Arboricity Matters in Triangle Counting with Random Edges

Arijit Bishnu
Indian Statistical Institute, Kolkata, India

Debarshi Chanda
Indian Statistical Institute, Kolkata, India

Gopinath Mishra
Institute of Mathematical Sciences, Chennai, India

Abstract

Given a simple, unweighted, undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, and parameters $0 < \varepsilon, \delta < 1$, along with `Degree`, `Neighbour`, `Pair` and `RandomEdge` query access to G, we provide a query-based randomized algorithm to generate an estimate $\widehat{T}$ of the number of triangles T in G, such that $\widehat{T} \in [(1-\varepsilon)T, (1+\varepsilon)T]$ with probability at least $1-\delta$. The query complexity of our algorithm is $\widetilde{O}\left(m\alpha \log(1/\delta)/\varepsilon^3 T\right)$, where α is the arboricity of G. Our work can be seen as a natural progression to the line of recent works [Eden et al., SIAM J Comp., 2017; Assadi et al., ITCS 2019; Eden et al., SODA 2020] that considered subgraph or triangle counting with or without the use of `RandomEdge` query. Of these works, Eden et al. [SODA 2020] considers the role of arboricity. Our work is the first to consider how `RandomEdge` query can leverage the structural property of arboricity. Furthermore, continuing in the line of work of Assadi et al. [APPROX/RANDOM 2022], we also provide a lower bound of $\widetilde{\Omega}\left(m\alpha \log(1/\delta)/\varepsilon^2 T\right)$ that matches the upper bound exactly on arboricity, δ, and almost on ε.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Triangle counting, Sublinear algorithms, Subgraph counting

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.55

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/abs/2502.15379`

1 Introduction

Counting the number of triangles present in graphs is a classical problem [20, 5, 53, 4, 10]. Researchers in the streaming (sub-linear space) community have studied this problem for the last two decades; see [10, 11, 35, 2, 17, 6, 36, 18, 22, 55, 41] for a sample of such works. The problem has also been studied by researchers in the property testing (sub-linear time) community in the last decade [26, 48, 39, 11, 13, 28, 35]. Here, graphs can be accessed only through certain queries. On the other hand, starting with the work of [20], the graph parameter called arboricity has also found its use in triangle counting [28]. Our work shows the use of arboricity in triangle counting where the graph can be accessed through random edge queries along with other standard queries like `Degree`, `Neighbour` and `Pair`.

Counting triangles has been a well-motivated problem with applications across various domains like optimizing query size in database join problems [10, 9, 7], computing clustering coefficients, transitivity ratio [1, 46, 60, 44], studying structures in web graphs [25, 23] etc. For a more thorough overview, see the surveys [3, 57]. The parametrization based on arboricity is also of practical interest due to the occurrence of low-arboricity graphs in various real-world scenarios [24, 42, 50, 31, 12, 23, 54].

Formally, we are given a simple, unweighted, undirected graph $G = (V, E)$ with $|V| = n$, and $|E| = m$. The neighbours of v is denoted $\mathsf{N}(v) = \{u | (u, v) \in E\}$ and the degree of the vertex v is denoted $\deg_v = |\mathsf{N}(v)|$. We are given the following queries to access the graph:

- `Degree`(v): Given a vertex v, return $\deg_v$.
- `Neighbour`(v, i): Given a vertex $v \in V$ and $i \in [n]$, return the i-th vertex $u \in \mathsf{N}(v)$ if it exists; otherwise, we get $\perp$.
- `Pair`(u, v): Given two vertices $u, v \in V$, return 1 if $(u, v) \in E$, and 0, otherwise.
- `RandomEdge`: Return an edge $e \in E$ uniformly at random.

We will call the `Degree`, `Neighbour` and `Pair` queries taken together to be *local queries* [7]. Now, we formally state our problem:

Triangle Estimation:
Given ε, δ with $0 < \varepsilon, \delta < 1$, obtain an estimate $\widehat{T}$ of the number of triangles T in the graph G, such that $\widehat{T} \in [(1-\varepsilon)T, (1+\varepsilon)T]$ with probability at least $1 - \delta$.

From now on, we refer $\widehat{T}$ to be an (ε, δ) estimate of T. One of the graph parameters that has been relevant to quantify the complexity of triangle counting algorithms within the context of RAM model [20], streaming model [13], distributed model [34, 56, 30, 45] and query model [29], is the arboricity of the graph G, denoted α. The arboricity of a graph can be seen as a measure of the density of the graph.

▶ **Definition 1** (Arboricity(α)). *The arboricity of a graph $G = (V, E)$, denoted α, is the minimum number of spanning forests that E can be partitioned into.*

Recent works have established various ways to construct lower bounds incorporating the impact of ε and δ both in graph property testing [8] and other problems [58]. Our lower bounds will also involve ε and δ. To illustrate the dependence of ε, δ, and polylog terms on the query complexity of our algorithm, we will use two notations at times in place of $O(\bullet)$ – $\widetilde{O}(\bullet)$ hides polylog factors only but shows dependence on ε and δ, whereas $\widetilde{O}_{\varepsilon,\delta}(\bullet)$ hides both ε, δ and polylog terms.

1.1 Our results

In this section, we present the main results of this work.

Upper bound

We show that using random edge queries in addition to local queries, we can obtain an (ε, δ) estimate of T where the query complexity is parametrized by arboricity apart from the usual parameters of edge, vertex, the number of triangles, ε, and δ. The work of [20] established the role of arboricity in the problem of triangle counting giving an $O(m\alpha)$ time algorithm to exactly count the number of triangles T in the RAM model. In recent works in the property testing model, [26] gave an algorithm that uses local queries and requires $\widetilde{O}_{\varepsilon,\delta}\left(\frac{n}{T^{1/3}} + \min\left\{m, \frac{m^{3/2}}{T}\right\}\right)$ queries to obtain an (ε, δ) estimate $\widehat{T}$ of T. Using `RandomEdge` in addition to the local queries, [7] proposed an algorithm that uses $\widetilde{O}\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ queries. Following this, [28] was introduces the parameter of arboricity to triangle counting in the query based setup. They proposed an algorithm that uses $\widetilde{O}\left(\frac{n}{T^{1/3}} + \frac{m\alpha\log^3(1/\delta)}{\varepsilon^5 T}\right)$ local queries to obtain an (ε, δ) estimate $\widehat{T}$ of T. Our work follows in the line of the works of [26, 7, 28], where we use `RandomEdge` in addition to *local* queries to obtain an algorithm that uses $\widetilde{O}\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^3 T}\right)$ queries (see the last row in Table 1). Formally,

▶ **Theorem 2** (Upper bound). *Given a simple, unweighted and undirected graph $G = (V, E)$ with $|V| = n$, $|E| = m$ and arboricity α, and access to the graph via* `Degree`, `Neighbour`, `Pair` *and* `RandomEdge` *queries, an (ε, δ) estimate $\widehat{T}$ of T can be obtained using $\widetilde{O}\left(\frac{m\alpha \log(1/\delta)}{\varepsilon^3 T}\right)$ queries.*

A constant-pass $\widetilde{O}\left(\frac{m\kappa}{\varepsilon^4 T}\right)$ space streaming algorithm was proposed in [13] for obtaining an (ε, δ)-estimate of the number of triangles in a graph, where κ is the degeneracy of the graph. Our algorithm can be directly extended to a constant pass streaming algorithm using $\widetilde{O}\left(\frac{m\alpha}{\varepsilon^3 T}\right)$ space. By the fact that $\alpha \leq \kappa \leq 2\alpha$, our algorithm's space complexity is as good as the algorithm of [13] in terms of the graph parameters, and asymptotically better in terms of the approximation factor ε.

Lower bound

We also establish a lower bound involving the arboricity and the parameters ε and δ. Reductions from communication complexity is a well studied method to establish lower bound results in terms of query complexity [15, 29, 32]. Using only `Degree` and `Neighbour` queries, triangle counting requires $\Omega(m)$ queries [33]. Including `Pair` with the previous two queries, [26] established a lower bound of $\Omega\left(\frac{n}{T^{1/3}} + \frac{m^{3/2}}{T}\right)$ queries which matches their upper bound up to $poly\left(\frac{\log(n/\delta)}{\varepsilon}\right)$ factors. Later, a simpler proof was proposed for the same bound through reduction from communication complexity problems [29]. This was subsequently extended to incorporate arboricity with a bound of $\Omega\left(\frac{n}{T^{1/3}} + \frac{m\alpha}{T}\right)$ queries [28] which again matched their proposed upper bound up to $poly\left(\frac{\log(n/\delta)}{\varepsilon}\right)$ factors. If `RandomEdge` query is allowed along with all three local queries, the above lower bounds of $\Omega\left(\frac{n}{T^{1/3}} + \frac{m^{3/2}}{T}\right)$ and $\Omega\left(\frac{n}{T^{1/3}} + \frac{m\alpha}{T}\right)$ become $\Omega\left(m^{3/2}/T\right)$ [29] and $\Omega(m\alpha/T)$ [28], respectively. [1] As we have mentioned, for general graphs, there is an algorithm that makes $\widetilde{O}\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ queries, where each query is either a local query or a `RandomEdge` query [7]. Recently, in [8], it has been shown that the algorithm of [7] is optimal (up to $poly(\log(n))$ factors) in terms of all parameters including ε and δ through a lower bound of $\Omega\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ queries for obtaining an (ε, δ) estimate of T using local and `RandomEdge` queries. However, this does not consider the dependency of the query complexity on arboricity of the graph. Like our upper bound, the lower bound of our work also captures the role of arboricity and the parameters ε, and δ when the queries allowed are the local and the `RandomEdge` queries. Moreover, our lower bound combines the parametrization of [8] and [28] in terms of ε, δ, and α, respectively, to establish an improved lower bound.

▶ **Theorem 3** (Lower bound). *Given a simple, unweighted and undirected graph $G = (V, E)$ with $|V| = n$, $|E| = m$ and arboricity α, and access to the graph via* `Degree`, `Neighbour`, `Pair` *and* `RandomEdge` *queries, obtaining an (ε, δ) estimate $\widehat{T}$ of T requires $\Omega\left(\frac{m\alpha \log(1/\delta)}{\varepsilon^2 T}\right)$ queries.*

Table 1 describes the upper and lower bounds across different query accesses and parametrization for the triangle counting problem, with our contributions in this work highlighted with an asterisk $(*)$ mark. Table 1 shows that it is only natural to study the problem of

[1] Though [28] does not mention about $\Omega(m\alpha/T)$ explicitly, it is not difficult to see it from their construction. Also, this lower bound of $\Omega(m\alpha/T)$ matches with our upper bound (stated in Theorem 2) in terms of dependence on m, T, and α.

Table 1 Upper and lower bounds for the Triangle Counting problem. Entries marked with an asterisk (∗) are our results proved in this paper.

