]>0$. If such an entry exists, output YES. Otherwise, output NO.

Table 2: Our (k,ℓ)-Clique-Detection exponent for various values of k,ℓ with the best current bound on ω and rectangular matrix multiplication [38]. See also [35] for a way to bound $\omega(a,b,c)$ for arbitrary $a,b,c>0$ from values of $\omega(1,x,1)$. The (k,ℓ)th entry corresponds to the exponent α such that the runtime to detect a k-clique is $\tilde{O}(\Delta_\ell^\alpha)$, where Δ_ℓ is the number of ℓ-cliques.

ℓ \ k	3	4	5	6	7	8	9	10	11	12
1	2.372	3.251	4.086	4.744	5.590	6.397	7.115	7.952	8.745	9.487
2	1.407	1.657	2.057	2.372	2.795	3.199	3.558	3.976	4.373	4.744
3	-	1.248	1.422	1.668	1.918	2.149	2.372	2.651	2.915	3.163
4	-	-	1.174	1.298	1.487	1.657	1.840	2.028	2.205	2.372
5	-	-	-	1.130	1.232	1.377	1.503	1.660	1.811	1.953

by $O(n^\ell)$. Then we can use $\mathcal{A}$ to solve the $(k,1)$-Clique-Detection instance in $O((n^\ell)^{\beta(k)/\ell-\varepsilon}) = n^{\beta(k)-\varepsilon\ell}$ time, a contradiction. □

Example 3.2 ((3, 2)-Clique-Detection). In this case, the algorithm can only choose $a=b=c=1$, and it would naturally choose $x_a=x_b=x_c$. The time it takes to bound the number of 1-cliques (nodes) is $\tilde{O}(\Delta_2^{1+x_a(g(2,1)-1)}) = \tilde{O}(m^{1+x_a})$. Then we have $S_a,S_b,S_c \leq \Theta(m^{1-x_a})$. Thus, the running time for the matrix multiplication of dimensions S_a,S_b,S_c is $\tilde{O}(m^{(1-x_a)\omega})$. Overall, the running time is $\tilde{O}(m^{\frac{2\omega}{\omega+1}})$ by setting $x_a=\frac{\omega-1}{\omega+1}$. This is essentially Alon, Yuster and Zwick [6]'s triangle detection algorithm for sparse graphs.

Example 3.3 ((4, 2)-Clique-Detection). In this case, the algorithm can only choose $a=2, b=c=1$, and it would naturally choose $x_b=x_c$. The algorithm uses Lemma 2.4 to (trivially) bound the number of edges as m. The time it takes to bound the number of nodes is $\tilde{O}(\Delta_2^{1+x_b(g(3,1)-1)}) = \tilde{O}(m^{1+x_b(\omega-1)})$. Then we have $S_a \leq \Theta(m), S_b, S_c \leq \Theta(m^{1-x_b})$. Thus, the running time for the matrix multiplication of dimensions S_a,S_b,S_c is $\tilde{O}(m^{\omega(1,1-x_b,1-x_b)})$. The algorithm chooses x_b so that $1+x_b(\omega-1)=\omega(1,1-x_b,1-x_b)$. If we simply bound $\omega(1,1-x_b,1-x_b)$ by $x_b+\omega(1-x_b)$, we can get $g(4,2) \leq \frac{\omega+1}{2}$ by setting $x_b=\frac{1}{2}$. For the current best bound of square and rectangular matrix multiplication [38], we can set $x_b=0.478$ to get an upper bound $g(4,2)\leq 1.657$. This is an improvement over the previous best algorithm of Eisenbrand and Grandoni [19], which runs in $O(m^{1.668})$ time. The key difference between our algorithm and [19]'s algorithm is that, after they perform a similar first stage, they recursively call a $(4,1)$-Clique-Detection algorithm on graphs with S_b nodes, losing the information that the graph has $S_a=m$ edges to begin with. We instead utilize this information with rectangular matrix multiplication to get a better running time.

