Approximate Triangle Counting via Sampling and Fast Matrix Multiplication

Jakub Tětek
Basic Algorithms Research Copenhagen, University of Copenhagen, Denmark

Abstract

There is a simple $O(\frac{n^3}{\epsilon^2 T})$ time algorithm for $1\pm\epsilon$-approximate triangle counting where T is the number of triangles in the graph and n the number of vertices. At the same time, one may count triangles exactly using fast matrix multiplication in time $\tilde{O}(n^\omega)$. Is it possible to get a negative dependency on the number of triangles T while retaining the state-of-the-art n^ω dependency on n? We answer this question positively by providing an algorithm which runs in time $O\left(\frac{n^\omega}{T^{\omega-2}}\right) \cdot \text{poly}(n^{o(1)}/\epsilon)$. This is optimal in the sense that as long as the exponent of T is independent of n, T, it cannot be improved while retaining the dependency on n. Our algorithm improves upon the state of the art when $T \gg 1$ and $T \ll n$.

We also consider the problem of approximate triangle counting in sparse graphs, parameterized by the number of edges m. The best known algorithm runs in time $\tilde{O}_\epsilon\left(\frac{m^{3/2}}{T}\right)$ [Eden et al., SIAM Journal on Computing, 2017]. An algorithm by Alon et al. [JACM, 1995] for exact triangle counting that runs in time $\tilde{O}(m^{2\omega/(\omega+1)})$. We again get an algorithm whose complexity has a state-of-the-art dependency on m while having negative dependency on T. Specifically, our algorithm runs in time $O\left(\frac{m^{2\omega/(\omega+1)}}{T^{2(\omega-1)/(\omega+1)}}\right) \cdot \text{poly}(n^{o(1)}/\epsilon)$. This is again optimal in the sense that no better constant exponent of T is possible without worsening the dependency on m. This algorithm improves upon the state of the art when $T \gg 1$ and $T \ll \sqrt{m}$.

In both cases, algorithms with *time* complexity matching *query* complexity lower bounds were known on some range of parameters. While those algorithms have optimal *query* complexity for the whole range of T, the time complexity departs from the query complexity and is no longer optimal (as we show) for $T \ll n$ and $T \ll \sqrt{m}$, respectively. We focus on the time complexity in this range of T. To the best of our knowledge, this is the first paper considering the discrepancy between query and time complexity in graph parameter estimation.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Approximate triangle counting, Fast matrix multiplication, Sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.107

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2104.08501`

Funding The author was supported by the VILLUM Foundation grant 16582 and by the Bakala Foundation.

Acknowledgements I would like to thank several people: Václav Rozhoň, who gave early feedback and who suggested using a recursive approach; Rasmus Pagh for helping improve a draft of this paper; my supervisor Mikkel Thorup for helpful discussions and for his support; and Talya Eden for advice regarding related work.

1 Introduction

The problem of counting triangles in a graph is both a fundamental problem in graph algorithms and a problem with many applications, for example in network science [5], biology [8, 19, 21] or sociology [25, 15]. Triangle counting is, also for these reasons, one of the basic procedures in graph mining. Consequently, triangle counting has received a lot of attention both in the theoretical and applied communities.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 107; pp. 107:1–107:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are algorithms for *exact* triangle counting that have time complexity $\tilde{O}(n^\omega)$ [1] and $\tilde{O}(m^{2\omega/(\omega+1)})$ [3] where ω is the smallest constant such that two $n \times n$ matrices can be multiplied in time $n^{\omega+o(1)}$.

A natural variant of this problem is *approximate triangle counting*. In this problem, we are given a graph with T triangles and want to output $\hat{T} = (1 \pm \epsilon)T$ with probability 2/3. Several algorithms are known for approximate triangle counting that have *negative dependence* on the number of triangles T. Assuming the algorithm may sample random vertices and edges, there are algorithms that run in time $O_\epsilon(\frac{n^3}{T})$ [2] and $\tilde{O}_\epsilon(\frac{m^{3/2}}{T})$ [9][3]. However, no algorithm is known that would get the best of both worlds – algorithms with state-of-the-art dependency on n or m while being negatively dependent on T. We show two such algorithms in this paper. Moreover, our algorithms are optimal in the following sense: the dependence on T in the time complexity of our algorithms cannot be improved without worsening the dependency on n or m as long as the exponent of T is constant (i.e., is independent of n, m, T). This optimality follows from the lower bound shown by Eden, Levi, Ron, and Seshadhri [9], as we show in the full version of this paper.

The main contribution of our paper is an algorithm for approximate triangle counting that runs in time $\tilde{O}_\epsilon\left(\frac{n^\omega}{T^{\omega-2}}\right)$. We then use it to give an algorithm for the same problem running in time $\tilde{O}_\epsilon\left(\frac{m^{2\omega/(\omega+1)}}{T^{2(\omega-1)/(\omega+1)}}\right)$. This improves upon our first algorithm for sparse graphs. Note that both time complexities are sublinear for sufficiently large values of T. To the best of our knowledge, this is the first work that uses fast matrix multiplication in sublinear-time algorithms. Assuming constant ϵ, our algorithms improve upon the state-of-the-art when $T \ll n$, and $T \gg 1$ or $T \ll \sqrt{m}$ and $T \gg 1$ (for Algorithm 4 and Algorithm 5, respectively). In other words, our algorithms improve upon the state of the art when the time complexity (of both our and the state-of-the-art algorithms) is $\gg n^2$ and $\gg m$, respectively. Figure 1 shows a plot of the complexity of our algorithms, in comparison to other algorithms for approximate triangle counting.

The basic approach we use is to sample vertices, recursively count triangles in the subgraph induced by these sampled vertices, and estimate the total number of triangles based on this count. The main hurdle in this approach is that when a vertex is contained in many triangles (we call such a vertex triangle-heavy, and triangle-light otherwise), this results in a poor concentration of the number of triangles in the induced subgraph as the inclusion of one vertex can make a large difference in the number of triangles. We get around this by introducing a procedure based on fast matrix multiplication which finds all vertices contained in many triangles, and a procedure for approximately counting triangles that contain at least one such vertex. We count triangles that consist of light vertices separately. The fact that the vertices are light helps us ensure concentration of our estimate. These triangles are counted using a recursive approach. An obstacle to overcome is that in some cases the time spent in each successive level of recursion may grow exponentially.

---

[1] We use $\tilde{O}(f(x))$ with the meaning that it ignores an $f(x)^{o(1)}$ factor; this is necessary as ω only determines the time complexity up to $n^{o(1)}$. This differs from the usual use in which $\tilde{O}(f(x)) = \cup_{c>0} O(f(x)\log^c f(x))$. $\tilde{O}_\epsilon$ in addition ignores $\text{poly}(\frac{1}{\epsilon})$.

[2] This algorithm follows from a paper of Lipton, Naughton, Schneider, and Seshadhri [20]. In this paper, the authors show an algorithm that gives a relative approximation to the bias of a Bernoulli trial. One may reduce approximate triangle counting to this problem by picking at random three vertices and checking whether they form a triangle. See also the presentation of Watanabe [24].

[3] The main result of the paper is an algorithm and lower bound in a slightly different setting. This algorithm is only mentioned in the paper briefly.

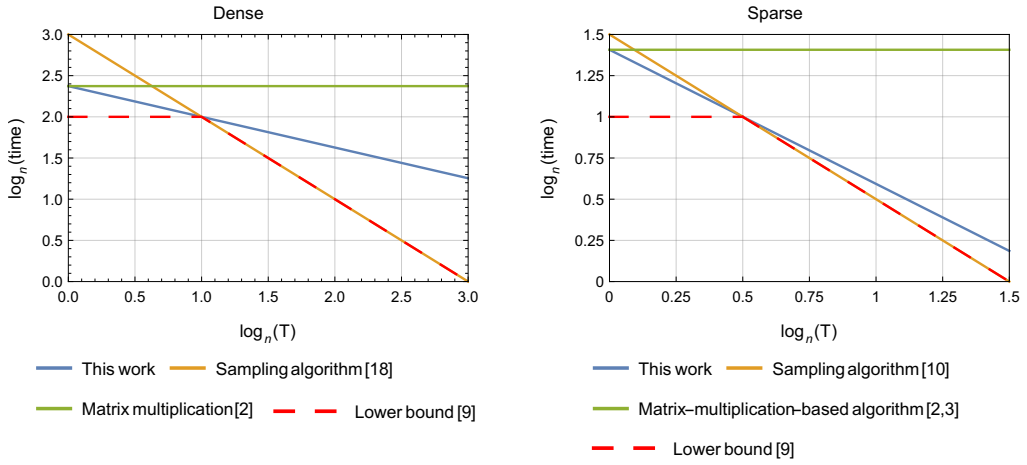

Figure 1 Comparison of the time complexities, assuming ϵ is constant. This plot assumes the best currently known bound $\omega < 2.3728596$ from [2]. For more detail, see Table 1.