	Without `RandomEdge`	**With `RandomEdge`**
Without Arboricity	UB: $\widetilde{O}_{\varepsilon,\delta}\left(\frac{n}{T^{1/3}} + \frac{m^{3/2}}{T}\right)$ [26] LB: $\widetilde{\Omega}\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ [8]	UB: $\widetilde{O}\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ [7] LB: $\widetilde{\Omega}\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ [8]
With Arboricity	UB: $\widetilde{O}\left(\frac{n}{T^{1/3}} + \frac{m\alpha\log^3(1/\delta)}{\varepsilon^5 T}\right)$ [28] (∗) LB: $\Omega\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^2 T}\right)$	(∗) UB: $\widetilde{O}\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^3 T}\right)$ (∗) LB: $\Omega\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^2 T}\right)$

triangle estimation with the use of `RandomEdge` and arboricity as a parameter after the works of [26, 28, 7, 8]. We believe our work also raises some new questions: (a) it opens up the question of quantifying the impact of α to count other subgraphs (e.g., k-cliques) using `RandomEdge` and other local queries; (b) there remains a small gap of $1/\varepsilon$ between our upper and lower bounds that remain to be resolved.

1.2 Related work in other models

The problem of triangle counting is a problem that has been extensively studied across various models of computation. Here, we discuss the works in RAM and streaming models.

RAM model. The work of [20] proposed an $O\left(\frac{m\alpha}{T}\right)$ algorithm in the RAM model that remains the best known till date. Recent works have shown it to be conditionally optimal under the 3SUM [43] and APSP hypotheses [59].

Streaming model. In the data streaming model, a long line of work [51, 52] has culminated in an $\widetilde{O}\left(\frac{m}{T}\left(\Delta_E + \sqrt{\Delta_V}\right)\right)$-space one-pass streaming algorithm for insertion only model [39] that has been shown to be optimal through the combined lower bounds proposed in [16, 39]. Here, Δ_E (Δ_V) denote the maximum number of triangles incident on an edge e (a vertex v) in the graph G. There have been works on triangle counting in multi-pass setup as well as in other streaming models (e.g., turnstile, cash-register etc.) [48, 11, 39, 37, 38, 40, 13].

1.3 Paper organization

Section 2 discusses a broad overview of the techniques we use, and contextualizes it with respect to the relevant prior works. Section 3 sets up the preliminaries for our proofs, including the structural results involving arboricity of a graph. Section 4 discusses the query-based algorithm by breaking the discussion into several parts: a weight function (Section 4.1), an oracle based algorithm (Sections 4.2) and its implementation (Section 4.3) and the final algorithm (Section 4.4). The lower bounds are discussed in Section 5. Some of the proofs are deferred to the Appendix. We have also included the full version of the paper in the Appendix.

2 Technical overview

In this section, we give a broad overview of the techniques used in this work. We denote by T_e the number of triangles incident on the edge e, and the degree of an edge by $\deg_e = \deg_{(u,v)} = \min\{\deg_u, \deg_v\}$.

2.1 Upper bound

Our starting point is to use `RandomEdge` queries to obtain S, a random sample of edges of size s and try to estimate the number of triangles T incident on the edges of S. As the `RandomEdge` query samples edges uniformly at random, the expected number of triangles each sampled edge participates in is $\frac{T}{m}$. Hence, an obvious approach to estimate T would be to count the number of triangles T_e each edge e participates in, and take their average for the sampled edges and scale it appropriately. However, there are two issues with this approach.

Firstly, counting the number of triangles an edge e participates in exactly would require $\Omega(\deg_e)$ *local* queries. Therefore, we have to sample sufficiently many neighbour vertices for each edge e and check if it forms a triangle or not to obtain a good estimate of T_e.

Secondly, even though the expected number of triangles incident on each edge is $\frac{T}{m}$, individual edges e can participate in a large number of triangles, i.e., T_e can be $\Omega(T)$. This makes the variance of the estimate too high, and requires s to be large to obtain a good estimate. To work around this issue, we fix a threshold τ based on the arboricity α of the graph, that has a connection to the number of triangles in the graph. We call edges that participates in more than τ triangles *τ-heavy edges*. Correspondingly, any edge that participates in less than τ triangles is called *τ-light edges*. For now, assume we are given access to an oracle `Heavy` to decide whether an edge is light or heavy. We call the triangles consisting exclusively of heavy edges as *heavy triangles*, and the remaining triangles as *light triangles*. The threshold τ is set at an appropriate multiple of α so that heavy triangles constitute only ε fraction of the triangles. Thus, we can forget about the *τ-heavy edges* and focus only on counting the light triangles. This brings us to the issue of sampling light triangles uniformly at random. We define a weight function on the edges that assigns each light triangle to one of its constituent light edges. The weight function is defined to be the number of light triangles assigned to the edge, with each heavy edge being assigned a value of 0.

Now the algorithm's objective is to estimate this weight function, which cannot be accessed or sampled directly. Observe that there can be multiple assignments of the light triangles to the edges, and consequently multiple weight functions. It suffices for the algorithm to estimate any one of them. In that regard, for all the light edges in our sample S, we maintain a partial non-conflicting assignment. Such an assignment can be extended to a valid assignment, and consequently, a valid weight function. Based on this assignment, we obtain an estimate of the corresponding weight function, which, given an appropriately fixed threshold $\tau = \Theta\left(\frac{\alpha}{\varepsilon}\right)$, and setting s to $\widetilde{O}\left(\frac{m\alpha}{\varepsilon^3 T}\log\frac{1}{\delta}\right)$ will give an (ε, δ) estimate of T.

To design the oracle `Heavy` for a given edge e, we estimate the number of triangles each edge participates in. We can estimate it by sampling each triangle independently through the corresponding neighbouring vertex of e. The probability of obtaining a triangle for an edge e through a random neighbour is $\frac{T_e}{\deg_e}$. Hence, for the heavy edges where $T_e \geq \tau$, T_e can be estimated using sufficiently small number of queries. For the light edges where $T_e \leq \tau$, the probability of obtaining a triangle through a random neighbour can be arbitrarily low. However, note that the lower probability of obtaining triangles allows for high approximation error. We then design a bucketing trick to exploit this trade-off to implement an efficient algorithm to decide whether an edge is heavy or light. For full details, see Section 4.

2.2 Lower bound

Several lower bounds for triangle counting problems based on reductions from the *disjointness* problem in communication complexity [29, 7, 28] are often close to the best possible upper bounds. It is not clear whether this approach can give tight bounds involving ε and δ. Recently, Assadi and Nguyen [8] showed that any algorithm to find an (ε, δ)-estimate of the number of triangles T must make at least $\Omega\left(\frac{m^{3/2}\log(1/\delta)}{\varepsilon^2 T}\right)$ queries, where each query is either a local query or a `RandomEdge` query.[2] To prove the lower bound, they introduced a new query problem called the *Popcount Thresholding Problem* (`PTP`), and gave reductions from this problem to triangle estimation, instead of using disjointness. We also show a reduction from `PTP`. In particular, we show that for any value of arboricity α, we can design a graph G with arboricity α such that finding an (ε, δ)-estimate in it using $\Omega\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^2 T}\right)$ queries would violate the lower bound of `PTP`.

In the $\texttt{PTP}_{M,k,\gamma}$ problem, we are given query access to a binary string $x \in \{0,1\}^M$ and must decide whether it was generated from one of two similar distributions. In the first distribution, denoted $\mathcal{D}_0$, each bit of the string is independently set to 1 with probability $(1-2\gamma)k/M$; in the second distribution, denoted $\mathcal{D}_1$, each bit is independently set to 1 with probability $(1+2\gamma)k/M$. The objective is to distinguish between these two cases by adaptively querying bits of x. For suitable choice of k,γ, and δ, with probability at least $1-\delta$, a string from $\mathcal{D}_0$ will have fewer than $(1-\gamma)k$ 1s and a string from $\mathcal{D}_1$ will have more than $(1+\gamma)k$ 1s. Hence, to distinguish whether x is generated from $\mathcal{D}_0$ or $\mathcal{D}_1$, it suffices to estimate the number of 1s present in x, i.e. $\|x\|_1$ upto multiplicative approximation factor γ. Assadi et al. [8] showed that (under suitable choice of parameters) $\Omega\left(\frac{M\log(1/\delta)}{\gamma^2 k}\right)$ queries to x are needed to distinguish between the case whether $x \sim \mathcal{D}_0$ and $x \sim \mathcal{D}_1$ with probability $1-\delta$.

We now describe the hard graph instance $G_x(V_x, E_x)$ used in the reduction. The core structure of V_x consists of three vertex sets: A, B, and R, where $|A| = |B| = \Theta(m/\alpha)$ and $|R| = \Theta(\alpha)$. Every vertex in $A \cup B$ is connected to all vertices in R, forming a bipartite clique between $A \cup B$ and R. The subgraph induced by $A \cup B$ contains $\Theta(m)$ edges, with each vertex having degree at most $O(\alpha)$. We associate each index $i \in [M]$ from the $\texttt{PTP}_{M,k,\gamma}$ instance with a potential edge between A and B, and include the edge if and only if $x_i = 1$. Thus, the number of edges between A and B equals $\|x\|_1$, and the total number of triangles in the graph G_x is $\|x\|_1 \cdot |R|$. By choosing parameters such that $M = \Theta(m)$, $k = \Theta(T/\alpha)$, and $\gamma = \Theta(\varepsilon)$, distinguishing whether x is drawn from $\mathcal{D}_0$ or $\mathcal{D}_1$ becomes equivalent to distinguishing whether the number of triangles in the graph is at most $(1-\varepsilon)T$ or at least $(1+\varepsilon)T$. To complete the reduction, we show that each local query and `RandomEdge` query to G_x can be simulated using only a constant number of queries to x. To enable this, we extend the construction by introducing two additional disjoint vertex sets A' and B', resulting in five disjoint sets: A, B, R, A', and B'. All triangles in the graph are confined to the sets A, B, and R, while the sets A' and B' are added to ensure that the degree of each vertex in V_x remains identical regardless of whether x is drawn from $\mathcal{D}_0$ or $\mathcal{D}_1$. With this construction, every allowable query to the graph can be simulated with $O(1)$ queries to x. For full details, see Section 5.

[2] Note that this lower bound matches exactly with the corresponding upper bound in [7].

2.3 Contextualizing with relevant prior work

Our work follows as a natural progression to the works of [26, 7, 28] in the property testing framework. So, let us review the algorithmic techniques at play in them.

We start by describing a simple randomized algorithm due to Assadi et al. [7] for estimating the number of triangles in a graph when we have access to queries like `Degree`, `Neighbour`, `Pair`, and `RandomEdge`. First, pick a random edge $\{u, v\}$ and let $\deg_u \leq \deg_v$. If $\deg_u \leq \tau_d$, pick a random neighbour w of u and check if $\{u, v, w\}$ forms a triangle in the correct order, i.e., in the increasing degree sequence. The probability of sampling a triangle in this case is $\frac{1}{2m} \cdot \frac{1}{\deg_u} \cdot \frac{\deg_u}{\tau_d} = \frac{1}{2m\tau_d}$. If $\deg_u > \tau_d$, pick a vertex w at random with probability proportional to its degree. This is possible because `RandomEdge` allows sampling vertices proportional to their degrees. The probability of sampling a triangle in this case is $\frac{1}{2m} \cdot \frac{\deg_w}{2m} \geq \frac{\tau_d}{4m^2}$. By setting $\tau_d = \sqrt{2m}$, we can ensure that the algorithm works effectively. This process finds a triangle with probability $\Omega(T/m^{3/2})$. Finally, by repeating this process $\widetilde{O}(m^{3/2} \log(1/\delta)/\varepsilon^2 T)$ times, we can estimate the total number of triangles.