Example 3.4 ((5, 2)-Clique-Detection). In this case, let the algorithm choose $a=b=2, c=1$ (the choice $a=3, b=c=1$ gives a worse bound). The algorithm uses Lemma 2.4 to (trivially) bound

the number of edges as m. The time it takes to bound the number of nodes is $\tilde{O}(\Delta_2^{1+x_c(g(4,1)-1)}) = \tilde{O}(m^{1+x_c(\omega(1,2,1)-1)})$. Then we have $S_a, S_b \leq \Theta(m), S_c \leq \Theta(m^{1-x_c})$. Thus, the running time for the matrix multiplication of dimensions S_a, S_b, S_c is $\tilde{O}(m^{\omega(1,1,1-x_c)})$. The algorithm chooses x_c so that $1+x_c(\omega(1,2,1)-1) = \omega(1,1,1-x_c)$. If we simply bound $\omega(1,2,1)$ by $\omega+1$ and $\omega(1,1,1-x_c)$ by $2x_c+(1-x_c)\omega$, we can get $g(5,2) \leq \frac{\omega+2}{2}$ by setting $x_c = \frac{1}{2}$. For the current best bound of rectangular matrix multiplication [38], we can set $x_c = 0.469$ to get an upper bound $g(5,2) \leq 2.057$. This is an improvement over the previous best known algorithm of Eisenbrand and Grandoni [19], which runs in $O(m^{2.096})$ time.

Example 3.5 (More Small Examples). See Tables 2 for more examples of the running times of our algorithm. These running times were obtained by finding the optimal values of a, b, c using dynamic programming.

From previous examples, one might wonder whether the algorithm always sets a, b, c as close to $k/3$ as possible. The following example shows that it is not the case (for $\omega = 2$).

In $(8,4)$-Clique-Detection, if the algorithm chooses $a = 4, b = c = 2$, then the running time is

$$\tilde{O}\left(\Delta_4^{1+x_b(g(6,2)-1)} + \Delta_4^{1+x_c(g(6,2)-1)} + \Delta_4^{\omega(1,1-x_b,1-x_c)}\right).$$

By setting $x_b = x_c = \frac{1}{2}$, this running time is bounded by $\tilde{O}(\Delta_4^{3/2})$ when $\omega = 2$ (See Table 2 for the value of $g(6,2)$ when $\omega = 2$).

However, if the algorithm chooses a more balanced choice $a = b = 3, c = 2$, then the running time is

$$\tilde{O}\left(\Delta_4^{1+x_a(g(5,1)-1)} + \Delta_4^{1+x_b(g(5,1)-1)} + \Delta_4^{1+x_c(g(6,2)-1)} + \Delta_4^{\omega(1-x_a,1-x_b,1-x_c)}\right).$$

One optimal way to set the parameters when $\omega = 2$ is $x_a = x_b = \frac{1}{5}$ and $x_c = \frac{3}{5}$, which only gives an $\tilde{O}(\Delta_4^{8/5})$ running time when $\omega = 2$ (See Table 2 for the values of $g(5,1)$ and $g(6,2)$ when $\omega = 2$).

We provide more analyses for Algorithm 1 in the full version of the paper.

4 OPTIMAL LISTING ALGORITHMS FOR GRAPHS WITH MANY k-CLIQUES

In this section, we give a $(k,1)$-Clique-Listing algorithm that is optimal for graphs with many k-cliques under Hypothesis 1.3. This algorithm can be seen as a generalization of the densifying and sparsifying paradigm of [9].

We then show how we can extend this algorithm to obtain the conditionally optimal algorithms for all (k,ℓ)-Clique-Listing for graphs with many k-cliques.

4.1 Algorithm

First, we describe the algorithm for $(k,1)$-Clique-Listing in Algorithm 2.

In the Dense algorithm, we use matrix multiplication to enumerate all k-cliques containing light edges, i.e. edges that are part of very few k-cliques. These edges are then removed to result in a *sparse* graph with only edges that are part of many k-cliques.

In the Sparse algorithm, we enumerate all k-cliques containing low-degree nodes by recursively listing all $(k-1)$-cliques in their neighborhoods, and delete all such nodes. Deleting these nodes results in a *dense* graph with only high degree nodes. While one could brute-force the $(k-1)$-cliques in the neighborhoods, our key insight is that we can instead recursively use a $(k-1,1)$-Clique-Listing algorithm to be more efficient. We defer the correctness of Algorithm 2.

(k,ℓ)-Clique-Listing when $\ell \geq 2$. To generalize this algorithm to (k,ℓ)-Clique-Listing for $\ell \geq 2$, we recursively use $(k-1,\ell-1)$-Clique-Listing to reduce the problem to $(k,1)$-Clique-Listing. At a high level, the algorithm considers all nodes v in fewer than x ℓ-cliques and recursively calls $(k-1,\ell-1)$-Clique-Listing to list all k-cliques containing v. See Algorithm 3. The correctness of Algorithm 3 is deferred to the full version.