We find it surprising that it is sufficient to use matrix multiplication to find the vertices contained in many triangles. That is, the matrix multiplication is only used to decide which triangles are going to be counted by which subroutine, while the counting itself is done without the help of matrix multiplication.

The setting. When an algorithm runs in sublinear time, it is of importance how the algorithm is allowed to access the graph. This is because the algorithm does not have the time to pre-process the whole graph. We assume the following standard setting: the algorithm may in constant time (1) get the i-th vertex, (2) get the j-th neighbor of a given vertex, (3) query the degree of a given vertex, and (4) given two vertices, tell whether they are adjacent or not (this is called the pair query). In the second part of this paper, we assume a setting where instead of the query (1), we can in constant time (1') get the i-th edge. This setting is also standard in the area of sublinear algorithms. As the main contribution of this paper is in the superlinear regime (as this is where we improve upon the state of the art), we do not describe the settings in more detail. See, for example, [13] for a more detailed description.

Time complexity vs. query complexity. In the area of sublinear algorithms, it is customary to focus on the *query complexity* (that is the number of queries, in the above sense, performed) of an algorithm, as opposed to the time complexity. The time complexity is then usually (near-)linear in the query complexity. However, this is the case only in the sublinear regime, and it breaks down when the time complexity is superlinear[4]. In this case, one may always read the whole graph in $O(n+m)$ queries. The time complexity is then usually significantly larger than the query complexity. This regime is our main focus.

Specifically, an algorithm is described in [9] that uses the edge access query (1') and has query complexity $\tilde{O}_\epsilon(\min(n+m, m^{3/2}/T))$. This is known to be near-optimal[5] [11]. In the sublinear regime (that is, for T large enough), the asymptotic time complexity is the same

[4] It may seem confusing talking about a sublinear-time algorithm having superlinear time complexity. The reason is that an algorithm is usually said to be sublinear time if its complexity is sublinear for some setting of the parameters (but not necessarily for all).

[5] Whenever we talk about optimality, we consider the case of ϵ being constant.

as the query complexity and it is, therefore, also near-optimal. However, in the superlinear regime, the time complexity is polynomially larger than the query complexity. Up until now, nothing was known about whether the time complexity can be improved in this case. We answer this question positively. In doing this, we give, to the best of our knowledge, the first algorithm answering a question of this type (that is, a question regarding the discrepancy between the time and query complexities in the superlinear regime).

1.1 Our results

The main contributions of this paper are the following two algorithms for approximate triangle counting. The first algorithm (Algorithm 4) is suitable for dense graphs and its time complexity is parameterized by n, T, and ϵ. The second algorithm (Algorithm 5) is suitable for sparse graphs and its time complexity is parameterized by m and T. Both of our algorithms have a state-of-the-art dependency on n, m while being negatively dependent on T. Our algorithms improve upon the existing algorithms when $1 \ll T \ll n$ and $1 \ll T \ll \sqrt{m}$, respectively.

We compare these results to the previous work in the following table. These time complexities are also plotted in Figure 1.

Table 1 Algorithms for approximate triangle counting. Algorithms with complexity independent of ϵ in fact solve exact triangle counting. (Note that algorithms with dependency on ϵ have the dependency hidden in O_ϵ in this table.)

	Dense	Sparse
This work	$\tilde{O}_\epsilon\left(\frac{n^\omega}{T^{\omega-2}}\right)$	$\tilde{O}_\epsilon\left(\frac{m^{2\omega/(\omega+1)}}{T^{2(\omega-1)/(\omega+1)}}\right)$
Sampling algorithm [20]	$O_\epsilon(\frac{n^3}{T})$	
Fast matrix multiplication	$\tilde{O}(n^\omega)$	
Alon, Yuster, and Zwick[3]		$\tilde{O}(m^{2\omega/(\omega+1)})$
Eden et. al. [9]	$\Omega\big(\min(n^2, \frac{n^3}{T})\big)$	$\tilde{O}_\epsilon(\frac{m^{3/2}}{T})$, $\Omega\big(\min(m, \frac{m^{3/2}}{T})\big)$

1.2 Related work

Algorithms. The number of triangles in a graph can be trivially counted in time $O(n^3)$. Itai and Rodeh [16] obtained an algorithm that runs in time $O(m^{3/2})$, a significant improvement for sparse graphs. A simpler algorithm with the same complexity, based on bounds on arboricity has been given by Chiba and Nishizeki [7]. A more efficient algorithm for *approximate* triangle counting has been obtained by Kolountzakis, Miller, Peng, and Tsourakakis [18]. In their paper, the authors presented an algorithm that runs in time $\tilde{O}_\epsilon(m + \frac{m^{3/2}}{T})$. This has been later improved by Eden et. al. [9] to $\tilde{O}_\epsilon(\frac{n}{T^{1/3}} + \frac{m^{3/2}}{T})$ when having vertex queries and not edge queries (having query (1) but not (1′)) and to $\tilde{O}_\epsilon(\frac{m^{3/2}}{T})$ when having edge queries (query (1′)).

While fast matrix multiplication gives a $\tilde{O}(n^\omega)$ algorithm for exact triangle counting, it is not immediately clear an improvement can be achieved in sparse graphs. In their paper, Alon, Yuster, and Zwick [3] show an algorithm for exact triangle counting that runs in time $\tilde{O}(m^{2\omega/(\omega+1)})$. This algorithm works by counting triangles whose all vertices have high degree using matrix multiplication while using a naïve algorithm for the rest of the graph. We use a variant of this approach for approximate triangle counting in sparse graphs in Section 3.

Sampling from graphs in order to estimate the number of triangles has been considered many times [23, 22, 17, 6]; for more details, see the introduction of [17]. The bounds then often depend on an upper bound on the number of triangles containing any single edge or similar parameters, for example in [22] or [23]. In this paper, we go further in the sense that we use a similar sampling approach (in fact, we use the same sampling scheme that was used in [6] but use a finer analysis) but explicitly ensure the bound on the number of triangles containing any single vertex. In other papers using similar sampling schemes, it is just assumed that this or a similar bound is already satisfied for the input instance.

Lower bounds. From the side of lower bounds, Eden et. al. [9] have proven a lower bound on the query complexity, thus also implying a lower bound on the time complexity. This lower bound is presented for the setting when random edge queries are not allowed, but the lower bound of $\Omega(\min(m, m^{3/2}/T))$ also holds in the setting when they are. This was made explicit by Eden and Rosenbaum [12] who presented a significantly simpler proof based on communication complexity.

▶ **Theorem 1** (Theorem 4.7 of [12]). *For any $n, m \in O(n^2), T \in O(m^{3/2})$, it holds that any multiplicative-approximation algorithm for the number of triangles in a graph must perform $\Omega\left(\min\left(m, \frac{m^{3/2}}{T}\right)\right)$ queries, where the allowed queries are degree queries, pair queries, random neighbor queries, random vertex queries, and random edge queries.*

We use this lower bound to prove the claim that the exponent of T in Theorem 12 is optimal, out of all algorithms running in time $\tilde{O}(n^\omega/T^c)$ for some constant c and similarly the exponent in Theorem 15 is optimal out of all algorithms running in time $\tilde{O}(m^{2\omega/(\omega+1)}/T^c)$.

By choosing $m = \Theta(n^2)$ in the above theorem, it follows that

▶ **Corollary 2.** *For any n and $T \in O(n^3)$, it holds that any multiplicative-approximation algorithm for the number of triangles in a graph must perform $\Omega\left(\min\left(n^2, \frac{n^3}{T}\right)\right)$ queries, where the allowed queries are degree queries, pair queries, neighbor queries, random vertex queries, and random edge queries.*

Fast matrix multiplication. There is a large literature on fast matrix multiplication. We only mention here the asymptotically most efficient matrix multiplication algorithm which is currently the algorithm by Alman and Williams [2] which runs in time $\tilde{O}(n^\omega)$ for some $\omega < 2.3728596$.

1.3 Technical overview

1.3.1 Triangle counting in dense graphs

Sampling and (lack of) concentration. Suppose we sample each vertex independently with some probability p. The expected number of triangles in the subgraph induced by the sampled vertices is then p^3T. If we were able to show concentration around the expected value, we could count the number of triangles in this subgraph and based on that, estimate the number of triangles in the original graph.

Unfortunately, the number of triangles is not concentrated; for example, in the case when there is one vertex that is contained in all triangles. We show that we get concentration if we limit the number of triangles containing any single vertex. We, therefore, have some threshold τ and call all vertices that are contained in more than τ triangles triangle-heavy. Assume that the graph does not have any triangle-heavy vertices. By choosing τ sufficiently

small and by choosing p sufficiently large, we may get a good estimate of the number of triangles in the original graph, based on the number of triangles in the subgraph induced by the sampled vertices.

In the final algorithm, we separately count triangles that contain at least one triangle-heavy vertex and triangles that do not. For this, we need to be able to find all triangle-heavy vertices in the graph and then estimate the number of triangles that contain at least one triangle-heavy vertex. We efficiently count triangles with no triangle-heavy vertex by reducing the problem to one on a smaller graph by the sampling procedure we just described.