The setting $\tau_d = \Theta(\sqrt{m})$ seems to be the optimal choice for the general case. We do not see a way to generalize the above idea when parametrized by arboricity. In bounded arboricity graphs, the structure and degree distribution of the graph may not align with this approach. So, a different strategy would be required for graphs with bounded arboricity to achieve the desired bound of $\widetilde{O}(m\alpha \log(1/\delta)/\varepsilon^3 T)$.

Our approach is more closely aligned with the framework of triangle counting developed by Eden et al. [26] with access to local queries only. The algorithm by Eden et al. [26] starts by picking a small set of vertices at random and for each sampled vertex, the algorithm tries to estimate the number of triangles it participates in. Some vertices appear in many triangles and are called heavy, while others are called light. To keep the estimate as desired, the algorithm limits the contribution of heavy vertices and focuses mainly on light ones. It gives more weight to triangles that have more light vertices. Since the number of triangles a vertex participates in cannot be obtained directly, the algorithm estimates this by sampling edges incident on the sampled vertices and checking if they close triangles with neighbouring vertices of those edges. The above framework has also been subsequently extended for clique counting in general graphs [28] and that in bounded arboricity graphs [28].

In summary, the idea of dividing the triangles in terms of their constituent edges or vertices into heavy and light, and dealing with them separately has been a long-standing technique for subgraph counting in sublinear models [48, 26, 11, 27, 38, 28, 13]. The algorithmic techniques vary on how to set and leverage this thresholding behaviour for the specific problems and models. Furthermore, allowing `RandomEdge` queries has yielded simpler algorithms for subgraph counting problems [7]. The usage of `RandomEdge` enables us to design a simple algorithm that uses the structural parameter of arboricity.

The lower bounds for the triangle counting problem in the query model have come from indistinguishability arguments [26] or reduction from communication problems, particularly *set-disjointness* [29, 7, 28] with [28] establishing arboricity-sensitive lower bounds. On the other hand, several recent works [14, 8] have established lower bounds for property testing problems through reductions to query problems that are sensitive to ε and δ. In this work, we extend the ε, δ-sensitive lower bound of [8] to the triangle counting problem for any value of arboricity α.

3 Preliminaries

3.1 Notations

The set of positive integers $\{1, 2, \ldots, x\}$ is denoted $[x]$. The set of triangles in G is denoted Δ, and individual triangles are denoted $\mathtt{t}$. (v, e) denotes a triangle formed by the vertices v and the endpoints of the edge e. $\text{Unif}(S)$ denotes an element of S is chosen uniformly at random.

3.2 Arboricity and its properties

As arboricity plays a crucial role in our work, we put together all the structural results that involve arboricity here. Let us restate the definition once more.

▶ **Definition 4** (Arboricity(α)). *The arboricity of a graph $G = (V, E)$, denoted by α_G, is the minimum number of spanning forests that E can be partitioned into.*

The arboricity of a graph can be seen as a measure of the density of the graph. α_G can be at least $\lceil m/(n-1) \rceil$. Also, $\alpha_G \geq \alpha_H$ where H is any subgraph of G. We will write α instead of α_G when the underlying graph is understood. We introduce the following result due to [20] on the sum of edge degrees over all edges in the graph.

▶ **Lemma 5** (Sum of deg_e [20]). *Given a graph $G = (V, E)$ with arboricity α and $|E| = m$, $\sum_{e \in E} deg_e \leq 2m\alpha$.*

The following result due to [28] builds on the work of [20] to bound the number of triangles based on the number of edges m and arboricity α.

▶ **Lemma 6** (Triangle upper bound [28]). *Given a graph $G = (V, E)$ with arboricity α and $|E| = m$, the graph G has at most $m\alpha$ triangles.*

Note that this upper bound is also tight, i.e., there exists graphs that contain m edges and $\Omega(m\alpha)$ triangles. Additionally, arboricity α can be at most $O(\sqrt{m})$ [28]. Thus, all our results can be reformulated by plugging in this upper bound.

3.3 Chernoff bounds

We will be using the following variation of the Chernoff bound that bounds the deviation of the sum of independent Poisson trials [49].

▶ **Lemma 7** (Multiplicative Chernoff bound). *Given i.i.d. random variables $X_1, X_2, ..., X_t$ where $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = (1 - p)$, define $X = \sum_{i \in [t]} X_i$. Then, we have:*

$$\Pr[X \leq (1-\varepsilon)\,\mathbb{E}[X]] \leq \exp\left(-\frac{\mathbb{E}[X]\,\varepsilon^2}{3}\right) \qquad 0 \leq \varepsilon < 1$$

$$\Pr[X \geq (1+\varepsilon)\,\mathbb{E}[X]] \leq \exp\left(-\frac{\varepsilon^2\,\mathbb{E}[X]}{2+\varepsilon}\right) \qquad 0 \leq \varepsilon$$

4 Algorithm

This section describes the algorithm and its related concepts. Section 4.1 formally defines the weight function for edges and associated structural results. Section 4.2 describes the algorithm assuming access to an idealized oracle that can decide if an edge is heavy or light

(based on whether the edge has many or few incident triangles). Section 4.3 describes how to actually implement this oracle within the problem setup. Section 4.4 puts everything together to develop the final algorithm.

4.1 Weight function

In this section, we formalize the ideas of heavy and light edges and weight function for the edges.

Heavy and light edges and triangles

First, we define heavy and light edges and correspondingly, heavy and light triangles.

▶ **Definition 8** (τ-heavy and τ-light edges). *An edge $e \in E$ is defined to be a τ-heavy (resp. τ-light) edge if it participates in more than τ (resp. $\leq \tau$) triangles.*

▶ **Definition 9** (τ-heavy and τ-light triangles). *A triangle $\mathsf{t} \in \Delta$ is called a τ-heavy triangle if all its three edges are τ-heavy edges. A triangle that is not τ-heavy is a τ-light triangle.*

We denote by T_L^τ and T_H^τ the number of τ-light and τ-heavy triangles in the graph G, respectively. The following lemma bounds the number of τ-heavy triangles in a graph using the parameter of arboricity.

▶ **Lemma 10** (Upper bound on T_H^τ). *Given a graph $G = (V, E)$ with T triangles, the number of τ-heavy triangles is at most $\frac{3T\alpha}{\tau}$.*

Proof. Note that the graph $G = (V, E)$ has T triangles containing at most $3T$ edges. By Definition 9, τ-heavy triangles have all three of their edges to be τ-heavy edges, τ or more triangles incident on each of them. Hence, the number of τ-heavy edges in G is at most $\frac{3T}{\tau}$.

Now consider the subgraph $H = (V_H, E_H)$ of G induced by the τ-heavy edges in G. We know, $E_H \leq \frac{3T}{\tau}$. Also, $\alpha_H \leq \alpha_G = \alpha$ (see Section 3.2). Hence, by Lemma 6, we know that H contains at most $\frac{3T\alpha}{\tau}$ triangles. ◀

The following corollary follows from Lemma 10 and the fact that $T = T_L^\tau + T_H^\tau$.

▶ **Corollary 11** (Lower bound on T_L^τ). *Given a graph $G = (V, E)$, there are at least $(1 - \frac{3\alpha}{\tau})T$ τ-light triangles.*

Weight function for the edges

Now, we define a weight function for the edges. As a triangle is incident to multiple edges, the objective of a weight function is to charge each of the light triangles to exactly one of its participating light edges. To ensure this, we do not charge any light triangle to any heavy edge that it may be incident on. Each light edge e can be charged by at most T_e triangles, i.e., all the triangles the edge e participates in. To avoid over-counting, we ensure that the sum of the weight function over all edges is equal to the number of light triangles, T_L^τ. (t, e) denotes that the triangle t is charged to the edge e.

▶ **Definition 12** (Triangle weight function (w)). *We define a function $w : E \to \mathbb{N}$ to be a triangle weight function if it satisfies the following two conditions:*

$$\begin{aligned} &\textit{Condition (1)}: \quad w(e) \quad \leq \quad \begin{cases} T_e & \textit{if } e \textit{ is a light edge} \\ 0 & \textit{if } e \textit{ is a heavy edge} \end{cases} \\ &\textit{Condition (2)}: \quad \sum_{e \in E} w(e) \quad = \quad T_L^\tau \end{aligned}$$

Observe that there are multiple such weight functions (e.g., a triangle with 3 light edges can be assigned to any one of these 3 edges). Henceforth, we denote the set consisting of valid triangle weight functions by $\mathcal{W}$. We now state a property of valid weight functions. Note that this property is true for any $w \in \mathcal{W}$.

▶ **Lemma 13** (Expectation of $w(e)$)**.** *Consider any triangle weight function $w \in \mathcal{W}$. If we select an edge $e \in E$ uniformly at random, then the expected value of $w(e)$ is $\frac{T_L^\tau}{m}$, i.e., $\mathbb{E}_{e\sim \mathrm{Unif}(E)} w(e) = \frac{T_L^\tau}{m}$.*

Proof. Given that the edges have been chosen uniformly at random and the condition of the triangle weight function (Definition 12), we have:

$$\mathop{\mathbb{E}}_{e\sim \mathrm{Unif}(E)} w(e) \;=\; \sum_{e\in \mathrm{Unif}(E)} \frac{1}{m} w(e) \;=\; \frac{1}{m}\sum_{e\in E} w(e) \;=\; \frac{T_L^\tau}{m}. \qquad \blacktriangleleft$$

4.2 Oracle based algorithm

Our algorithm, due to its usage of the *triangle weight function*, requires knowledge of whether an edge is *heavy* or *light*. The problem of deciding whether an edge is heavy is non-trivial as it directly relates to number of triangles the edge participates in. In this section, we assume black-box access to an oracle `HeavyOracle` that helps us determine if an edge is heavy.

$$\texttt{HeavyOracle}(e,\alpha,\varepsilon) = \begin{cases} 1 & \text{if edge } e \text{ is a } \dfrac{h\alpha}{\varepsilon}\text{-heavy edge, i.e., } \tau = \dfrac{h\alpha}{\varepsilon} \\ 0 & \text{if edge } e \text{ is a } \dfrac{l\alpha}{\varepsilon}\text{-light edge, i.e., } \tau = \dfrac{l\alpha}{\varepsilon} \end{cases}$$

Here h and l ($h > l$) are constants to be determined later. We will discuss an efficient implementation of this oracle later.

Let us first consider the case where we are given an oracle that given an edge e, returns the exact value of a weight function, $w(e)$. Given we can sample edges uniformly at random through the `RandomEdge` query, we can compute $\mathbb{E}_{e\sim \mathrm{Unif}(E)}[w(e)] = T_L^{\frac{l\alpha}{\varepsilon}}$ using this oracle on the sampled edges. However, no such oracle exists in our model. Hence, we try to simulate one through an empirical estimate of the weight function, $\widehat{w}(e)$. Recall the fact that there are many valid weight functions, each being characterized by every light triangle being charged to one of its constituent light edges. Our algorithm will work if the empirical weight function $\widehat{w}$ estimates any one of these weight functions. We achieve this by ensuring that a triangle is not sampled through more than one edge (see Line 17 of Algorithm 1). Thus, the assignments that we consider can be extended to a valid weight function w, and our algorithm can be thought of as estimating this weight function through $\widehat{w}$. We develop our initial algorithm (Algorithm 1) assuming $\widetilde{T}$, an estimate of T satisfying $\widetilde{T} \le 2T$.

