Massively Parallel Algorithms for Small Subgraph Counting

Amartya Shankha Biswas
CSAIL, MIT, Cambridge, MA, USA

Talya Eden
CSAIL, MIT, Cambridge MA, USA
Boston University, MA, USA

Quanquan C. Liu
Northwestern University, Evanston, IL, USA

Ronitt Rubinfeld
CSAIL, MIT, Cambridge, MA, USA

Slobodan Mitrović
University of California Davis, CA, USA

Abstract

Over the last two decades, frameworks for distributed-memory parallel computation, such as MapReduce, Hadoop, Spark and Dryad, have gained significant popularity with the growing prevalence of large network datasets. The Massively Parallel Computation (MPC) model is the de-facto standard for studying graph algorithms in these frameworks theoretically. Subgraph counting is one such fundamental problem in analyzing massive graphs, with the main algorithmic challenges centering on designing methods which are both scalable and accurate.

Given a graph $G = (V, E)$ with n vertices, m edges and T triangles, our first result is an algorithm that outputs a $(1 + \varepsilon)$-approximation to T, with asymptotically *optimal round and total space complexity* provided any $S \geq \max\left(\sqrt{m}, n^2/m\right)$ space per machine and assuming $T = \Omega(\sqrt{m/n})$. Our result gives a quadratic improvement on the bound on T over previous works. We also provide a simple extension of our result to counting *any* subgraph of k size for constant $k \geq 1$. Our second result is an $O_\delta(\log\log n)$-round algorithm for exactly counting the number of triangles, whose total space usage is parametrized by the *arboricity* α of the input graph. We extend this result to exactly counting k-cliques for any constant k. Finally, we prove that a recent result of Bera, Pashanasangi and Seshadhri (ITCS 2020) for exactly counting all subgraphs of size at most 5 can be implemented in the MPC model in $\tilde{O}_\delta(\sqrt{\log n})$ rounds, $O(n^\delta)$ space per machine and $O(m\alpha^3)$ total space.

In addition to our theoretical results, we simulate our triangle counting algorithms in real-world graphs obtained from the Stanford Network Analysis Project (SNAP) database. Our results show that both our approximate and exact counting algorithms exhibit improvements in terms of round complexity and approximation ratio, respectively, compared to two previous widely used algorithms for these problems.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Computing methodologies → Massively parallel algorithms

Keywords and phrases triangle counting, massively parallel computation, clique counting, approximation algorithms, subgraph counting

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.39

Category APPROX

Related Version *Full Version*: `https://arxiv.org/abs/2002.08299`

Funding *Slobodan Mitrović*: S. Mitrović was supported by the Swiss NSF grant No. P400P2_191122/1 and FinTech@CSAIL.

1 Introduction

Estimating the number of small subgraphs, cliques in particular, is a fundamental problem in computer science, and has been extensively studied both theoretically and from an applied perspective. Given its importance, the task of counting subgraphs has been explored in various computational settings, e.g., sequential [7, 91, 28], distributed and parallel [89, 78, 68, 80, 72], streaming [16, 66, 24, 76], and sublinear-time [44, 5, 13, 45]. There are usually two perspectives from which subgraph counting is studied: first, optimizing the running time (especially relevant in the sequential and sublinear-time settings) and, second, optimizing the space or query requirement (relevant in the streaming, parallel, and distributed settings). In each of these perspectives, there are two, somewhat orthogonal, directions that one can take. The first is *exact* counting. However, in most scenarios, algorithms that perform exact counting are prohibitive, e.g., they require too much space or too many parallel rounds to be implementable in practice.

Hence, the second direction of obtaining an *estimate/approximation* on the number of small subgraphs is both an interesting theoretical problem and of practical importance. If $H_{\#}$ is the number of subgraphs isomorphic to H, the main question in approximate counting is whether we can design algorithms that, under given resource constraints, provide approximations that concentrate well. This concentration is usually parametrized by $H_{\#}$ (and potentially some other parameters). In particular, most known results do not provide a strong approximation guarantee when $H_{\#}$ is very small, e.g., $|H_{\#}| = O(1)$. So, the main attempts in this line of work is to provide an estimation that concentrates well while imposing as small a lower bound on $H_{\#}$ as possible.

Due to ever increasing sizes of data stores, there has been an increasing interest in designing scalable algorithms. The *Massively Parallel Computation* (MPC) model is a theoretical abstraction of popular frameworks for large-scale computation such as MapReduce [41], Hadoop [93], Spark [95] and Dryad [62]. MPC gained significant interest recently, most prominently in building algorithmic toolkits for graph processing [57, 74, 17, 8, 18, 59, 4, 83, 61, 38, 11, 12, 51, 58, 30, 14, 29, 21, 19, 23, 9, 15, 53, 50, 55, 71, 63, 34, 52, 54]. Efficiency of an algorithm in MPC is characterized by three parameters: round complexity, the space per machine in the system, and the number of machines/total memory used. Our work aims to design efficient algorithms with respect to all three parameters and is guided by the following question:

How does one design efficient massively parallel algorithms for small subgraph counting?

1.1 The MPC Model

In this paper, we are working in the Massively Parallel Computation (MPC) model introduced by [67, 57, 17]. The model operates as follows. There exist $\mathcal{M}$ machines that communicate with each other in synchronous rounds. The graph input is initially distributed across the machines in some organized way such that machines know how to access the relevant information via communication with other machines. During each round, the machines first perform computation locally without communicating with other machines. The computation done locally can be unbounded (although the machines have limited space so any reasonable program will not do an absurdly large amount of computation). At the end of the round, the machines exchange messages to inform the computation for the next round. The total size of all messages that can be received by a machine is upper bounded by the size of its local memory, and each machine outputs messages of sufficiently small size that can fit into its memory. If N is the total size of the data and each machine has S words of space, we are interested in the settings when S is sublinear in N. We use *total space* to refer to $\mathcal{M} \cdot S$, which is the total space that is available across all the machines.

1.2 Our Contributions

Table 1 Summary of our main MPC triangle counting results compared to previous work. Our results are **bolded**. "ALB" refers to the approximation lower bound on the number of triangles required to obtain a $(1+\varepsilon)$-approximation, with high probability. α is the arboricity of the input graph and is generally small (logarithmic) in real-world networks. Parameter $\delta > 0$ is *any* constant.

Problem	Work	MPC Rounds	Space Per Machine	Total Space	ALB
Exact Triangle Counting	[89]	2	$O(\sqrt{m})$	$O(m^{3/2})$	-
	[89]	1	$o(m)$	$\omega(m)$	-
	[36]	$O(n)$	$O(n)$	$O(m)$	-
	folklore	$O(\log n)$	$\Omega(\alpha^2)$	$O(m\alpha)$	-
	Ours	$\mathbf{O_\delta(\log\log n)}$	$\mathbf{O(n^\delta)}$	$\mathbf{O(m\alpha)}$	-
Approximate Triangle Counting	[78]	$O(1)$	$\Omega(m)$	$O(m)$	$\Omega(d_{avg})$
	[85]	$O(1)$	$O(n^\delta)$	$O(m)$	$\Omega\left(\sum_{v \in V} \deg(v)^2\right)$
	Ours	$\mathbf{O(1)}$	$\mathbf{\widetilde{O}(n)}$	$\mathbf{\widetilde{O}(m)}$	$\mathbf{\Omega(\sqrt{d_{avg}})}$

1.2.1 Triangle Counting

We provide a number of results for triangle counting in both the approximate and exact settings. Let $G = (V, E)$ be a graph with n vertices, m edges and T triangles. First we study the question of approximately counting the number of triangles under the restriction that the round and total space complexities are essentially *optimal*, i.e., $O(1)$ and $\widetilde{O}(m)$, where $\widetilde{O}$ hides $O(\text{poly} \log n)$ factors, respectively. Here and throughout, we use O_δ and O_ε to hide factors of δ and ε, respectively, where we consider constant factors of $\delta, \varepsilon > 0$ in this paper.

Our algorithm is surprisingly simple with a more complicated analysis, but improves on the previous best-known result by giving a $(1+\varepsilon)$-approximation, with high probability, while achieving a *quadratic* improvement on the number of triangles required to ensure this approximation. The specific bounds are given in Table 1.

▶ **Theorem 1.** *Let $G = (V, E)$ be a graph with n vertices, m edges, and let T be the number of triangles in G. Assuming*

(i) $T = \widetilde{\Omega}\left(\sqrt{\frac{m}{S}}\right)$, **(ii)** $S = \widetilde{\Omega}\left(\max\left\{\frac{\sqrt{m}}{\varepsilon}, \frac{n^2}{m}\right\}\right)$,

there exists an MPC *algorithm, using $\mathcal{M}$ machines, each with local space S, and total space $\mathcal{M}S = \widetilde{O}_\varepsilon(m)$, that outputs a $(1 \pm \varepsilon)$-approximation of T, with high probability, in $O(1)$ rounds.*

For $S = \Theta(n \log n)$ (specifically, $S > 100 n \log n$) in Theorem 1, we derive the following corollary.

▶ **Corollary 2.** *Let G be a graph and T be the number of triangles it contains. If $T \geq \sqrt{d_{avg}}$, then there exists an MPC algorithm that in $O(1)$ rounds with high probability outputs a $(1+\varepsilon)$-approximation of T. This algorithm uses a total space of $\widetilde{O}(m)$ and space $\widetilde{O}(n)$ per machine. d_{avg} is the average degree of the vertices in the graph.*

There is a long line of work on computing approximate triangle counting in parallel computation [37, 90, 89, 94, 78, 69, 79, 85, 10, 80, 68, 64, 42] and references therein. Despite this progress, and to the best of our knowledge, on one hand, each MPC algorithm for exact triangle counting either requires strictly super-polynomial in m total space, or the number of rounds is super-constant (as seen in Table 1). On the other hand, the best-known, classic algorithm for approximate triangle counting by Pagh and Tsourakakis [78] requires $T \geq d_{avg}$ even when the space per machine is $\Theta(n)$. We design an algorithm that has essentially optimal total space and round complexity, while at least quadratically improving the requirement on T.

Furthermore, since the amount of messages sent and received by each machine is bounded by $O(n)$, by [20], our algorithm directly implies an $O(1)$-rounds algorithm in the Congested-Clique model[1] under the same restriction $T = \Omega(\sqrt{m/n})$. The best known (to our knowledge) triangle approximation algorithm for general graphs in this model, is an $O(n^{1/3}/T^{2/3})$-rounds algorithm by [43]. The best-known previous bound only results in constant round complexity when $T = \Omega(\sqrt{n})$.

