On Asymptotic Cost of Triangle Listing in Random Graphs

Di Xiao, Yi Cui, Daren B.H. Cline, and Dmitri Loguinov
Texas A&M University, College Station, TX 77843 USA
{di, yicui, dmitri}@cse.tamu.edu, dcline@stat.tamu.edu

ABSTRACT

Triangle listing has been a long-standing problem, with many heuristics, bounds, and experimental results, but not much asymptotically accurate complexity analysis. To address this issue, we introduce a novel stochastic framework, based on Glivenko-Cantelli results for functions of order statistics, that allows modeling cost of in-memory triangle enumeration in families of random graphs. Unlike prior work that usually studies the $O(.)$ notation, we derive the exact limits of CPU complexity of all vertex/edge iterators under arbitrary acyclic orientations as graph size $n \to \infty$. These results are obtained in simple closed form as functions of the degree distribution. This allows us to establish optimal orientations for all studied algorithms, compare them to each other, and discover the best technique within each class.

1. INTRODUCTION

With an ever increasing flood of data, it is no longer sufficient just to process input correctly; instead, the underlying algorithms must exhibit scalability and high performance when operating on enormous datasets. This becomes quite evident in *graph mining*, which aims to discover interesting patterns within the connectivity network of participating agents. One specific problem that has recently gained attention [3], [14], [18], [22], [25] is enumeration of small subgraphs whose occurrence in nature is much more frequent than in classical random graphs [5], [19].

The most widely considered subgraphs are triangles, which have applications in numerous areas – databases, computer graphics, information retrieval, graph theory, algorithm complexity, and bioinformatics [4], [6], [7], [10], [16], [20], [24], [31], [37], [38], [40], [43]. Triangle listing has a four-decade history [23]; however, even for in-memory operation, this problem remains poorly understood, both mathematically and computationally. Open issues include how to accurately model the overhead, optimally arrange the nodes, select the best neighbor-traversal pattern, and minimize the runtime. Our goal in this paper is to shed light on these questions.

PODS'17, May 14-19, 2017, Chicago, IL, USA
© 2017 ACM. ISBN 978-1-4503-4198-1/17/05. . . $15.00
DOI: `http://dx.doi.org/10.1145/3034786.3034790`

1.1 Deterministic Graphs

Assume $G = (V, E)$ is a simple undirected graph with n nodes and m edges. Triangle listing involves a large number of comparisons during verification of edge existence. Expressing this complexity as a function of n and/or m has captivated the community for a long time [13], [28].

Exhaustively checking all 3-node subsets is the most obvious solution, but its $\approx n^3/6$ overhead is far from optimal unless G is complete. The first widely known algorithm with $O(m^{1.5})$ cost is [23]; however, it requires n^2 RAM to store the adjacency matrix, certainly an impossibility for large graphs. An improvement to $O(m)$ space is [13], which visits its nodes in descending order of degree and removes them from the graph. The best result about its CPU complexity is $O(\delta m)$, where δ is the *arboricity* of G, i.e., minimum number of edge-disjoint spanning forests into which G can be decomposed. Unfortunately, δ is an elusive quantity, only known to be $O(1)$ for trees and $O(\sqrt{m})$ otherwise.

In subsequent literature, two competing algorithms emerged – *vertex iterator* and *edge iterator* – which we review in the next section. Without enforcing order among the nodes in each triangle, these methods examine $\Theta(\sum_{i=1}^{n} d_i^2)$ candidate edges, where d_i is the degree of node i. Following [2], the majority of subsequent implementations run node/edge iterators after performing some type of acyclic orientation on G, which is a technique that assigns direction to each edge and ensures that no cycles are present in the resulting structure. Experimentally, it was observed that certain orientations reduced the runtime; however, their usage did not lead to any improvement to the asymptotic bound $O(\delta m) = O(m^{1.5})$ [2], [28].

Each acyclic orientation on G can be viewed as some permutation θ_n on V, where an undirected edge (i, j) becomes $i \to j$ if and only if $\theta_n(i) > \theta_n(j)$. After the shuffle, let the out-degree of the node in position i be $X_i(\theta_n)$ and its total degree be $d_i(\theta_n)$. Intuitively, orientations reduce overhead because they replace second moments of d_i with those of $X_i(\theta_n)$, $d_i(\theta_n) - X_i(\theta_n)$, or some combination thereof. Since the directed degree can be made significantly smaller, the resulting cost is orders of magnitude better.

In general, per-node complexity of triangle enumeration can be expressed using

$$c_n(\mathcal{M}, \theta_n) = \frac{1}{n} \sum_{i=1}^{n} f(X_i(\theta_n), d_i(\theta_n)), \qquad (1)$$

where f is some non-linear function that depends on the listing method $\mathcal{M}$. The main challenge with directly minimizing (1) is the likelihood that building such permutations is NP-hard. This intuition comes from the relationship between acyclic orientations and graph-coloring problems [26]. While optimization of (1) has not been considered before, there exist so-called *degenerate* orientations that minimize the largest out-degree, i.e., $\min_{\theta_n} \max_i X_i(\theta_n)$. This can be done in $O(m)$ time [2], [29]; however, compared to simpler strategies, the negligible improvement in cost often fails to justify the extra computation [32], [33].

As a result, attaining the lowest sum in (1) for a given deterministic graph may be a moot objective. Instead, more insight can be gleaned from the analysis of random graphs, which is our next topic.

1.2 Stochastic Graphs

Assume $F(x)$ is a CDF on integers in $[1,\infty)$, function $t_n \to \infty$ is monotonically increasing, and $F_n(x) = F(x)/F(t_n)$ is the original distribution truncated to the range $[1, t_n]$. Now suppose $\mathbf{D}_n = (D_{n1}, \dots, D_{nn})$ is an iid (independent and identically distributed) sequence drawn from $F_n(x)$, which we assume is graphic with probability $1 - o(1)$, or can be made such by removal of one edge. This allows construction of a random graph $G_n = (V_n, E_n)$ that realizes $\mathbf{D}_n$. Since (1) is now a random variable, performance analysis focuses on its expectation over all G_n, which we write as $E[c_n(\mathcal{M}, \theta_n)|\mathbf{D}_n]$.

Let $D_n \sim F_n(x)$ be a random variable with the same distribution as the degree in G_n. Early results [28] with Pareto $F(x) = 1 - (1 + \lfloor x \rfloor/\beta)^{-\alpha}$ typically bound the expected cost using a growing function of n. Recent work [9] obtains tighter bounds by focusing on a particular version of vertex iterator, which we call T_1, and using the descending-degree orientation θ_D. Assuming $t_n = \sqrt{n}$, they derive that $E[c_n(T_1, \theta_D)|\mathbf{D}_n]$ converges as $n \to \infty$ to

$$\frac{E[(Z_1^2 - Z_1)Z_2Z_3\mathbf{1}_{\min(Z_2,Z_3)>Z_1}]}{2E^2[D]}, \tag{2}$$

where $D, Z_1, Z_2, Z_3 \sim F(x)$ are iid variables and $\mathbf{1}_A$ is an indicator variable of event A. While useful in general, these results are limited to a single method T_1 and one specific permutation θ_D. Due to the complexity of derivation in [9], it is unclear if the same theoretical techniques can be extended into more general scenarios.

As the field stands today, it remains unknown which orientations are fundamentally better than others, under what conditions these advantages hold, whether orientations change the asymptotics of complexity or just the constants inside $O(.)$, and how the degree distribution impacts the overhead of alternative methods. We address these questions next.

1.3 Technical Results

To understand the overhead of triangle listing, our first contribution is to dissect the various ways that vertex and edge iterators can be executed in an acyclic digraph. We find that there are a total of 18 different neighbor-search patterns, but only four of them – which we call T_1, T_2, E_1, and E_4 – are non-isomorphic when considering CPU complexity. The former two are vertex iterators, while the latter are scanning edge iterators. We discuss how to efficiently implement them and outline a three-step algorithmic framework that does exactly that.

Conditioning on the degree sequence $\mathbf{D}_n$, our second contribution is to model $E[X_i(\theta_n)|\mathbf{D}_n]$ and introduce a family of asymptotically accurate approximations that admit simple analysis of cost. We show that under certain constraints on the maximum degree and for sufficiently large n all methods have expected overhead in the form of

$$E[c_n(\mathcal{M}, \theta_n)|\mathbf{D}_n] \approx \frac{1}{n}\sum_{i=1}^{n} g(d_i(\theta_n))h(q_i(\theta_n)), \tag{3}$$

where $g(x) = x^2 - x$, $q_i(\theta_n) = \sum_{j=1}^{i-1} d_j(\theta_n)/\sum_{k=1}^{n} d_k$ and h is a function that depends on $\mathcal{M}$.

Our third contribution is to analyze convergence of (3) using Glivenko-Cantelli results for functions of order statistics [39], [44] and obtain closed-form limits for all four methods under both ascending/descending-degree permutations. For example, vertex iterator T_1 under θ_D converges to

$$\lim_{n\to\infty} E[c_n(T_1, \theta_D)|\mathbf{D}_n] = \frac{E[g(D)(1 - J(D))^2]}{2}, \tag{4}$$

where $D \sim F(x)$ and $J(x) = 1/E[D]\int_0^x ydF(y)$ is the spread distribution from renewal theory. The result in (4) is finite iff $\alpha > 4/3$, which agrees with the conclusion of [9], [12]. While (2) captures the same limit, our derivation and result are both much simpler.