In Algorithm 1, we assume that we know m exactly. In fact, our algorithm and analysis work even if we have a constant factor approximation of m. This can be achieved by using $\widetilde{O}_{\varepsilon,\delta}(1)$ queries [7].

We now state and analyse the algorithm.

▶ **Lemma 14** (Expectation of Algorithm 1)**.** *Algorithm 1 ensures that $\mathbb{E}[\widehat{w}(e_i)] = T_L^{\frac{l\alpha}{\varepsilon}}/m$, and $\mathbb{E}\left[\widehat{T}\right] = T_L^{\frac{l\alpha}{\varepsilon}}$.*

Proof. Let $\mathcal{E}_i$ be the event that the edge e is chosen as the i-th edge sample. We proceed with the proof by considering two different cases: $\deg_e < \alpha$ and $\deg_e \ge \alpha$.

Algorithm 1 Triangle Counting Algorithm - with access to `Heavy`, and knowledge of $\widetilde{T}$.

Require: `Degree`, `Neighbour`, `Pair`, and `RandomEdge` query access to a graph G. Parameters $\widetilde{T}, \alpha, \varepsilon, m$ and oracle access to `HeavyOracle` with threshold constants l, h

$s \leftarrow c(1+h)\varepsilon^{-3}(m\alpha/\widetilde{T})\log n$
$S \leftarrow \emptyset$ ▷ S is the set of random edges sampled. $|S|$ grows up to s
$S_\Delta \leftarrow \emptyset$ ▷ S_Δ is the set of light triangles sampled through the edges in S
for $i \in [s]$ **do**
  $e_i \leftarrow$ `RandomEdge`
  $S \leftarrow S \cup e_i$
  Let $e_i = (v_i, x)$ where $\deg_{v_i} < \deg_x$ ▷ Requires two `Degree` queries
  if (`HeavyOracle`$(e_i, \alpha, \varepsilon) = 0$) **then** ▷ e_i is a $\frac{l\alpha}{\varepsilon}$-light edge
    $r_{e_i} \leftarrow 0$ ▷ r_e denotes the number of queries for each edge e
    if $\deg_{v_i} \leq \alpha$ **then**
      set $r_{e_i} \leftarrow 1$ with probability $\frac{\deg_{v_i}}{\alpha}$
    else
      set $r_{e_i} \leftarrow \left\lceil \frac{\deg_{v_i}}{\alpha} \right\rceil$
    for $j \in [r_{e_i}]$ **do**
      Choose $k \leftarrow \text{Unif}\left(\{1, 2, ..., \deg_{v_i}\}\right)$
      $u \leftarrow$ `Neighbour`(v_i, k)
      if (`Pair`$(u, x) = 1$ and $\mathtt{t} = (\mathtt{u}, \mathtt{e_i})$ is not in S_Δ through another edge e') **then**
        $S_\Delta \leftarrow S_\Delta \cup \mathtt{t}$
  ▷ A triangle $\mathtt{t}$ may occur in S_Δ multiple times, but each time it will be through the same edge e
  else
    $r_{e_i} \leftarrow 0$
for $i \in [s]$ **do**
  if $r_{e_i} > 0$ **then**
    $\widehat{w}(e_i) = \frac{1}{r_{e_i}} \sum_{(\mathtt{t}, \mathtt{e_i}) \in \mathtt{S}_\Delta} \max\left(\alpha, \deg_{e_i}\right)$
  else
    $\widehat{w}\left(e_i\right) = 0$
return $\widehat{T} = \frac{m}{s} \sum_{i \in s} \widehat{w}(e_i)$

Case I: ($\deg_e < \alpha$). When $\deg_e < \alpha$, r_e is set to 0 with probability $1 - \deg_e/\alpha$ and 1 with probability $\deg_e/\alpha$, and $\widehat{w}(e)$ is evaluated through a single query. Thus,

$$\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i] = \frac{\deg_e}{\alpha} \sum_{k \in [w(e)]} \frac{1}{\deg_e}\alpha + \left(1 - \frac{\deg_e}{\alpha}\right) \cdot 0 = w(e) \tag{1}$$

Case II: ($\deg_e \geq \alpha$). On the other hand, when $\deg_e \geq \alpha$, let $Z_j, j \in [r_e]$ be the contribution of each r_e queries made for the edge e to the weight function estimate. Then,

$$\mathbb{E}[Z_j|\mathcal{E}_i] = \sum_{k \in [w(e)]} \frac{1}{\deg_e} \deg_e = w(e) \tag{2}$$

Correspondingly, we have by linearity of expectation and Equation (2):

$$\mathbb{E}\left[\widehat{w}(e_i)|\mathcal{E}_i\right] = \frac{1}{r_e}\sum_{j\in[r_e]}\mathbb{E}\left[Z_j|\mathcal{E}_i\right] = \mathbb{E}\left[Z_j|\mathcal{E}_i\right] = w(e) \tag{3}$$

Given that we draw each edge $e \in S$ uniformly at random, we now have by Equations (1), (3), and Lemma 13:

$$\mathbb{E}\left[\widehat{w}(e_i)\right] = \mathop{\mathbb{E}}_{e\sim\mathrm{Unif}(E)} w\left(e\right) = T_L^{\frac{l\alpha}{\varepsilon}}/m$$

By linearity of expectations, we obtain

$$\mathbb{E}\left[\widehat{T}\right] = \mathbb{E}\left[\frac{m}{s}\sum_{i\in s}\widehat{w}(e_i)\right] = T_L^{\frac{l\alpha}{\varepsilon}}$$ ◀

Note that Lemma 14 holds irrespective of whether the assumption $\widetilde{T} \leq 2T$ is satisfied or not. Next, we turn our attention to the variance of $\widehat{w}(e)$.

▶ **Lemma 15** (Variance of Algorithm 1). *Algorithm 1 ensures that* $\mathrm{Var}[\widehat{w}(e_i)] \leq \frac{(1+h)\alpha}{\varepsilon} \cdot \mathbb{E}_{e\sim\mathrm{Unif}(E)}[w(e)]$.

Proof. Again, we consider two different cases as in the proof of Lemma 14.

Case I: ($\deg_e < \alpha$). When $\deg_e < \alpha$, Algorithm 1 makes at most 1 query as $r \leq 1$. As $\widehat{w}(e) \leq \alpha$, we have

$$\mathrm{Var}[\widehat{w}(e_i)|\mathcal{E}_i] \leq \mathbb{E}[\widehat{w}(e_i)^2|\mathcal{E}_i] \leq \alpha\,\mathbb{E}\left[\widehat{w}\left(e_i\right)|\mathcal{E}_i\right] \tag{4}$$

Case II: ($\deg_e \geq \alpha$). Now we consider the case $\deg_e \geq \alpha$ which is more involved as $r \geq 1$. We first estimate the variance of each Z_j individually as defined in Lemma 14. We condition on the event that edge e is chosen as the i-th edge sample, denoted by $\mathcal{E}_i$. If $\texttt{HeavyOracle}(e,\alpha,\varepsilon) = 1$, then $Var[Z_j] = 0, \forall j \in [r]$. Hence, we condition on the event that $\texttt{HeavyOracle}(e,\alpha,\varepsilon) = 0$ from now on. As $Z_j \leq \deg_e$,

$$\mathrm{Var}[Z_j|\mathcal{E}_i] \leq \mathbb{E}[Z_j^2|\mathcal{E}_i] \leq \deg_{e_i}\mathbb{E}[Z_j|\mathcal{E}_i] \tag{5}$$

After having bounded the variance of the contribution of each query, we now obtain the variance of the weight estimate of the edge.

$$\begin{aligned}
\mathrm{Var}[\widehat{w}(e_i)|\mathcal{E}_i] &= \mathrm{Var}\left[\frac{1}{r_{e_i}}\sum_{j\in r_{e_i}} Z_j|\mathcal{E}_i\right] \\
&= \frac{1}{r_{e_i}^2}\sum_{j\in r_{e_i}}\mathrm{Var}\left[Z_j|\mathcal{E}_i\right] && (Z_j \text{ are i.i.d. given } \mathcal{E}_i) \\
&= \frac{\deg_{e_i}}{r_{e_i}}\sum_{j\in r_{e_i}}\frac{1}{r_{e_i}}\mathbb{E}[Z_j|\mathcal{E}_i] && (\text{by Equation (5) and linearity of expectation}) \\
&\leq \alpha\,\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i] && \left(\text{as } r_{e_i} = \left\lceil\frac{\deg_e}{\alpha}\right\rceil\right)
\end{aligned} \tag{6}$$

Now, we remove the conditioning on $\mathcal{E}_i$ using the law of total variance:

$$\begin{aligned}
\mathrm{Var}[\widehat{w}(e_i)] &= \mathop{\mathbb{E}}_{e_i}[\mathrm{Var}[\widehat{w}(e_i)|\mathcal{E}_i]] + \mathop{\mathrm{Var}}_{e_i}[\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i]] && \text{(by the law of total variance)}\\
&\leq \mathop{\mathbb{E}}_{e_i}[\alpha\,\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i]] + \mathop{\mathbb{E}}_{e_i}[\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i]^2] && \text{(by Equations (4) and (6))}\\
&\leq \alpha \mathop{\mathbb{E}}_{e_i}[\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i]] + \frac{h\alpha}{\varepsilon}\cdot\mathop{\mathbb{E}}_{e_i}[\mathbb{E}[\widehat{w}(e_i)|\mathcal{E}_i]] && \text{(as } \texttt{HeavyOracle}(e)=0,\ \frac{h\alpha}{\varepsilon}\geq|T_e|)\\
&\leq \frac{(1+h)\alpha}{\varepsilon}\cdot\mathbb{E}[w(e)] && \blacktriangleleft
\end{aligned}$$

▶ **Theorem 16** (Performance guarantees of Algorithm 1). *Algorithm 1 makes* $c(1+h)\varepsilon^{-3}(m\alpha/\widetilde{T})\log n$ *queries, and* $c(1+h)\varepsilon^{-3}(m\alpha/\widetilde{T})\log n$ *calls to* `HeavyOracle`*, and given* $\widetilde{T}$ *satisfying the assumption that* $\widetilde{T}\leq 2T$*, returns* $\widehat{T}$ *such that* $\Pr(|\widehat{T}-T_L^{\frac{l\alpha}{\varepsilon}}|\geq\varepsilon T_L^{\frac{l\alpha}{\varepsilon}})\leq\frac{1}{c\log n}$.

Proof. The algorithm calls `HeavyOracle` for each of the s edges in S, resulting in a total of $c(1+h)\varepsilon^{-3}(\log n)(m\alpha/\widetilde{T})$ calls.