Finding triangle-heavy vertices by matrix multiplication. We now sketch how to find the set of vertices that are contained in many triangles. We sample each vertex with some small probability p'. This gives a subset of vertices of size $\approx p'n$. We find all vertices that are contained in a triangle in the subgraph induced by the sampled vertices in time $O\big((p'n)^\omega\big)$ by using matrix multiplication. We repeat this experiment an appropriate number of times. We report vertices that were contained in at least one triangle in at least one repetition of the experiment. If a vertex is contained in few triangles, the probability that it is reported can be bounded by using union bound over all triangles it is contained in and over all iterations. The intuition for why vertices contained in many triangles are reported with good probability is the following. We bound the variance of the total number of triangles that the vertex will be in over all repetitions. The repetitions are independent, but there are correlations between triangles within one repetition. If we set the probability p' to be small enough, we can show that these correlations are small. This allows us to show concentration of the number of triangles containing any single vertex that are observed in the sampled subgraphs.

Counting triangles with triangle-heavy vertices. Suppose we are given a subset of vertices such that each of those vertices is contained in many triangles. We now sketch how to give a factor $3+\epsilon$ approximation to the number of triangles that contain at least one of these vertices. In the body of the paper, we then give a $(1+\epsilon)$-approximation by a more careful argument.

We estimate separately for each vertex the number of triangles it is contained in. The fact that each vertex individually is in many triangles allows us to employ a concentration argument separately for each vertex. We then use union bound over all vertices. Specifically, for each vertex v from the set of vertices that is given to us, we perform the following experiment many times: we sample two vertices and we check whether they form a triangle with v. We use the proportion of the experiments that ended up with a triangle to estimate the number of triangles v is contained in. We add together the estimates for all vertices in the set. This is where the factor 3 comes from – a triangle may be counted from all its 3 vertices.

Recursive algorithm. The last trick we introduce is that we use recursion to approximately count the triangles in the sampled subgraph. Doing this in such a way to give a correct algorithm with the desired complexity is the technically most challenging part of our paper. An obstacle is that we need the precision to be increasing in the depth of recursion. Specifically, in our algorithm, the allowed error decreases at an exponential rate in the depth of recursion. Moreover, we need to use probability amplification, meaning that the number of recursive calls also grows exponentially with the depth of recursion. This leads, in some situations, to the time complexity of the k-th level of recursion increasing exponentially in k; we bound this time.

1.3.2 Triangle counting in sparse graphs

We set a parameter θ and say a vertex is degree-heavy if its degree is at least θ and degree-light otherwise. We count separately *(a)* triangles on the subgraph induced by the degree-heavy vertices and *(b)* triangles that contain at least one degree-light vertex. This is basically the idea used by Alon, Yester, and Zwick [3] for their exact triangle-counting algorithm. We use our algorithm for approximate triangle counting in dense graphs to count the triangles from case *(a)* and sampling to count the triangles *(b)*. The sampling-based estimation is based on ideas from [7, 9]. Specifically, we assign each triangle to its vertex with the lowest degree (breaking ties arbitrarily). We then repeatedly perform the following experiment: pick an edge uniformly at random; consider its endpoint v with the lower degree; pick one of its neighbors at random; check whether it forms a triangle together with the edge we have sampled and whether the triangle is assigned to v. If so, return $d(v)$, otherwise return 0. We estimate the number of triangles based on the average of the returned values of the experiments.

1.4 Notation

Throughout the paper, we denote the number of vertices and edges of a graph by n and m, respectively. We use T to denote the number of triangles. We define $a \pm b = [a - b, a + b]$. This can be used for example in $(1 \pm \epsilon)a$ which is then equal to $[(1 - \epsilon)a, (1 + \epsilon)a]$. We call $(1 \pm \epsilon)$-approximation to a number a any number b such that $b \in (1 \pm \epsilon)a$. Similarly, we call *additive* $\pm c$ approximation to a number a any number b such that $b \in a \pm c$. We call *diamond* a graph isomorphic to  and *butterfly* a graph isomorphic to  . By "the number of diamonds in G" we mean the number of, not necessarily induced, subgraphs of G isomorphic to a diamond. We similarly talk about "the number of butterflies in a graph". We denote the subgraph of G induced by $V' \subseteq V$ by $G[V']$.

2 Triangle counting in dense graphs

In this section, we present the main result of our paper – an algorithm for approximate triangle counting, parameterized by n and the number of triangles T. In the lemmas that follow, we prove, among other things, bounds on the expectation of the estimators. We do this because our algorithm requires some "advice" and having a bound on the expectation will allow us to remove the need for this advice, as we describe in Section 2.5. We now define some notation that we will need.

We call a vertex *triangle-dense* for a parameter τ if there are at least τ triangles that contain the vertex and *triangle-light* if at most $\tau/20$ triangles contain it (there can thus be vertices that are neither triangle-light nor triangle-heavy). We define T_v to be the number of triangles containing the vertex v. We abuse notation and also denote by T_v the set of all triangles containing v. We use $T_v(G)$ instead when we want to explicitly specify the graph. T denotes the number of triangles in G but we again abuse notation and use this to also denote the set of triangles. Again, if we want to specify the graph, we use $T(G)$.

We now prove a simple lemma on fractional moments of the binomial distribution. We will use this to bound the time complexity of matrix multiplication executed on a random subgraph of some given graph. Specifically, we will use it on a subgraph obtained by keeping each vertex with some appropriately chosen probability and removing the other vertices. The lemma also holds for other exponents than ω, but fixing the exponent allows for a simpler proof.

▶ **Lemma 3.** *Assume $p \geq 1/n$. It holds that $E(Bin(n,p)^\omega) = O(n^\omega p^\omega)$.*

Proof. $Bin(n,p)$ has its third moment equal to

$$(n(n-1)(n-2)p^3) + 3(n(n-1)p^2) + np = O(n^3p^3)$$

where the equality holds because $p \geq 1/n$. Since $\omega < 3$, the function $x^{\omega/3}$ is concave. By the Jensen's inequality, we then get $E(Bin(n,p)^\omega) \leq \big(E(Bin(n,p)^3)\big)^{\omega/3} = O(n^\omega p^\omega)$. ◀

2.1 Triangle-light subgraph

We show a reduction from the problem of approximately counting triangles in a graph with no triangle-heavy vertices to the problem of approximately counting triangles in a smaller graph. Before that, we need to prove the following lemma.

▶ **Lemma 4.** *Let G be a graph with T triangles such that any vertex is contained in at most τ triangles and let D, B be the number of diamonds and butterflies in G, respectively. Then $D + B \leq \frac{3}{2}\tau T$.*

Proof. Let S be the number of pairs $(\triangle_1, \triangle_2)$ of triangles in G such that $\triangle_1$ and $\triangle_2$ share at least one vertex. It holds $D + B \leq S/2$ (the factor of 2 is there because the pairs are ordered). Consider a fixed triangle $\triangle_1 = uvw$. How many triangles are there that share a vertex with $\triangle_1$? Each of the vertices u, v, w can be contained in at most $\tau - 1$ other triangles. There can be, therefore, at most $3(\tau - 1)$ incidences between $\triangle_1$ and other triangles containing at least one of the vertices u, v, w. Therefore $D + B \leq S/2 \leq 3T(\tau - 1)/2$. ◀

We are now ready to prove the following lemma. Note that to use it (in order to set the probability p), one has to have a lower bound on the number of triangles. We resolve this issue later.

▶ **Lemma 5.** *Let $G' = (V', E')$ be the induced subgraph of G resulting from keeping each vertex with probability $p \geq \max(5\frac{1}{\sqrt[3]{\epsilon^2 T}}, 10\frac{1}{\epsilon\sqrt{T/\tau}})$ and removing it otherwise. Let $\hat{T} = |T(G')|/p^3$. Then $\hat{T}$ is an unbiased estimate of T. Moreover if any vertex $v \in G$ is contained in at most τ triangles, then $\hat{T} \in (1 \pm \epsilon)T$ with probability at least $19/20$.*

Proof. Let $\triangle$ be a triangle in G. Let $X_\triangle = 1$ if $\triangle \in T(G')$ and 0 otherwise. Let $T' = T(G')$. It holds that

$$E(\hat{T}) = E(T'/p^3) = \sum_{\triangle \in T} E(X_\triangle)/p^3 = T$$

and the estimate is, therefore, unbiased.

We now bound $Var(T')$. Recall that $T(G)$ is the set of triangles in G. Let $D(G)$ the set of diamonds in G, $B(G)$ the set of butterflies in G, and $\ddot{T}(G)$ the set of disjoint (unordered) pairs of triangles in G. Let $D = |D(G)|$, $B = |B(G)|$, and $\ddot{T} = |\ddot{T}(G)|$. By $\{\triangle_1, \triangle_2\} \in D(G)$ we mean that $\triangle_1 \cup \triangle_2 = A$ for some $A \in D(G)$ and analogously for $B(G)$ and $\ddot{T}(G)$.