To understand the effect of θ_n, our fourth contribution is to develop a novel framework for modeling the limit of permutations. To this end, suppose $u, v \in [0, 1]$ are constants. We define sequence $\{\theta_n\}_{n=1}^{\infty}$ to be *admissible* if $P(\theta_n(\lceil un \rceil) < vn)$ converges to a continuous, measure-preserving probability kernel $K(v; u)$. Letting $\xi(u)$ be a (possibly degenerate) random variable with distribution $K(v; u)$, we show that (3) converges for any admissible sequence of permutations to

$$\lim_{n\to\infty} E[c_n(\mathcal{M}, \theta_n)|\mathbf{D}_n] = E[g(D)h(\xi(J(D)))]. \tag{5}$$

Armed with (5), our fifth contribution is to analyze optimality of θ_n for the algorithms under consideration. We establish that the expected cost of both T_1 and E_1 is minimized using the descending-degree permutation. Their optimal complexity is respectively (4) and

$$\lim_{n\to\infty} E[c_n(E_1, \theta_D)|\mathbf{D}_n] = \frac{E[g(D)(1 - J^2(D))]}{2}. \tag{6}$$

Method T_2 requires a new ordering we call *Round-Robin* (RR), which spreads large degree towards the outside of the range $[1, n]$. Its best cost is half of E_1's in (6). On the other hand, E_4 is optimized by the *Complementary Round-Robin* (CRR) permutation, which positions large degree towards the middle. If θ_n is optimal for a given $\mathcal{M}$, we also explain how to construct its opposite that achieves the worst cost.

Comparison across the four methods is our final contribution. We first prove that T_1 is faster than T_2 and E_1 is better than E_4 for all $F(x)$. However, the choice between T_1 and E_1 must take into account additional factors beyond cost – the speed of elementary comparison instructions on target hardware and in a specific graph G_n. This tradeoff arises because scanning edge iterators perform as many operations as both vertex iterators combined, i.e., $c_n(E_1, \theta_n) = c_n(T_1, \theta_n) + c_n(T_2, \theta_n)$, but at significantly higher speed [17]. The decision is simple only for $\alpha \in (4/3, 1.5]$ and $n \to \infty$, where T_1 *always* beats E_1 in the limit (i.e., the former has finite cost, while the latter has infinite). Although it is commonly believed that vertex and edge iterators have the same asymptotic complexity [28], [33], we prove that this is not true for $\alpha \in (4/3, 1.5]$ and scanning edge iterators.

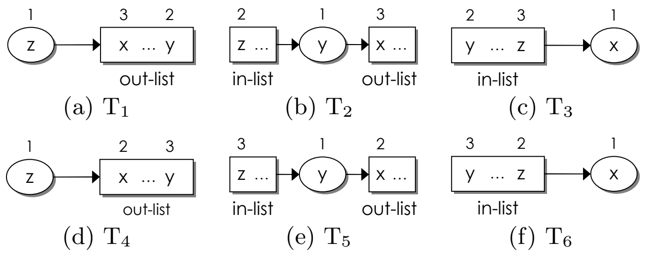

Figure 1: Search order in vertex iterator.

We conclude with developing an algorithm for efficient computation of our models, examining their accuracy under finite n, and highlighting the impact of Pareto α on the scaling behavior of cost.

2. UNIFYING FRAMEWORK

In this section, we condition on G and treat it as deterministic. We start by generalizing all previous work under a novel umbrella of 18 baseline algorithms and discussing how to extract the highest performance out of them.

Throughout the paper, we assume that adjacency lists in graphs are sorted ascending by node ID, which corresponds to left-to-right orientation in the figures. The omitted proofs can be found in [42].

2.1 Preprocessing

Each triangle $\triangle_{xyz}$ can be listed in $3! = 6$ equivalent ways by permuting its node sequence (x, y, z). To avoid this redundancy and improve efficiency, it is common [3], [22], [25], [34] to consider only triangles that satisfy some global transitive order $\mathcal{O}$, which results in exactly one listing per actual triangle. Three most-commonly used orders are random (e.g., hash-based) [14], [23], descending-degree [3], [22], [28], [33], and ascending-degree [34]. We propose that such techniques be implemented by a three-step process – 1) sort the nodes by $\mathcal{O}$ and sequentially assign IDs from sequence $(1, 2, \ldots, n)$; 2) create a directed graph in which the out-neighbors of each y have smaller labels and in-neighbors have larger, which is denoted by $y \to x$ and $y \leftarrow z$, respectively; and 3) in the directed graph, list triangles in ascending order of node ID, i.e., $x < y < z$.

The first step, called *relabeling* [28], [34], requires sorting n items using $\mathcal{O}$ and rewriting the labels of all edges in G. The second step, known as *orientation* [22], [33], splits each list of undirected neighbors $N(y)$ into in/out sets $N^-(y)$ and $N^+(y)$, respectively. These two steps can be modeled by a permutation $\theta_n : V \to V$ that always starts with ascending-degree order and maps each node in position i to a label $\theta_n(i)$. After re-writing the IDs, i's out-list consists of original neighbors j whose permuted label precedes that of i, i.e., $\theta_n(j) < \theta_n(i)$. In the directed graph $G(\theta_n) = (V, E(\theta_n))$, suppose i has out-degree $X_i(\theta_n)$, in-degree $Y_i(\theta_n)$, and total degree $d_i(\theta_n) = X_i(\theta_n) + Y_i(\theta_n)$.

While prior work is mostly concerned with the third step, i.e., triangle listing, we begin by noting that deciding on θ_n is an important factor for runtime, despite not having been studied in much detail before. With $n!$ possible permutations, it is unclear which of them produces the smallest overhead, how to approach modeling complexity of triangle listing in a relabeled/oriented graph, and which of the methods are better under various circumstances. The rest of the section lays a foundation for addressing these issues.

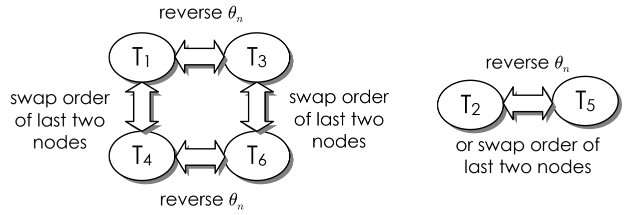
reverse θ_n
T$_1$
T$_3$
swap order of last two nodes
swap order of last two nodes
T$_4$
T$_6$
reverse θ_n
reverse θ_n
T$_2$
T$_5$
or swap order of last two nodes

Figure 2: Vertex-iterator equivalence classes.

2.2 Vertex Iterator

Vertex iterator is a classical algorithm [18], [22], [33], [36] that visits each node and verifies edge existence between all pairs of non-redundant neighbors. For an oriented graph $G(\theta_n)$, vertex iterator admits six distinct search patterns, each specifying a different order in which the nodes (x, y, z) are traversed.

In the first pattern, which we call T$_1$, the algorithm considers the first node to be the largest in the triangle. This is illustrated in Figure 1(a), where the search starts from z, continues to $y < z$, and finishes with $x < y$. This method generates candidate edges $y \to x$, which are checked against $E(\theta_n)$ using a hash table. Each match indicates a unique triangle $\triangle_{xyz}$. Assuming the overhead is measured in the volume of tuples (y, x), the per-node cost of T$_1$ is

$$c_n(T_1, \theta_n) = \frac{1}{n} \sum_{i=1}^{n} \frac{X_i(\theta_n)(X_i(\theta_n) - 1)}{2}. \tag{7}$$

The second search order, which we call T$_2$, treats the initial node as being neither the smallest, nor the largest in the triangle. The corresponding technique, shown in Figure 1(b), verifies existence of edges $z \to x$, where pairs (z, x) sweep all possible combinations of y's in and out neighbors. The number of such tuples is given by

$$c_n(T_2, \theta_n) = \frac{1}{n} \sum_{i=1}^{n} X_i(\theta_n) Y_i(\theta_n). \tag{8}$$

As sketched in Figure 1(c), method T$_3$ starts from the smallest node x and examines presence of edges $z \to y$ between all pairs of in-neighbors such that $y < z$. Its overhead is similar to that of T$_1$, but with $X_i(\theta_n)$ and $Y_i(\theta_n)$ trading places, i.e.,

$$c_n(T_3, \theta_n) = \frac{1}{n} \sum_{i=1}^{n} \frac{Y_i(\theta_n)(Y_i(\theta_n) - 1)}{2}. \tag{9}$$

The remaining three methods in the figure (i.e., T$_4$-T$_6$) change the order in which the last two neighbors are visited; however, their cost formulas are identical to those of the counter-parts above them. Symmetry of T$_1$ and T$_3$ suggests even wider *equivalence classes*, which are groups of methods that have the same complexity and speed of elementary operations. Let the *reverse permutation* $\theta'_n : V \to V$ be such that $\theta'_n(i) = n + 1 - \theta_n(i)$ and consider the following.

PROPOSITION 1. *Reversing the permutation results in the overhead function swapping $X_i(\theta_n)$ with $Y_i(\theta_n)$.*

Since $c_n(T_1, \theta_n) = c_n(T_3, \theta'_n)$, we conclude that T$_1$ and T$_3$ are indeed equivalent. In addition, T$_2$ and T$_5$ reverse

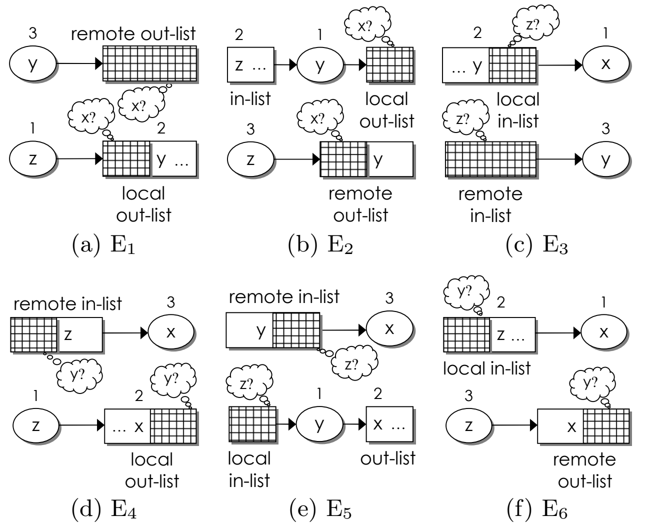

Figure 3: Search order in SEI.