▶ **Corollary 3.** *Given a graph $G = (V, E)$ with T triangles, if $T = \Omega(\sqrt{m/n})$, then there exists a $O(1)$-rounds algorithm in the Congested-Clique model that gives a $(1+\varepsilon)$-approximation of T with high probability.*

The second question we consider is the question of exact counting, for which we present an algorithm whose total space depends on the arboricity of the input graph. The arboricity of a graph (roughly) equals the average degree of its densest subgraph. The class of graphs with bounded arboricity includes many important graph families such as planar graphs, bounded degree graphs and randomly generated preferential attachment graphs. In addition, many real-world graphs exhibit bounded arboricity [56, 48, 87], making this property important also in practical settings. For many problems, a bound on the arboricity of the graph allows for much more efficient algorithms and/or better approximation ratios [6, 48].

Specifically for the task of subgraph counting, in a seminal paper, Chiba and Nishizeki [35] prove that triangle enumeration can be performed in $O(m\alpha)$ time, and assuming 3SUM-hardness this result is optimal up to dependencies in $O(\text{poly}\log n)$ [81, 70]. Many applied algorithms also rely on the property of having bounded arboricity in order to achieve better space and time bounds, e.g., [84, 36, 73]. Our main theorem with respect to this question is the following.

▶ **Theorem 4.** *Let $G = (V, E)$ be a graph with n vertices, m edges and arboricity α. Count-Triangles(G) takes $O_\delta(\log\log n)$ rounds, $O(n^\delta)$ space per machine for any $\delta > 0$, and $O(m\alpha)$ total space.*

It is interesting to note that our total space complexity matches the time complexity (both upper and conditional lower bounds) of combinatorial[2] triangle counting algorithms in the sequential model [35, 81, 70]. The best-known previous algorithm in this setting is the folklore algorithm of placing each vertex and its out-neighbors in the same machine and counting the incident triangles. Such an approach requires $O(\log n)$ rounds and $\Omega(\alpha^2)$ space per machine (summarized in Table 1). We prove the above theorem in Section 4.

[1] A distributed model where nodes communicate with each other over a complete network using $O(\log n)$ bit messages [75].

[2] Combinatorial algorithms, usually, refer to algorithms that do not rely on fast matrix multiplication.

1.2.2 Clique Counting

All of our above triangle counting results can be extended to k-clique counting. In our full paper [27], we prove that our exact triangle counting result can be extended to exactly counting k-cliques for any constant k:

▶ **Theorem 5.** *Let $G = (V, E)$ be a graph with n vertices, m edges and arboricity α. Count-Cliques(G) takes $O_\delta\left(\log\log n\right)$ rounds, $O\left(n^\delta\right)$ space per machine for any $\delta > 0$, and $O\left(m\alpha^{k-2}\right)$ total space.*

We can improve on the total space usage if we are given machines where the memory for each individual machine satisfies $\alpha < n^{\delta'/2}$ where $\delta' < \delta$. In this case, we obtain an algorithm that counts the number of k-cliques in G using $O(n\alpha^2)$ total space and $O_\delta(\log\log n)$ communication rounds.

Furthermore, our approximate triangle counting results can be extended to counting *any* subgraph of size K where K is constant. Specifically, we obtain the following result:

▶ **Theorem 6.** *Let $G = (V, E)$ be a graph with n vertices, m edges, and let B be the number of occurrences of a subgraph H with K vertices in G. If $B \geq d_{avg}^{K/2-1}$, then there exists an MPC algorithm that gives a $(1+\varepsilon)$-approximation of B in $O(1)$ rounds, total space $\widetilde{O}(m)$, and $\widetilde{O}(n)$ space per machine, with high probability. Here, d_{avg} is the average degree of the vertices in the graph.*

1.3 Other Small Subgraphs

Finally, we consider the problem of exactly counting subgraphs of size at most 5, and show that the recent result of Bera, Pashanasangi and Seshadhri [25] for this question in the sequential model, can be implemented in the MPC model. Ours is the first result for counting any arbitrary subgraph of size at most 5 in poly$(\log n)$ rounds in the MPC model. Here too, our total space complexity matches the time complexity of the sequential model algorithm. It is an interesting open question whether our results can be extended to more general subgraphs following the results of [32, 26]. Section 6 summarizes the difficulties of implementing these algorithms in the MPC model and we present this question as interesting future work.

▶ **Theorem 7.** *Let $G = (V, E)$ be a graph with n vertices, m edges, and arboricity α. The algorithm of BPS for counting the number of occurrences of a subgraph H over $k \leq 5$ vertices in G can be implemented in the MPC model in $O_\delta(\sqrt{\log n}\log\log m)$ rounds, with high probability. The space requirement per machine is $O(n^{2\delta})$ and the total space is $O(m\alpha^3)$.*

1.4 Related Work

There has been a long line of work on small subgraph counting in massive networks in the MapReduce model whose results translate to the MPC model. We first describe the works for *exact* triangle and k-clique counting. [89] first designed an algorithm for triangle counting, but their approach requires a super-linear total space of $O(m^{3/2})$. Another work, [2], shows how to count small subgraphs by using b^3 machines, each requiring $O(m/b^2)$ space per machine. Hence, it uses a total space of $O(mb)$. Therefore, this approach either requires super-linear total space or almost $O(m)$ space per machine. [89] were the first to achieve constant number of rounds in MPC, where they design two algorithms. The first of those algorithms, that runs in 2 rounds, requires $O(\sqrt{m})$ space per machine and total space $O(m^{3/2})$. Their second algorithm requires only one round for exact triangle counting, total space $O(\rho m)$ and space per machine $O(m/\rho^2)$. Therefore, for this algorithm to work with polynomially less than space m per machine, it has to allow for a total space that is polynomially larger than m. [36] focus on algorithms that require a total space of $O(m)$. In the worst case, their algorithm performs $O(|E|/S)$ MPC rounds to output the exact count where S is the maximum space per machine. [49] extended and provided new algorithms for clique counting but they also require $\Omega(m^{3/2})$ total space.

[90, 10] designed randomized algorithms for *approximate* triangle counting also in the MapReduce model (whose results, again, can be translated rather straightforwardly to the MPC model). Their approach first sparsifies the input graph by sampling a subset of edges, and executes some of the known algorithms for triangle counting on the sampled subgraph. Denoting their sampling probability by p, their approach outputs a $(1+\varepsilon)$-approximate triangle count with probability at most $1 - 1/(\varepsilon^2 p^3 T)$. [3] To contrast this result with our approach, consider a graph G where $m = \Theta(n^2)$. Let G' be the edge-sparsified graph as explained above. To be able to execute the first algorithm of [89] on G' such that the total space requirement is $O(m)$, one can verify that it is needed to set $p = \Theta(n^{-2/3})$. This in turn implies that the result in [90, 10] outputs the correct approximation with constant probability only if $T = \Omega(n^2)$. An improved lower-bound can be obtained by using the second algorithm of [89]. By balancing out ρ and p and for $S = O(n)$, one can show that the sparsification results in a constant probability of success for $T = \Omega(n)$. On the other hand, for $S = O(n)$, our approach obtains the same guarantee even when $T = \Theta(\sqrt{d_{avg}(G)}) = \Theta(\sqrt{n})$.

The best-known algorithm of [78] is a randomized algorithm for approximate triangle counting based on graph partitioning. The graph is partitioned into $1/p$ pieces, where p is at least the ratio of the maximum number of triangles sharing an edge and T. When all the triangles share one edge, then $p \geq 1$, and hence such an approach would require the space per machine to be $\Omega(m)$. Furthermore, this approach requires the number of triangles to be lower bounded by $T = \Omega\left(d_{avg}\right)$. Another more recent work of [85] uses wedge sampling and provides a $(1+\varepsilon)$-approximation of the triangle count in $O(1)$ rounds when T is a constant fraction of the sum of squares of degrees. The comparison of our bounds with these previous results are summarized in Table 1.

Other related work. Subgraph counting (primarily triangles) was also extensively studied in the streaming model, see [16, 66, 31, 65, 76, 24, 13] and references therein. This culminated in a result that requires space $\widetilde{O}\left(m^{3/2}/(T\varepsilon^2)\right)$ to estimate the number of triangles within a $(1+\varepsilon)$-factor. In the semi-streaming setting it is assumed that one has $\widetilde{O}(n)$ space at their disposal. This result fits in this regime for $T \geq m^{3/2}/n = d_{avg} \cdot m^{1/2}$. As a reminder, our MPC result requires $T \geq \sqrt{d_{avg}}$ when $S = \widetilde{O}(n)$.

In a celebrated result, [7] designed an algorithm for triangle counting in the sequential settings that runs in $O(m^{2\omega/(\omega+1)})$ time, where ω is the best-known exponent of matrix multiplication. Since then, several important works have extended this result to k-clique counting [46, 91]. In the work-depth (shared-memory parallel processors) model, several results are known for this problem. There has been significant work on practical parallel algorithms for the case of triangle counting (e.g. [10, 89, 79, 80, 88] among others). There is even an annual competition for parallel triangle counting algorithms [1]. For counting $k = 4$ and $k = 5$ cliques, efficient practical solutions have also been developed [3, 40, 47, 60, 82]. [39] recently implemented the Chiba-Nishizeki algorithm [35] for k-cliques in the parallel setting; although, their work does not achieve polylogarithmic depth. Even more recently, [86]

[3] The actual probability is even smaller and also depends on pairs of triangles that share an edge.

enumerated k-cliques in the work-depth model in $O\left(m\alpha^{k-2}\right)$ expected work and $O\left(\log^{k-2} n\right)$ depth with high probability, using $O(m)$ space. Among other distinctions from our setting, the work-depth model assumes a shared, common memory.

In the CONGESTED-CLIQUE model, [33] present an $\tilde{O}(n^{1-2/\omega}) = \tilde{O}(n^{0.158})$ rounds algorithm for matrix multiplication, implying the same complexity for exact triangle counting. [43] present an algorithm for approximate triangle counting in general graphs whose expected running time is $O(n^{1/3}/T^{2/3})$. They also present an $O(\alpha^2/n)$-rounds algorithm for bounded arboricity graphs.

2 Preliminaries

Counting Duplicates. We make use of interval trees for certain parts of our paper to count the number of repeating elements in a sorted list, given bounded space per machine. We use the interval tree implementation given by [57] to obtain our count duplicates algorithm in the MPC model. We prove the following theorem in the MPC model regarding our count duplicates tree implementation. The proofs of the following claims are given in our full paper [27].