We first bound the second moment of T'

$$\begin{aligned} E(T'^2) =& E\left(\Big(\sum_{\triangle \in T(G)} X_\triangle\Big)^2\right) \\ =& E\Big(\sum_{\triangle \in T(G)} X_\triangle^2\Big) + E\Big(\sum_{\substack{\triangle_1, \triangle_2 \in T(G) \\ \triangle_1 \neq \triangle_2}} X_{\triangle_1} X_{\triangle_2}\Big) \end{aligned}$$

$$
\begin{aligned}
&= E\Big(\sum_{\triangle \in T(G)} X_\triangle^2\Big) + E\Big(2 \sum_{\{\triangle_1,\triangle_2\}\in D(G)} X_{\triangle_1} X_{\triangle_2}\Big) \\
&\quad + E\Big(2 \sum_{\{\triangle_1,\triangle_2\}\in B(G)} X_{\triangle_1} X_{\triangle_2}\Big) + E\Big(2 \sum_{\{\triangle_1,\triangle_2\}\in \ddot{T}(G)} X_{\triangle_1} X_{\triangle_2}\Big) \\
&= p^3 T + 2p^4 D + 2p^5 B + 2p^6 \ddot{T} \\
&\leq p^3 T + 2(D+B)p^4 + p^6 T^2 \\
&\leq p^3 T + 3p^4 \tau T + p^6 T^2
\end{aligned}
$$

where we have used that $D + B \leq \frac{3}{2}\tau T$ (by Lemma 4) and $2\ddot{T} \leq T^2$. At the same time, $E(T') = p^3 T$. Therefore,

$$Var(T') = E(T'^2) - E(T')^2 \leq p^3 T + 3p^4 \tau T.$$

This means that $E(\hat{T}) = T$ and $Var(\hat{T}) = Var(\frac{1}{p^3}T') = \frac{1}{p^3}T + \frac{3}{p^2}\tau T$. Since $p \geq \max(5\frac{1}{\sqrt[3]{\epsilon^2 T}}, 10\frac{1}{\epsilon\sqrt{T/\tau}})$, it holds $Var(\hat{T}) \leq (1/5^3 + 3/10^2)\epsilon^2 T^2 < \frac{1}{20}\epsilon^2 T^2$. It, therefore holds by the Chebyshev inequality that $P(|\hat{T} - T| \geq \epsilon T) \leq 1/20$. ◀

2.2 Triangle-heavy subgraph

We now show an algorithm that approximately counts triangles that contain at least one vertex from set V_H where V_H is some given set that does not contain any triangle-light vertices.

Algorithm 1 Count $(1 \pm \epsilon)$-approximately triangles in G containing a vertex from V_H.

```
for v ∈ V_H do
    T̂_v ← 0
    repeat 360 n² log n / (ε²τ) times
        Sample u, w ∈ V
        Let ℓ = |{u, v, w} ∩ V_H|
        If uvw forms a triangle, increment T̂_v by ε²τ / (360 ℓ log n)
return Σ_{v∈V_H} T̂_v
```

▶ **Lemma 6.** *Given a set V_H, Algorithm 1 returns an unbiased estimate of the number of triangles containing at least one vertex from V_H.*

Assume V_H contains all triangle-heavy vertices of G and no triangle-light vertices. Then Algorithm 1 returns $(1 \pm \epsilon)$-approximation of the number of triangles that contain at least one triangle-heavy vertex with probability at least $1 - O(\frac{1}{n})$. It runs in time $O(\frac{Tn^2 \log n}{\epsilon^2 \tau^2})$.

Proof. We introduce charges on vertices. For any triangle $\triangle$ that contains at least one vertex that is in V_H, we divide and charge single unit to the vertices in $\triangle \cap V_H$, dividing it fairly (if there are, e.g., two vertices from V_H in the triangle, they are both charged 1/2). Let χ_v be the charge on vertex v. The total amount charged (that is, $\sum_{v\in V_H} \chi_v$) is equal to the number of triangles that contain at least one vertex from V_H. Note that this is the quantity we want to estimate.

The algorithm considers $v \in V_H$ and samples one pair of vertices u, w. Consider the amount charged by uvw to v; call it χ'_v. It holds that $E(\chi'_v) = \chi_v/n^2$. When uvw form a triangle, the algorithm increments $\hat{T}_v$ by $\frac{\epsilon^2\tau}{360\ell \log n} = \frac{\chi'_v \epsilon^2 \tau}{360 \log n}$. Therefore, the increment is, in expectation, equal to $\frac{\chi_v \epsilon^2 \tau}{360 n^2 \log n}$. Since there are $\frac{n^2 \log n}{360\epsilon^2\tau}$ repetitions, it holds

$$E(\hat{T}_v) = \frac{360n^2 \log n}{\epsilon^2\tau} \cdot \frac{\chi_v \epsilon^2 \tau}{360 n^2 \log n} = \chi_v.$$

By the linearity of expectation, we have $E(\sum_{v \in V_H} \hat{T}_v) = \sum_{v \in V_H} \chi_v$ which we said above is equal to the number of triangles with empty intersection with V_H.

We now prove concentration around mean. For each vertex, we have $\frac{360n^2 \log n}{\epsilon^2\tau}$ independent random variables corresponding to the increments in $\hat{T}$. These random variables have values between 0 and $\frac{\epsilon^2\tau}{360 \log n}$ and their sum has expectation $\chi_v \geq \frac{T_v}{3} \geq \frac{\tau}{60}$ where the second inequality holds from the assumption that V_H contains no triangle-light vertices. By the Chernoff bound,

$$P(|\hat{T}_v - T_v| \geq \epsilon T_v) \leq 2 \exp\Big(- \frac{\epsilon^2\tau/60}{3\epsilon^2\tau/(360 \log n)} \Big) \leq \frac{2}{n^2}.$$

By the union bound, it holds for all v that $\hat{T}_v$ is an $(1 \pm \epsilon)$-approximate of T_v, with probability at least $1 - \frac{2}{n}$. On this event, the algorithm correctly gives a $(1 \pm \epsilon)$-approximation of the number of triangles containing at least one triangle-heavy vertex. ◀

2.3 Finding the triangle-heavy subgraph

In this section, we show how to find a set of vertices that contains each triangle-heavy vertex with probability at least 2/3 and each triangle-light vertex with probability at most 1/3. This guarantee may be strengthened by probability amplification (applying the probability amplification separately to each vertex) to make sure that, with high probability, all triangle-heavy vertices are reported, and none of the triangle-light vertices are. This only adds a $O(\log n)$ factor to the time complexity of this subroutine.

We solve separately the case $\tau \leq n$ and $\tau \geq n$. On the range $\tau \leq n$, we get an algorithm running in time $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$. On the range $\tau \geq n$, we get time complexity $O(\frac{n^3}{\tau}) \subseteq \tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$ (the inclusion holds on this range of τ). This means that on this range, our bound is not tight. However, this is the case only on the range of T for which a near-optimal algorithm was already known, and we only show this for completeness; the reader may wish to skip the case of $\tau \geq n$. Putting the guarantees for the two ranges together, it follows that

▶ **Corollary 7.** *There is an algorithm that with probability at least $1 - O(\frac{1}{n})$ reports all triangle-heavy vertices and no triangle-light vertices while having time complexity $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$.*

2.3.1 The case $\tau \leq n$

▶ **Lemma 8.** *Assuming $\tau \leq n$, Algorithm 2 lists any triangle-heavy vertex with probability at least 2/3 and any triangle-light vertex with probability at most 1/3. It has expected time complexity $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$.*

---

[6] This may be done as follows. Raise the adjacency matrix of $G[V_i]$ to the third power. Each diagonal element in this matrix corresponds to a vertex (as rows and columns correspond to vertices and diagonal vertices have both the row and column corresponding to the same vertex). Consider the vertices that have a non-zero value on the corresponding diagonal position. These are exactly the vertices contained in some triangle by standard equivalence between taking powers of adjacency matrices and between counting walks in graphs.

Algorithm 2 Distinguish triangle-heavy vertices from triangle-light vertices.

$\mathcal{M} \leftarrow \emptyset$
for $i \in [k = 6\tau^2]$ **do**
  $V_i \leftarrow$ sample each vertex $v \in V$ independently with probability $p = \tau^{-1}$
  $V_i' \leftarrow$ find all vertices contained in a triangle in $G[V_i]$ in time $\tilde{O}(|V_i|^\omega)$ [6]
  $\mathcal{M} \leftarrow \mathcal{M} \cup V_i'$
return $\mathcal{M}$

Proof. Consider a triangle-light vertex v. That is, there are at most $\tau/20$ triangles in G containing v. The probability that a triangle is discovered in one iteration is, by the union bound, at most $\tau p^3/20$. Taking a union bound over the k iterations, the probability that a triangle containing v is found (and thus v reported) is at most $k\tau p^3/20 < 1/3$.