	E$_1$	E$_2$	E$_3$	E$_4$	E$_5$	E$_6$
Local cost	T_1	T_2	T_3	T_1	T_2	T_3
Remote cost	T_2	T_1	T_2	T_3	T_3	T_1

Table 1: Intersection complexity of SEI.

into each other and thus form a standalone equivalence class. This is schematically shown in Figure 2. From this point on, we dismiss T$_3$-T$_6$ and keep only T$_1$-T$_2$ under consideration.

2.3 Edge Iterator

Instead of generating candidate pairs and checking them later, it is possible to traverse each directed edge and intersect the sets of neighbors at both incident vertices [14], [28], [33]. A *scanning edge iterator* (SEI) sequentially rolls through both neighbor lists, performing comparison using two pointers. An alternative, which we call *lookup-based edge iterator* (LEI), intersects using hash tables.

We start analysis with SEI, which also admits six search orders in the oriented graph. Using Figure 3(a), algorithm E$_1$ visits node z, examines each out-neighbor y, and intersects $N^+(y)$ with $N^+(z)$ to discover nodes x that complete triangle $\triangle_{xyz}$. Note that intersection at z does not involve all of its out-neighbors; only those smaller than y (shown with a grid pattern). This is a consequence of the $x < y < z$ relationship and transitivity of $\mathcal{O}$.

The cost of SEI is based on the number of comparisons in list intersection. We split this overhead into *local*, which takes into account scans over z's out-neighbor list $N^+(z)$, and *remote*, which does the same for y.

Proposition 2. *The CPU cost of E$_1$ is given by $c_n(E_1, \theta_n) = c_n(T_1, \theta_n) + c_n(T_2, \theta_n)$, where the former term is local overhead and the latter is remote.*

The remaining SEI methods E$_2$-E$_6$ in Figure 3 are self-explanatory. Their complexity is shown in Table 1, which covers all ordered ways to choose local and remote overhead from the first three vertex-iterator options. It should be noted that LEI has the same six algorithms, which we call L$_1$-L$_6$, except that it hashes the local neighbor list of the first visited node and performs lookups against it for all remote nodes [17]. Complexity of populating the hash tables is $\sum_{i=1}^n X_i(\theta_n) = \sum_{i=1}^n Y_i(\theta_n) = m$, while the lookup overhead is given by Table 2 (which is also the second row of Table 1). Since LEI can be reduced to vertex iterator (in terms of both operation speed and cost), there is no need to consider it separately in the remainder of the paper.

	L$_1$	L$_2$	L$_3$	L$_4$	L$_5$	L$_6$
Cost	T_2	T_1	T_2	T_3	T_3	T_1

Table 2: Lookup complexity of LEI.

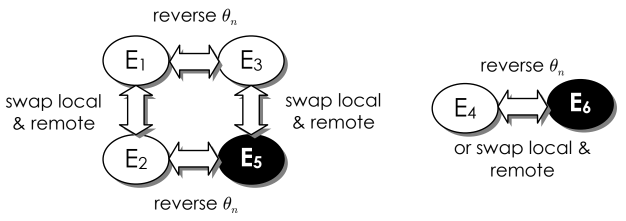

Figure 4: SEI equivalence classes.

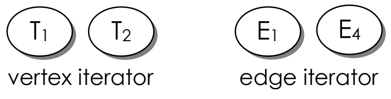

Figure 5: Competing algorithms.

Complexity of scanning edge iterator combines two terms, where the equivalence classes are built either by reversing the permutation or swapping local/remote overhead. The result is shown in Figure 4, where we again find just two distinct families of algorithms. However, SEI requires more careful pruning than vertex iterator since additional factors must be taken into account, which we discuss next.

For E$_5$, notice that the starting location of the intersection (i.e., y) is buried in the middle of x's in-neighbor list. Therefore, compared to E$_1$-E$_3$, this method requires either an extra binary search through $N^-(x)$ or intersection backwards (on the Intel i7-2600K, a backwards search is 26% slower than forward, which may be explained by poor CPU prefetch and/or less efficient auto-vectorization in the compiler). For similar reasons, E$_6$ is not competitive against E$_4$. Next, reversal of θ_n has no impact on runtime, which means that E$_1$ and E$_3$ are identical in all aspects. However, deciding between E$_1$ and E$_2$ requires modeling I/O complexity under a specific graph-partitioning scheme, which is a topic for another paper [17]. For now, we drop E$_2$-E$_3$ and obtain the four fundamental techniques in Figure 5.

2.4 Discussion

We now translate prior techniques into our notation. The oldest method for which this mapping is possible is [13], which proposes a variation of L$_3$ where acyclic orientation holds only for two of the three edges in each triangle. As a result, its complexity is $c_n(E_1, \theta_n)$ rather than $c_n(T_2, \theta_n)$. Incomplete orientation is remedied in [33] using a dynamically growing set of vectors to implement E$_2$ under the name *Forward*. This is extended to *Compact Forward* in [28] by removing the vector constraint. In more recent studies, T$_1$ appears in [22], [25], [36], E$_1$ in [3], [21], [35], E$_3$ in [14], and E$_5$ in [34]. At least six methods, i.e., T$_1$-T$_3$, E$_1$, E$_3$, E$_4$ are identified in [32].

Family of algorithms	Operations	Speed
Vertex iterator	Hash table	19
Lookup edge iterator (LEI)	Hash table	19
Scanning edge iterator (SEI)	SIMD intersection	1,801

Table 3: Single-core speed (million nodes/sec) using an Intel i7-3930K @ 4.4 GHz.

The majority of literature [3], [21], [22], [25], [33], [35], [36] omits relabeling and performs only orientation. Consequently, nodes in each of the directed neighbor lists are not ordered in any particular way *against each other*, which doubles the cost of all terms that depends on T_1 or T_3. For example, T_1 must check all pairs $x, y \in N^+(z)$ instead of only those with $x < y$. Similarly, scanning of the local list in E_1 cannot stop at y and must traverse the entire $N^+(z)$. On the other hand, T_2 takes advantage of preprocessing that separates in/out-neighbors into different sets, which allows it to keep complexity unchanged.

A limited number of studies [28], [33], [34] utilize relabeling; however, they omit orientation. In this setting, some of the methods require a binary search in remote neighbor lists to locate the boundary between the candidates larger and smaller than the current node. Vertex iterators T_1 and T_3 are not impacted, but T_2 must perform an additional $\zeta = \sum_{i=1}^n \log_2 d_i$ random memory accesses. Scanning edge iterators E_1/E_2 use the same ζ extra jumps, while E_3/E_5 and E_4/E_6 take a larger performance hit – one binary search per edge. This adds up to $\sum_{i=1}^n X_i \log_2 d_i$ or $\sum_{i=1}^n Y_i \log_2 d_i$ unnecessary memory lookups depending on the method. Using backwards-sorted lists, this overhead can be reduced to ζ for E_3/E_5, but not for E_4/E_6. Besides slowing down the computation, binary search may be impossible altogether in certain graphs (e.g., with compressed neighbor lists).

Our three-step framework ensures the smallest cost within each class of techniques, while imposing no restrictions on adjacency lists. The 18 introduced algorithms are converted to pseudocode and benchmarked in [17]. Table 3 summarizes these results using neighbor lists of sufficiently large size (i.e., best-case scenario for intersection). Even though SEI executes up to 95 times faster *per node*, Table 1 show that it requires more operations. To understand this tradeoff, define w_n to be the ratio of the lowest cost in SEI to that in the other two families of algorithms. It then follows that SEI, assuming it can match the speed in Table 3 over the adjacency lists of $G_n(\theta_n)$, has a better runtime on modern Intel CPUs iff $w_n < 95$. Since both instruction speed and w_n are functions of $G_n(\theta_n)$, this decision cannot be made unless the specific graph (or at least its degree distribution) is known. The only exception is $n \to \infty$ and graphs with $w_n \to \infty$, where SEI is always slower in the limit. We identify and discuss these cases later in the paper.

3. MODELING OUT-DEGREE

We now examine how to model the out-degree $X_i(\theta_n)$ in a family of random graphs $G_n(\theta_n)$.

3.1 Degree Growth

Recall that $F(x)$ is some fixed CDF, $t_n \to \infty$ is a monotonic function, and $F_n(x) = F(x)/F(t_n)$ is a truncated distribution. Suppose $D \sim F(x)$ represents the limiting degree and let $\mathbf{D}_n = (D_{n1}, \ldots, D_{nn})$ be an iid degree sequence drawn from $F_n(x)$. Note that $t_n \le n-1$ is required for $\mathbf{D}_n$ to be graphic (i.e., realizable by a graph). For the results that follow, it is convenient to sort $\mathbf{D}_n$ in ascending order, produce a new sequence $\mathbf{A}_n = (A_{n1}, \ldots, A_{nn})$, where $A_{ni} \le A_{n,i+1}$, and apply permutation to the result. Then, our earlier notation $d_i(\theta_n)$ refers to A_{nj} such that $\theta_n(j) = i$.

As we show below, the cost of triangle listing depends on one crucial parameter – the probability of edge existence between each pair of nodes (i, j) in G_n. We assume a traditional random-graph model that realizes a given degree sequence [8], [30], in which this metric is proportional to the product of corresponding degrees [1], [15]

$$p_{ij}(\theta_n) \approx \frac{d_i(\theta_n) d_j(\theta_n)}{2m}. \tag{10}$$

For this to be a probability, the numerator must not exceed the denominator, which we formalize next.

Definition 1. *Suppose $L_n = \max_i\{D_{ni}\}$ is the largest degree in G_n. A sequence of distributions $\{F_n(x)\}$ is called* asymptotically max-root-constrained (AMRC) *if $P(L_n > \sqrt{n}) \to 0$ as $n \to \infty$.*

AMRC sequences ensure that (10) is accurate for sufficiently large n. One option for satisfying this condition is to use distributions $F(x)$ with finite variance (i.e., $E[D^2] < \infty$), which is a consequence of the following result.