4.2 Analysis for $(k,1)$-Clique-Listing

For $k \geq 2$, define

$$x_k = k\prod_{j=2}^{k}((5-2j)+(j-2)\omega) \tag{1}$$

$$y_k = (3-\omega)^{k-2} + \sum_{j=2}^{k-1}(3-\omega)^{k-1-j}x_j \tag{2}$$

Theorem 4.1. *Let $\alpha_k = x_k/y_k$. For any $k \geq 2$ and large $t \geq n^{\gamma_k}$ where*

$$\gamma_k = \begin{cases} 0 & \text{if } k = 2 \\ k\left(1-\frac{3-\omega}{k-\alpha_k}\right) & \text{if } k \geq 3 \end{cases},$$

there exists an algorithm that lists all t k-cliques in time $\tilde{O}(n^{\alpha_k}t^{1-\frac{\alpha_k}{k}})$. If $\omega = 2$, we have that $x_k = k$ and $y_k = \frac{k(k-1)}{2}$, therefore giving a runtime of $\tilde{O}(n^{\frac{2}{k-1}}t^{1-\frac{2}{k(k-1)}})$ for $t \geq n^{k-1-\frac{2}{k^2-k-2}}$.

We defer the analysis to the full version of the paper.

4.3 Analysis for (k,ℓ)-Clique-Listing for $\ell \geq 2$

We have shown an algorithm for $(k,1)$-Clique-Listing that is conditionally optimal when $\Delta_k \geq n^{\gamma_k}$. Now, we use this to show that there exists a (k,ℓ)-Clique-Listing algorithm for all ℓ that is conditionally optimal for $\Delta_k \geq n^{\gamma_{k,\ell}}$, for some $0 \leq \gamma_{k,\ell} < \frac{k}{\ell}$. First, we define the following variable

$$z_{k,\ell} = x_k \sum_{i=0}^{\ell-1}\frac{k-\ell}{k-i-1}\cdot\frac{y_{k-i}}{x_{k-i}}$$

where x_k and y_k are just as defined in (1) and (2). From this definition, the following identity is immediate.

Claim 4.2. *For $\ell \geq 2$ and $k > \ell$, $z_{k,\ell} = \frac{x_k}{x_{k-1}}z_{k-1,\ell-1} + \frac{k-\ell}{k-1}y_k$.*

Theorem 4.3. *Fix any constant integers $k-1 \geq \ell \geq 1$. Let $\alpha_{k,\ell} = x_k/z_{k,\ell}$. Then, there exists some $\gamma_{k,\ell} = (1-\varepsilon_{k,\ell})k/\ell$ for $\varepsilon_{k,\ell} > 0$ such that for large $t \geq n^{\gamma_k}$ there exists an algorithm that lists all t k-cliques given the ℓ-cliques in time $\tilde{O}(\Delta_\ell^{\alpha_{k,\ell}}t^{1-\frac{\ell\alpha_{k,\ell}}{k}})$.*

Algorithm 2 $(k,1)$-Clique-Listing Algorithm for large $t \geq n^{\gamma_k}$, where γ_k is defined in Theorem 4.1

$\texttt{Dense}(G := (V,E), n, t)$:

- **Input:** Graph $G = (V,E)$ with $|V| \leq n$ and at most t k-cliques.
- **Output:** List of k-cliques in G.
- **The Algorithm:**

(1) If $n < k$, it returns no k-cliques.
(2) Choose a parameter λ. Let an edge be λ-light if it is in fewer than λ k-cliques.
(3) Use the algorithm in Lemma 2.4 to obtain a list L of all $(k-2)$-cliques (there are at most n^{k-2} such cliques).
(4) Initialize an empty list T.
(5) Repeat the following $O(\lambda \log n)$ times:
  – Sample a subset L' of L of size $|L|/\lambda$.
  – Construct adjacency matrices A and $\overline{A}$ where the rows are indexed by V and columns are indexed by L'.
  – Let $A[v,C] = 1$ if node v is distinct from and adjacent to every node in the $(k-1)$-clique C, and set $A[v,C] = 0$ otherwise.
  – Let $\overline{A}[v,C] = A[v,C] \cdot C$, i.e. column C contains entries 0 or C.
  – Compute $B = A \cdot A^T$ and $\overline{B} = A \cdot \overline{A}^T$. This takes $O(\mathsf{MM}(n, |L'|, n))$ time.
  – For every edge $(u,v) \in E$ that is λ-light, if $B[u,v] = 1$, add $(u, v, \overline{B}[u,v])$ to T.
(6) Output T.
(7) Delete all λ-light edges from E to obtain E' (all λ-light edges are found in Step 5 w.h.p.).
(8) Call $\texttt{Sparse}(G' := (V,E'), \binom{k}{2}t/\lambda, t)$.