Consider a triangle-heavy vertex v. Let X_i for $i \in [k]$ be the number of triangles containing v discovered in the i-th iteration. For $\triangle \in T$, let $X_{\triangle,i}$ be an indicator that triangle $\triangle$ is discovered in i-th iteration. It now holds $X_i = \sum_{\triangle \in T, v \in \triangle} X_{\triangle,i}$. Let $X = \sum_{i=1}^{k} X_i$.

By the linearity of expectation, it holds that

$$E(X) = E\Big(\sum_{i=1}^{k} X_i\Big) = kp^3 T_v = 6T_v/\tau.$$

We now bound the variance of X. We first bound

$$Var(X_i) \leq E(X_i^2) = E\Bigg(\sum_{\triangle_1, \triangle_2 \in T_v} X_{\triangle_1,i} X_{\triangle_2,i}\Bigg) \leq p^3 T_v + p^4 T_v^2$$

where the last inequality holds because there are $\leq T_v^2$ terms (i.e. pairs of triangles containing v) that are 1 with probability $\leq p^4$ (such pairs have at least 4 vertices) and there are T_v terms (i.e. pairs of triangles containing v) that are 1 with probability p^3 (these are pairs that have $\triangle_1 = \triangle_2$ and the pair thus has 3 vertices). Since the iterations are independent, it holds that

$$Var(X) = Var\Big(\sum_{i=1}^{k} X_i\Big) = kVar(X_i) \leq 6\tau^2(p^3 T_v + p^4 T_v^2) = \frac{6}{\tau} T_v + \frac{6}{\tau^2} T_v^2 \leq \frac{12 T_v^2}{\tau^2}$$

where the last inequality holds because $T_v \geq \tau$. It, therefore, holds by the Chebyshev inequality that

$$P(X = 0) \leq \frac{Var(X)}{E(X)^2} \leq \frac{12T_v^2/\tau^2}{(6T_v/\tau)^2} = 1/3.$$

We now argue the time complexity. It holds that $|V_i| \sim Bin(n, p)$. By Lemma 3, each iteration has expected time complexity $\tilde{O}(n^\omega p^\omega)$ because of the assumption $p = 1/\tau \geq 1/n$. There are $O(p^{-2})$ iterations. The total time complexity is thus $\tilde{O}(p^{-2} p^\omega n^\omega) = \tilde{O}\Big(\frac{n^\omega}{\tau^{\omega-2}}\Big)$. ◀

2.3.2 The case $\tau > n$

We repeat that this case only applies to the range where an optimal algorithm is already known (see the beginning of Section 2.3) and we only show this for completeness; the reader may wish to skip this part.

Algorithm 3 Distinguish triangle-heavy vertices from triangle-light vertices.

```
1 M ← ∅
2 for v ∈ V do
3     repeat k = 2n²/τ times
4         Sample u, w ∈ V
5         if uvw ∈ T then
6             M ← M ∪ {v}
7 return M
```

▶ **Lemma 9.** *Algorithm 3 lists any triangle-heavy vertex with probability at least* $2/3$ *and any triangle-light vertex with probability at most* $1/3$*. It runs in time* $\tilde{O}(\frac{n^3}{\tau})$*.*

Proof. Consider the probability that a triangle-light vertex v is reported. Similarly to the case $\tau \leq n$, it holds by the union bound over all incident triangles and over the k iterations that this probability is at most $\tau/20 \cdot k \cdot 1/n^2 \leq 1/3$.

Consider now the probability a triangle-heavy vertex v is reported. In each iteration, the probability that we find an incident triangle is $T_v/n^2 \geq \tau/n^2$. The probability that we report v is now at least $1 - (1 - \tau/n^2)^{2n^2/\tau} \geq 1 - 1/e^2 \geq 2/3$. ◀

2.4 Recursive algorithm

We now take the subroutines presented above and put them together into one recursive algorithm. Our proof of correctness works by induction on the depth of recursion. For this reason, the statement assumes the input graph can be random, as we use the inductive hypothesis on a sampled subgraph of the input graph. This algorithm requires advice $\tilde{T}$ such that $\tilde{T} \leq E(T)$ (note that T is a random variable as the input graph may be random); we will remove the need for advice later. As we already mentioned, for the advice removal, we will need to prove a bound on the expectation of the estimate.

We show an algorithm that gives additive approximation. The reason is that this is better suited for performing recursive calls. Specifically, the time complexity of getting relative approximation is worse when T is small. If it happened that most triangles contained a triangle-heavy vertex, we would recurse on a subgraph with few triangles, making it more costly to get the relative approximation. We then show how this algorithm can be turned into an algorithm that gives relative approximation.

Note that line 11 can be efficiently implemented as follows: We pick $X \sim Bin(n, p)$, then sample X vertices without replacement, keep only the vertices from $V \setminus V_H$.

▶ **Lemma 10.** *Suppose* G *is a (possibly random) graph. Given parameters* A*,* $\tilde{T}$*, Algorithm 4 returns* $\hat{T}$ *such that* $E(\hat{T}) \leq E(T)$ [7]*. Moreover, if* $\tilde{T} \geq E(T)$*, Algorithm 4 returns* $\hat{T} \in T \pm A$ *with probability at least* $4/5$*. It runs in expected time* $\tilde{O}\big(\frac{n^\omega}{(A^2/\tilde{T})^{\omega-2}} + \frac{\tilde{T}^{4+1/3}Tn^2}{A^6}\big)$*.*

Proof. The structure of the proof is as follows. We first prove that $E(\hat{T}) \leq E(T)$. We prove this by induction. We then go on to prove correctness (that is, that the returned answer is $(1 \pm \epsilon)$-approximation with probability $4/5$), also by induction. In both cases, the induction is on the depth of recursion. Note that, while the instances on which the algorithm

[7] Since the graph can be random, T is a random variable

Algorithm 4 Estimate the number of triangles in G, advice $\tilde{T}$, and error parameter A.

1 $q \leftarrow 1/\log(nT/A)$
2 **if** $A^2 \leq \frac{10^7}{q}\tilde{T}$ **then**
3 **return** *number of triangles in G, computed using matrix multiplication*
4 **if** $\tilde{T} < 1/5$ **then**
5 **return** 0
6 $q \leftarrow 1/\log\left(\frac{nT}{A}\right)$
7 $\tilde{T}' = 20\tilde{T}$
8 $\tau \leftarrow \frac{A^2q^2}{40000\tilde{T}'}$
9 $V_H \leftarrow$ triangle-heavy vertices w.r.t. τ using Corollary 7
10 $\hat{T}_H \leftarrow$ estimate number of triangles with non-empty intersection with V_H using Algorithm 1 with $\epsilon = qA/(2\tilde{T}')$
11 $V' \leftarrow$ sample each vertex from $V \setminus V_H$ independently with probability $p = \frac{20\sqrt{\tau\tilde{T}'}}{qA} = 1/10$
12 $\hat{T}_{V'} \leftarrow$ estimate number of triangles in $G[V']$ by a recursive call with $A = (1-q)p^3A$ and with $\tilde{T} = \tilde{T}p^3$; amplify success probability to 19/20 by taking median of 7 independent executions
13 **return** $\hat{T}_H + \hat{T}_{V'}/p^3$

is called in the recursive calls are random, the recursion tree is deterministic. This means that performing induction on the depth of recursion is formally correct. Finally, we prove the time complexity; in this part, we analyze the whole recursion tree instead of using induction.

Estimate's expected value. We now prove $E(\hat{T}|G) \leq E(T|G)$. We prove this by induction on the depth of recursion. If the condition on line 2 is satisfied, then $\hat{T} = T$ and the inequality thus holds. If the condition on line 4 is satisfied, then $\hat{T} = 0$ and the inequality also holds. Consider now the case when neither of these conditions are satisfied. By the inductive hypothesis, $E(\hat{T}_{V'}|V', G) \leq E(T_{V'}|V', G)$. Moreover, by Lemma 5, $E(T_{V'}/p^3|V_H, G) = E(T(G[V \setminus V_H])|V_H, G)$ and therefore

$$\begin{aligned} E(\hat{T}_{V'}/p^3|V_H, G) &= E(E(\hat{T}_{V'}/p^3|V', G)|V_H, G) \\ &\leq E(E(T_{V'}/p^3|V', G)|V_H, G) \\ &= E(T_{V'}/p^3|V_H, G) = E(T(G[V \setminus V_H])|V_H, G). \end{aligned}$$

It follows from Lemma 6 that $E(\hat{T}_H|V_H, G)$ is an unbiased estimate of the number of triangles in G containing at least one vertex from V_H. This implies that $E(\hat{T}_H|G) = T - T(G[V \setminus V_H])$. We now put this all together:

$$\begin{aligned} E(\hat{T}) &= E\big(E(\hat{T}_H|V_H, G) + E(\hat{T}_{V'}/p^3|V_H, G)\big) \\ &\leq E\big(E(\hat{T}_H|V_H, G)\big) + E\big(E(T(G[V \setminus V_H])|V_H, G)\big) \\ &= E(T - T(G[V \setminus V_H])) + E(T(G[V \setminus V_H])) = E(T). \end{aligned}$$

Note that this bound on the expectation is not using in any way that V_H contains all triangle-heavy vertices and no triangle-light vertices.