Proposition 3. *For $n \to \infty$ and some constant $c > 0$, $P(L_n > n^c) \to 0$ if $E[D^{1/c}] < \infty$.*

The second option is to scale t_n slowly enough, but without placing any restrictions on $F(x)$. To this end, define truncation to be *linear* if $t_n = n - 1$ and *root* if $t_n = \sqrt{n}$. Now observe that the latter case deterministically yields $L_n \le \sqrt{n}$, which keeps $p_{ij}(\theta_n) \le 1$ for all n.

If Definition 1 fails to hold, we call sequence $\{F_n(x)\}$ *unconstrained*. Unfortunately, the probability of edge existence in G_n built by such degree distributions is extremely difficult (if not impossible) to obtain in closed-form due to the high levels of dependency in the edge-construction process. At this point, it is even unclear if useful approximations to $p_{ij}(\theta_n)$ can be made for such cases. However, depending on the objectives, this may not be required.

Assume $\mathcal{M}$ is a particular triangle-listing method. Throughout the paper, we develop two sets of results – those that rely on $F_n(x)$ to establish $E[c_n(\mathcal{M}, \theta_n)|\mathbf{D}_n]$ and those that use $F(x)$ to arrive at the corresponding limit as $n \to \infty$. The former models are accurate for finite (but sufficiently large) n if the degree sequence is AMRC. In contrast, the asymptotic limits hold *unconditionally* because the rate at which $F_n(x)$ approaches $F(x)$ has no impact on the convergence point, i.e., linear and root truncation yield the same result. Therefore, all models in the form of (4)-(6) are exact even in unconstrained graphs.

3.2 Expected Degree and Cost

Notice that the expected out-degree at node i is the summation of $p_{ij}(\theta_n)$ for all nodes j smaller than i. Excluding self-loops in the denominator, this leads to

$$E[X_i(\theta_n)|\mathbf{D}_n] \approx d_i(\theta_n) \frac{\sum_{j=1}^{i-1} d_j(\theta_n)}{2m - d_i(\theta_n)}. \tag{11}$$

While (11) is asymptotically precise, it may exhibit large errors for finite n in unconstrained graphs. This occurs

T_1	T_2	E_1	E_4
$\frac{x^2}{2}$	$x(1-x)$	$\frac{x(2-x)}{2}$	$\frac{x^2+(1-x)^2}{2}$

Table 4: Function $h(x)$.

because it over-estimates the number of edges delivered to high-degree nodes, in essence treating G_n as allowing duplicate links. To curb this tendency, we propose to extend (11) by applying some positive and monotonically non-decreasing function $w(x)$ to the degree of potential neighbors, i.e.,

$$E[X_i(\theta_n)|\mathbf{D}_n] \approx d_i(\theta_n)\frac{\sum_{j=1}^{i-1} w(d_j(\theta_n))}{\sum_{k=1}^{n} w(d_k) - w(d_i(\theta_n))}. \quad (12)$$

For example, $w(x) = \min(x, a)$, where a is a constant, is one such option we consider below. By tuning $w(x)$ to suit $F(x)$, it may be possible to make (12) accurate in unconstrained graphs under finite n; however, discovery of this relationship is not necessary for proving optimality of different permutations/methods since our results below cover a wide range of $w(x)$.

Define the fraction of i's neighbors with smaller ID as

$$q_i(\theta_n) = \frac{E[X_i(\theta_n)|\mathbf{D}_n]}{d_i(\theta_n)} \quad (13)$$

and consider the next result.

Proposition 4. *In asymptotically large AMRC graphs, all four triangle-listing techniques in Figure 5 are covered by one formula*

$$E[c_n(\mathcal{M}, \theta_n)|\mathbf{D}_n] \approx \frac{1}{n}\sum_{i=1}^{n} g(d_i(\theta_n))h(q_i(\theta_n)), \quad (14)$$

where $g(x) = x^2 - x$ and $h(x)$ is given by Table 4.

4. CONVERGENCE OF COST

We next examine the limit of (14) as $n \to \infty$. Note that we use Lebesgue-Stieltjes integrals and treat CDFs as measures, i.e., $dF(x)$ applies to both discrete and continuous distributions.

4.1 Functions of Order Statistics

For now, assume $\theta_n(i) = i$ is the ascending-degree permutation, which means that $d_i(\theta_n) = A_{ni}$. Let $\{\phi_n(t)\}_{n=1}^{\infty}$ be a sequence of functions that for all $t \in [0, 1]$ satisfies

$$\lim_{n\to\infty} \int_0^t \phi_n(u)du = \int_0^t \phi(u)du, \quad (15)$$

where $\phi(u)$ is some integrable function. Then, given a sufficiently smooth $g(x)$, results in the field of L-estimators [39], [44] show that

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} g(A_{ni})\phi_n(i/n) = \int_0^1 g(F^{-1}(u))\phi(u)du. \quad (16)$$

Limits in the form of (16) are known as Glivenko-Cantelli results for functions of order statistics. Letting U be uniform in $[0, 1]$, the integral in (16) can be written shorter as $E[g(F^{-1}(U))\phi(U)]$. When $F(x)$ is continuous, it is often convenient to represent the expectation as $E[g(D)\phi(F(D))]$, where $D \sim F(x)$.

Our first result extends (16) to cover partial sums.

Lemma 1. *For a fixed $u \in [0, 1]$,*

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{\lceil nu\rceil} g(A_{ni}) = \int_0^u g(F^{-1}(x))dx. \quad (17)$$

To allow (16) to handle summations in the form of (14), it is important to understand the asymptotic properties of $q_i(\theta_n)$. The following result sheds light on this issue.

Lemma 2. *For the ascending permutation θ_A and fixed $u \in [0, 1]$, random variable $q_{\lceil nu\rceil}(\theta_A)$ converges to $J(F^{-1}(u))$ as $n \to \infty$, where*

$$J(x) := \frac{1}{E[w(D)]}\int_0^x w(y)dF(y). \quad (18)$$

For $E[w(D)] < \infty$, (18) defines a CDF. We study its properties next.

Proposition 5. *Assume a process that picks node i in G_n in proportion to $w(D_{ni})$ and let S_n be the random degree of the chosen nodes. Then, $P(S_n \le x) \to J(x)$ as $n \to \infty$.*

Variable $S \sim J(x)$ is known as *spread* in renewal process theory. Given n intervals of size $w(d_1), \ldots, w(d_n)$, suppose a random point is thrown into $[0, \sum_{k=1}^{n} w(d_k)]$. Then, the length of the interval that the point hits follows the spread distribution as $n \to \infty$. The selected intervals are biased towards larger $w(d_i)$, which is known as the *inspection paradox* [41]. In graphs $G_n(\theta_n)$, the bias reflects the probability to select neighbors in proportion to their weight $w(d_i)$. For $w(x) = x$, variable S represents the degree of nodes adjacent to a random link, or that seen by a random walk on the graph. In such cases, spread $J(x)$ is well-studied, e.g., exponential D produces $S \sim$ Erlang(2) and Pareto(α, β) yields

$$J(x) = 1 - \frac{\beta + \alpha x}{\beta}\Big(1 + \frac{x}{\beta}\Big)^{-\alpha}, \quad (19)$$

which has Pareto-like tails with a heavier shape $\alpha - 1$.

4.2 Monotonic Permutations

We are now ready to derive the expected cost of triangle listing for the ascending/descending permutations.

Theorem 1. *The ascending permutation produces*

$$\lim_{n\to\infty} E[c_n(\mathcal{M}, \theta_A)|\mathbf{D}_n] = E[g(D)h(J(D))]. \quad (20)$$

This means that random variable (14) becomes independent of $\mathbf{D}_n$ as $n \to \infty$, which allows us to treat it as a constant for sufficiently large n. The descending case is handled similarly since $q_i(\theta_D) = 1 - q_i(\theta_A)$. This implies $q_{\lceil nu\rceil}(\theta_D)$ converges to $1 - J(F^{-1}(u))$ and consequently

$$\lim_{n\to\infty} E[c_n(\mathcal{M}, \theta_D)|\mathbf{D}_n] = E[g(D)h(1 - J(D))]. \quad (21)$$

Applying these observations and recalling Table 4,

$$\lim_{n\to\infty} E[c_n(T_1, \theta_A)|\mathbf{D}_n] = \frac{E[g(D)J^2(D)]}{2}, \quad (22)$$

$$\lim_{n\to\infty} E[c_n(T_1, \theta_D)|\mathbf{D}_n] = \frac{E[g(D)(1 - J(D))^2]}{2}. \quad (23)$$

For T_2, symmetry $h(1 - x) = h(x)$ shows that both permutations have the same limiting cost. It is thus sufficient to consider only the descending case

$$\lim_{n\to\infty} E[c_n(T_2, \theta_D)|\mathbf{D}_n] = E[g(D)J(D)(1 - J(D))]. \quad (24)$$

Interestingly, T_1 under θ_A is finite iff $\alpha > 2$, which is shown by expanding the integral in (22) using $1 - J(x) \sim x^{1-\alpha}$. Thus, θ_A offers no finiteness benefits over omitting orientation, where the cost is $E[D^2 - D]/2$. Next, T_2 in (24) is finite iff $\alpha > 1.5$, which is a noticeable improvement over (22). But this is eclipsed by T_1 under θ_D, where the finiteness condition is $\alpha > 4/3$. Therefore, as $n \to \infty$, vertex iterator exhibits at least four regimes of operation, i.e., $\alpha \leq 4/3$, $\alpha \in (4/3, 1.5]$, $\alpha \in (1.5, 2]$, and $\alpha > 2$.

Note that (20) and (21) require no constraints on $\{F_n(x)\}$. For finite n, however, accurate results are guaranteed only for AMRC sequences. In these cases, Theorem 1 yields the following approximation to the expected cost across all degree sequences drawn from $F_n(x)$

$$E[c_n(\mathcal{M}, \theta_A)] \approx E[g(D_n)h(J_n(D_n))], \tag{25}$$

where spread $J_n(x)$ is computed from the truncated distribution $F_n(x) = F(x)/F(t_n)$ and $D_n \sim F_n(x)$.