▶ **Theorem 8.** *Given a sorted list of N elements implemented on processors where the space per processor is S and the total space among all processors is $O(N)$, for each unique element in the list, we can compute the number of times it repeats in $O\left(\log_S N\right)$ communication rounds.*

We also use the following two new MPC primitives in proving our bounds. These primitives may be of use in other algorithms beyond the scope of our paper.

▶ **Lemma 9.** *Given two sets of tuples Q and C (both of which may contain duplicates), for each tuple $q \in Q$, we return whether $q \in C$ in $O(|Q \cup C|)$ total space and $O_\delta(1)$ rounds given machines with space $O(n^\delta)$ for any $\delta > 0$.*

▶ **Lemma 10.** *Given a machine M that has space $O(n^{2\delta})$ for any $\delta > 0$ and contains data of $O(n^\delta)$ words, we can generate x copies of M, each holding the same data as M, using $O(M \cdot x)$ machines with $O(n^\delta)$ space each in $O(\log_{n^\delta} x)$ rounds.*

3 Overview of Our Techniques

3.1 Exact Triangle Counting

Let $G = (V, E)$ be a graph with n vertices, m edges and arboricity at most α. We tackle the task of exactly counting the number of triangles in G in $O_\delta(\log\log n)$ rounds using the following ideas. In each round i, we partition the vertices into low-degree vertices A_i and high-degree vertices, according to a degree threshold γ_i, which grows doubly exponentially in the number of rounds. We then count the number of triangles incident to the set of low degree vertices A_i. Each low-degree vertex $v \in A_i$ sends a list of its neighbors to all its neighbors. Then, any neighbor u of v that detects a common neighbor w to u and v, adds the triangle (u, v, w) to the list of discovered triangles.

Once all triangles incident to the vertices in A_i are processed, we remove this set from the graph and continue with the now smaller graph. This removal of the already processed vertices allows us to handle larger and larger degrees from step to step while using a total space of $O(m\alpha)$. This behavior also leads to the $O_\delta(\log\log n)$ round complexity, as after this many rounds all vertices are processed. The key insight in our proof that we maintain $O(m\alpha)$ total space *even when we increase the degree threshold doubly exponentially.* Such insight allows us to obtain our improved number of rounds while maintaining the same total space as the previous folklore algorithm. Finally, we achieve improved space per machine to $O(n^{\delta})$ for any constant $\delta > 0$ via a number of new MPC primitives. Our algorithm and its analysis are provided in Section 4.

3.2 Approximate Triangle Counting

Our work reduces approximate triangle counting to exact triangle counting in multiple (randomly chosen) induced subgraphs of the original graph. In our work, and in contrast to prior approaches (e.g., [78]), the induced sugraphs on different machines might *overlap* in both vertices and edges. This enables us to obtain better concentration bounds compared to prior work, but also brings many challenges. Surprisingly, our algorithm is very simple (with a more complicated analysis), but is able to achieve a better lower bound on the number of triangles required to achieve a $(1+\varepsilon)$-approximation with high probability.

The high level idea is that each machine M_i samples a subset of vertices V_i by including each vertex in V_i with probability $\hat{p}$. Then, each machine computes the induced subgraph $G[V_i]$ and the number of triangles in that subgraph. The total number of triangles seen across all the machines is used as an estimator. We repeat in parallel this sampling process $O(\log n)$ times and return the median of the estimates. The main challenge this approach raises is: *How do we efficiently collect* overlapping *induced subgraphs?* (Indeed, approximate triangle counting, even when the number of triangles is $O(1)$, can be reduced to counting the number of edges in *sparse* induced subgraphs with the total size of subgraphs being $\widetilde{O}(m)$.) We now describe how to handle this task in our case.

Computing induced overlapping subgraphs. It is unclear how to compute the induced subgraph on each machine in $O(1)$ rounds without exceeding the total allowed space of $\tilde{O}(m)$. This task becomes easier if the subgraphs are disjoint. For example, such an issue is avoided when the graph is *partitioned* across machines as in the algorithm of Pagh and Tsourakakis [78] since there is one copy of each vertex among all the machines. This is not the case for our algorithm.

The trivial strategy of sampling vertices into the machines and querying for all possible edges between any pair of two vertices takes total space at least $\sum_{i=1}^{\mathcal{M}} X_i^2$ where X_i is the number of vertices sampled to each machine i. In general, this approach requires much larger than $\widetilde{O}(m)$ space. We tackle this challenge by using a *globally known* hash function $h : V \times [\mathcal{M}] \to \{0,1\}$, to indicate whether vertex v is sampled in the i^{th} machine. By requiring that the hash function is known to all machines, we can efficiently compute which edges to send to each machine, i.e., which edges belong to the subgraph $G[V_i]$. However, in order for all machines to be able to compute the hash function, the hash function has to use limited space. Hence, we cannot hope for a fully independent function, rather we can only use an $(S/\log n)$-wise independent hash function. Still, we manage to show that we are able to handle the dependencies introduced by the hash function, even if we allow as little as $O(\log n)$-independence.

3.3 Counting k-cliques and 5-subgraphs

We use similar techniques for both problems of exactly counting the number of k-cliques and of exactly counting subgraphs up to size 5. Our final result is the first MPC algorithm for counting any *arbitrary* subgraph H of size at most 5 in poly$(\log n)$ MPC rounds.

Let H denote the subgraph of interest. We say that a subgraph that can be mapped to a subset of H of size i is a *i-subcopy of H*. Our main contribution in this section is a new MPC procedure that in each round, tries to extend i-subcopies of H to $(i+1)$-subcopies of H by increasing the total space by a factor of at most α. This is possible by ordering the vertices in H such that each vertex has at most $O(\alpha)$ outgoing neighbors so that in each iteration only α possible extensions should be considered per each previously discovered subcopy.

Challenges. The major challenge we face here is dealing with *finding* and *storing* copies of small (constant-sized) subgraphs in individual machines. This is a challenge due to the fact that an entire neighborhood of a vertex v may not fit on one machine (recall that we have no restrictions on how large the constant δ in $O(n^{\delta})$ machine size can be). Thus, we cannot compute all such small subgraphs on one machine. However, if not done carefully, computing small subgraphs across many machines could potentially result in many rounds of computation (since we potentially have to try all combinations of vertices in a neighborhood). We solve this issue by formulating a new MPC procedure (Lemma 10) in which we carefully duplicate neighborhoods of vertices across machines. The detailed analysis of our algorithm is given in our full paper [27].

4 Exact Triangle Counting in $O(m\alpha)$ Total Space

In this section we describe our algorithm for (exactly) counting the number of triangles in graphs $G = (V, E)$ of arboricity α and prove Theorem 4, restated here, in Appendix A.1. We first provide an overview of our algorithm and its challenges.

▶ **Theorem 11.** *Let $G = (V, E)$ be a graph over n vertices, m edges and arboricity α. COUNT-TRIANGLES(G) takes $O_{\delta}(\log\log n)$ rounds, $O\left(n^{\delta}\right)$ space per machine for some constant $0 < \delta < 1$, and $O(m\alpha)$ total space.*

Importantly, unlike previous methods, we *do not* need to assume knowledge of the arboricity of the graph α as input into our algorithm. The arboricity only shows up in our space bound as a property of the graph but we do not need to have knowledge of its value as we run the algorithm. The folklore algorithm shown in Table 1 requires an assumption of an upper bound on α since in order to achieve $O(\log n)$ rounds, we must count triangles incident to and remove all vertices with degree less than or equal to 2α in each round. The procedure gets stuck if we remove vertices with degree c where $c < \alpha$ in each round because there exists an induced subgraph with degree at least α in a graph with arboricity α. One can estimate the arboricity of the graph using $O(\log n)$ additional rounds or an $O(\log n)$ additional factor in space. Our algorithm does not require this additional step.

In this section, we assume that individual machines have space $\Theta(n^{\delta})$ where δ is some constant $0 < \delta < 1$. Given this setting, there are several challenges associated with this problem.

▶ **Challenge 12.** *The entire subgraph neighborhood of a vertex may not fit on a single machine. This means that all triangles incident to a particular vertex cannot be counted on one machine. Even if we are considering vertices with degree at most α, it is possible that $\alpha > n^{\delta}$. Thus, we need to have a way to count triangles efficiently when the neighborhood of a vertex is spread across multiple machines.*

The second challenge is to avoid over-counting.

▶ **Challenge 13.** *When counting triangles across different machines, over-counting the triangles might occur, e.g., if two different machines count the same triangle. We need some way to deal with duplicate counting of the triangles to obtain the exact count of the triangles.*

We deal with the above challenges in our procedures below. We assume in our algorithm that each vertex can access its neighbors in $O(1)$ rounds of communication; such can be ensured via standard MPC techniques. Let $d_Q(v)$ be the degree of v in the subgraph induced by vertex set Q, i.e. in $G[Q]$. Our main algorithm consists of the following COUNT-TRIANGLES(G) procedure.

Algorithm 1 Count-Triangles$(G = (V, E))$.

```
1:  Let $Q_i$ be the set of vertices not yet processed by iteration $i$. Initially set $Q_0 \leftarrow V$.
2:  Let $T$ be the current count of triangles. Set $T \leftarrow 0$.
3:  for $i = 0$ to $i = \lceil \log_{3/2}(\log_2(n)) \rceil$ do
4:      $\gamma_i \leftarrow 2^{(3/2)^i}$.
5:      Let $A_i$ be the list of vertices $v \in Q_i$ where $d_{Q_i}(v) \leq \gamma_i$. Set $Q_{i+1} \leftarrow Q_i \setminus A_i$.
6:      parfor $v \in A_i$ do
7:          Retrieve the list of neighbors of $v$ and denote it by $L_v$.
8:          Send each of $v$'s neighbors a copy of $L_v$.
9:      end parfor
10:     parfor $w \in Q_i$ do
11:         Let $\mathcal{L}_w = \bigcup_{v \in (N(w) \cap A_i)} L_v$ be the union of neighbor lists received by $w$.
12:         Set $T \leftarrow T + $ FIND-TRIANGLES$(w, \mathcal{L}_w)$.          ▷ Algorithm 2
13:     end parfor
14: Return $T$.
```

Round compression is a technique formulated by [77, 38] that randomly partitions the vertices in a graph across machines where each machine then stores the induced subgraph induced by the partition. Then, a problem (e.g. maximum matching) is solved locally in each induced subgraph in each machine. The solutions in each machine allows one to remove certain vertices, reducing the degree of the remaining graph. In each round compression step, the maximum degree of the graph drops by a polynomial factor. This degree reduction then allows for more aggressive sampling in the next round compression step. This leads to $O(\log \log \Delta)$ round compression steps until the maximum degree is poly$(\log n)$; in this case, the remaining graph can be placed on a single machine.