Correctness. We first focus on the base case. Consider the case $A^2 \leq \tilde{T}$. In this case, the condition on line 2 is satisfied. The algorithm is then clearly correct.

We now focus on the case when the condition on line 4 is satisfied. Consider one call of the algorithm in the tree of recursion where this is the case. Let k be the depth of this call in the tree. We denote by $G, T, n, A, \tilde{T}, \epsilon$ the respective values in this specific call. We use the subscript $_0$ to denote the respective values in the original call. For example, T_0 would be the number of triangles in the whole graph on which the algorithm was executed.

It holds that $E(T) \leq 1/10^{3k} T_0 \leq 1/10^{3k} \tilde{T}_0 = \tilde{T}$. It holds by the Markov's inequality that $P(5\tilde{T} \geq T) \geq 4/5$. When $\tilde{T} < 1/5$ (this is the condition on line 4), this means that $P(T = 0) \geq 4/5$, meaning that the algorithm is correct in this case and runs in time $O(1)$.

We now prove the inductive step. Consider one recursive call, regardless of the level of recursion. Again, we use $G, T, n, A, \tilde{T}, \epsilon$ to denote the respective values. We assume that $\tilde{T}' \geq T$ in this call. This holds with probability at least 19/20 by the Markov inequality as $\tilde{T}' = 20\tilde{T} \geq 20E(T)$. We now bound the error probability by 3/20. Together with the fact that $P(\tilde{T}' \geq T) \geq 19/20$, this gives a bound on the probability of the call returning an invalid answer of 4/20. Let $\epsilon = \frac{qA}{2\tilde{T}'}$. It then holds $\epsilon \leq \frac{qA}{2T}$. We have $p = 1/10$. We show that $p \geq \max(5\frac{1}{\sqrt[3]{\epsilon^2 T}}, 10\frac{1}{\epsilon\sqrt{T/\tau}})$ (this is the assumption of Lemma 5). It holds $\frac{1}{10} \geq 10\frac{1}{\epsilon\sqrt{T/\tau}}$ from the way we set τ. Because $A^2 \geq \frac{10^7}{q}\tilde{T}$, it also holds

$$5\frac{1}{\sqrt[3]{\epsilon^2 T}} \leq \frac{5}{\sqrt[3]{qA^2/(4\tilde{T}')}} \leq \frac{5}{\sqrt[3]{q\frac{10^7}{q}\tilde{T}/(4\tilde{T}')}} = \frac{1}{10}.$$

By Lemma 5, it holds with probability at least 19/20 that

$$T(G[V']) / p^3 \in (1 \pm \epsilon) T(G[V \setminus V_H]) \subseteq T(G[V \setminus V_H]) \pm qA/2.$$

By the induction hypothesis, it holds that with probability at least 19/20, $\hat{T}_{V'} = T(G[V']) \pm (1-q)p^3 A$. Note that $E(T_{k+1}) = E(E(T_{V'}|G)) \leq E(T/10^3) \leq \tilde{T}/10^3 = \tilde{T}_{k+1}$, where the subscript $_{k+1}$ refers to the respective values at a call on recursion depth $k+1$. This means that the assumption on $\tilde{T}$ is satisfied. Also note that to get success probability 19/20, it does suffice to take the median of 7 independent executions (as can be easily checked, $P(Bin(7, 1/5) \geq g) < 1/20$). Putting this together, with probability at least 18/20, it holds that

$$\hat{T}_{V'}/p^3 \in T(G[V'])/p^3 \pm (1-q)A \subseteq T(G[V \setminus V_H]) \pm (1 - q/2)A.$$

Moreover, with probability at least 19/20, $\hat{T}_H = T(V_H) \pm \epsilon T(V_H) \subseteq T(V_H) \pm qA/2$ by Lemma 6 where the inclusion holds because we have set $\epsilon = qA/(2\tilde{T}') \leq qA/(2T) \leq qA/(2T(V_H))$. By the union bound, with probability at least 17/20, $|\hat{T}_H - T(V_H)| \leq qA/2$ and $|T(V')/p^3 - T(V \setminus V_H)| \leq (1 - q/2)A$, in which case $\hat{T}_H + \hat{T}_L/p^3 = T \pm A$ – the resulting answer is correct. Including the probability that $\tilde{T}' \leq T$ (we assumed in the analysis that this is not the case), we get that the answer is correct with probability at least 16/20.

Time complexity. We again consider one recursive call and use $G, T, n, A, \tilde{T}, \epsilon$ to denote the respective values in this one call, and the subscript $_0$ is used to refer to the respective values in the original call. We first show that we may ignore in the analysis the time spent on line 3. Counting triangles exactly by using fast matrix multiplication takes $\tilde{O}(n^\omega) \subseteq \tilde{O}\big(\frac{n^\omega}{\tau^{\omega-2}} + \frac{\tilde{T}^4 T n^2}{A^6}\big)$ time (the inclusion holds thanks to the assumption $A^2 \leq \tilde{T}$ which is satisfied on line 3). This is equal to the bound on the time complexity of the call that we use below. In the rest of the proof, we therefore ignore the time spent on line 3.

The time complexity of line 9 is $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$ and complexity of line 10 is $\tilde{O}(\frac{Tn^2}{\epsilon^2\tau^2}) = \tilde{O}(\frac{\tilde{T}'^4Tn^2}{q^2A^6})$ by Corollary 7 and Lemma 6, respectively (note that ϵ is defined on line 10). Since $p = 1/10$, the number of vertices in a depth k recursive call is stochastically dominated by $Bin(n, 1/10^k)$ [8] while the value of A is in a depth k recursive call multiplied by factor of $\big((1-q)p^3\big)^k = \big((1-q)/10^3\big)^k$ and $\tilde{T}' = 1/10^{3k}\tilde{T}'_0$. In each recursive call, we make 7 recursive calls for the probability amplification. The number of calls at recursion depth k is then 7^k. The expected time spent in the depth k recursive calls on line 9 is

$$E\Big(7^k\frac{n^\omega\log n}{\tau^{\omega-2}}\Big) \le \tilde{O}\Big(7^k\frac{n_0^\omega/10^{\omega k}}{(q^2\epsilon_0^2(1-q)^k * \tilde{T}'_0/10^{3k})^{\omega-2}}\Big) = \tilde{O}\Big(q^{4-2\omega}\Big(\frac{7\cdot 10^{3(\omega-2)}}{(1-q)^{\omega-2}10^\omega}\Big)^k\cdot\frac{n_0^\omega}{\epsilon_0^2T_0^{\omega-2}}\Big).$$

This decreases exponentially since $\omega \le 2.5$ [9]and, therefore, the time on line 9 is dominated by the first call. (By changing constants in the algorithm, this argument can be made to work even for asymptotically slower sub-cubic matrix multiplication algorithms.) However, the time spent on line 10 is

$$E\Big(7^k\frac{\tilde{T}'^4Tn^2}{q^2A^6}\Big) = 7^k\frac{\tilde{T}'^4E(T)n^2}{q^2A^6} \le 7^k\frac{\tilde{T}'^4_0/10^{4\cdot3k}\cdot T_0/10^{3k}\cdot n^2/10^{2k}}{q^2A_0^6(1-q)^{6k}/10^{6\cdot3k}} = \Big(\frac{10}{1-q}\Big)^k\cdot\frac{\tilde{T}'^4_0T_0n_0^2}{q^2A_0^6}$$

which increases at an exponential rate. We now upper bound the number of recursive calls (in other words, we upper bound k). In each subsequent recursive call, $\tilde{T}$ decreases by a factor of $1/p^3 = 10^3$. This means that after $\log_{1000}\tilde{T} + O(1)$ recursive calls, it will hold that $\tilde{T} \le 1/20$, in which case the algorithm finishes on line 4 in time $O(1)$ with no additional recursive calls. Therefore $k \le \log_{1000}\tilde{T} + O(1)$. Since the amount of work at each level of recursion increases exponentially, the work is dominated by the last level of recursion. Thanks to our bound on k, the amount of time spent on line 8 is

$$\begin{aligned}O\Big(\Big(\frac{10}{1-q}\Big)^{\log_{1000}\tilde{T}_0+O(1)}\cdot\frac{\tilde{T}_0^4T_0n^2}{q^2A_0^6}\Big) &= O\Big((1-q)^{-\log_{1000}\tilde{T}_0}\frac{\tilde{T}_0^{4+1/3}T_0n^2}{q^2A_0^6}\Big)\\ &= O\Big(\frac{\tilde{T}_0^{4+1/3+\log_{1000}(1/(1-q))}T_0n^2}{q^2A_0^6}\Big)\end{aligned}$$

and substituting $q = 1/\log(nT/A)$, we get that the total time complexity is

$$\tilde{O}\left(\frac{q^{4-2\omega}n^\omega}{\epsilon^2T^{\omega-2}} + \frac{\tilde{T}^{4+1/3+\log_{1000}(1/(1-q))}Tn^2}{q^2A^6}\right) = \tilde{O}\left(\frac{n^\omega}{\epsilon^2T^{\omega-2}} + \frac{\tilde{T}^{4+1/3}Tn^2}{A^6}\right). \quad ◀$$

We now get the following claim, which guarantees relative approximation, unlike Lemma 10.