5. CONVERGENCE OF PERMUTATIONS

Our initial investigation into cost of triangle listing handles only the simplest permutations. The next goal is to understand what makes sequences $\{\theta_n\}_{n=1}^{\infty}$ convergent and propose a framework for modeling their limits.

5.1 Admissibility

We start with a background on measures.

Definition 2. *For a set S and its σ-algebra Σ, function $\mu : \Sigma \to \mathbb{R}$ is called a* measure *if a) $\mu(C) \geq 0$ for all elements $C \in \Sigma$; b) $\mu(\emptyset) = 0$; c) $\mu(\cup_{i=1}^{\infty} C_i) = \sum_{i=1}^{\infty} \mu(C_i)$ for pairwise disjoint $C_i \in \Sigma$.*

It is sufficient for us to consider just two cases of S. For finite sets, the σ-algebra consists of all subsets of S, in which case $\mu(C) = |C|$. For continuum S, we use the Lebesgue measure $\mu([a, b]) = b - a$.

Definition 3. *For a set S, function $K(v; u) : \mathbb{R} \times S \to \mathbb{R}$ is called a* probability kernel *if for each $u \in S$ it is a CDF in variable v, i.e., non-decreasing, defined for all $v \in \mathbb{R}$, and compliant with $K(-\infty; u) = 0$, and $K(\infty; u) = 1$.*

Kernels are useful mechanisms for capturing the distribution of non-iid collections of random variables; however, only some of them will be suitable for our purposes.

Definition 4. *Let S be a set with measure μ and U be a uniformly random variable in S. Kernel $K(v; u)$ is called* measure-preserving *if it satisfies for all $v \in \mathbb{R}$*

$$E[K(v; U)] = \frac{\mu(\{s \in S : s \leq v\})}{\mu(S)}, \tag{26}$$

in which case the corresponding variable $\xi(u) \sim K(v; u)$ is a random map*, i.e., $P(\xi(u) \leq v) = K(v; u)$.*

Note that $\xi(u)$ can be viewed as a random bijection on S. As before, sequence of functions $\theta_n : [1, n] \to [1, n]$ specifies a relationship between the position in the ascending-order vector $\mathbf{A}_n$ and that in the permuted sequence. If these functions are deterministic, $i = \theta_n(j)$ means that $d_i(\theta_n) = A_{nj}$. Since each θ_n is a bijection, its inverse exists and satisfies $\theta_n^{-1}(\theta_n(j)) = j$. For more general cases, where specifying a deterministic relationship is inconvenient, θ_n may be random. Either way, there must exist a measure-preserving kernel $M_n(j; i)$ such that $P(\theta_n(i) \leq j) = M_n(j; i)$. However, to have a reasonable mapping as $n \to \infty$, we need to impose certain asymptotic constraints.

Definition 5. *Suppose sequence $k(n) \to \infty$ is such that $k(n)/n \to 0$ as $n \to \infty$. If for all $u, v \in [0, 1]$ the fraction of values in the $k(n)$-neighborhood of u that are mapped to the interval $[0, v]$, i.e.,*

$$K_n(v; u) := \frac{1}{2k(n) + 1} \sum_{i=-k(n)}^{k(n)} M_n(nv; \lceil nu \rceil + i), \tag{27}$$

converges weakly to some limit $K(v; u)$, sequence $\{\theta_n\}$ is called admissible.

Outside of specially crafted counter-examples (e.g., $\theta_n = \theta_A$ for odd n and θ_D for even), most reasonable permutation sequences are admissible. For such cases, the limiting behavior of θ_n is a random process $\{\xi(u)\}$ with distribution $K(v; u) = P(\xi(u) \leq v)$. Note that if (27) converges, the limit must be measure-preserving. The opposite is true as well – any such kernel has some sequence of permutations $\{\theta_n\}$ that converges to it.

5.2 Cost Under General Permutations

We start by generalizing (16) beyond monotonic θ_n.

Lemma 3. *For an admissible sequence $\{\theta_n\}$,*

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} g(d_i(\theta_n))h(i/n) = E[g(F^{-1}(U))h(\xi(U))]. \tag{28}$$

Re-writing (28) as $E[g(D)h(\xi(F(D)))]$ and using the same logic as in Theorem 1, we get the following.

Theorem 2. *For an admissible sequence $\{\theta_n\}$,*

$$\begin{aligned} c(\mathcal{M}, \xi) &:= \lim_{n \to \infty} E[c_n(\mathcal{M}, \theta_n) | \mathbf{D}_n] \\ &= E[g(D)h(\xi(J(D)))]. \end{aligned} \tag{29}$$

For AMRC graphs, we can use the limiting map $\xi(u)$ to achieve accurate cost analysis even under finite n

$$E[c_n(\mathcal{M}, \theta_n)] \approx E[g(D_n)h(\xi(J_n(D_n)))]. \tag{30}$$

Not surprisingly, the ascending map $\xi(u) = u$ and descending $\xi(u) = 1 - u$ produce in (29) the already-established (22)-(24). Additional cases are covered next.

5.3 Non-Monotonic Permutations

Besides θ_A and θ_D, another previously used option is *uniform* [14], where the direction of edges in the acyclic orientation is based on original (e.g., hashed) node IDs. In this case, $\xi_U(u)$ is a uniform variable in $[0, 1]$, independent of the starting position u. This yields

$$c(\mathcal{M}, \xi_U) = E[g(D)]E[h(U)] = E[D^2 - D]E[h(U)], \tag{31}$$

where U is uniform in $[0, 1]$. Simple calculations show that $E[h(U)] = 1/6$ for both vertex iterators and $1/3$ for both edge iterators. Compared to not performing orientation at all, where the corresponding cost is $E[D^2 - D]/2$ and $E[D^2 - D]$, both families of algorithms reduce complexity by a factor of 3. This agrees with common sense since orientation avoids counting each triangle three times.

Analysis of T$_2$'s function $h(x) = x(1-x)$ suggests that larger values of degree should be scattered towards the outside of the range $[1, n]$ in an effort to pair them with smaller products $q_i(\theta_n)(1-q_i(\theta_n))$ in (14). This leads to a new permutation we call *Round-Robin* (RR)

$$\theta_n(i) = \begin{cases} \lceil \frac{n+i}{2} \rceil & i \text{ is odd} \\ \lfloor \frac{n-i}{2} \rfloor + 1 & i \text{ is even} \end{cases}, \quad (32)$$

whose map is derived next.

Proposition 6. *Permutation* (32) *converges to a random map $\xi_{RR}(u)$ that equals $(1-u)/2$ or $(1+u)/2$, each with probability 1/2.*

Re-writing (29), we have a general result

$$c(\mathcal{M}, \xi_{RR}) = \frac{E\Big[g(D)\Big(h\big(\frac{1-J(D)}{2}\big) + h\big(\frac{1+J(D)}{2}\big)\Big)\Big]}{2}. \quad (33)$$

Applying this to T$_2$ yields

$$c(T_2, \xi_{RR}) = \frac{E[g(D)(1-J^2(D))]}{4}, \quad (34)$$

which is finite iff $\alpha > 1.5$. Since the complexity of E$_1$ combines that of T$_1$ and T$_2$, it is interesting whether it performs better under the descending or RR permutation. Full comparison is delayed until the next section, but Proposition 6 can be used to reveal the likely outcome. Combining (23) and (24) leads to

$$c(E_1, \xi_D) = \frac{E[g(D)(1-J^2(D))]}{2} \quad (35)$$

and expanding (33) using $h(x) = x(2-x)/2$ produces

$$c(E_1, \xi_{RR}) = \frac{E[g(D)(3-J^2(D))]}{8}. \quad (36)$$

While (35) is finite iff $\alpha > 1.5$, this cannot be said about (36), where the condition shifts to $\alpha > 2$. This can be explained by the fact that T$_1$ suffers a significant cost increase under RR, which more than offsets the benefits that T$_2$ may be gaining.

For E$_4$, function $h(x)$ behaves the opposite of that for T$_2$, i.e., smaller values are found towards the center. To handle such cases, suppose the *complementary permutation* $\theta''_n(i) = \theta_n(n-i+1)$ applies the same mapping as θ_n, but starting from the descending order of degree rather than ascending. With this in mind, it might be interesting to examine E$_4$ under the *Complementary Round-Robin* (CRR) permutation. To determine the corresponding map, consider the following.

Proposition 7. *Suppose sequence $\{\theta_n\}$ converges to some map $\xi(u)$ as $n \to \infty$. Then, its reverse $\{\theta'_n\}$ converges to $\xi'(u) = 1-\xi(u)$ and its complement $\{\theta''_n\}$ to $\xi''(u) = \xi(1-u)$.*

This leads to $\xi_{CRR}(u) = \xi''_{RR}(u) = \xi_{RR}(1-u)$, which equals $u/2$ or $1-u/2$ with probability 1/2 each. Expanding (29), it can be shown that CRR coupled with any of the considered methods has finite cost iff $\alpha > 2$.

In summary, we have five different permutations (i.e., ascending, descending, RR, CRR, and uniform) that cover cases where $\theta_n(i)$ and $\xi(u)$ are both deterministic, both random, and one is deterministic but the other is random. Our next task is create a framework for minimizing (29) over all measure-preserving maps $\{\xi(u)\}$.

Algorithm 1: Construction of optimal permutations

```
Function OPT (h)
    for (i = 1; i ≤ n; i++) do
        z[i].key = h(i/n)
        z[i].index = i
    if r(x) is an increasing function then
        sort array z descending by key
    else
        sort array z ascending by key
    for (i = 1; i ≤ n; i++) do
        theta[i] = z[i].index
    return theta
```

6. COMPARISON AND OPTIMALITY

This section obtains the optimal permutation and cost under various monotonic relationships among $g(x)$, $h(x)$, and $w(x)$. It also compares the best methods within each triangle-listing class.