The algorithm makes s `RandomEdge` queries, $2s$ `Degree` queries, and $2r_e$ `Neighbour` query for each edge $e\in S$. All edges in S are sampled uniformly at random from E. Hence, the expected number of `Neighbour` queries made are:

$$\mathop{\mathbb{E}}_{e\sim\mathrm{Unif}(E)}\left[\left\lceil\frac{\deg_e}{\alpha}\right\rceil\right]=\sum_{e\in E}\frac{1}{m}\left\lceil\frac{\deg_e}{\alpha}\right\rceil\leq\frac{1}{m}\sum_{e\in E}1+\frac{\deg_e}{\alpha}=1+\frac{1}{m}\sum_{e\in E}\frac{\deg_e}{\alpha}\leq 1+\frac{2m\alpha}{m\alpha}=3$$

The last inequality in the above step follows from Lemma 5. Hence, the algorithm makes at most $9s=c(1+h)\varepsilon^{-3}(m\alpha/\widetilde{T})\log n$ queries in expectation. Here the inequality is due to the fact that for a heavy edge e in S, $r_e=0$. Furthermore, we bound the variance of the estimate $\widehat{T}$ as:

$$\mathrm{Var}[\widehat{T}]=\mathrm{Var}\left[\frac{m}{s}\sum_{i\in s}\widehat{w}(e_i)\right]\leq\frac{(1+h)\alpha m^2\,\mathbb{E}[w(e)]}{\varepsilon s}\leq\frac{(1+h)m\alpha\,\mathbb{E}[\widehat{T}]}{\varepsilon s}.$$

The last two steps follow from Lemma 15 and the fact that $\mathbb{E}\left[\widehat{T}\right]=m\,\mathbb{E}\left[w(e)\right]$. We now use Chebyshev's inequality on $\widehat{T}$:

$$\begin{aligned}
\Pr\left(|\widehat{T}-\mathbb{E}[\widehat{T}]|\leq\varepsilon\,\mathbb{E}[\widehat{T}]\right)&\leq\frac{\mathrm{Var}[\widehat{T}]}{\varepsilon^2\,\mathbb{E}[\widehat{T}]^2} && \text{(Chebyshev's inequality)}\\
&\leq\frac{(1+h)m\alpha}{\varepsilon^3 s\,\mathbb{E}[\widehat{T}]} && \text{(by Lemma 15)}\\
&\leq\frac{(1+h)m\alpha}{\varepsilon^3\cdot 4c\,(1+h)\,\varepsilon^{-3}(m\alpha/\widetilde{T})\log n\cdot T_L^{\frac{l\alpha}{\varepsilon}}} && \text{(by Lemma 14)}\\
&\leq\frac{1}{c\log n} && \text{(as } T_L^{\frac{l\alpha}{\varepsilon}}>T/2\geq\widetilde{T}/4)
\end{aligned}$$

where the last inequality uses the fact $T/2\geq\widetilde{T}/4$ due to the assumption that $\widetilde{T}\leq 2T$. ◀

4.3 Implementing the oracle

Rather than the exact oracle (`HeavyOracle`) that we assumed in Algorithm 1, we would design an oracle (called `Heavy`) that given arboricity α, and parameters ε and δ, accepts edges participating in at most $\frac{\alpha}{2\varepsilon}$ triangles as light, and rejects edges participating in at least

$\frac{2\alpha}{\varepsilon}$ triangles as heavy, with probability at least $1-\delta$. The algorithm works by estimating the number of triangles each edge participates in. However, in this case, each `Neighbour` and `Pair` queries generate i.i.d. random variables denoting the existence of triangle through that neighbouring vertex. Thus, our analysis uses the multiplicative Chernoff bound to obtain high probability guarantees for each individual edge.

Algorithm 2 `Heavy`$(e, \alpha, \varepsilon, \delta)$.

Require: `Degree`, `Neighbour`, `Pair`, and `RandomEdge` query access to a graph G

$r \leftarrow \left\lceil \frac{16\varepsilon \deg_e}{\alpha} \log\left(\frac{1}{\delta}\right) \right\rceil$ ▷ r denotes the number of queries for each edge e
Let $e = (u, x)$ where $\deg_u < \deg_x$ ▷ Requires 2 `Degree` queries
$Y \leftarrow 0$
for $i \in [r]$ **do**
  Choose $k \leftarrow \text{Unif}(\{1, 2, ..., \deg_u\})$
  $v_i \leftarrow$ `Neighbour`(u, k) ▷ Requires 1 `Neighbour` query
  if `Pair`$(v_i, x) = 1$ **then** ▷ Requires 1 `Pair` query
    $Y_i \leftarrow 1$ ▷ Found triangle (v_i, e)
  else
    $Y_i \leftarrow 0$
  $Y \leftarrow Y + Y_i$
$Y \leftarrow \frac{Y}{r}$
if $Y \geq \frac{\alpha}{\varepsilon \cdot \deg_e}$ **then**
  return 1
else
  return 0

▶ **Lemma 17** (Estimation guarantee of `Heavy`). *The algorithm* `Heavy`$(e, \alpha, \varepsilon, \delta)$ *satisfies the following properties with probability at least* $1-\delta$*: (i) returns* 1 *edge* e *if it is* $\frac{2\alpha}{\varepsilon}$*-heavy; (ii) returns* 0 *edge* e *if it is* $\frac{\alpha}{2\varepsilon}$*-light.*

Proof. For (i), we only consider edges that have at least $\frac{2\alpha}{\varepsilon}$ triangles. In this case, the random variables Y_i (Y_i as in Algorithm 2; $Y_i = 1$ if v_i and e form a triangle; 0, otherwise) are i.i.d. Bernoulli random variables taking value 1 with probability at least $\frac{2\alpha}{\varepsilon \deg_e}$. Hence, we have the following using linearity of expectation: $\mathbb{E}[Y] = \mathbb{E}\left[\frac{1}{r}\sum_{i\in[r]} Y_i\right] = \mathbb{E}[Y_i] > \frac{2\alpha}{\varepsilon \deg_e}$

As $Y = \frac{1}{r}\sum_i Y_i$, we can upper bound the probability of the algorithm returning 0 for the edge e, by a multiplicative Chernoff bound (Lemma 7) as follows:

$$\begin{aligned}
\Pr\left[Y \leq \frac{\alpha}{\varepsilon \deg_e}\right] &\leq \Pr\left[Y \leq \left(1-\frac{1}{2}\right)\mathbb{E}[Y]\right] && \left(\mathbb{E}[Y] > \frac{2\alpha}{\varepsilon \deg_e}\right) \\
&\leq \exp\left(-\frac{r\,\mathbb{E}[Y]}{12}\right) && \text{(by multiplicative Chernoff bound)} \\
&\leq \exp\left(-\frac{r\alpha}{6\deg_e \varepsilon}\right) && \left(\mathbb{E}[Y] > \frac{2\alpha}{\varepsilon \deg_e}\right) \\
&\leq \delta && \left(r = \frac{16\varepsilon \deg_e}{\alpha}\log\left(\frac{1}{\delta}\right)\right)
\end{aligned}$$

For (ii), we do not have a lower bound on the probability of success ($Y_i = 1$) in general and hence we cannot obtain a lower bound on $\mathbb{E}[Y]$. Now, observe that for the edges that participate in very few triangles, the probability of finding triangles (i.e., getting $Y_i = 1$) is

low. However, such light edges can still tolerate a high approximation error to be accepted (as a light edge). To account for the trade-off between the lower bound on the probability of $Y_i = 1$ and the upper bound on the approximation factor, we divide the edges participating in at most $\frac{\alpha}{2\varepsilon}$ triangles into $\left\lceil \log(\frac{\alpha}{\varepsilon}) \right\rceil$ buckets with each bucket being defined as the set of edges $\mathcal{B}_k = \left\{ e | \frac{\alpha}{2^{k+1}\varepsilon} \leq T_e < \frac{\alpha}{2^k \varepsilon} \right\}$. For each of these buckets, observe that when `Heavy` is called for an edge belonging to the bucket, we have $\Pr[Y_i = 1] = \frac{\mathsf{t_e}}{\deg_e} \geq \frac{\alpha}{2^{k+1}\varepsilon \deg_e}$, and hence $\mathbb{E}[Y] = \mathbb{E}[Y_i] \geq \frac{\alpha}{2^{k+1}\varepsilon \deg_e}$.

On the other hand, we have $\mathbb{E}[Y] < \frac{\alpha}{2^k \varepsilon \deg_e}$, and hence, $\left(1 + 2^{k-1}\right)\mathbb{E}[Y] < \frac{\alpha}{\varepsilon \deg_e}$. Using these observations, we complete the proof by considering any e in these buckets:

$$
\begin{aligned}
\Pr\left[Y \geq \frac{\alpha}{\varepsilon \deg_e} \right] &\leq \Pr\left[Y \geq \left(1 + 2^{k-1}\right)\mathbb{E}[Y] \right] && \left(\left(1 + 2^{k-1}\right)\mathbb{E}[Y] < \frac{\alpha}{\varepsilon \deg_e} \right) \\
&\leq \exp\left(- \frac{2^{2k-2} r \, \mathbb{E}[Y]}{2 + 2^{k-1}} \right) && \text{(by multiplicative Chernoff bound)} \\
&\leq \exp\left(- \frac{2^{2k} r \, \mathbb{E}[Y]}{2^{k+3}} \right) && (k \geq 1) \\
&\leq \exp\left(- \frac{r\alpha}{16\varepsilon \deg_e} \right) && \left(\mathbb{E}[Y] > \frac{\alpha}{2^{k+1}\varepsilon \deg_e} \right) \\
&\leq \delta && \left(r = \frac{16\varepsilon \deg_e}{\alpha} \log\left(\frac{1}{\delta}\right) \right) \quad \blacktriangleleft
\end{aligned}
$$

Lemma 17 shows that Algorithm 2 can decide whether an edge is heavy or not with probability at least $1 - \delta$ using $\frac{16\varepsilon}{\alpha} \deg_e \log\left(\frac{1}{\delta}\right)$ iterations; making 2 `Degree`, 1 `Neighbour` and 1 `Pair` queries in each iteration. For an individual edge, $\deg_e$ can be at most $n - 1$, and hence the subroutine might contribute to additional queries for each edge. However, as part of Algorithm 1, `Heavy` is called on a set of edges drawn uniformly at random. Hence, in the following lemma, we bound the expected number of queries made by `Heavy` when called on edges that were drawn uniformly at random.

▶ **Lemma 18** (Query complexity of `Heavy`). *`Heavy`$(e, \alpha, \varepsilon, \delta)$ makes $c\varepsilon \log\left(\frac{1}{\delta}\right)$ queries in expectation on an edge $e \sim \mathrm{Unif}(E)$.*

Proof. For edge e, `Heavy`$(e, \alpha, \varepsilon, \delta)$ makes $\left\lceil \frac{16\varepsilon \deg_e}{\alpha} \log\left(\frac{1}{\delta}\right) \right\rceil$ iterations with each iteration making 4 queries. Hence, the expected number of queries is:

$$
\begin{aligned}
\mathop{\mathbb{E}}_{e \sim \mathrm{Unif}(E)} \left[4 \left\lceil \frac{16\varepsilon \deg_e}{\alpha} \log\left(\frac{1}{\delta}\right) \right\rceil \right] &\leq \frac{64\varepsilon}{\alpha} \log\left(\frac{1}{\delta}\right) \mathop{\mathbb{E}}_{e \sim \mathrm{Unif}(E)} [\deg_e] + 4 \\
&= \frac{64\varepsilon}{\alpha} \log\left(\frac{1}{\delta}\right) \frac{1}{m} \sum_{e \in E} \deg_e + 4 \\
&\leq \frac{64\varepsilon}{\alpha} \log\left(\frac{1}{\delta}\right) \frac{1}{m} 2m\alpha + 4 && \text{(by Lemma 5)} \\
&\leq 132\varepsilon \log\left(\frac{1}{\delta}\right) && \left(\varepsilon \log\left(\frac{1}{\delta}\right) \geq 1 \right) \quad \blacktriangleleft
\end{aligned}
$$

4.4 Finalizing the algorithm

In this section, we combine the algorithms discussed above to obtain our final algorithm. Our goal is to call `Heavy` with appropriate parameters instead of `HeavyOracle` so that the approximation error due to the heavy triangles not being counted remains small. We also derive the query complexity of the algorithm including the queries made through `Heavy`.