▶ **Proposition 11.** *There is an algorithm that, given $\epsilon > 0$ and $\tilde{T}$ returns an estimate $\hat{T}$ of T, such that $E(\hat{T}) \le E(T)$ of T and if, moreover, $T \le \tilde{T} \le 2T$, it holds $\hat{T} \in (1\pm\epsilon)T$ with probability at least $4/5$. It runs in expected time $\tilde{O}(\frac{n^\omega}{(\epsilon^2T)^{\omega-2}} + \frac{n^2}{\epsilon^6T^{2/3}})$.*

Proof. By substituting $A = \epsilon\tilde{T}/2$, it is sufficient to calculate approximation with additive error A. This can be done by Lemma 10, giving us the desired bounds. ◀

[8] This is the case as in each level of recursion, we keep each triangle-light vertex with probability $p = 1/10$ while removing all other vertices with high probability.

[9] The argument works for $\omega > 2.5$ if some constants in the algorithms are modified.

2.5 Removing advice

We remove the dependency on $\tilde{T}$ by performing a geometric search. This method has been used many times before, for example in [9, 14, 1, 10, 4, 11]. In our case, it is slightly more complicated in that our algorithm requires both a lower and an upper bound on T. For this reason, we describe the method in completeness. We also prove a variant that gives a guarantee of absolute approximation. We will need this for triangle counting in sparse graphs.

▶ **Theorem 12.** *There is an algorithm that, given $\epsilon > 0$ (respectively A) returns $\hat{T} \in (1 \pm \epsilon)T$ (respectively $\hat{T} \in T \pm A$) with probability at least $2/3$. It runs in expected time $\tilde{O}(\frac{n^\omega}{(\epsilon^2 T)^{\omega-2}} + \frac{n^2}{\epsilon^6 T^{2/3}})$ (respectively $\tilde{O}(\frac{n^\omega}{(A^2/\tilde{T})^{\omega-2}} + \frac{T^{5+1/3}n^2}{A^6})$).*

Proof. We present the argument for the relative approximation. The exact same argument gives the absolute approximation by using Lemma 10 in place of Proposition 11.

We start with $\tilde{T} = \binom{n}{3}$ and, in each step, divide $\tilde{T}$ by a factor of two. When it holds that $\hat{T} \geq \tilde{T}$ where $\hat{T}$ is the estimate returned by the algorithm, we return $\hat{T}$ as the final estimate of T.

We now argue correctness. The estimate $\hat{T}$ given by the algorithm is always non-negative and it holds $E(\hat{T}) \leq T$. By Markov's inequality, $P(\hat{T} \geq 2T) \leq 1/2$, regardless of the choice of $\tilde{T}$. We amplify this probability to $\Theta(1/\log n)$ by taking the median of $\Theta(\log\log n)$ independent executions. When $\hat{T} \geq \tilde{T}$ (this is when we return $\hat{T}$ as the final estimate), it holds with probability at least $1 - O(1/\log n)$ that $\tilde{T} \leq 2T$ as otherwise, it would be the case that $\hat{T} \geq \tilde{T} \geq 2T$ which holds with probability $O(1/\log n)$. Conditioned on $\tilde{T} \leq 2T$, the algorithm gives a $(1 \pm \epsilon)$-approximate estimate (respectively additive $\pm A$ estimate) by Proposition 11 (respectively Lemma 10). By the union bound, the failure probability is then arbitrarily small if we make the constants in Θ sufficiently small.

We now argue the time complexity. The probability amplification only increases the time complexity by a $O(\log\log n)$ factor. When $\tilde{T} \leq T$, with probability $1 - O(1/\log n)$ we return the estimate and quit. We now consider the calls of the algorithm with $\tilde{T} \leq T$. The time complexity increases exponentially (as T is decreasing exponentially) while the probability of performing a call is decreasing at a rate faster than exponential. Therefore, the time complexity is dominated by the first call when $\tilde{T} \leq T$. This time complexity is as claimed by Proposition 11 (respectively Lemma 10). ◀

2.6 Optimality

The lower bound of Eden et. al. [9] implies that our algorithm is in certain sense optimal. Specifically, the exponent in the dependency on T cannot be improved without worsening the dependency on n, as long as the exponent of T is constant. That still leaves open the possibility of improving the dependency on n and getting an algorithm with a non-constant exponent of T.

▶ **Proposition 13.** *Suppose there is an algorithm which runs in time $\tilde{O}\left(\frac{n^\omega}{T^\delta}\right)$ for some constant δ, and returns a constant-factor approximation of T with probability at least $2/3$. Then $\delta \leq \omega - 2$.*

Proof. Suppose $\delta > \omega - 2$. Then, for $T = \Theta(n)$, the algorithm runs in time $o(n)$. This is in contradiction to the lower bound of from [9] which states that any such algorithm has to run in time $\Omega(n)$. ◀

3 Triangle counting in sparse graphs

We now show how Algorithm 4 can be used to get an efficient algorithm for counting triangles in a sparse graph. The algorithm finds all vertices with degree at least some parameter θ, then it uses Algorithm 4 to count triangles in the subgraph induced by these vertices and uses sampling in the rest of the graph. We set $\theta = \frac{m^{(\omega-1)/(\omega+1)}}{T^{(\omega-3)/(\omega+1)}\epsilon^{(2\omega-6)/(\omega+1)}}$.

For sake of simplicity, we show an algorithm with an additional $O(m)$ term in its complexity. As the time complexity of the algorithm from [4] is lower in the sublinear regime than what we claim, we may obtain an algorithm with the desired complexity by running the two algorithms in parallel and terminating when one of them finishes. In this section, we use a total order $\prec$ defined as $u \prec v$ if $d(u) \leq d(v)$ and ties broken arbitrarily in a consistent way.

Algorithm 5 Estimate the number of triangles in G and advice $\tilde{T}$.

1 $\theta \leftarrow \frac{m^{(\omega-1)/(\omega+1)}}{\tilde{T}^{(\omega-3)/(\omega+1)}\epsilon^{(2\omega-6)/(\omega+1)}}$
2 $V_H \leftarrow$ find all vertices v with $d(v) \geq \theta$
3 $\hat{T}_H \leftarrow$ count triangles in $G[V_H]$ using Algorithm 4 with error $\pm\epsilon\tilde{T}/2$; amplify success probability to $\frac{5}{6}$
4 $M \leftarrow 0$
5 **repeat** $k = 12\frac{\theta m}{\epsilon^2\tilde{T}}$ **times**
6 $\quad e \leftarrow$ pick an edge uniformly at random
7 $\quad uv \leftarrow e$, such that $u \succ v$
8 $\quad w \leftarrow$ pick a random neighbor of v
9 $\quad$ **if** $v \notin V_H$ *and* $w \succ v$ *and* $uvw \in T$ **then**
10 $\quad\quad M \leftarrow M + d(v)$
11 $\hat{T}_L = \frac{m}{2k}M$
12 **return** $\hat{T}_H + \hat{T}_L$

We now prove the main theorem for triangle counting in sparse graphs. The last part in the following theorem is there to enable us to remove advice like we did in Section 2.5.

▶ **Lemma 14.** *Let us have a graph G. Given parameters $1 > \epsilon > 0$ and advice $\tilde{T}$, Algorithm 5 returns $\hat{T}$ such that $P(\hat{T} \geq 2T) \leq 2/3$. Moreover, if $\tilde{T} \leq T$, Algorithm 5 returns $\hat{T} \in (1 \pm \epsilon)T$ with probability at least $4/5$. It runs in expected time*

$$\tilde{O}\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}} + \frac{m^{4/(\omega+1)}}{\epsilon^{2(9+\omega)/(\omega+1)}T^{4(5-\omega)/(3(\omega+1))}} + m\Big).$$

With the current best bound on ω, we get running time of $\tilde{O}(\frac{m^{1.408}}{\epsilon^{1.628}T^{0.814}} + \frac{m^{1.186}}{\epsilon^{6.744}T^{1.038}})$. For constant ϵ, the second term is significantly smaller than the first one.

Proof. Let T_L be the set of triangles that are not contained in $G[V_H]$ – or equivalently, that have their $\prec$-minimal vertex outside of V_H – and, by an abuse of notation, also the number of such triangles. Let T_H be the number of triangles contained in $G[V_H]$.