6.1 Minimizing Cost

Assume that $J(x)$ is a continuous spread distribution from (18). This implies that $J(S)$, where $S \sim J(x)$, is a uniform variable in $[0, 1]$. For heavy-tailed $F(x)$, such as Pareto, this assumption holds iff $E[w(D)] < \infty$. For the results below, it is convenient to define $r(x) = g(J^{-1}(x))/w(J^{-1}(x))$. We will be mostly concerned with its monotonicity, which is the same as that of $g(x)/w(x)$ since $J(x)$ is a CDF.

Lemma 4. *Model* (29) *can be written as*

$$c(\mathcal{M}, \xi) = E[w(D)]E[r(U)h(\xi(U))], \quad (37)$$

where U is uniform in [0,1].

Note that (37) is better-suited for our purposes in this section because it replaces a combination of D and $J(D)$ with a simpler variable U. The next result shows that there exists a percolation point in the behavior of $r(x)$ at which all permutations are equal.

Proposition 8. *For a constant function $r(x) = b$ and a fixed method $\mathcal{M}$, all permutations yield the same complexity*

$$c(\mathcal{M}, \xi) = E[g(D)]E[h(U)]. \quad (38)$$

Interestingly, this is exactly the same overhead as under the random permutation in (31). When $r(x)$ deviates from being a constant in either direction (i.e., becomes increasing or decreasing in x), there exists a simple technique for deciding the optimal order. Assume $r(x)$ is monotonic and consider Algorithm 1, which creates a sequence

$$\mathbf{z} := (h(1/n), h(2/n), \ldots, h(1)) \quad (39)$$

and sorts it in the opposite order of monotonicity of $r(x)$. If $\mathbf{z}$ transforms into $h(i_1/n), h(i_2/n), \ldots, h(i_n/n)$, the algorithm assigns $\theta_n(j) = i_j$, where ties are broken arbitrarily.

Theorem 3. *When $r(x)$ is monotonic, Algorithm 1 builds permutations that minimize* (37).

This result leads to several useful conclusions.

Corollary 1. *If $h(x)$ has the same monotonicity in $[0, 1]$ as $g(x)/w(x)$ in $[0, \infty)$, descending order is optimal. If monotonicity of these functions is opposite of each other, ascending order is optimal.*

For triangle listing with $w(x) = \min(x, a)$, where $a > 0$ is a constant, ratio $g(x)/w(x) = (x^2 - x)/\min(x, a)$ is monotonically increasing. Recalling Table 4, as well as the various relationships in Figures 2 and 4, we get that θ_D is optimal for $T_1/E_1/E_2$ and [13], while θ_A is for $T_3/E_3/E_5$.

Corollary 2. *Suppose $h(1/2 + x) = h(1/2 - x)$ for all $x \in [0, 1/2]$. Then, if $h(x)$ has the same monotonicity in $[0, 1/2)$ as $g(x)/w(x)$ in $[0, \infty)$, RR is optimal. If these functions have opposing monotonicity, CRR is optimal.*

This implies that the cost of T_2 is indeed minimized by RR and that of E_4/E_6 by CRR. Theorem 3 also provides a means for creating permutations with the highest cost.

Corollary 3. *Map $\xi(u)$ is the best for a given method iff its complement $\xi''(u)$ is the worst.*

6.2 Comparison

We now decide which vertex iterator is better under their respective optimal permutations. Re-write (23) and (34) as

$$c(T_1, \xi_D) = \frac{E[r(U)(1-U)^2]}{2}, \tag{40}$$

$$c(T_2, \xi_{RR}) = \frac{E[r(U)(1-U^2)]}{4} \tag{41}$$

and consider the next result.

Theorem 4. *If $r(x)$ is increasing, (40) is smaller than (41). If $r(x)$ is decreasing, the inequality is reversed. If $r(x)$ is a constant, they produce the same cost.*

For $w(x) = \min(x, a)$, we already know that $r(x)$ is increasing, which implies that T_1 is faster than T_2. For edge iterator, we have

$$c(E_1, \xi_D) = \frac{E[r(U)(1-U^2)]}{2}, \tag{42}$$

$$c(E_4, \xi_{CRR}) = \frac{E[r(U)(U^2 - 2U + 2)]}{4}, \tag{43}$$

with the corresponding comparison next.

Theorem 5. *If $r(x)$ is increasing, (42) is smaller than (43). If $r(x)$ is decreasing, the opposite holds. If $r(x)$ is a constant, they are identical.*

6.3 Asymptotics

Our study revealed that the best vertex iterator is T_1 and the fastest scanning edge iterator is E_1, with the corresponding optimal cost

$$c(T_1, \xi_D) = \frac{E[(D^2 - D)(1 - J(D))^2]}{2}, \tag{44}$$

$$c(E_1, \xi_D) = \frac{E[(D^2 - D)(1 - J^2(D))]}{2}. \tag{45}$$

Using Pareto $F(x)$, we established earlier that the finiteness condition was $\alpha > 4/3$ for (44) and $\alpha > 1.5$ for (45). This means that as $n \to \infty$, T_1 is provably faster than E_1 in all graphs with $\alpha \in (4/3, 1.5]$, *no matter how these algorithms are implemented.* For $\alpha \in (1.5, \infty)$, both methods produce finite cost and the winner must be decided by taking into account the speed of their elementary operations, i.e., hash-table lookups vs scanning intersection similar to Table 3, but customized to a particular graph $G_n(\theta_n)$.

When α drops below the finiteness thresholds of each method, the scaling rate of cost is determined by the tail of the spread distribution

$$1 - J_n(x) \sim \begin{cases} x^{1-\alpha} & \alpha > 1 \\ 1 - \log(x)/\log(t_n) & \alpha = 1 \\ 1 - x^{1-\alpha}/t_n^{1-\alpha} & 0 < \alpha < 1 \end{cases}, \tag{46}$$

where the last two cases arise due to $E[D_n] \to \infty$. For root truncation, we get that $E[c_n(T_1, \theta_D)|\mathbf{D}_n]/a_n \to 1$, where

$$a_n = \begin{cases} \log n & \alpha = 4/3 \\ n^{2-1.5\alpha} & 1 < \alpha < 4/3 \\ \sqrt{n}/\log^2 n & \alpha = 1 \\ n^{1-\alpha/2} & 0 < \alpha < 1 \end{cases}, \tag{47}$$

and $E[c_n(E_1, \theta_D)|\mathbf{D}_n]/b_n \to 1$, where

$$b_n = \begin{cases} \log n & \alpha = 1.5 \\ n^{1.5-\alpha} & 1 < \alpha < 1.5 \\ \sqrt{n}/\log n & \alpha = 1 \\ n^{1-\alpha/2} & 0 < \alpha < 1 \end{cases}. \tag{48}$$

These results show that T_1 grows slower than E_1 for all $\alpha \in [1, 1.5)$; however, interestingly enough, they have the same scaling behavior when $\alpha \in (0, 1)$. Determining the corresponding rates under *linear* truncation requires modeling families of unconstrained graphs under finite n, which is beyond the scope of our investigation here.

7. EVALUATION

We now use finite n, where the models are only approximate, to examine the impact of α, truncation, and graph size on the distance between the model and actual cost.

7.1 Model Computation

Recall that we start with a continuous Pareto distribution $F^*(x) = 1 - (1 + x/\beta)^{-\alpha}$ defined on $[0, \infty)$ and discretize it by rounding up each generated value. This produces $F(x) = 1 - (1 + \lfloor x \rfloor/\beta)^{-\alpha}$ defined on natural numbers, which we employ in the construction of random graphs and comparison against the model. In general, (30) expands into a double Lebesgue-Stieltjes integral

$$\int_0^{t_n} g(x) h\Big(\xi\Big(\frac{\int_0^x w(y) dF_n(y)}{\int_0^{t_n} w(z) dF_n(z)}\Big)\Big) dF_n(x), \tag{49}$$

which can be easily computed in Matlab if $F_n(x)$ is continuous, e.g., using $F_n(x) = F^*(x)/F^*(t_n)$. However, this is only a crude approximation to the outcome observed in simulations. Therefore, a more prudent approach is to write these integrals as summations

$$\sum_{i=1}^{t_n} g(i) h\Big(\xi\Big(\frac{\sum_{j=1}^{i} w(j) p_j}{\sum_{k=1}^{t_n} w(k) p_k}\Big)\Big) p_i, \tag{50}$$

where $p_i := F_n(i) - F_n(i-1)$ is the PMF (probability mass function) of truncated degree.

Even though (50) contains two nested sums and appears to exhibit quadratic complexity, it can be computed in linear time and $O(1)$ space. This works well for many scenarios, except when a good estimate of the limiting cost is needed under linear truncation and a slowly-decaying tail $1 - F(x)$. For such cases, the runtime can be improved to

Algorithm 2: Quick computation of (50)

```
Function DiscreteModel (Fn, g, w, h, xi; tn, eps)
    EDn = J = cost = 0
    for (i=1; i ≤ tn; i += jump) do                    ▷ $E[D_n]$
        jump = ceil(eps ∗ i)
        EDn += w(i)∗(Fn(i + jump − 1) − Fn(i − 1))
    for (i=1; i ≤ tn; i += jump) do                    ▷ $E[c_n(\mathcal{M}, \theta_n)]$
        jump = ceil(eps ∗ i)
        p = Fn(i + jump − 1) − Fn(i − 1)
        J += w(i) ∗ p / EDn
        cost += w(i) ∗ h(xi(J)) ∗ p
    return cost
```

n	$F^*(x)$ in (49)		$F(x)$ in (50)		Algorithm 2	
	value	time	value	time	value	time
10^3	144.86	7.4	142.85	<0.01	142.85	<0.01
10^4	245.29	7.6	241.15	<0.01	241.15	<0.01
10^7	353.92	7.8	346.92	1.2	346.92	0.04
10^8	359.85	7.9	352.73	12	352.73	0.05
10^9	362.18	7.9	354.94	117	354.94	0.07
10^{10}	363.06	8.0	355.79	1170	355.79	0.08
10^{12}	363.51	8.2			356.22	0.10
10^{13}	363.56	8.2	too slow		356.26	0.11
10^{14}	363.57	8.2			356.28	0.12
10^{17}	363.57	8.3			356.28	0.13

Table 5: Model results and computation time (in seconds) for $\mathbf{T}_1$ under descending order ($\alpha = 1.5$, $\epsilon = 10^{-5}$, linear truncation).