Our algorithm, although similar, is simpler than the round compression technique. We do not require sampling since vertices are assigned to machines by degree, deterministically. The crux of our argument is showing that allowing for total space in terms of the arboricity α leads to a simpler and deterministic argument. Furthermore, for this specific problem, we also do not need to place the induced subgraph on one machine. In the next section, we show an implementation that allows us to operate in the sublinear space per machine regime. We hope our algorithm and analysis will lead to other deterministic algorithms for bounded arboricity graphs in sublinear space per machine and $O(\log \log n)$ rounds.

4.1 MPC Implementation Details

In order to implement COUNT-TRIANGLES(G) in the MPC model, we define our FIND-TRIANGLES$(w, \mathcal{L})$ procedure and provide additional details on sending and storing neighbor lists across different machines. We define *high-degree* vertices to be the set of vertices whose degree is $> \gamma$ and *low-degree* vertices to be ones whose degree is $\leq \gamma$ (for some γ defined in our algorithm). We now define the function FIND-TRIANGLES$(w, \mathcal{L})$ used in the above procedure:

Algorithm 2 Find-Triangles$(w, \mathcal{L}_w)$.

```
1: Sort all elements in $(\mathcal{L}_w \cup (N(w) \cap Q_i))$ lexicographically, using the procedure given in
   Lemma 4.3 of [57]. Let this sorted list of all elements be $S$.
2: Let $T$ denote the corrected[4] number of duplicates in $S$ using Theorem 8.
3: Return $T$.
```

Allocating machines for sorting. Since each $v \in Q_i$ could have multiple neighbors whose degrees are $\leq \gamma$, the total size of all neighbor lists v receives could exceed their allowed space $\Theta\left(n^{\delta}\right)$. Thus, we allocate $O\left(\frac{\gamma d_{Q_i}(v)}{n^{\delta}}\right)$ machines for each vertex $v \in Q_i$ to store all neighbor lists that v receives.

The complete analysis for Theorem 11 is given in Appendix A.1.

We provide two additional extensions of our triangle counting algorithm to counting k-cliques:

▶ **Theorem 14.** *Given a graph $G = (V, E)$ with arboricity α, we can count all k-cliques in $O(m\alpha^{k-2})$ total space, $O_\delta(\log\log n)$ rounds, on machines with $O(n^{2\delta})$ space for any $0 < \delta < 1$.*

We can prove a stronger result when we have some bound on the arboricity of our input graph. Namely, if $\alpha = O(n^{\delta'/2})$ for any $\delta' < \delta$, then we obtain the following result:

▶ **Theorem 15.** *Given a graph $G = (V, E)$ with arboricity α where $\alpha = O(n^{\frac{\delta'}{2}})$ for any $\delta' < \delta$, we can count all k-cliques in $O\left(n\alpha^2\right)$ total space and $O_\delta(\log\log n)$ rounds, on machines with $O(n^{\delta})$ space for any $0 < \delta < 1$.*

The proofs of these theorems are provided in our full paper [27].

5 Approximate Triangle Counting in General Graphs

In this section we provide our algorithm for estimating the number of triangles in general graphs (see Algorithms 3 and 6) and hence prove Theorem 1.

▶ **Theorem 1.** *Let $G = (V, E)$ be a graph with n vertices, m edges, and let T be the number of triangles in G. Assuming*

(i) $T = \widetilde{\Omega}\left(\sqrt{\frac{m}{S}}\right)$, **(ii)** $S = \widetilde{\Omega}\left(\max\left\{\frac{\sqrt{m}}{\varepsilon}, \frac{n^2}{m}\right\}\right)$,

there exists an MPC *algorithm, using $\mathcal{M}$ machines, each with local space S, and total space $\mathcal{M}S = \tilde{O}_\varepsilon(m)$, that outputs a $(1 \pm \varepsilon)$-approximation of T, with high probability, in $O(1)$ rounds.*

The rationale behind the lower bound constraints in Theorem 1 will become clear when we discuss the challenges and analysis (formally presented in the following sections).

5.1 Overview of the Algorithm and Challenges

Our approach is to use the collection of machines to repeat the following experiment multiple times in parallel. Each machine M_i samples a subset of vertices V_i, and then counts the number of triangles $\hat{T}_i$ seen in each induced graph $G[V_i]$. We then use the sum $\hat{T}$ of all $\hat{T}_i$'s as an unbiased estimator (after appropriate scaling) for the number of triangles T in the original graph.

Algorithm 3 Approximate-Triangle-Counting(G=(V,E)).

```
1: $R \leftarrow 0$
2: parfor $i \leftarrow 1 \ldots \mathcal{M}$ do
3:   Let $V_i$ be a random subset of $V$      ▷ See Section 5.2 for details about the sampling
4:   if size of $G[V_i]$ exceeds machine space $S$ then
5:     Ignore this sample and set $\hat{T}_i \leftarrow 0$
6:   else
7:     Let $\hat{T}_i$ be the number of triangles in $G[V_i]$
8:     $R \leftarrow R + 1$
9: end parfor
10: Let $\hat{T} = \sum_{i=1}^{\mathcal{M}} \hat{T}_i$
11: return $\frac{1}{\hat{p}^3 R}\hat{T}$
```

Moving forwards, for the most part, we will focus on a specific machine M_i containing V_i (a single experiment). We list the main challenges in the analysis of this algorithm, along with the sections that describe them.

1. **Section 5.2:** The induced subgraph $G[V_i]$ fits into the memory S of M_i (thus allowing us to count the number of triangles in $G[V_i]$ in one round).
2. **Section 5.3:** We can efficiently (in one round) collect all the edges in the induced subgraph $G[V_i]$. This involves presenting an MPC protocol such that the number of messages *sent and received* by any machine is at most the *space per machine* S.
3. **Section 5.4** With high constant probability, the number of messages sent and received by each machine M_i is at most S.
4. **Section 5.5:** With high *constant* probability (of at least 0.9), the sum of triangles across all machines, $\hat{T}$, is close to its expected value. Then, repeating the algorithm polylogarithmic number of times with only a polylogarithmic increase in total space, and by using the median trick, allows us to get a high probability bound. The specifics are discussed in Appendix A.1.7.

In each of the following sections, we first present a high level overview of the challenges that we need to solve and then follow these high-level descriptions with detailed proofs.

5.2 Challenge (1): Ensuring That $G[V_i]$ Fits on a Single Machine

Ensuring that *edges* fit on a machine

Our algorithm constructs V_i by including each $v \in V$ with probability $\hat{p}$, which implies that the expected number of edges in $G[V_i]$ is $\hat{p}^2 m$. Since we have to ensure that each induced subgraph $G[V_i]$ fits on a single machine, we obtain the constraint $\hat{p}^2 m = O(S)$. Concretely, we achieve this by defining:

$$\hat{p} \overset{\text{def}}{=} \frac{1}{10} \cdot \sqrt{\frac{S}{mk}}\,, \tag{1}$$

where the parameter $k = O(\log n)$ will be exactly determined later (See Section 5.3).

Ensuring that *vertices* fit on a machine

In certain regimes of values of n and m, the *expected number of vertices* ending up in an induced subgraph – $\hat{p}n$, may exceed the space limit S. Avoiding this scenario introduces an additional constraint $\hat{p}n = O(S) \iff S = \Omega(kn^2/m)$.

Getting a high probability guarantee

As discussed above, the value of $\hat{p} = \widetilde{\Theta}_\varepsilon(\sqrt{S/m})$ is chosen specifically so that the *expected number of edges* in the induced subgraphs $G[V_i]$ is $\hat{p}^2 m \leq \Theta(S)$, thus using all the available space (asymptotically). In order to guarantee that this bound holds *with high probability* (see Appendix A.1.4), we require additional constraints on the space per machine $S = \widetilde{\Omega}_\varepsilon(\sqrt{m})$. We remark that this lower bound $S = \widetilde{\Omega}_\varepsilon(\sqrt{m})$ is essentially saying that $\mathcal{M} = \widetilde{O}_\varepsilon(\sqrt{m})$, i.e. the *space per machine* is much larger than the *number of machines*. This is a realistic assumption as in practice we can have machines with 10^{11} words of local random access memory, however, it is unlikely that we also have as many machines in our cluster.

Lower Bound on *space per machine*

Combining the above two constraints, we get:

$$S > \max\left\{15\frac{\sqrt{mk}}{\varepsilon}, \frac{100kn^2}{m}\right\} \implies S = \widetilde{\Omega}_\varepsilon\left(\max\left\{\sqrt{m}, \frac{n^2}{m}\right\}\right) \tag{2}$$

Note that Eq. (2) always allows *linear space per machine*, as long as $m = \Omega(n)$. The following sections, Appendices A.1.4 and A.1.5 present a detailed analysis, showing that the number of vertices and edges in each subgraph is at most S *with high probability.* In this high-level overview of the challenges, we defer a detailed analysis of these bounds to the later sections (Appendices A.1.4 and A.1.5) since the formal proof of these bounds also require a discussion of Section 5.3.

5.3 Challenge (2): Using k-wise Independence to Compute the Induced Subgraph $G[V_i]$ in MPC

For each sub-sampled set of vertices V_i, we need to compute $G[V_i]$, i.e. we need to send all the edges in the induced subgraph $G[V_i]$ to the machine M_i. Let Q_u denote the set of all machines containing u. Each edge (u, w) then needs to be sent to all machines that contain both u and w, $Q_u \cap Q_w$. Naively, one could try to send the sets Q_u and Q_w to the edge $e = (u, w)$, for all $e \in E$. However, this strategy could result in Q_v being replicated $d(v)$ times. Since the expected size of Q_v is $|Q_v| = \hat{p}\mathcal{M}$ the total expected memory usage of this strategy would be $\sum_{v \in V} |Q_v| \cdot d(v) = \widetilde{\Theta}_\varepsilon(m \cdot \hat{p}\mathcal{M}) = \widetilde{\omega}_\varepsilon(m)$, since $\hat{p} = \widetilde{\Theta}(1/\sqrt{\mathcal{M}})$. This defies our goal of optimal total memory.