$\texttt{Sparse}(G := (V,E), m, t)$:

- **Input:** Graph $G = (V,E)$ with $|E| \leq m$ and at most t k-cliques.
- **Output:** List of k-cliques in G.
- **The Algorithm:**

(1) If $m < \binom{k}{2}$, it returns no k-cliques.
(2) Choose a parameter x.
(3) Find all nodes such that $\deg(v) \leq x$, and call the $(k-1,1)$-Clique-Listing algorithm in the neighbourhoods of all such nodes with $n' = \deg(v)$.
(4) Delete all nodes in V of degree less than x to obtain set V'.
(5) Call $\texttt{Dense}(G' := (V', E \cap (V' \times V')), 2m/x, t)$

Algorithm 3 (k,ℓ)-Clique-Listing Algorithm for large $t \geq n^{\gamma_{k,\ell}}$, where $\gamma_{k,\ell}$ is defined in Theorem 4.3

Input: A graph G and a list L of all ℓ-cliques.
Output: All k-cliques in the graph.
The Algorithm:

(1) Call a node v light if $\Delta_\ell(v) \leq x$, for some parameter x.
(2) For all light nodes, call $(k-1,\ell-1)$-Clique-Listing in the neighbourhoods to find all k-cliques incident to x.
(3) Delete all light nodes and incident edges from G.
(4) Call the $(k,1)$-Clique-Listing algorithm $\texttt{Dense}(G' := (V', E'), \ell\Delta_\ell/x, t)$ (from Algorithm 2).

If $\omega = 2$, we have $x_k = k$ and $z_{k,\ell} = \frac{k\ell(k-\ell)}{2}$, giving a runtime of $\tilde{O}(\Delta_\ell^{\frac{2}{\ell(k-\ell)}} t^{1-\frac{2}{k(k-\ell)}})$ for all $t \geq n^{\gamma_{k,\ell}}$, where

$$\gamma_{k,\ell} = \frac{k(k^2-2k-1)}{\ell(k^2-k-\ell-1)}.$$

We defer the analysis to the full version of the paper.

5 EXTENDING THE ALGORITHM TO GRAPHS WITH FEWER k-CLIQUES

In this section, we show how to apply our algorithm in Section 4 which only works for very large t (or rather, does not have improved runtime for smaller t) to other ranges of t as well, via black-box reductions. In the full version, we will also give some examples to show how to use Theorem 5.1.

Theorem 5.1. *Suppose for every $1 \leq \ell < k$, (k,ℓ)-Clique-Listing can be solved in $\tilde{O}(\Delta_\ell^{\alpha_{k,\ell}} t^{1-\frac{\ell\alpha_{k,\ell}}{k}})$ time when $t \geq \Delta_\ell^{\gamma_{k,\ell}}$. Then for every $1 \leq \ell \leq k$ and $1 \leq s < k$ where $\lceil \frac{k}{s} \rceil \neq \lceil \frac{\ell}{s} \rceil$, (k,ℓ)-Clique-Listing can be solved in*

$$\tilde{O}\left(\left(\Delta_\ell^{\frac{s}{\ell}\lceil\frac{\ell}{s}\rceil}\right)^{\alpha_{k',\ell'}} t^{1-\frac{\ell'\alpha_{k',\ell'}}{k'}}\right)$$

time for $t \geq \left(\Delta_\ell^{\frac{s}{\ell}\lceil\frac{\ell}{s}\rceil}\right)^{\gamma_{k',\ell'}}$, where $k' = \lceil \frac{k}{s} \rceil$ and $\ell' = \lceil \frac{\ell}{s} \rceil$.

ACKNOWLEDGMENTS

Mina Dalirrooyfard is supported by a Google Faculty Research Award and an Akamai MIT CS Theory Group Fellowship while at MIT. Surya Mathialagan is supported by the Siebel Scholars program, by DARPA under Agreement No. HR00112020023 and by NSF grant CNS-2154149. Virginia Vassilevska Williams is partially

supported by NSF Career Award CCF-1651838, NSF Grant CCF-2129139, a Sloan Research Fellowship and a Google Faculty Research Award. Yinzhan Xu is supported by NSF Grant CCF-2129139.

REFERENCES

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2018. If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser. *SIAM J. Comput.* 47, 6 (2018), 2527–2555.

[2] Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. 2018. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 253–266.

[3] Amir Abboud, Karl Bringmann, and Nick Fischer. 2023. Stronger 3-SUM Lower Bounds for Approximate Distance Oracles via Additive Combinatorics. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 391–404. https://doi.org/10.1145/3564246.3585240

[4] Amir Abboud, Seri Khoury, Oree Leibowitz, and Ron Safier. 2022. Listing 4-Cycles. *arXiv preprint arXiv:2211.10022* (2022).