4.4.1 Algorithm with an estimate $\widetilde{T}$

Algorithm 3 Triangle Counting Algorithm - with $\widetilde{T}$.

Require: `Degree`, `Neighbour`, `Pair`, and `RandomEdge` query access to a graph G. Parameters $\widetilde{T}, \alpha, \varepsilon, m$

1: Call Algorithm 1 with parameters $\widetilde{T}, \alpha, \varepsilon/2$, and thresholds $l = 6$, and $h = 24$. We also implement the oracle $\texttt{HeavyOracle}(e, \alpha, \varepsilon)$ through $\texttt{Heavy}\left(e, \alpha, \varepsilon/6, \frac{1}{mn}\right)$. ▷
Due to Algorithm 1 being called with parameter $\varepsilon/2$ and the implementation of `Heavy` as stated, the threshold parameters are evaluated w.r.t. call to `Heavy` with parameter $l = 6$ and $h = 24$

▶ **Theorem 19** (Performance guarantees of Algorithm 3)**.** *Algorithm 3 makes at most* $c\varepsilon^{-3}(m\alpha/\widetilde{T})\log^2 n$ *queries in expectation and returns* $\widehat{T}$ *such that* $\Pr\left[\widehat{T} \in [(1-\varepsilon)T, (1+\varepsilon)T]\right] \geq 1 - \frac{1}{c\log n}$.

Proof. Algorithm 3 will make the same number of calls to `Heavy` as did Algorithm 1 to `HeavyOracle`. By Theorem 16, Algorithm 3 makes $c(1+h)\varepsilon^{-3}(m\alpha/\widetilde{T})\log n$ queries and $c(1+h)\varepsilon^{-3}\log(n)(m\alpha/\widetilde{T})$ calls to `Heavy`. By Lemma 18, each call to `Heavy` requires $c\varepsilon\log n$ queries in expectation. Combining with the fact that we have $h = 24$, the algorithm requires $(c + c\varepsilon\log n)\left(\varepsilon^{-3}(m\alpha/\widetilde{T})c\log n\right) \leq c\varepsilon^{-3}(m\alpha/\widetilde{T})\log^2 n$ queries in expectation.

By Lemma 17, we can use $\texttt{Heavy}\left(e, \alpha, \varepsilon/6, \frac{1}{mn}\right)$ as an implementation of the idealized $\texttt{HeavyOracle}(e, \alpha, \varepsilon)$. This ensures that a $\frac{6\alpha}{\varepsilon}$-light edge is accepted and a $\frac{24\alpha}{\varepsilon}$-heavy edge is rejected with probability at least $1 - \frac{1}{mn}$. By union bound, the algorithm accepts all $\frac{6\alpha}{\varepsilon}$-light edges and rejects all $\frac{24\alpha}{\varepsilon}$-heavy edges in S with probability at least $1 - \frac{1}{n}$. Hence, by Corollary 11 and Lemma 14, we have,

$$\mathbb{E}\left[\widehat{T}\right] = T_L^{\frac{6\alpha}{\varepsilon}} \in [(1-\varepsilon/2)T, T] \tag{7}$$

By Theorem 16, we have that:

$$\Pr\left[\widehat{T} \in \left[(1-\varepsilon/2)T_L^{\frac{6\alpha}{\varepsilon}}, (1+\varepsilon/2)T_L^{\frac{6\alpha}{\varepsilon}}\right]\right] \geq 1 - \frac{1}{c\log n} \tag{8}$$

Combining Equations (7) and (8) and using union bound on the event that all oracle calls were executed correctly, for large enough n:

$$\Pr\left[\widehat{T} \in [(1-\varepsilon)T, (1+\varepsilon)T]\right] \geq 1 - \frac{1}{c\log n}$$ ◀

4.4.2 Getting rid of the assumption on $\widetilde{T}$

In this section, we describe the steps to obtain Theorem 22 from Theorem 19. First, we remove the assumption on the knowledge of $\widetilde{T}$ for Algorithm 3. To achieve that, we search for an appropriate choice of $\widetilde{T}$ starting from $\bar{t} = n^3$ and halving it each time we fail to find a $\widetilde{T}$ satisfying the assumption that $\widetilde{T} \leq 2T$. We must also bound the probability of $\bar{t}$ deviating significantly below T. To ensure that, for each value of $\bar{t}$, we run Algorithm 3 using values of $\widetilde{T}$ as $n^3, n^3/2, ..., \bar{t}$.

First we bound the probability that the Algorithm 4 terminates at a choice of $\widetilde{T}$ that does not satisfy the assumption that $\widetilde{T} \leq 2T$.

Algorithm 4 Triangle Counting Algorithm.

Require: `Degree`, `Neighbour`, `Pair`, and `RandomEdge` query access to a graph G. Parameters α, ε, m

1: **for** $\bar{t}$ in $\left[n^3, n^3/2, ..., 1\right]$ **do**
2: **for** $\widetilde{T}$ in $\left[n^3, n^3/2, ..., \bar{t}\right]$ **do**
3: **for** $i \in [2\log(c\log(n))]$ **do**
4: $X_i \leftarrow$ Solution to Algorithm 3 with parameters $\widetilde{T}, \alpha, \varepsilon$
5: $X \leftarrow \min_i X_i$
6: **if** $X \geq \widetilde{T}$ **then**
7: **return** X

▶ **Lemma 20** (Termination bound of Algorithm 4). *When $\widetilde{T} > 2T$, the algorithm terminates with probability at most $\frac{1}{c\log^2 n}$.*

Proof. By Lemma 14, we have from Markov's inequality:

$$\Pr\left[\widehat{T} > 2T\right] \leq \frac{\mathbb{E}\left[\widehat{T}\right]}{2T} \leq \frac{1}{2}$$

Now, we have $\Pr\left[X \geq \widetilde{T}\right] \leq \Pr\left[\cap_i X_i \geq \widetilde{T}\right] \leq \frac{1}{c\log^2 n}$ ◀

▶ **Lemma 21** (Lower bound of $\bar{t}$ for Algorithm 4). *The algorithm reaches a value of $\bar{t} \leq T/2^k$ $(k \geq 1)$ with probability at most $\frac{1}{(c\log n)^k}$.*

Proof. For every such value of $\bar{t}$, we have a $\widetilde{T}$ in Line 2 of Algorithm 4 such that $\widetilde{T} \leq T/2$. By Theorem 19, the algorithm returns an estimate $X_i \geq T/2 \geq \widetilde{T}$ with probability at least $1 - \frac{1}{c\log n}$. Taking a union bound over possible values of $i \in [2\log(c\log n)]$, we have Algorithm 4 returning an X such that $X \geq T/2 \geq \widetilde{T}$ with probability at least $1 - \frac{1}{c\log n}$, and the algorithm terminates.

Hence, to get to a $\bar{t} \leq \frac{T}{2^k}$, Algorithm 4 must have failed to terminate for all previous values of $\bar{t}$, which happens with probability at most $\frac{1}{(c\log n)^k}$. ◀

Finally we combine Lemmas 20 and 21 to obtain the following theorem quantifying the estimation guarantees and query complexity of Algorithm 4. The following theorem formalizes the Theorem 2 of Section 1.1.

▶ **Theorem 22** (Upper bound - Formal). *There exists an algorithm that makes $\widetilde{O}\left(\frac{m\alpha\log\frac{1}{\delta}}{\varepsilon^3 T}\right)$ queries in expectation, and returns $X \in [(1-\varepsilon)T, (1+\varepsilon)T]$ with probability at least $1-\delta$.*

Proof. The algorithm runs through at most $\log^2 n$ values of $\widetilde{T}$ such that $\widetilde{T} > 2T$. By Lemma 20, the algorithm terminates in each such case with probability at most $1/c\log^2 n$. For each value of $\bar{t} \leq 2T$, by Theorem 19, the algorithm returns a wrong value with probability at most $1/c\log n$. There are at most $3\log n$ such values of $\bar{t}$. Taking an union bound and fixing the constant c appropriately, the algorithm returns a correct output with probability at least $5/6$.

For each $\bar{t}$ in Line 1 of Algorithm 4, by Theorem 19, the algorithm makes $\widetilde{O}\left((m\alpha/\varepsilon^3\bar{t})\right)$ queries. Hence, till $\bar{t} \leq T/2$, the algorithm makes $\widetilde{O}\left((m\alpha/\varepsilon^3 T)\right)$ queries. Additionally, by Lemma 21, the queries beyond $\bar{t} \leq T$ can be bounded as:

$$\sum_{i\in\log(n)} \frac{1}{(c\log n)^k} \cdot 2^k \cdot \widetilde{O}\left(\frac{m\alpha}{\varepsilon^3 T}\right) = \widetilde{O}\left(\frac{m\alpha}{\varepsilon^3 T}\right)$$

By the well-known median trick [19], the median of $O\left(\log\frac{1}{\delta}\right)$ independent runs of Algorithm 4 establishes the result. ◀

5 Lower bound

In this section, we prove a lower bound that accounts for ε, δ, and the arboricity α of the graph by extending the ideas proposed in [8]. Our lower bound almost matches our proposed upper bound stated in Theorem 22. We first state the problem Popcount Thresholding Problem (referred to as PTP) in the query framework:

▶ **Definition 23** ($\texttt{PTP}_{M,k,\gamma}$). *Given a string $x \in \{0,1\}^M$, the problem $\textit{PTP}_{M,k,\gamma}$ is to distinguish whether x is generated by i.i.d. samples from $\mathcal{D}_0$ or $\mathcal{D}_1$ defined as follows:*

- $\boldsymbol{\mathcal{D}_0}$ *: For all $i \in [M]$, x_i is set to 1 with probability $(1-2\gamma)\frac{k}{M}$, and set to 0, otherwise.*
- $\boldsymbol{\mathcal{D}_1}$ *: For all $i \in [M]$, x_i is set to 1 with probability $(1+2\gamma)\frac{k}{M}$, and set to 0, otherwise.*

The problem is to decide if x is generated from $\mathcal{D}_0$ or $\mathcal{D}_1$ by querying x at any of its M bits.

We state the following lemma [8] quantifying the query complexity of $\texttt{PTP}_{M,k,\gamma}$.

▶ **Lemma 24** (PTP Query Lower Bound [8]). *For any $\gamma \in (0,1/4)$, $\delta \in (0,1/100)$, and integers $M \geq 1$, $\log(1/\delta)\cdot 12/\gamma^2 \leq k \leq M/6$, $R_\delta(\textit{PTP}_{M,k,\gamma}) \geq \frac{M\log(1/4\delta)}{24\gamma^2 k}$ where $R_\delta(\textit{PTP}_{M,k,\gamma})$ is the randomized query complexity to decide $\textit{PTP}_{M,k,\gamma}$ problem with probability at least $1-\delta$.*

We also state the following lemma, establishing bounds on the ℓ_1-norm of x, which was partially proved in [8].