Instead, we address this challenge by using globally known hash functions to sample the vertices on each machine. That is, we let $h : V \times [\mathcal{M}] \to \{0, 1\}$ (formally presented in Definition 16) be a hash function known globally to all the machines. Then we can compute the induced subgraphs $G[V_i]$ as follows.

▶ **Definition 16.** *The hash function $h(v, i)$ indicates whether vertex v is sampled in V_i or not. Specifically, $h : V \times [\mathcal{M}] \to \{0, 1\}$ such that $\mathbb{P}[h(v, i) = 1] = \hat{p}$ for all $v \in V$ and $i \in [\mathcal{M}]$. Recall that $\mathcal{M}$ is the number of machines, and $\hat{p} = \frac{1}{10} \cdot \sqrt{\frac{S}{mk}}$ is the sampling probability set in Eq.* (1).

Algorithm 4 Compute-Induced-Subgraphs.

```
1: $Q_v \leftarrow \{i \in [\mathcal{M}] \mid h(v,i) = 1\}$.
2: $Q_w \leftarrow \{i \in [\mathcal{M}] \mid h(w,i) = 1\}$.
3: parfor $i \in Q_v \cap Q_w$ do
4:     Send $e$ to machine $M_i$, containing $V_i$.
5: end parfor
```

Using limited independence. Ideally, we would want a perfect hash function, which would allow us to sample the V_i's i.i.d. from the uniform distribution on V. However, since the hash function needs to be known globally, it must fit into each of the machines. This implies that we *cannot* use a fully independent perfect hash function. Rather, we *can* use one that has a high level of independence. Specifically, given that the space per machine is S, we can have a globally known hash function h that is k-wise independent[5] for any $k < \Theta(S/\log n)$. In fact, we can get away with as little as $(6 \log n)$-wise independence (i.e., $k = 6 \log n$). Recalling Eq. (1), this also fixes the sampling probability to be $\hat{p} = \sqrt{S/600m \log n}$.

5.4 Challenge (3): Showing that, with high constant probability, the size of the sent/received messages is bounded

We need to show that the number of edges sent and received by any machine M_i is at most S *with high constant probability.* To this end, we partition the vertex set V into V_{light} and V_{heavy} by picking a threshold degree τ for the vertices. Following this, we define *light edges* as ones that have both end-points in V_{light}, and conversely, any edge with at least one end-point in V_{heavy} is designated as *heavy.* In order for the protocol to suceed, the following must hold:
(A) The number of *light edges* concentrates (see Appendix A.1.4).
(B) The number of *heavy edges* concentrates (see Appendix A.1.5).
(C) The number of sent messages is at most S (see Appendix A.1.6).

The first two items ensure that each machine M_i *receives* at most S messages, and the last item ensures that each machine *sends* at most S messages. Given the above, we proceed to address the last challenge.

5.5 Challenge (4): $\hat{T}$ is close to its expected value

In this section, we provide merely a brief discussion of this challenge for intuition, and we fully analyze the approximation guarantees of our algorithm in Appendix A.1.3. That analysis also makes clear the source of our advertised lower-bound on T for which an estimated count concentrates well.

Lower Bound on Number of Triangles. In order to output *any* approximation (note that we are ignoring all factors of ε and $O(\operatorname{poly}\log n)$ here) to the triangle count, we must see $\Omega(1)$ triangles amongst all of the induced subgraphs on all the machines. The expected number of triangles in a specific induced $G[V_i]$ is $\hat{p}^3 T$, and therefore, the expected number of triangles overall is $\hat{p}^3 T \mathcal{M}$ which must be $\Omega(1)$ for some setting of T. Since we set $\hat{p}$ such that $\hat{p}^2 m = \Theta(S)$, this gives that $\hat{p}^2 = O(S/m)$ which implies $\hat{p}^2 \cdot \mathcal{M} = \hat{p}^2 \cdot (m/S) = \Theta(1)$.

[5] A k-wise independent hash function is one where the hashes of *any* k distinct keys are guaranteed to be independent random variables (see [92]).

This then immmediately implies that to show that $\hat{p}^3 T$ is $\Omega(1)$, we need only show that $\hat{p} \cdot T$ is $\Omega(1)$. Specifically, we show in Lemma 20 that when $T > 1/\hat{p}$, we can obtain a $(1 \pm \varepsilon)$-approximation. To get some intuition for this lower bound on T, note that, in the linear memory regime, when $S = \Theta(n)$, this translates to $T > \sqrt{d_{avg}}$, where d_{avg} is the average degree of G.

$$T > \frac{1}{\hat{p}} = \widetilde{\Theta}\left(\sqrt{\frac{m}{S}}\right) \xRightarrow{\text{for } S=\widetilde{\Theta}(n)} T > \widetilde{\Theta}\left(\sqrt{d_{avg}}\right).$$

6 Open Questions

There are many interesting open questions that result from our study; among these open questions include improving the bounds presented in our algorithm: the round complexity and total space usage in our exact algorithms and the space per machine in our approximation algorithms. In addition to these questions, we also discussion two additional open questions with a larger research scope.

Small subgraph counting counting for a broader class of small subgraphs

Two recent works of [32, 26] extend the result of [25] to a broader set of small subgraphs in the sequential model. However, their results depend crucially on a *DAG tree decomposition* which is non-trivial to implement in the MPC model. Furthermore, even given this DAG tree decomposition, their approach requires iterating through the tree from the leaf level by level up the tree. Such a procedure when implemented in the MPC model requires number of rounds that is $O(depth)$ where *depth* is the depth of the tree. The depth may not be $\text{poly}(\log n)$. In order to obtain efficient MPC implementation of these new algorithms, we must find novel solutions to the above two challenges.

Counting in the AMPC model

A new (stronger) model of MPC, called the *adaptive* MPC model, was recently introduced by [22]. The AMPC model allows access to a *shared* distributed hash table at the end of every round; additionally, the algorithms are allowed *adaptive* access to this hash table. Such a model has shown to be very practical and have led to improvements in the number of rounds over previous MPC algorithms. Such a model seems to be quite relevant to our work since one of the main challenges in our approximation algorithms is to find the set of edges to give to each machine. (Such a challenge may no longer exist given a shared-memory distributed hash table.) We leave as an interesting open question to obtain better, more round efficient approximate triangle counting algorithms in the AMPC model.

Triangle Counting in $O(1)$ Rounds in Sparse Graphs

For sparse graphs where $m = \tilde{O}(n)$, our approximation algorithm requires $\tilde{\Omega}(n)$ space per machine which means that (almost) the entire graph can fit on one machine. This naturally leads to an interesting open question for whether we can obtain an approximate or exact triangle counting algorithm in $O(1)$ rounds in sparse graphs while using *sublinear* space per machine (n^δ space for any constant $\delta > 0$).

References

1 GraphChallenge. URL: `http://graphchallenge.mit.edu/`.

2 Foto N Afrati, Dimitris Fotakis, and Jeffrey D Ullman. Enumerating subgraph instances using map-reduce. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 62–73. IEEE, 2013.

3 Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, Nick G. Duffield, and Theodore L. Willke. Graphlet decomposition: framework, algorithms, and applications. *Knowl. Inf. Syst.*, 50(3):689–722, 2017.

4 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *SPAA*, pages 202–211, 2015.

5 Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica*, 80(2):668–697, 2018.

6 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009. `doi:10.1007/s00453-008-9204-0`.

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

8 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, pages 574–583, 2014.

9 Alexandr Andoni, Clifford Stein, and Peilin Zhong. Log diameter rounds algorithms for 2-vertex and 2-edge connectivity. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.14`.

10 Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. Patric: A parallel algorithm for counting triangles in massive networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 529–538, 2013.

11 Sepehr Assadi. Simple round compression for parallel vertex cover. *arXiv preprint*, 2017. `arXiv:1709.04599`.

12 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019.

13 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *ITCS*, volume 124 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

14 Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. *arXiv preprint*, 2018. `arXiv:1805.02974`.

15 Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 461–470. ACM, 2019. `doi:10.1145/3293611.3331596`.

16 Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

17 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 273–284, 2013.

18 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 212–223, 2014.

19 Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively parallel computation of matching and MIS in sparse graphs. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 481–490. ACM, 2019. `doi:10.1145/3293611.3331609`.

20 Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Semi-mapreduce meets congested clique. *arXiv preprint*, 2018. `arXiv:1802.10297`.

21 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ESA.2019.15`.

22 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab Mirrokni, and Warren Schudy. Massively parallel computation via remote memory access. *ACM Transactions on Parallel Computing*, 8(3):1–25, 2021.

23 Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. Exponentially faster massively parallel maximal matching. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1637–1649. IEEE Computer Society, 2019. `doi:10.1109/FOCS.2019.00096`.

24 Suman K Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *34th Symposium on Theoretical Aspects of Computer Science*, 2017.

25 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear Time Subgraph Counting, Graph Degeneracy, and the Chasm at Size Six. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ITCS.2020.38`.

26 Suman K Bera, Noujan Pashanasangi, and C Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: the barrier of long induced cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2315–2332. SIAM, 2021.

27 Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Slobodan Mitrovic, and Ronitt Rubinfeld. Parallel algorithms for small subgraph counting. *CoRR*, abs/2002.08299, 2020. `arXiv:2002.08299`.

28 Andreas Bjöklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. *Algorithms - ESA 2009*, pages 578–586, 2009. `doi:10.1007/978-3-642-04128-0_52`.

29 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: quantum and MapReduce. In *SODA*, pages 1170–1189, 2018.

30 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the linear-memory barrier in MPC: Fast MIS on trees with n^ϵ memory per machine. *arXiv preprint*, 2018. `arXiv:1802.06748`.

31 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *International Colloquium on Automata, Languages, and Programming*, pages 244–254. Springer, 2013.

32 Marco Bressan. Faster subgraph counting in sparse graphs. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

33 Keren Censor-Hillel, Petteri Kaski, Janne H Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019.

34 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\Delta+1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 471–480. ACM, 2019. doi:10.1145/3293611.3331607.

35 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on computing*, 14(1):210–223, 1985.

36 Shumo Chu and James Cheng. Triangle listing in massive networks and its applications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 672–680, 2011.

37 Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

38 Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1197655.

39 Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs*. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 589–598, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. doi:10.1145/3178876.3186125.

40 V. S. Dave, N. K. Ahmed, and M. Hasan. PE-CLoG: Counting edge-centric local graphlets. In *IEEE International Conference on Big Data*, pages 586–595, 2017.