[5] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. 2014. Consequences of Faster Alignment of Sequences. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*. 39–51.

[6] Noga Alon, Raphael Yuster, and Uri Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223.

[7] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. 2016. Tight Hardness Results for Maximum Weight Rectangles. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*. 81:1–81:13.

[8] Arturs Backurs and Christos Tzamos. 2017. Improving Viterbi is Hard: Better Runtimes Imply Faster Clique Algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 311–321.

[9] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. 2014. Listing triangles. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*. 223–234.

[10] Karl Bringmann, Nofar Carmeli, and Stefan Mengel. 2022. Tight Fine-Grained Bounds for Direct Access on Join Queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. 427–436. https://doi.org/10.1145/3517804.3526234

[11] Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2020. Tree Edit Distance Cannot be Computed in Strongly Subcubic Time (Unless APSP Can). *ACM Trans. Algorithms* 16, 4 (2020), 48:1–48:22.

[12] Karl Bringmann and Philip Wellnitz. 2017. Clique-Based Lower Bounds for Parsing Tree-Adjoining Grammars. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*. 12:1–12:14.

[13] Timothy M. Chan and Yinzhan Xu. 2024. Simpler Reductions from Exact Triangle. In *Proceedings of the 2024 SIAM Symposium on Simplicity in Algorithms (SOSA)*. to appear.

[14] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223.

[15] Shumo Chu and James Cheng. 2011. Triangle listing in massive networks and its applications. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 672–680.

[16] Seshadhri Comandur, Joshua Wang, Rishi Gupta, and Tim Roughgarden. 2014. Counting small cliques in social networks via triangle-preserving decompositions. *Sandia Technical Report SAND2014-1516C 504950* (2 2014). https://www.osti.gov/biblio/1141233

[17] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing K-Cliques in Sparse Real-World Graphs. In *Proceedings of the 2018 World Wide Web Conference (WWW)*. 589–598.

[18] Ran Duan, Hongxun Wu, and Renfei Zhou. 2023. Faster Matrix Multiplication via Asymmetric Hashing. In *Proceedings of the 64th IEEE Symposium on Foundations of Computer Science (FOCS)*.

[19] Friedrich Eisenbrand and Fabrizio Grandoni. 2004. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.* 326, 1-3 (2004), 57–67.

[20] Alon Itai and Michael Rodeh. 1978. Finding a Minimum Circuit in a Graph. *SIAM J. Comput.* 7, 4 (1978), 413–423.

[21] Ce Jin, Virginia Vassilevska Williams, and Renfei Zhou. 2024. Listing 6-Cycles. In *Proceedings of the 2024 SIAM Symposium on Simplicity in Algorithms (SOSA)*. to appear.

[22] Ce Jin and Yinzhan Xu. 2023. Removing Additive Structure in 3SUM-Based Reductions. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 405–418. https://doi.org/10.1145/3564246.3585157

[23] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*. 85–103.

[24] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1272–1287.

[25] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.* 407, 1 (2008), 458–473.

[26] François Le Gall and Florent Urrutia. 2018. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1029–1046.

[27] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. 2018. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1236–1252.

[28] Jaroslav Nešetřil and Svatopluk Poljak. 1985. On the complexity of the subgraph problem. *Comment. Math. Univ. Carol.* 26, 2 (1985), 415–419.

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

[30] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. 1431–1440.

[31] Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. 2015. Finding the Hierarchy of Dense Subgraphs using Nucleus Decompositions. In *Proceedings of the International Conference on the World Wide Web (WWW)*. 927–937.

[32] Thomas Schank and Dorothea Wagner. 2005. Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study. In *Proceedings of the 4th International Conference on Experimental and Efficient Algorithms (WEA)*. 606–609.

[33] Julian Shun and Kanat Tangwongsan. 2015. Multicore triangle computations without tuning. In *Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE)*. 149–160.

[34] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of the International Conference on the World Wide Web (WWW)*. 1122–1132.

[35] Jan van den Brand and Danupon Nanongkai. 2019. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *Proceedings of the 2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. 436–455.

[36] Virginia Vassilevska Williams. 2018. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, Vol. 3. World Scientific, 3431–3472.

[37] Virginia Vassilevska Williams and Yinzhan Xu. 2020. Monochromatic triangles, triangle listing and APSP. In *Proceedings of the 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. 786–797.

[38] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. 2024. New Bounds for Matrix Multiplication: from Alpha to Omega. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. to appear.