▶ **Lemma 25.** *In $\textit{PTP}_{M,k,\gamma}$, for any $\gamma \in (0,1/4)$, $\delta \in (0,1/100)$, and integers $M \geq 1$, $\log(1/\delta)\cdot 12/\gamma^2 \leq k \leq M/6$, let $\|x\|_1$ denote the number of 1's in the string x, then:*

(i) $\Pr[\|x\|_1 > (1-\gamma)\cdot k \,|\, x \sim \mathcal{D}_0] \leq \delta$;

(ii) $\Pr[\|x\|_1 < (1+\gamma)\cdot k \,|\, x \sim \mathcal{D}_1] \leq \delta$;

(iii) $\Pr[\|x\|_1 < (1-4\gamma)\cdot k] \leq \delta$;

In [8], (i) and (ii) have been shown using Chernoff bound. (iii) can also be shown by using Chernoff bound. We combine two results to obtain the result. First we state the lemma due to [8]:

▶ **Lemma 26** ([8]). *In $\textit{PTP}_{M,k,\gamma}$, for any $\gamma \in (0,1/4)$, $\delta \in (0,1/100)$, and integers $M \geq 1$, $\log(1/\delta)\cdot 12/\gamma^2 \leq k \leq M/6$,*

(i) $\Pr[\|x\|_1 > (1-\gamma)\cdot k \mid x \sim \mathcal{D}_0] \leq \delta$;

(ii) $\Pr[\|x\|_1 < (1+\gamma)\cdot k \mid x \sim \mathcal{D}_1] \leq \delta$

where $\|x\|_1$ denotes the number of 1's in the string x.

We introduce another lemma to lower bound the ℓ_1 norm of x for $\mathcal{D}_0$.

▶ **Lemma 27.** *In $\textit{PTP}_{M,k,\gamma}$, for any $\gamma \in (0,1/4)$, $\delta \in (0,1/100)$, and integers $M \geq 1$, $\log(1/\delta)\cdot 12/\gamma^2 \leq k \leq M/6$, $\Pr[\|x\|_1 < (1-4\gamma)\cdot k \mid x \sim \mathcal{D}_0] \leq \delta$*

Proof. By definition of $\mathcal{D}_0$ and linearity of expectations, we have:

$$\mathbb{E}[\|x\|_1] = \sum_{i\in[M]} \mathbb{E}[x_i] = (1-2\gamma)\,k$$

Now, given the independence of each x_i, we can use Chernoff bound (Lemma 7) to obtain:

$$\begin{aligned}
&\Pr\left[\|x\|_1 < (1-4\gamma)\,k\right] \\
&= \Pr\left[\|x\|_1 - \mathbb{E}\left[\|x\|_1\right] > \frac{2\gamma}{1-2\gamma}\,\mathbb{E}\left[\|x\|_1\right]\right] \\
&\leq \exp\left(-\frac{4\gamma^2\,\mathbb{E}\left[\|x\|_1\right]}{3\,(1-2\gamma)^2}\right) && \text{(by Chernoff bound)} \\
&\leq \exp\left(-\frac{48\gamma^2\log\left(1/\delta\right)}{3\gamma^2\,(1-2\gamma)}\right) && \left(\mathbb{E}\left[\|x\|_1\right] = (1-2\gamma)\,k, k \geq \log(1/\delta)\cdot 12/\gamma^2\right) \\
&\leq \delta && (\gamma < 1/4)\ \blacktriangleleft
\end{aligned}$$

Combining Lemma 26 and 27, we obtain the desired result.

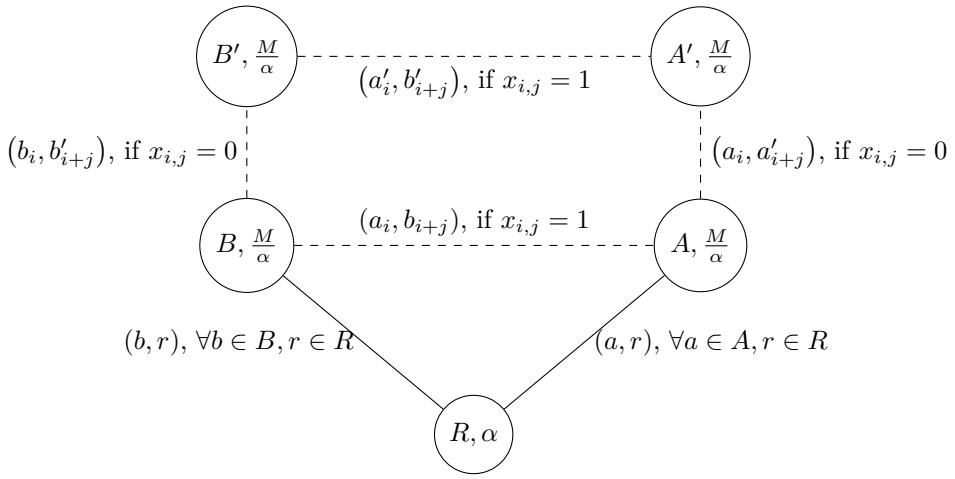

Figure 1 Lower Bound Graph Construction.

The following theorem formalizes the Theorem 3 of Section 1.1.

▶ **Theorem 28** (Lower bound - Formal). *Any algorithm that solves triangle estimation problem with $\varepsilon \leq \frac{1}{8}$ using* `Degree`, `Neighbour`, `Pair` *and* `RandomEdge` *for any graph with m edges, T triangles, and arboricity α such that $T = \Omega\left(\frac{\alpha\log(1/\delta)}{\varepsilon^2}\right)$ requires $\Omega\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^2 T}\right)$ queries.*

Proof. We want to show that given a graph G, it takes $\Omega\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^2 T}\right)$ queries to obtain an estimate $\widetilde{T}$ such that $\widetilde{T} \in (1\pm\varepsilon)T$ with probability at least $1-\delta$. For a contradiction, let us assume that there exists an algorithm $\mathcal{A}_T$ that, for some arboricity α, computes an estimate $\widetilde{T}$ such that $\widetilde{T} \in (1\pm\varepsilon)T$ with probability at least $1-\delta$ using $o\left(\frac{m\alpha\log(1/\delta)}{\varepsilon^2 T}\right)$ queries.

To show the contradiction, we design an algorithm that solves $\mathtt{PTP}_{M,k,\gamma}$ in less than $\frac{M\log(1/4\delta)}{24\gamma^2 k}$ queries using $\mathcal{A}_T$. Given an instance of x generated from $\mathtt{PTP}_{M,k,\gamma}$, we consider a graph $G_x = (V_x, E_x)$ defined as:

The vertex set V_x. The vertex set V_x consists of 5 sets of disjoint and independent set of vertices A, A', B, B', and R, with $|A| = |A'| = |B| = |B'| = M/\alpha$, and $|R| = \alpha$. Vertices of A, A', B and B' will be named as a, a', b and b', respectively.

The edge set E_x. There exists an edge between every vertex in $A \cup B$ to every vertex in R. Observe that this makes sure that the arboricity of G_x is at least the average degree of the subgraph induced by $A \cup B \cup R$, i.e., $\Omega(\alpha)$. If a bit of x is 1 (resp. 0), there will be an edge from a vertex in A (resp. A) to a vertex in B (resp. A'), and an edge from a vertex in A' (resp. B) to a vertex in B' (resp. B').

We index the problem instance x of length M as $\left[\frac{M}{\alpha}\right] \times [\alpha]$, denoting $x_{(i-1)\times\alpha+j}$ as $x_{i,j}$, where $i \in \left[\frac{M}{\alpha}\right]$ and $j \in [\alpha]$. We add an edge $\left(a_i, a'_{i+j}\right)$, and an edge $\left(b_i, b'_{i+j}\right)$ if $x_{i,j} = 0$. We add an edge (a_i, b_{i+j}), and an edge $\left(a'_i, b'_{i+j}\right)$ if $x_{i,j} = 1$. All $(i+j)$ additions here are modulo $\left[\frac{M}{\alpha}\right]$.

Consider the orientation of edges such that all edges from R is oriented outwards, and other edges are oriented arbitrarily. This gives a DAG with the out-degree of each edge being bounded by 2α. As a DAG with out-degree at most α ensures that the arboricity of the graph G_x is at most α[28, 47, 21], the graph has arboricity at most 2α. Hence, the arboricity of G_x is $\Theta(\alpha)$, as we have already shown it to be $\Omega(\alpha)$.

From the construction, observe that the number of edges between A and B is $\|x\|_1$. Moreover, each edge between A and B adds α triangles. Hence, the number of triangles in G_x is exactly $\|x\|_1 \alpha$. Also note that we have added $2M$ edges between $A \cup B$ and R, and further $2M$ edges (in the subgraph induced by $V_x \setminus S$) according to the entries of x, 2 for each bit x_i. Hence, we have $m = 4M$, and we fix $\varepsilon = \gamma$.

We now describe $\mathcal{A}_T^{\mathtt{PTP}}$ an algorithm that uses $\mathcal{A}_T$ to solve $\mathtt{PTP}_{M,k,\gamma}$ using less than $\frac{M \log(1/4\delta)}{24\gamma^2 k}$ queries. Given a string x, we generate G_x and estimate the number of triangles $\widehat{T}$ by calling $\mathcal{A}_T$ using $\frac{m \log(1/4\delta)}{200\varepsilon^2 k} = \frac{M \log(1/4\delta)}{50\gamma^2 k}$ queries. If $\widehat{T} < \left(1-\gamma^2\right) k\alpha$, it outputs $x \sim \mathcal{D}_0$; otherwise, it outputs $x \sim \mathcal{D}_1$. Also, observe that by Lemma 25, we have $T = \alpha \|x\|_1 \geq \alpha(1-4\gamma)k \geq \frac{\alpha k}{2}$ with probability at least $1-\delta$, where the last inequality follows from the fact that $\gamma = \varepsilon \leq \frac{1}{4}$. Lemma 25 is applicable due to the fact that $T = \Omega\left(\frac{\alpha \log(1/\delta)}{\varepsilon^2}\right)$. Hence, we have $\frac{m \log(1/4\delta)}{200\varepsilon^2 k} \geq \frac{m\alpha \log(1/4\delta)}{200\varepsilon^2 T}$. Thus, $\mathcal{A}_T$ is allowed to make $o\left(\frac{m\alpha \log(1/\delta)}{\varepsilon^2 T}\right)$ queries under the given query bound.