41 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

42 Laxman Dhulipala, Quanquan C. Liu, Julian Shun, and Shangdi Yu. Parallel batch-dynamic k-clique counting. In Michael Schapira, editor, *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021*, pages 129–143. SIAM, 2021. doi:10.1137/1.9781611976489.10.

43 Danny Dolev, Christoph Lenzen, and Shir Peled. "Tri, Tri again": Finding triangles and small subgraphs in a distributed setting. *Distributed Computing*, pages 195–209, 2012. doi:10.1007/978-3-642-33651-5_14.

44 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

45 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1467–1478, 2020. doi:10.1137/1.9781611975994.89.

46 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1–3):57–67, October 2004. doi:10.1016/j.tcs.2004.05.009.

47 Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed estimation of graph 4-profiles. In *International Conference on World Wide Web (WWW)*, pages 483–493, 2016.

48 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithms*, 18(3):364–375, 2013. doi:10.1145/2543629.

49 Irene Finocchi, Marco Finocchi, and Emanuele G Fusco. Clique counting in mapreduce: Algorithms and experiments. *Journal of Experimental Algorithmics (JEA)*, 20:1–20, 2015.

50 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500. ACM, 2019. `doi:10.1145/3293611.3331603`.

51 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *PODC*. `arXiv:1802.08237`, 2018.

52 Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1650–1663. IEEE Computer Society, 2019. `doi:10.1109/FOCS.2019.00097`.

53 Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. Improved parallel algorithms for density-based network clustering. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2201–2210, Long Beach, California, USA, 09–15 June 2019. PMLR. URL: `http://proceedings.mlr.press/v97/ghaffari19a.html`.

54 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1260–1279. SIAM, 2020. `doi:10.1137/1.9781611975994.77`.

55 Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653. SIAM, 2019. `doi:10.1137/1.9781611975482.99`.

56 Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, pages 159–167, 2006. `doi:10.1007/11917496_15`.

57 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Proceedings of the 22Nd International Conference on Algorithms and Computation*, ISAAC'11, pages 374–383, Berlin, Heidelberg, 2011. Springer-Verlag. `doi:10.1007/978-3-642-25591-5_39`.

58 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the MapReduce model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 43–52, New York, NY, USA, 2018. ACM. `doi:10.1145/3210377.3210386`.

59 James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to MapReduce. *Theoretical Computer Science*, 608:268–281, 2015.

60 Tomaz Hocevar and Janez Demsar. A combinatorial approach to graphlet counting. *Bioinformatics*, pages 559–65, 2014.

61 Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient massively parallel methods for dynamic programming. In *STOC*, pages 798–811, 2017.

62 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41(3), pages 59–72. ACM, 2007.

63 Giuseppe F. Italiano, Silvio Lattanzi, Vahab S. Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 49–58. ACM, 2019. `doi:10.1145/3323165.3323202`.

64 Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th International Conference on World Wide Web*, pages 441–449, 2017.

65 Madhav Jha, Comandur Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 589–597, 2013.

66 Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 598–609. Springer, 2012.

67 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

68 Tamara G Kolda, Ali Pinar, Todd Plantenga, C Seshadhri, and Christine Task. Counting triangles in massive graphs with mapreduce. *SIAM Journal on Scientific Computing*, 36(5):S48–S77, 2014.

69 Mihail N Kolountzakis, Gary L Miller, Richard Peng, and Charalampos E Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.

70 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.

71 Jakub Lacki, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. *CoRR*, abs/1907.05391, 2019. arXiv:1907.05391.

72 Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. Scalable subgraph enumeration in mapreduce. *Proceedings of the VLDB Endowment*, 8(10):974–985, 2015.

73 Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical computer science*, 407(1-3):458–473, 2008.

74 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA*, pages 85–94, 2011.

75 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in o (log log n) communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

76 Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 401–411, 2016.

77 Krzysztof Onak. Round compression for parallel graph algorithms in strongly sublinear space. *CoRR*, abs/1807.08745, 2018. arXiv:1807.08745.

78 Rasmus Pagh and Charalampos E Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

79 Ha-Myung Park and Chin-Wan Chung. An efficient mapreduce algorithm for counting triangles in a very large graph. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 539–548, 2013.

80 Ha-Myung Park, Francesco Silvestri, U Kang, and Rasmus Pagh. Mapreduce triangle enumeration with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1739–1748, 2014.

81 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610, 2010. doi:10.1145/1806689.1806772.

82 Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *International Conference on World Wide Web (WWW)*, pages 1431–1440, 2017.

83 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits: (on lower bounds for modern parallel computation). In *SPAA*, pages 1–12, 2016.

84 Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer, 2005.

85 Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 10–18. SIAM, 2013.

86 Jessica Shi, Laxman Dhulipala, and Julian Shun. Parallel clique counting and peeling algorithms. *CoRR*, abs/2002.10047, 2020. `arXiv:2002.10047`.

87 Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems*, 54(3):677–710, 2018.

88 J. Shun and K. Tangwongsan. Multicore triangle computations without tuning. In *2015 IEEE 31st International Conference on Data Engineering*, pages 149–160, April 2015. `doi:10.1109/ICDE.2015.7113280`.

89 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614, 2011.

90 Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 837–846, 2009.

91 Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254–257, 2009. `doi:10.1016/j.ipl.2008.10.014`.

92 Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

93 Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

94 Jin-Hyun Yoon and Sung-Ryul Kim. Improved sampling for triangle counting with mapreduce. In *International Conference on Hybrid Information Technology*, pages 685–689. Springer, 2011.

95 Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

A Exact Triangle Counting Analysis

A.1 Detailed Analysis

In this section we give the full details and analysis of algorithm Algorithm 1 given in Section 4, for exactly counting the number of triangles in the graph.

We first provide a detailed version of Algorithm 2 that also takes into account over counting due to the fact that each triangle might be counted by several endpoints, and then continue to prove the main theorem of this section, Theorem 4.

A.1.1 Details about finding duplicate elements using Theorem 8

Find-Triangles$(w, \mathcal{L}_w)$ finds triangles by counting the number of duplicates that occur between elements in lists. Theorem 8 provides a MPC implementation for finding the count of all occurrences of every element in a sorted list. Provided a sorted list of neighbors of $v \in Q_i$ and neighbor lists in $\mathcal{L}_v$, this function counts the number of intersections between a neighbor list sent to v and the neighbors of v. Every intersection indicates the existence of a triangle. As given, Find-Triangles$(w, \mathcal{L}_w)$ (see v Algorithm 2) returns a 6-approximation of the number of triangles in any graph. We provide a detailed and somewhat more complicated algorithm Find-Triangles-Exact$(w, \mathcal{L}_w)$ that accounts for over-counting of triangles and returns the exact number of triangles.

Since Theorem 8 returns the total count of each element, we subtract the value returned by 1 to obtain the number of intersections. Finally, each triangle containing one low-degree vertex will be counted twice, each containing two low-degree vertices will be counted 4 times, and each containing three low-degree vertices will be counted 6 times. Thus, we need to divide the counts by 2, 4, and 6, respectively, to obtain the exact count of unique triangles.

Algorithm 5 Find-Triangles-Exact$(w, \mathcal{L}_w)$.

```
1:  Set the number of triangles $T_i \leftarrow 0$.
2:  Sort all elements in $(\mathcal{L}_w \cup (N(w) \cap Q_i))$ lexicographically using the procedure given in
    Lemma 4.3 of [57]. Let this sorted list of all elements be $S$.
3:  Count the duplicates in $S$ using Theorem 8.
4:  parfor all $v \in N(w)$ do
5:      Let $R$ be the number of duplicates of $v$ returned by Theorem 8.
6:      if $d_{Q_i}(v) > \gamma_i$ and $d_{Q_i}(w) > \gamma_i$ then
7:          Increment $T_i \leftarrow T_i + \frac{R-1}{2}$.
8:      else if $(d_{Q_i}(v) > \gamma_i$ and $d_{Q_i}(w) \leq \gamma_i)$ or $(d_{Q_i}(v) \leq \gamma_i$ and $d_{Q_i}(w) > \gamma_i)$ then
9:          Increment $T_i \leftarrow T_i + \frac{R-1}{4}$.
10:     else
11:         Increment $T_i \leftarrow T_i + \frac{R-1}{6}$.
12: end parfor
13: Return $T_i$.
```

Substituting Find-Triangles-Exact in Count-Triangles finds the exact count of triangles in graphs with arboricity α using $O(m\alpha)$ total space.

A.1.2 Proof of Theorem 4

First, all proofs below assume we start at a cutoff of $\gamma = 4\alpha$. Because we increase the cutoff bound doubly exponentially, we can reach such a bound in $O(\log \log \alpha)$ rounds. Thus, in the following proofs, we ignore all rounds before we get to a round where $\gamma \geq 4\alpha$. Before proving the theorem, we provide several useful lemmas stating that the number of vertices and edges remaining at the beginning of each iteration is bounded.

▶ **Lemma 17.** *At the beginning of iteration i of Count-Triangles, given $\gamma_i = 2^{(3/2)^i} \cdot (2\alpha)$ as stated in Algorithm 1, the number of remaining vertices $N_i = |Q_i|$ is at most $\frac{n}{2^{2\cdot((3/2)^i-1)}}$.*

Proof. Let N_i be the number of vertices in Q_i at the beginning of iteration i. Since the subgraph induced by Q_i must have arboricity bounded by α, we can bound the total degree of Q_i,

$$\sum_{v \in Q_i} d_{Q_i}(v) < 2\alpha|Q_i| = 2N_i\alpha.$$

At the end of the iteration, we only keep the vertices in $Q_{i+1} = \{v \in Q_i \mid d_{Q_i}(v) > \gamma_i\}$. If we assume that $|Q_{i+1}| > \frac{N_i}{\gamma_i/(2\alpha)}$, then we obtain a contradiction since this implies that

$$\sum_{v \in Q_{i+1}} d_{Q_i}(v) > |Q_{i+1}| \cdot \gamma_i > 2N_i\alpha > \sum_{v \in Q_i} d_{Q_i}(v).$$

Then, the number of remaining vertices follows directly from the above by induction on i with base case $N_1 = n$,

$$N_i \le \frac{N_{i-1}}{\gamma_i/(2\alpha)} = \frac{N_{i-1}}{2^{(3/2)^{i-1}}} \le \frac{n}{\prod_{j=0}^{i-1} 2^{(3/2)^j}} = \frac{n}{2^{2\cdot((3/2)^i-1)}}. \qquad ◀$$

We can show a similar statement for the number of edges that remain at the start of the i^{th} iteration.