Let X_i be the increment in M in the i-th iteration of the loop on line line 5. We now calculate the expectation of M after all k iterations. Let us fix a triangle $\triangle \in T$ and define $X_i' = X_i$ if $uvw = \triangle$ in the i-th iteration and $X_i' = 0$ otherwise (note that u, v, w are defined in the algorithm). If a is the vertex of $\triangle$ minimal w.r.t. $\prec$, it now holds that

$E(X_i') = d(a)P(X_i' = d(a))$. It holds that $X_i' = d(a)$ if and only if $v = a$ and $uvw = \triangle$. For this to happen, the algorithm has to sample in the i-th iteration the edge uv or wv. This happens with probability $2/m$. Then, it has to be the case that the random neighbor sampled on line 7 is the single vertex of $\triangle$ not in e. This happens with probability $1/d(a)$. This gives us that $E(X_i') = \frac{d(a)}{md(a)} = 2/m$. By summing over all triangles in T_L and by linearity of expectation, it holds that $E(X_i) = 2T_L/m$. Therefore, the estimate $\hat{T}_L = \frac{m}{2k}M = \frac{m}{2k}\sum_{i=1}^{k} X_i$ is an unbiased estimate of T_L.

We now bound the variance. We use the inequality[10] $Var(X) \leq \sup(X)E(X)$ to get $Var(X_i) \leq \theta E(X_i) = 2\theta T_L/m$. Therefore $Var(mX_i/2) = \frac{1}{2}\theta m T_L$. Taking average of $12\frac{\theta m}{\epsilon^2 \tilde{T}}$ independent trials, we get variance $\frac{1}{24}\epsilon^2 \tilde{T} T_L \leq \frac{1}{24}\epsilon^2 T^2$. By the Chebyshev's inequality, it therefore holds that

$$P(|\hat{T}_L - T_L| \geq \epsilon T/2) \leq \frac{Var(\hat{T}_L)}{(\epsilon T/2)^2} \leq 1/6.$$

At the same time, by Lemma 10, it holds that $\hat{T}_H \in T_H \pm \epsilon\tilde{T}/2 \subseteq T_H \pm \epsilon T/2$ with probability at least $5/6$ (note that we have amplified the success probability). By the union bound, with probability at least $2/3$, it holds that both $|\hat{T}_L - T_L| \leq \epsilon T/2$ and $|\hat{T}_H - T_H| \leq \epsilon T/2$. Therefore, the algorithm returns a $(1 \pm \epsilon)$-approximate answer with probability at least $2/3$.

We now argue the time complexity. We spend $O(n)$ time on line 2. There are no more than m/θ vertices with degree at least θ. Triangles in $G[V_H]$ are therefore counted, by Theorem 12, in time

$$\tilde{O}\Big(\frac{(m/\theta)^\omega}{(A^2/\tilde{T})^{\omega-2}} + \frac{T_H^{5+1/3}(m/\theta)^2}{A^6}\Big) \leq \tilde{O}\Big(\frac{(m/\theta)^\omega}{(\epsilon^2\tilde{T})^{\omega-2}} + \frac{(m/\theta)^2}{\epsilon^6 T^{2/3}}\Big)$$
$$= \tilde{O}\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}} + \frac{m^{4/(\omega+1)}}{\epsilon^{2(\omega+1)/(\omega+1)}T^{(7-\omega)/(\omega+1)}}\Big).$$

The time spent in each iteration of the loop on line 5 is $O(1)$. The total time spent in the loop is therefore

$$O\Big(\frac{\theta m}{A^2/\tilde{T}}\Big) = O\Big(\frac{\theta m}{\epsilon^2\tilde{T}}\Big) = O\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}}\Big).$$

Putting these two time complexities together, we get the desired running time.

We now prove that $P(\hat{T} \geq 2T) \leq 2/3$ [11]. Since $\epsilon < 1$, it holds that $P(\hat{T}_H \geq 2T_H) \leq P(\hat{T}_H \geq (1+\epsilon)T_H) \leq 1/6$ as we have amplified the success probability of Algorithm 4 to $\frac{5}{6}$. Moreover, as we have shown, $E(\hat{T}_L) = T_L$ and, therefore, $P(\hat{T}_L \geq 2T_L) \leq 1/2$ ◀

By the same argument as in Section 2.5, we may remove the need for advice [12]. We run the algorithm in parallel with the algorithm from [4], resulting in an algorithm with its time complexity being the minimum of the complexities of the two respective algorithms. This gives us the following claim.

---

[10] $\sup(X)$ is the smallest x such that $P(X > x) = 0$.

[11] We cannot use the Markov's inequality in a straightforward way. The reason is that it is not clear how to bound the expectation of the estimate coming from Theorem 12. While we do have a bound on the expectation of the estimate given by, Algorithm 4 it is not clear a bound of this type carries over when we perform the advice removal.

[12] Instead of directly using the Markov inequality in the argument, we may use the last part of Lemma 14 to get the same guarantee.

▶ **Theorem 15.** *There is an algorithm that returns a* $(1 \pm \epsilon)$*-approximation of the number of triangles with probability at least* $2/3$*. It runs in time*

$$\tilde{O}\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}} + \frac{m^{4/(\omega+1)}}{\epsilon^{2(9+\omega)/(\omega+1)}T^{4(5-\omega)/(3(\omega+1))}}\Big).$$

3.1 Optimality

Like in the case of dense graphs, we show that our algorithm is in certain sense optimal.

▶ **Proposition 16.** *Suppose there is an algorithm that uses both random vertex and random edge queries, which runs in time* $\tilde{O}(\frac{m^{2\omega/(\omega+1)}}{T^\delta})$ *for some constant* δ*, and returns a constant-factor approximation of* T *with probability at least* $2/3$*. Then* $\delta \leq 2(\omega-1)/(\omega+1)$*.*

Proof. Suppose $\delta > 2(\omega-1)/(\omega+1)$. Then, for $T = \Theta(\sqrt{m})$, the algorithm runs in time $o(m)$. This is in contradiction to the lower bound from [9] which states that any such algorithms has to run in time $\Omega(m)$. ◀

References

1. 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, February 2018. doi:10.1007/s00453-017-0287-3.
2. Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 522–539. Society for Industrial and Applied Mathematics, January 2021. doi:10.1137/1.9781611976465.32.
3. N. Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica (New York)*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
4. Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *Leibniz International Proceedings in Informatics, LIPIcs*, 124, November 2019. doi:10.4230/LIPIcs.ITCS.2019.6.
5. Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016. URL: http://barabasi.com/networksciencebook/.
6. Amartya Shankha Biswas, T. Eden, Q. Liu, Slobodan Mitrović, and R. Rubinfeld. Parallel algorithms for small subgraph counting. *ArXiv*, abs/2002.08299, 2020. arXiv:2002.08299.
7. N. Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.
8. Fan Chung and Linyuan Lu. *Complex Graphs and Networks (Cbms Regional Conference Series in Mathematics)*. American Mathematical Society, USA, 2006.
9. Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017. doi:10.1137/15M1054389.
10. Talya Eden, Dana Ron, and C. Seshadhri. Sublinear Time Estimation of Degree Distribution Moments: The Degeneracy Connection. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.7.
11. Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 722–734, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188810.

12 Talya Eden and Will Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.11.

13 Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

14 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 363–374, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

15 Paul W Holland and Samuel Leinhardt. A method for detecting structure in sociometric data. In *Social networks*, pages 411–432. Elsevier, 1977.

16 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 1–10, New York, NY, USA, 1977. Association for Computing Machinery. doi:10.1145/800105.803390.

17 John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1778–1797, USA, 2017. Society for Industrial and Applied Mathematics.

18 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. doi:10.1080/15427951.2012.625260.

19 Tong Ihn Lee, Nicola J Rinaldi, François Robert, Duncan T Odom, Ziv Bar-Joseph, Georg K Gerber, Nancy M Hannett, Christopher T Harbison, Craig M Thompson, Itamar Simon, et al. Transcriptional regulatory networks in saccharomyces cerevisiae. *science*, 298(5594):799–804, 2002.

20 Richard J. Lipton, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1):195–226, 1993. doi:10.1016/0304-3975(93)90224-H.

21 Duncan T Odom, Nora Zizlsperger, D Benjamin Gordon, George W Bell, Nicola J Rinaldi, Heather L Murray, Tom L Volkert, Jorg Schreiber, P Alexander Rolfe, David K Gifford, et al. Control of pancreas and liver gene expression by hnf transcription factors. *Science*, 303(5662):1378–1381, 2004.

22 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Inf. Process. Lett.*, 112(7):277–281, March 2012. doi:10.1016/j.ipl.2011.12.007.

23 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*, KDD '09, pages 837–846, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1557019.1557111.

24 Osamu Watanabe. Sequential sampling techniques for algorithmic learning theory. *Theoretical Computer Science*, 348(1):3–14, 2005. Algorithmic Learning Theory (ALT 2000). doi:10.1016/j.tcs.2005.09.003.

25 Andreas Wimmer and Kevin Lewis. Beyond and below racial homophily: Erg models of a friendship network documented on facebook. *American Journal of Sociology*, 116(2):583–642, 2010. URL: http://www.jstor.org/stable/10.1086/653658.