$O((1+\log(\epsilon t_n))/\epsilon)$ by compressing all summands from large intervals $[i, (1+\epsilon)i]$ into a single term, where $1/t_n \le \epsilon < 1$ is a chosen parameter. This is demonstrated in Algorithm 2, in which $\epsilon = 1/t_n$ yields the exact result and larger values offer varying degrees of approximation.

Table 3 compares the continuous result, the exact discrete model, and Algorithm 2. Using two decimal digits of precision, notice that the continuous model does not converge until $n = 10^{14}$. Based on the result in columns 4-5 of the table, this value of n would require an extrapolated four months in the exact model (50). On the other hand, Algorithm 2 computes this case in a fraction of a second due to its $\log n$ complexity. Additionally, the table shows that the continuous model indeed deviates from the discrete version by non-negligible amounts (i.e., $1.5-2\%$).

7.2 Random Graph Generation

Traditional methods [8], [30] that aim to realize a random graph with a given degree sequence $(d_1, \ldots, d_n)$ place d_i copies (i.e., stubs) of each node i into an array and uniformly draw pairs of available nodes at each step. The two selected stubs are removed and the process is repeated. This, however, leads to self-loops and duplicate edges. Since we are interested in simple graphs, these extraneous edges must be removed, which has a noticeable impact on the realized degree, especially when Pareto α drops below 2 and truncation function $t_n = n-1$. Specifically, if the desired degree of node i is d_i but the constructed graphs G_n are allowed to implement smaller values in the range $[1, d_i]$, simulations may not match theoretical predictions of $E[X_i(\theta_n)|\mathbf{D}_n]$.

To overcome this problem, we employ a variation of the method from [11] that picks neighbors in proportion to their residual degree and excludes the already-attached neighbors

n	$\mathrm{T}_1 + \theta_A$			$\mathrm{T}_1 + \theta_D$		
	sim	(50)	error	sim	(50)	error
10^4	159.1	155.6	-2.2%	40.2	39.3	-2.2%
10^5	518.0	516.6	-0.3%	87.8	87.0	-0.9%
10^6	1,355.6	1,354.5	-0.1%	143.7	142.9	-0.6%
10^7	3,089.1	3,089.2	0.003%	196.9	196.2	-0.4%
∞		∞			356.3	

Table 6: Cost with $\alpha = 1.5$ and root truncation.

n	$\mathrm{T}_2 + \theta_D$			$\mathrm{T}_2 + \theta_{RR}$		
	sim	(50)	error	sim	(50)	error
10^4	102.3	103.7	1.4%	79.5	75.8	-4.6%
10^5	260.0	261.4	0.5%	186.4	181.8	-2.5%
10^6	467.0	467.4	0.1%	315.4	310.4	-1.6%
10^7	674.6	675.4	0.1%	436.1	432.4	-0.8%
∞		1,307.6			770.4	

Table 7: Cost with $\alpha = 1.7$ and root truncation.

n	$\mathrm{T}_1 + \theta_D$			$\mathrm{T}_2 + \theta_{RR}$		
	sim	(50)	error	sim	(50)	error
10^4	178.6	179.3	0.4%	318.9	371.9	16.6%
10^5	182.2	181.3	-0.5%	363.7	383.0	5.3%
10^6	182.6	181.5	-0.6%	382.0	384.2	0.6%
10^7	182.6	181.5	-0.6%	383.5	384.3	0.2%
∞		181.5			384.3	

Table 8: Cost with $\alpha = 2.1$ and linear truncation.

when performing selection. If implemented naively, this requires quadratic complexity; however, this can also be done in $n \log n$ time using interval trees that record the residual probability mass of degree on both sides of each node. As a result, graphs with 10M nodes to be generated in several seconds, which is sufficient for our analysis below. With the exception of possibly one last edge (i.e., if $\sum_{i=1}^n d_i$ is odd), G_n in our simulations implements $\mathbf{D}_n$ exactly.

7.3 Constrained Degree

We start with AMRC graphs, where the obtained results should be accurate even for finite n. All simulations in the rest of the paper are averaged over 100 random degree sequences $\mathbf{D}_n$, each with 100 random graphs G_n (i.e., 10K graph instances total). We keep Pareto $\beta = 30(\alpha-1)$, which yields $E[D] \approx 30.5$ after discretization. We use (50) for small n, while the corresponding limits as $n \to \infty$ are provided by Algorithm 2. Unless mentioned otherwise, $w(x) = x$. Note that simulation results averaged over 10K iterations are still pretty noisy, which explains non-monotonicity of error in certain cases as n increases.

Tables 6-7 examine $\alpha < 2$ under root truncation. Since these graphs deterministically limit the maximum degree to $\sqrt{n}$, (50) is accurate even for n as small as 10K. Table 8 shows another AMRC scenario, where $\alpha = 2.1$ and truncation is linear. Because these graphs are only asymptotically constrained, larger discrepancy for small n was expected. However, even the slowest-converging method studied here (i.e., $\mathrm{T}_2+\theta_{RR}$) exhibits less than 1% error for $n \ge$ 1M nodes.

7.4 Unconstrained Degree

We now transition to more challenging cases. Table 9 revisits the data in Table 6 under linear truncation. Both permutations now produce larger cost and quicker convergence

n	$T_1 + \theta_A$			$T_1 + \theta_D$		
	sim	(50)	error	sim	(50)	error
10^4	7,158	6,452	-9.9%	209.5	241.1	15.1%
10^5	25,770	24,303	-5.7%	261.0	302.1	15.8%
10^6	84,441	82,815	-1.9%	294.1	333.0	13.3%
10^7	274,876	270,125	-1.7%	317.0	346.9	9.4%
∞		∞			356.3	

Table 9: Cost with $\alpha = 1.5$ and linear truncation.

n	$T_2 + \theta_D$			$T_2 + \theta_{RR}$		
	sim	(50)	error	sim	(50)	error
10^4	499.4	854.4	71.1%	354.5	532.6	50.3%
10^5	725.4	1,096.6	51.2%	476.5	662.3	39.0%
10^6	907.7	1,216.7	34.0%	570.2	724.4	27.0%
10^7	1,041.5	1,270.0	21.9%	631.2	751.5	19.1%
∞		1,307.6			770.4	

Table 10: Cost with $\alpha = 1.7$ and linear truncation.

n	$T_1 + \theta_D$		$T_2 + \theta_D$		$T_2 + \theta_{RR}$	
	$w_1(x)$	$w_2(x)$	$w_1(x)$	$w_2(x)$	$w_1(x)$	$w_2(x)$
10^4	38%	-54.1%	304%	21.6%	216%	-3.1%
10^5	107%	-52.3%	619%	17.9%	458%	-2.2%
10^6	214%	-50.4%	1,207%	12.9%	856%	-2.3%
10^7	386%	-48.7%	2,353%	9.1%	4,105%	-0.5%

Table 11: Relative error of (50) under $\alpha = 1.2$ and linear truncation (asymptotically infinite cost).

towards their respective limits, which is especially noticeable under θ_A. A similar scenario for T_2 is shown in Table 10. Compared to Table 7, the error is much larger; however, it still monotonically decays towards zero as n increases. This is a consequence of the limiting cost being finite.

A drastically different result can be observed when the asymptotic complexity is infinite, i.e., $c(\mathcal{M}, \xi) = \infty$. This arises from the difference in the rates at which simulations and (50) scale as $n \to \infty$, which yields an error that grows with n. We use this opportunity to investigate how $w(x)$ can be used to create a more accurate result for such scenarios. Specifically, define $w_1(x) = x$ and $w_2(x) = \min(x, \sqrt{m})$. Table 11 examines $\alpha = 1.2$ under linear truncation. While $w_1(x)$ builds a hefty error against T_1 by the time n reaches 10M, its alternative $w_2(x)$ settles into a growth rate that is essentially the same as that of simulations. Furthermore, it eliminates most of the error in the other two cases.

Both functions w_1, w_2 have the same limit as $n \to \infty$, which can be shown using root-truncated $F_n(x)$, but the convergence speed is clearly different. As discussed earlier, finding $w(x)$ that keeps (50) provably accurate in unconstrained graphs of finite size is a topic for future work. However, if such functions exist and $(x^2 - x)/w(x)$ is monotonic, optimality and comparison results of the previous section apply to them just the same.

7.5 Real Graphs

We now examine our conclusions and model applicability to Twitter [27], which is a standard graph in this field with n = 41M nodes and m = 1.2B edges (9.3 GB). In addition to the five main permutations, i.e., ascending θ_A, descending θ_D, round-robin θ_{RR}, complementary round-robin θ_{CRR}, and uniform θ_U, we also consider the degenerate orientation, built using the algorithm from [29]. Because this orientation requires 5 hours to compute, which is two orders of magnitude longer than it takes E_1 to list all triangles [17], we do not consider it competitive. Instead, we provide the corresponding cost to shed light on how much theoretical improvement it may offer to $c_n(\mathcal{M}, \theta_n)$.

	Permutation					
	θ_D	θ_A	θ_{RR}	θ_{CRR}	θ_U	θ_{degen}
T_1	150B	123T	63T	31T	45T	136B
T_2	360B	360B	255B	62T	41T	815B
E_1	511B	123T	63T	93T	86T	951B
E_4	123T	123T	123T	62T	82T	123T

Table 12: CPU operations on Twitter.

Table 12 displays the total number of CPU instructions $nc_n(\mathcal{M}, \theta_n)$ on Twitter, highlighting in gray the optimal permutation for each row. These results agree with our analysis, in both the choice of best/worst orientation and comparison between the methods. For example, θ_{RR} is best for T_2, θ_A is worst for E_1, and the cost of E_1 under θ_D is double that of T_2 under θ_{RR}. The degenerate orientation reduces the best cost of T_1 by 10%, but increases that of the other methods by $2-3\times$. Note that our results minimize the expected cost over all graphs with a given degree distribution. This does not prevent existence of algorithms that can take a particular instance of G_n and customize θ_n to its edge structure, which explains how the degenerate permutation manages to beat θ_D for T_1.