▶ **Lemma 18.** *At the beginning of iteration i of* Count-Triangles*, given γ_i, the number of remaining edges m_i is at most $m_i \le \frac{m}{2^{2\cdot((3/2)^{i-1}-1)}}$.*

Proof. The number of vertices remaining at the beginning of iteration i is given by $|Q_i|$. Thus, because the arboricity of our graph is α, we can upper bound m_i by

$$m_i \le |Q_i|\alpha.$$

Then, we can also lower bound the number of edges at the beginning of iteration $i-1$ since the vertices that remain at the beginning of round i are ones which have greater than γ_{i-1} degree,

$$m_{i-1} \ge \frac{1}{2} \sum_{v \in Q_{i-1}} d_{Q_{i-1}}(v) \ge \frac{1}{2}|Q_i|\gamma_{i-1}.$$

Thus, we conclude that $m_i \le \frac{2\alpha m_{i-1}}{\gamma_{i-1}}$. By induction on i with base case $m_0 = m$, we obtain,

$$m_i \le 2\alpha\left(\frac{m_{i-1}}{\gamma_{i-1}}\right) \le \frac{m}{\prod_{j=0}^{i-2} 2^{(3/2)^j}} = \frac{m}{2^{2\cdot((3/2)^{i-1}-1)}}. \qquad ◀$$

The above lemmas allows us to bound the total space used by the algorithm.

▶ **Lemma 19.** Count-Triangles*(G) uses $O(m\alpha)$ total space when run on a graph G with arboricity α.*

Proof. The total space the algorithm requires is the sum of the space necessary for storing the neighbor lists sent by all vertices with degree $\le \gamma_i$ and the space necessary for all vertices to store their own neighbor lists. The total space necessary for each vertex to store its own neighbor list is $O(m)$.

Now we compute the total space used by the algorithm during iteration i. The number of vertices in Q_i at the beginning of this iteration is at most $N_i \le \frac{n}{2^{2\cdot((3/2)^i-1)}}$ by Lemma 17. Each vertex v with $d_{Q_i}(v) \le \gamma_i$, makes $d_{Q_i}(v)$ copies of its neighbor list $(N(v) \cap Q_i)$ and sends each neighbor in $N(v) \cap Q_i$ a copy of the list. Thus, the total space required by the messages sent by v is $d_{Q_i}(v)^2 \le \gamma_i^2$. v sends at most one message of size $d_{Q_i}(v) \le \gamma_i$ along each edge (v, w) for $w \in N(v) \cap Q_i$. Then, by Lemma 18 the total space required by all the low-degree vertices in round i is at most (as at most two messages are sent along each edge):

$$2m_i \cdot \gamma_i < \frac{m}{2^{2\cdot((3/2)^{i-1}-1)}} \cdot \left[2^{(3/2)^i}(2\alpha)\right] = 16m\alpha. \qquad ◀$$

We are now ready to prove Theorem 4.

Proof of Theorem 4. By Lemma 17, the number of vertices remaining in Q_i at the beginning of iteration i is $\frac{n}{2^{2\cdot((3/2)^i-1)}}$. This means that the procedure runs for $O(\log\log n)$ iterations before there will be no vertices. For each of the $O(\log\log n)$ iterations, $\textsc{Count-Triangles}(G)$ uses $O_\delta(1)$ rounds of communication for the low-degree vertices to send their neighbor lists to their neighbors. The algorithm then calls $\textsc{Find-Triangles-Exact}(w, \mathcal{L}_w)$ on each vertex $w \in Q_i$ (in parallel) to find the number of triangles incident to w and vertices in $A_i \subseteq Q_i$. $\textsc{Find-Triangles-Exact}(w, \mathcal{L}_w)$ requires $O\left(\log_{n^\delta}(m\alpha)\right) = O(1/\delta)$ rounds by Lemma 4.3 of [57] and Theorem 8. Therefore, the total number of rounds required by $\textsc{Count-Triangles}(G)$ is $O\left(\frac{\log\log n}{\delta}\right) = O_\delta(\log\log n)$. ◀

A.1.3 Showing Concentration for the Triangle Count

In the subsequent proofs, we will use the following assumptions from within Theorem 1 (note that we added specific constants).

$$T \geq 10\sqrt{\frac{mk}{S}} \qquad S \geq \max\left\{15\frac{\sqrt{mk}}{\varepsilon}, \frac{100kn^2}{m}\right\} \qquad \mathcal{M} = \frac{2000mk}{\varepsilon^2 S} \tag{3}$$

Note that we set the number of machines to a specific value, instead of lower bounding it. This is acceptable, because we can just ignore some of the machines.

Algorithm 3 outputs an estimate on the number of triangles in G (Line 11). It is not hard to show that in expectation this output equals T *even with limited independence* as discussed above. The main challenge is to show that this output also concentrates well around its expectation. Specifically, we show the following claim.

▶ **Lemma 20.** *Ignore Line 4 of Algorithm 3. Let $\hat{T}$ be as defined on Line 10 and $\mathcal{M} = \frac{20}{\varepsilon^2\hat{p}^2}$ be as defined in Eq. (3), and assume that $T \geq 1/\hat{p}$. Then, the following hold:*

(A) $\mathbb{E}\left[\hat{T}\right] = \hat{p}^3 \cdot R \cdot T$, *and*

(B) $\mathbb{P}\left[|\hat{T} - \mathbb{E}\left[\hat{T}\right]| > \varepsilon\mathbb{E}\left[\hat{T}\right]\right] < \frac{1}{10}$.

We will prove Property (B) of the claim by applying Chebyshev's inequality, for which we need to compute $\mathrm{Var}\left[\hat{T}\right]$. Let $\Delta(G)$ be the set of all triangles in G. For a triangle $t \in \Delta(G)$, let $\hat{T}_{i,t} = 1$ if $t \in V[G_i]$, and $\hat{T}_{i,t} = 0$ otherwise. Hence, $\hat{T}_i = \sum_{t\in\Delta(G)} \hat{T}_{i,t}$. We begin by deriving $\mathbb{E}\left[\hat{T}\right]$ and then proceed to showing that $\mathrm{Var}\left[\hat{T}\right] = \sum_{i=1}^{R} \mathrm{Var}\left[\hat{T}_i\right]$. After that we upper-bound $\mathrm{Var}\left[\hat{T}_i\right]$ and conclude the proof by applying Chebyshev's inequality.

A.1.3.1 Deriving $\mathbb{E}\left[\hat{T}\right]$

Let t be a triangle in G. Let $\hat{T}_t$ be a random variable denoting the total number of times t appears in $G[V_i]$, for all $i = 1 \ldots R$. Given that $\mathbb{P}\left[u \in V_i\right] = \hat{p}$, we have that $\mathbb{P}\left[t \in G[V_i]\right] = \hat{p}^3$. Therefore, $\mathbb{E}\left[\hat{T}_t\right] = R \cdot \hat{p}^3$.

Since $\hat{T} = \sum_{t\in\Delta(G)} \hat{T}_t$, we have

$$\mathbb{E}\left[\hat{T}\right] = \sum_{t\in\Delta(G)} \mathbb{E}\left[\hat{T}_t\right] = \hat{p}^3 \cdot R \cdot T. \tag{4}$$

This proves Property (A) of this claim.

A.1.3.2 Decoupling Var $\left[\hat{T}\right]$

To compute variance, one considers the second moment of a given random variable. So, to compute Var $\left[\hat{T}\right]$, we will consider products $\hat{T}_{i,t_1} \cdot \hat{T}_{j,t_2}$. Each of those products depend on at most 6 vertices. Now, given that we used a 6-wise independent function (see Section 5.3) to sample vertices in each V_i, one could expect that Var $\left[\hat{T}_i\right]$ and Var $\left[\hat{T}_j\right]$ for $i \neq j$ behave like they are independent, i.e., one could expect that it holds Var $\left[\hat{T}\right] = \sum_{i=1}^{R} \text{Var}\left[\hat{T}_i\right]$. As we show next, it is indeed the case. We have

$$\text{Var}\left[\hat{T}\right] = \mathbb{E}\left[\hat{T}^2\right] - \mathbb{E}\left[\hat{T}\right]^2 = \mathbb{E}\left[\left(\sum_{i=1}^{R} \sum_{t \in \Delta(G)} \hat{T}_{i,t}\right)^2\right] - \left(\sum_{i=1}^{R} \sum_{t \in \Delta(G)} \mathbb{E}\left[\hat{T}_{i,t}\right]\right)^2$$

Consider now $\hat{T}_{i,t_1}$ and $\hat{T}_{j,t_2}$ for $i \neq j$ and some $t_1, t_2 \in \Delta(G)$ not necessarily distinct. In the first summand of (5), we will have $\mathbb{E}\left[2\hat{T}_{i,t_1} \cdot \hat{T}_{j,t_2}\right]$. The vertices constituting t_1 and t_2 are 6 *distinct* copies of some (not necessarily all distinct) vertices of V. Since they are chosen by applying a 6-wise independent function, we have $\mathbb{E}\left[2\hat{T}_{i,t_1} \cdot \hat{T}_{j,t_2}\right] = 2\mathbb{E}\left[\hat{T}_{i,t_1}\right] \cdot \mathbb{E}\left[\hat{T}_{j,t_2}\right]$. On the other hand, the second summand of (5) also contains $2\mathbb{E}\left[\hat{T}_{i,t_1}\right] \cdot \mathbb{E}\left[\hat{T}_{j,t_2}\right]$, which follows by direct expansion of the sum. Therefore, all the terms $\mathbb{E}\left[2\hat{T}_{i,t_1} \cdot \hat{T}_{j,t_2}\right]$ in Var $\left[\hat{T}\right]$ for $i \neq j$ cancel each other. So, we can also write Var $\left[\hat{T}\right]$ as

$$\text{Var}\left[\hat{T}\right] = \sum_{i=1}^{R} \mathbb{E}\left[\left(\sum_{t \in \Delta(G)} \hat{T}_{i,t}\right)^2\right] - \sum_{i=1}^{R} \left(\sum_{t \in \Delta(G)} \mathbb{E}\left[\hat{T}_{i,t}\right]\right)^2 = \sum_{i=1}^{R} \text{Var}\left[\hat{T}_i\right].$$

Therefore, to upper-bound Var $\left[\hat{T}\right]$ it suffices to upper-bound Var $\left[\hat{T}_i\right]$.