By Lemma 25, to distinguish between whether x is generated from $\mathcal{D}_0$ or $\mathcal{D}_1$ with probability at least $1-\delta$, it suffices to show that we can use $\mathcal{A}_T$ to distinguish between the instance where $\|x\|_1 > (1+\gamma)\,k$ and the instance where $\|x\|_1 < (1-\gamma)\,k$ instance with probability at least $1-\frac{\delta}{2}$. We now show that $\mathcal{A}_T^{\mathtt{PTP}}$ outputs $\mathcal{D}_1$ with probability at least $1-\frac{\delta}{2}$ given $\|x\|_1 > (1+\gamma)\,k$, and outputs $\mathcal{D}_0$ with probability at least $1-\frac{\delta}{2}$ given $\|x\|_1 < (1-\gamma)\,k$. We consider the two cases separately:

Case I: $(\|x\|_1 > (1+\gamma)\,k)$. Our construction ensures that the number of triangles in G_x is $T = \|x\|_1 \alpha > (1+\gamma)\,k\alpha = (1+\varepsilon)\,k\alpha$. Additionally, by our assumption on $\mathcal{A}_T$, it outputs an estimate $\widehat{T} \geq (1-\varepsilon)T$ with probability $1-\frac{\delta}{2}$. Thus, we have $\widetilde{T} \geq \left(1-\varepsilon^2\right) k\alpha$ with probability at least $1-\frac{\delta}{2}$.

Case II: $(\|x\|_1 < (1-\gamma)\,k)$. Our construction ensures that the number of triangles in G_x is $T = \|x\|_1 \alpha < (1-\gamma)\,k\alpha = (1-\varepsilon)\,k\alpha$. Additionally, by our assumption on $\mathcal{A}_T$, it outputs an estimate $\widehat{T} \leq (1+\varepsilon)T$ with probability $1-\frac{\delta}{2}$. Thus, we have $\widetilde{T} \leq \left(1-\varepsilon^2\right) k\alpha$ with probability at least $1-\frac{\delta}{2}$.

Now we state how to simulate the required queries used by the algorithm:

- **`Degree`(v):** For a vertex $v \in A \cup B \cup A' \cup B'$, return α, and for a vertex $v \in R$, return $\frac{2M}{\alpha}$. Hence, for `Degree` queries, no queries are made to the string.
- **`Neighbour`(v, i):** For a vertex $v \in A \cup B \cup A' \cup B'$, w.l.o.g assume v to be $a_j \in A$. Return a'_{i+j} if $x_{i,j} = 0$, and b_{i+j} otherwise. For a vertex $v \in R$, if $i \leq \frac{m}{\alpha}$, return a_i, else return $b_{i-\frac{m}{\alpha}}$. Hence, for `Neighbour` queries, a single query is made to the string.

- **`Pair`(u, v):** Given a vertex pair (v, u), there can be 3 cases: (a) If the vertex $v \in A \cup B$, w.l.o.g assume v to be $a_i \in A$, if $u \in R$, return 1, else if $u = b_j \in B$ and $x_{i,j-i} = 1$, return 1, else if $u = a'_j \in A'$ and $x_{i,j-i} = 0$, return 1, else return 0. (b) If a vertex $v \in A' \cup B'$, w.l.o.g assume the v to be $a'_i \in A'$, if $u = a_j \in A$, return 1 if $x_{j,i-j} = 0$, else if $u = b'_j \in B'$, return 1 if $x_{j,i-j} = 1$, else return 0. If $v \in R$, return 1 if $u \in A \cup B$, else return 0. (c) If a vertex $v \in R$, return 1 if $u \in A \cup B$, return 0 otherwise. Hence, for `Pair` queries, a single query is made to the string.
- **`RandomEdge`:** Sample a vertex v with probability $\frac{\deg_v}{4M}$. Choose $i \in \deg_{(v)}$ uniformly at random, query $u =$ `Neighbour`(v, i) and return (v, u). Hence, for `RandomEdge` queries, a single query is made to the string for the `Neighbour` query. ◀

The following corollary is a direct implication from Theorem 28 due to the fact that we consider a weaker model (without access to `RandomEdge` queries) compared to Theorem 28.

▶ **Corollary 29** (Lower Bound for Local Queries). *Any algorithm that solves triangle estimation problem with $\varepsilon \leq \frac{1}{4}$ using `Degree`, `Neighbour`, and `Pair` requires $\Omega\left(\frac{m\alpha \log(1/\delta)}{\varepsilon^2 T}\right)$ queries.*

6 Conclusion

In this paper, we present an almost optimal algorithm for triangle counting in graphs of bounded arboricity, assuming access to both local queries and `RandomEdge` queries. In particular, the query complexity of our algorithm is $O(m\alpha/T)$.[3] Extending our result to clique counting remains an interesting open question. Eden et al. [28] showed that the number of k-cliques can be estimated using $O\left(n/T_k^{1/k} + m^{k/2}/T_k\right)$ local queries, where T_k denotes the number of k-cliques in the graph. When `RandomEdge` queries are also available, Assadi et al. [7] established a bound of $O\left(m^{k/2}/T_k\right)$. For graphs with arboricity bounded by α, Eden et al. [28] showed that $O\left(n/T_k^{1/k} + m\alpha^{k-2}/T_k\right)$ local queries suffice. We believe that, when `RandomEdge` queries are available in addition to local queries, the bound can be improved to $O\left(m\alpha^{k-2}/T_k\right)$. However, establishing the complexity of clique counting (when `RandomEdge` is available along with local queries) remains an open question. A similar open question was posed in the streaming model by Bera and Seshadri [13].

References

1 Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Comput. Surv.*, 47(1), May 2014. doi:10.1145/2601412.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '12, pages 5–14, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2213556.2213560.

3 Mohammad Al Hasan and Vachik S. Dave. Triangle counting in large networks: a review. *WIREs Data Mining and Knowledge Discovery*, 8(2):e1226, 2018. doi:10.1002/widm.1226.

4 N. Alon. Testing Subgraphs in Large Graphs. In *IEEE 54th Annual Symposium on Foundations of Computer Science*, page 434, Los Alamitos, CA, USA, October 2013. IEEE Computer Society. doi:10.1109/SFCS.2001.959918.

5 Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM J. Discret. Math.*, 22(2):786–819, 2008. doi:10.1137/07067917X.

[3] In this section, we suppress the dependence on polylogarithmic factors, for simplicity of presentation.

6 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

7 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2019.6.

8 Sepehr Assadi and Hoai-An Nguyen. Asymptotically optimal bounds for estimating h-index in sublinear time with applications to subgraph counting. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 48:1–48:20, Illinois, Urbana-Champaign, USA (Virtual Conference), 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.48.

9 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013. doi:10.1137/110859440.

10 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*, SODA '02, pages 623–632, USA, 2002. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=545381.545464.

11 Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 11:1–11:14, Hannover, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2017.11.

12 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph Coloring via Degeneracy in Streaming and Other Space-Conscious Models. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.11.

13 Suman K. Bera and C. Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, pages 457–467, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3375395.3387665.

14 Lorenzo Beretta and Jakub Tětek. Better sum estimation via weighted sampling. *ACM Trans. Algorithms*, 20(3), June 2024. doi:10.1145/3650030.

15 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *computational complexity*, 21(2):311–358, May 2012. doi:10.1007/s00037-012-0040-x.

16 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *40th International Colloquium on Automata, Languages, and Programming (ICALP 2013)*, ICALP'13, pages 244–254, Berlin, Heidelberg, 2013. Springer-Verlag. doi:10.1007/978-3-642-39206-1_21.

17 Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278, September 2016. doi:10.1007/s00453-015-0036-4.

18 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 253–262, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1142351.1142388.

19 Amit Chakrabarti. Data stream algorithms. *Computer Science*, 49:149, 2015.

20 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. doi:10.1137/0214017.

21 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243–266, 1991. doi:10.1016/0304-3975(91)90020-3.

22 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theoretical Computer Science*, 683:22–30, 2017. doi:10.1016/j.tcs.2016.06.020.

23 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.

24 Michal Dory, Mohsen Ghaffari, and Saeed Ilchi. Near-optimal distributed dominating set in bounded arboricity graphs. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, pages 292–300, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538437.

25 Jean-Pierre Eckmann and Elisha Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the National Academy of Sciences of the United States of America*, 99:5825–5829, 2001. URL: https://api.semanticscholar.org/CorpusID:2618949.

26 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.

27 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.

28 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 1467–1478, USA, 2020. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975994.89.

29 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 für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.11.

30 Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. Clique counting in mapreduce: Algorithms and experiments. *ACM J. Exp. Algorithmics*, 20, October 2015. doi:10.1145/2794080.

31 Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *Proceedings of the 32nd International Conference on Graph-Theoretic Concepts in Computer Science*, WG'06, pages 159–167, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11917496_15.

32 Oded Goldreich. *On the communication complexity methodology for proving lower bounds on the query complexity of property testing*, pages 87–118. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, Germany, April 2020. doi:10.1007/978-3-030-43662-9_7.

33 Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Discrete Mathematics*, 25(3):1365–1411, 2011. doi:10.1137/100783066.

34 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*, WWW '17, pages 441–449, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. doi:10.1145/3038912.3052636.

35 Rajesh Jayaram and John Kallaugher. An Optimal Algorithm for Triangle Counting in the Stream. In *APPROX/RANDOM 2021*, volume 207 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:11, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.11.

36 Madhav Jha, C. Seshadhri, and Ali Pinar. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Trans. Knowl. Discov. Data*, 9(3), February 2015. doi:10.1145/2700395.

37 John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 556–567. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00059.

38 John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 119–133. ACM, 2019. doi:10.1145/3294052.3319706.

39 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. doi:10.1137/1.9781611974782.116.

40 John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1223–1236. ACM, 2020. doi:10.1145/3357713.3384278.

41 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 598–609, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-31585-5_53.

42 Christian Konrad, Andrew McGregor, Rik Sengupta, and Cuong Than. Matchings in Low-Arboricity Graphs in the Dynamic Graph Stream Model. In Siddharth Barman and Sławomir Lasota, editors, *44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024)*, volume 323 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2024.29.

43 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*, SODA '16, pages 1272–1287, USA, 2016. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974331.CH89.

44 Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 462–470, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1401890.1401948.

45 Quanquan C. Liu and C. Seshadhri. Brief announcement: Improved massively parallel triangle counting in o(1) rounds. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 519–522, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3662158.3662819.

46 Duncan Luce and Albert D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14:95–116, 1949. URL: https://api.semanticscholar.org/CorpusID:16186758.

47 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983. doi:10.1145/2402.322385.

48 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*, PODS '16, pages 401–411, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2902251.2902283.

49 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge, 2005. doi:10.1017/CBO9780511813603.

50 Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic mis in uniformly sparse graphs. *ACM Trans. Algorithms*, 16(2), March 2020. doi:10.1145/3378025.

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

52 A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, 2013. doi:10.14778/2556549.2556569.

53 Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms*, pages 606–609, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11427186_54.

54 Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowl. Inf. Syst.*, 54(3):677–710, March 2018. doi:10.1007/s10115-017-1077-6.

55 Kijung Shin, Sejoon Oh, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Fast, accurate and provable triangle counting in fully dynamic graph streams. *ACM Trans. Knowl. Discov. Data*, 14(2), February 2020. doi:10.1145/3375392.

56 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*, WWW '11, pages 607–614, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1963405.1963491.

57 Charalampos Tsourakakis, Mihail Kolountzakis, and Gary Miller. Triangle sparsifiers. *Journal of Graph Algorithms and Applications*, 15(6):703–726, October 2011. doi:10.7155/jgaa.00245.

58 Jakub Tětek. Approximate Triangle Counting via Sampling and Fast Matrix Multiplication. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107:1–107:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.107.

59 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 786–797, Durham,NC,USA, 2020. IEEE. doi:10.1109/FOCS46700.2020.00078.

60 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998. doi:10.1038/30918.