The overhead ratio between the worst and best permutations is 817 for T_1, 241 for T_2 and E_1, and only 2 for E_4. The last case can be explained by the fact that E_4 is almost equally expensive under all permutations, exhibiting *at least* 62T/511B = 121 times more overhead than the best orientation for E_1. Also notice that T_1 and T_2 are only within a factor of 255B/150B = 1.7 of each other, which shows that keeping the graph non-relabeled would have doubled the cost of T_1 and made it worse than T_2. This also would have caused a 29% increase for E_1 and 100% for E_4. Our final observation is that lack of relabeling in prior work [36] explains reports of 300B tuples for T_1 on Twitter.

8. CONCLUSION

Our efforts produced the first accurate result on the asymptotics of triangle listing under arbitrary permutations. We proved that expected cost could be optimized using four different orientations and derived the corresponding complexity models. Our results showed that only two methods were generally worth considering – one from the vertex-iterator family and the other from the scanning edge-iterator. In many cases the winner will be determined by the edge-existence verification speed of these algorithms; however, we discovered degree sequences for which the former would always outperform the latter as $n \to \infty$.

Our results answered many open questions in the area of triangle listing; however, additional challenges remain. Among them is analysis of unconstrained graphs, design of better external-memory partitioning schemes, and modeling of I/O complexity in scenarios such as [17].

9. REFERENCES

[1] W. Aiello, F. R. K. Chung, and L. Lu, “A Random Graph

Model for Massive Graphs," in *Proc. ACM STOC*, May 2000, pp. 171–180.

[2] N. Alon, R. Yuster, and U. Zwick, "Finding and Counting Given Length Cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, Mar. 1997.

[3] S. Arifuzzaman, M. Khan, and M. Marathe, "PATRIC: A Parallel Algorithm for Counting Triangles in Massive Networks," in *Proc. ACM CIKM*, Oct. 2013, pp. 529–538.

[4] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, "Reductions in Streaming Algorithms, With an Application to Counting Triangles in Graphs," in *Proc. ACM-SIAM SODA*, Jan. 2002, pp. 623–632.

[5] A.-L. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[6] V. Batagelj and M. Zaversnik, "Short Cycle Connectivity," *Discrete Math.*, vol. 307, no. 3–5, pp. 310–318, Feb. 2007.

[7] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient Semi-streaming Algorithms for Local Triangle Counting in Massive Graphs," in *Proc. ACM SIGKDD*, Aug. 2008, pp. 16–24.

[8] E. A. Bender and E. R. Canfield, "The Asymptotic Number of Labeled Graphs with Given Degree Sequences," *J. Combin. Theory Ser. A*, vol. 24, pp. 296–307, May 1978.

[9] J. W. Berry, L. A. Fostvedt, D. J. Nordman, C. A. Phillips, C. Seshadhri, and A. G. Wilsone, "Why Do Simple Algorithms for Triangle Enumeration Work in the Real World?" *Internet Mathematics*, vol. 11, no. 6, pp. 555–571, May 2015.

[10] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips, "Tolerating the Community Detection Resolution Limit With Edge Weighting," *Physical Review E*, vol. 83, no. 5, p. 056119, 2011.

[11] J. Blitzstein and P. Diaconis, "A Sequential Importance Sampling Algorithm for Generating Random Graphs With Prescribed Degrees," *Internet Mathematics*, vol. 6, no. 4, pp. 489–522, 2011.

[12] P. Brach, M. Cygan, J. Łącki, and P. Sankowski, "Algorithmic Complexity of Power Law Networks," in *Proc. ACM-SIAM SODA*, Jan. 2016, pp. 1306–1325.

[13] N. Chiba and T. Nishizeki, "Arboricity and Subgraph Listing Algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, Feb. 1985.

[14] S. Chu and J. Cheng, "Triangle Listing in Massive Networks and Its Applications," in *Proc. ACM SIGKDD*, Aug. 2011, pp. 672–680.

[15] F. R. K. Chung and L. Lu, "Connected Components in Random Graphs with Given Expected Degree Sequences," *Annals of Combinatorics*, vol. 6, no. 2, pp. 125–145, Nov. 2002.

[16] J. Cohen, "Graph Twiddling in a MapReduce World," *Computing in Science & Engineering*, vol. 11, no. 4, pp. 29–41, Jul. 2009.

[17] Y. Cui, D. Xiao, and D. Loguinov, "On Efficient External-Memory Triangle Listing," in *Proc. IEEE ICDM*, Dec. 2016, pp. 101–110.

[18] R. Dementiev, "Algorithm Engineering for Large Data Sets," Ph.D. dissertation, Universität des Saarlandes, 2006.

[19] P. Erdös and A. Rényi, "On Random Graphs I," *Publication Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[20] I. Fudos and C. M. Hoffmann, "A Graph-Constructive Approach to Solving Systems of Geometric Constraints," *ACM Transactions on Graphics*, vol. 16, no. 2, pp. 179–216, Apr. 1997.

[21] I. Giechaskiel, G. Panagopoulos, and E. Yoneki, "PDTL: Parallel and Distributed Triangle Listing for Massive Graphs," in *Proc. IEEE ICPP*, Sep. 2015, pp. 370–379.

[22] X. Hu, Y. Tao, and C. Chung, "Massive Graph Triangulation," in *Proc. ACM SIGMOD*, Jun. 2013, pp. 325–336.

[23] A. Itai and M. Rodeh, "Finding a Minimum Circuit in a Graph," *SIAM Journal on Computing*, vol. 7, no. 4, pp. 413–423, Nov. 1978.

[24] Z. R. Kashani, H. Ahrabian, E. Elahi, A. Nowzari-Dalini, E. S. Ansari, S. Asadi, S. Mohammadi, F. Schreiber, and A. Masoudi-Nejad, "Kavosh: A New Algorithm for Finding Network Motifs," *Bioinformatics*, vol. 10, no. 1, p. 318, Oct. 2009.

[25] J. Kim, W. Han, S. Lee, K. Park, and H. Yu, "OPT: A New Framework for Overlapped and Parallel Triangulation in Large-scale Graphs," in *Proc. ACM SIGMOD*, 2014, pp. 637–648.

[26] A. Kostochka, E. Sopena, and X. Zhu, "Acyclic and Oriented Chromatic Numbers of Graphs," *Journal of Graph Theory*, vol. 24, no. 4, pp. 331–340, Apr. 1997.

[27] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, A Social Network or a News Media?" in *Proc. WWW*, Apr. 2010, pp. 591–600.

[28] M. Latapy, "Main-memory Triangle Computations for Very Large (Sparse (Power-law)) Graphs," *Theor. Comput. Sci.*, vol. 407, no. 1-3, pp. 458–473, Nov. 2008.

[29] D. W. Matula and L. L. Beck, "Smallest-Last Ordering and Clustering and Graph Coloring Algorithms," *Journal of the ACM*, vol. 30, no. 3, pp. 417–427, Jul. 1983.

[30] M. Molloy and B. Reed, "A Critical Point for Random Graphs with a Given Degree Sequence," *Random Structures and Algorithms*, vol. 6, no. 2/3, pp. 161–180, Mar./May 1995.

[31] M. E. Newman, D. J. Watts, and S. H. Strogatz, "Random Graph Models of Social Networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 1, pp. 2566–2572, Feb. 2002.

[32] M. Ortmann and U. Brandes, "Triangle Listing Algorithms: Back from the Diversion," in *Proc. ALENEX*, Jan. 2014, pp. 1–8.

[33] T. Schank and D. Wagner, "Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study," in *Proc. WEA*, May 2005, pp. 606–609.

[34] M. Sevenich, S. Hong, A. Welc, and H. Chafi, "Fast In-Memory Triangle Listing for Large Real-World Graphs," in *Proc. ACM SNA-KDD*, Aug. 2014, pp. 1–9.

[35] J. Shun and K. Tangwongsan, "Multicore Triangle Computations Without Tuning," in *Proc. IEEE ICDE*, Apr. 2015, pp. 149–160.

[36] S. Suri and S. Vassilvitskii, "Counting Triangles and the Curse of the Last Reducer," in *Proc. WWW*, Mar. 2011, pp. 607–614.

[37] N. Wang, J. Zhang, K.-L. Tan, and A. K. Tung, "On Triangulation-Based Dense Neighborhood Graph Discovery," *PVLDB*, vol. 4, no. 2, pp. 58–68, 2010.

[38] D. J. Watts and S. Strogatz, "Collective Dynamics of 'Small World' Networks," *Nature*, vol. 393, pp. 440–442, Jun. 1998.

[39] J. A. Wellner, "A Glivenko-Cantelli Theorem and Strong Laws of Large Numbers for Functions of Order Statistics," *The Annals of Statistics*, vol. 5, no. 3, pp. 473–480, May 1977.

[40] V. Williams and R. Williams, "Subcubic Equivalences Between Path, Matrix, and Triangle Problems," in *Proc. IEEE FOCS*, Oct. 2010, pp. 645–654.

[41] R. W. Wolff, *Stochastic Modeling and the Theory of Queues.* Prentice Hall, 1989.

[42] D. Xiao, Y. Cui, D. B. Cline, and D. Loguinov, "On Asymptotic Cost of Triangle Listing in Random Graphs," Texas A&M University, Tech. Rep. 2016-9-2, Sep. 2016. [Online]. Available: http://irl.cs.tamu.edu/publications/.

[43] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Zhao, and Y. Dai, "Uncovering Social Network Sybils in the Wild," in *Proc. ACM IMC*, Nov. 2011, pp. 259–268.

[44] W. van Zwet, "A Strong Law for Linear Functions of Order Statistics," *Annals of Probability*, vol. 8, no. 5, pp. 986–990, 1980.