A.1.3.3 Upper-bounding Var $\left[\hat{T}_i\right]$

We have

$$\begin{aligned} \text{Var}\left[\hat{T}_i\right] &= \mathbb{E}\left[\left(\sum_{t \in \Delta(G)} \hat{T}_{i,t}\right)^2\right] - \left(\sum_{t \in \Delta(G)} \mathbb{E}\left[\hat{T}_{i,t}\right]\right)^2 \le \mathbb{E}\left[\left(\sum_{t \in \Delta(G)} \hat{T}_{i,t}\right)^2\right] \\ &= \mathbb{E}\left[\sum_{t \in \Delta(G)} \hat{T}_{i,t}^2\right] + \mathbb{E}\left[\sum_{t_1, t_2 \in \Delta(G); t_1 \neq t_2} \hat{T}_{i,t_1} \cdot \hat{T}_{i,t_2}\right]. \qquad (5) \end{aligned}$$

Since each $\hat{T}_{i,t}$ is a 0/1 random variables, $\hat{T}_{i,t}^2 = \hat{T}_{i,t}$. Let $t_1 \neq t_2$ be two triangles in $\Delta(G)$. Let k be the number of distinct vertices they are consisted of, which implies $4 \le k \le 6$. Then, observe that $\mathbb{E}\left[\hat{T}_{i,t_1} \cdot \hat{T}_{i,t_2}\right] = \hat{p}^k \le \hat{p}^4$. We now have all ingredients to upper-bound Var $\left[\hat{T}_i\right]$. From (5) and our discussion it follows

$$\text{Var}\left[\hat{T}_i\right] \le T\hat{p}^3 + T^2\hat{p}^4 \le 2T^2\hat{p}^4, \qquad (6)$$

where we used our assumption that $T \ge 1/\hat{p}$.

A.1.3.4 Finalizing the proof

From (5) and (6) we have

$$\mathrm{Var}\left[\hat{T}\right] \leq 2RT^2\hat{p}^4.$$

So, from Chebyshev's inequality and (4) we derive

$$\mathbb{P}\left[|\hat{T} - \mathbb{E}\left[\hat{T}\right]| > \varepsilon\mathbb{E}\left[\hat{T}\right]\right] < \frac{\mathrm{Var}\left[\hat{T}\right]}{\varepsilon^2\mathbb{E}\left[\hat{T}\right]^2} \leq \frac{2RT^2\hat{p}^4}{\varepsilon^2\hat{p}^6R^2T^2} = \frac{2}{\varepsilon^2\hat{p}^2R}.$$

Hence, for $R \geq \frac{20}{\varepsilon^2\hat{p}^2}$ we get the desired bound.

A.1.4 Bounding the Number of Light Edges *Received* by a Machine

We will now bound the probability that any of the induced subgraphs *does not fit* on a machine. To that end, we set a degree threshold $\tau = \frac{k}{\hat{p}}$, and define the set of *light* vertices V_{light} to be the ones with degree less than τ. All other vertices are *heavy*, and we let them comprise the set V_{heavy}.

Fix a machine M_i. We prove that, with probability at least $9/10$, the number of edges in $G[V_i]$ is upper bounded by S.

We start with analyzing the contribution of the light vertices to the induced subgraphs. We first consider the simpler case of bounding the number of edges in $G[V_i]$ that have both end-points in V_{light}. We refer to such edges as *light edges* and denote them by E_{light}. For every edge $e \in E_{light}$, we define a random variable $Z_e^{(i)}$ as follows.

$$Z_e^{(i)} = \begin{cases} 1 & \text{if } e \in G[V_i], \\ 0 & \text{otherwise.} \end{cases}$$

We let $Z^{(i)}$ be the sum over all random variables Z_e^i, $Z^i = \sum_{e \in E_{light}} Z_e^i$, and we let m_ℓ denote the total number of edges with *light* endpoints in the original graph G, i.e., $m_\ell = |E_{light}|$. Due to space constraints, the proof of the following lemma can be found in our full paper [27].

We prove the following lemma.

▶ **Lemma 21.** *With probability at least* $9/10$, *for every* $i \in [\mathcal{M}]$, $G[V_i]$ *contains at most* $\frac{1}{4}S$ *light edges.*

We can now use Chebyshev's inequality to conclude that

$$\mathbb{P}\left[|Z^{(i)} - \mathbb{E}[Z^{(i)}]| > S/\sqrt{3}\right] \leq \frac{\mathrm{Var}\left[Z^{(i)}\right]}{S^2/3}$$
$$\implies \mathbb{P}\left[Z^{(i)} > 3S/4\right] \leq \frac{3}{30S} = \frac{1}{10S}$$

Finally, we can use union bound over all $\mathcal{M}$ machines to upper bound the probability that, *any* of the $Z^{(i)}$ values exceeds $3S/4$ (using the the constraints descrbed in Eq. (3) to simplify).

$$\frac{\mathcal{M}}{10S} = \frac{2000mk}{\varepsilon^2 S} \cdot \frac{1}{10S} \leq \frac{200mk}{\varepsilon^2} \cdot \frac{1}{(15\sqrt{mk}/\varepsilon)^2} = \frac{200mk}{\varepsilon^2 S^2},$$

Therefore, with probability at least $9/10$, none of the induced subgraphs $G[V_i]$ will contain more than $3S/4$ light edges.

A.1.5 Bounding the Number of Heavy Edges *Received* by a Machine

Next, we turn our attention to the edges that have at least one endpoint in V_{heavy} (we call such edges *heavy*). We will show that for each $v \in V_{heavy} \cap V_i$, the number of edges contributed by v concentrates around its expectation.[6] In this section, we will use $2m_h$ to denote the total degree of all the heavy vertices i.e. $2m_h = \sum_{v \in V_{heavy}} d(v)$. Due to space constraints, we present the proofs of the following theorems in our full paper [27].

▶ **Theorem 22** (Heavy edges). *With high probability, the number of edges in $G[V_i]$ that have some endpoint with degree larger than τ is at most $S/8$.*

Combining this result with Theorem 22, we conclude the following:

▶ **Theorem 23.** *With probability at least $9/10$, the maximum number of edges in any of the $G[V_i]$s (where $i \in [R]$) does not exceed S, and hence Algorithm 3 does not terminate on Line 4.*

A.1.6 Upper-Bounding the Number of Messages *Sent* by any Machine

Recalling Algorithm 4, we note that the number of messages *received* by the machine containing V_i, is equal to the number of edges in $G[V_i]$. Therefore, the last section essentially proved that the number of messages (edges) *received* by a particular machine is upper-bounded by S. Conversely, in this section, we will justify that the number of messages *sent* by any machine is $O(S)$. Since the number of edges stored in a machine is $\leq S$, it suffices to to show that for each edge e, Algorithm 4 sends only $O(1)$ messages (each message is a copy of the edge e). Our full proofs are included in our full paper [27].

Let $Z_i^{(e)}$ be the $\{0, 1\}$ indicator random variable for $e \in G[V_i]$, and let $Z^{(e)}$ be the sum of $Z_i^{(e)}$ for all $i \in [\mathcal{M}]$. Here, $Z^{(e)}$ represents the number of messages that are created by edge e. Additionally we make $r = S\mathcal{M}/m = O_\varepsilon(\log n)$ copies of each edge e, and ensure that all replicates reside on the same machine. We distribute the $Z^{(e)}$ messages evenly amongst the replicates, so that each replica is only responsible for $Z^{(e)}/r$ messages.

Since all replicates are on the same machine, this last step is purely conceptual, but it will simplify our arguemnt, by allowing us to charge the outgoing messages to each replicate (as opposed to each edge). Our goal will be show that each replicate is responsible for only $O(1)$ messages, which is the same as showing that w.h.p. $Z^{(e)}/r = O(1)$.

Clearly $\mu = \mathbb{E}[Z^{(e)}] = \hat{p}^2 \cdot \mathcal{M} = \frac{S\mathcal{M}}{100mk}$. With $\delta = \frac{100e^{1/3}mk^2}{S\mathcal{M}}$

$$\mathbb{P}\left[Z^{(e)} > \delta\mu\right] \leq e^{-\lfloor k/2 \rfloor} = \frac{1}{n^3} \implies \mathbb{P}\left[\frac{Z^{(e)}}{r} > \frac{e^{1/3}k}{r}\right] \leq \frac{1}{n^3}$$

Using the assumption (from Eq. (3)) that $\mathcal{M} > 2000mk/S \implies r > 2000k$, we see that with high probability, the number of messages sent by any replicate is bounded above by $e^{1/3}/2000 \leq 1$. So, the number of messages sent from any machine is bounded by S with high probability.

A.1.7 Getting the High Probability Bound

By building on Lemma 20 and Algorithm 3, we design Algorithm 6 that outputs an approximate triangle counting with high probability, as opposed with only constant success probability. It is important to note that in the below algorithm, all $O(\log n)$ *independent iterations* (Line 3) are done *in parallel*, simultaneously, not sequentially.

[6] Intuitively, this is because v has high degree, and therefore the number of its sampled neighbors ($|N(v) \cap V_i|$) will concentrate.

Algorithm 6 **Approximate Triangle Counting**.

```
1: function APPROX-TRIANGLES-MAIN($G = (V, E)$)
2:     Let $I \leftarrow 100 \cdot \log n$.
3:     parfor $i \leftarrow 1 \ldots I$ do    ▷ Perform all $I$ iterations in *parallel* simultaneously in $O(1)$ rounds.
4:         Let $Y_i$ be the output of Algorithm 3 invoked on $G$. We assume that each invocation of Algorithm 3 uses fresh randomness compared to previous runs.
5:     end parfor
6:     Let $\mathcal{Y}$ be the list of all $Y_i$, for $i = 1 \ldots I$.
7:     Sort $\mathcal{Y}$ in non-decreasing order.
8:     return the median of $\mathcal{Y}$
```

We have the following guarantee for Algorithm 6.

▶ **Theorem 24.** *Let Y be the output of Algorithm 6. Then, with high probability it holds*

$$|Y - T| \le \varepsilon T.$$

In the proof of this theorem we use the following concentration bound.

▶ **Theorem 25** (Chernoff bound)**.** *Let $X_1, \ldots, X_k$ be independent random variables taking values in $[0, 1]$. Let $X \overset{\text{def}}{=} \sum_{i=1}^{k} X_i$ and $\mu \overset{\text{def}}{=} \mathbb{E}[X]$. Then, or any $\delta \in [0, 1]$ it holds $\mathbb{P}[X \le (1 - \delta)\mu] \le \exp\left(-\delta^2 \mu / 2\right)$.*