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 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], [13], [19], [23], [24] is enumeration of small subgraphs whose occurrence in nature is much more frequent than in classical random graphs [7], [20].

The most widely considered subgraphs are triangles, which have applications in numerous areas – databases, computer graphics, information retrieval, graph theory, algorithm complexity, and bioinformatics [6], [8], [9], [12], [17], [21], [25], [32], [59], [60], [22], [44]. Triangle listing has a four-decade history [24]; 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.

1.1 Deterministic Graphs

Assume $G = (V,E)$ is an 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 [14], [29]. 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 [24]; however, it requires $n^2$ RAM to store the adjacency matrix, certainly an impossibility for large graphs.

An improvement to $O(m)$ space is [14], which visits nodes in descending order of degree and removes them from the graph after each step. As we show below, descending is the worst order for this specific algorithm; instead, the degree should be sorted ascending. Either way, the best result about CPU complexity of this approach is $O(\delta m)$, where $\delta$ is the arboricity of $G$, i.e., minimum number of edge-disjoint spanning forests into which $G$ can be decomposed. Unfortunately, $\delta$ 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 $O(\sum_{i=1}^{d_i} \theta_i)$ 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 theoretical cost [2, 29] has remained at the same bound $O(\delta m) = O(m^{1.5})$.

Each acyclic orientation on $G$ can be viewed as some permutation $\theta_n$ 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

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

where $f$ is some non-linear function that depends on the listing method $M$. 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 [27].

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 degenerate permutation offers negligible improvement and thus fails to justify the extra computation [33, 34].

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 $D_n = (D_1, \ldots, D_n)$ is an iid (independent and identically distributed) sequence drawn from $F_n(x)$, which we assume is graphic with probability $1 - o(1)$, and can be made such by removal of one edge. This allows construction of a random graph $G_n = (V_n, E_n)$ that realizes $D_n$, where analysis focuses on the expectation of $G_n$ over all graphs that realize a particular degree sequence, i.e., $E[c_n(M, \theta_n)|D_n]$.

Let $D_n \sim F_n(x)$ be a random variable with the same distribution as the degree in $G_n$. Early results [29] with Pareto $F(x) = 1 - (1 + |x|/\beta)^{-\alpha}$ typically bound the expected cost using a growing function of $n$. Recent work [11] obtains tighter bounds by focusing on a particular version of vertex iterator, which we call $T_1$, and using the descending-degree orientation $\theta_D$. Assuming $t_n = \sqrt{n}$, they derive that $E[c_n(T_1, \theta_D)|D_n]$ converges as $n \to \infty$ to

$$E[(Z_1^2 - Z_1)Z_2Z_31_{\min(Z_2, Z_3) > Z_1}]/2E^2[D],$$

(2)

where $D, Z_1, Z_2, Z_3 \sim F(x)$ are iid variables and $1_A$ 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 $\theta_D$. Due to the complexity of derivation in [11], it is unclear if the same 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 $O(\cdot)$, and how the degree distribution impacts overhead of the alternative methods. We address these questions next.

1.3 Contributions and Technical Results

To understand the overhead of triangle listing, our first contribution is to dissect the various ways that vertex and edge iterators have the same asymptotic complexity. We discuss how to efficiently implement them and propose a framework that does exactly that.

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

$$E[c_n(M, \theta_n)|D_n] \approx \frac{1}{n} \sum_{i=1}^{n} g(d_i(\theta_n))h(q_i(\theta_n)),$$

(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 $M$.

Our third contribution is to analyze convergence of (3) using Glimenko-Cantelli results for functions of order statistics [38]. Specifically, we obtain closed-form limits for all methods under both ascending/descending-degree permutations. For example, vertex iterator $T_1$ under $\theta_D$ converges to

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

(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 if $\alpha > 4/3$, which agrees with the conclusion of [11]. While $T_2$ captures the same limit, our derivation and result are both much simpler.

To understand the effect of $\theta_n$, 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 $P(\theta_n([0, u]) < vn)$ 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

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

(5)

Armed with (5), our fifth contribution is to analyze optimality of $\theta_n$ 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 $\alpha > 4/3$ and

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

(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 that of $E_1$ is in (5). Finally, $E_4$ is optimized by the Complementary Round-Robin (CRR) permutation, which positions large degree towards the middle. If $\theta_n$ is optimal for a given $M$, 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 faster than $E_4$ for all $F(x)$. However, selecting between the best vertex and edge iterators is more complicated. In general, $c_n(E_1, \theta_n) = c_n(T_1, \theta_n) + c_n(T_2, \theta_n)$ shows that the edge iterator performs as many operations as both vertex iterators combined; however, its elementary instructions are 95 times faster than those of vertex iterator on modern Intel CPUs [18]. Our results demonstrate that $T_1$ has finite cost if $\alpha > 4/3$ and $T_2$ if $\alpha > 0$. Consequently, for $\alpha \in (4/3, 1.5]$, $T_1$ always beats $E_1$ as $n \to \infty$, regardless of their internal implementation. Otherwise, the decision should be made based on the ratio of their cost and speed of underlying operations on target hardware. Although it is commonly believed that vertex and edge iterators have the same asymptotic complexity [29, 34], we prove that this is not true for extremely heavy-tailed degree with $\alpha \in (4/3, 1.5]$ and intersection-based edge iterators.

We conclude with simulations that highlight model accuracy under finite $n$ and examine the impact of Pareto $\alpha$ 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 (e.g., Vertex Iterator++) [15]. Forward [34]. Compact Forward [29] under a novel umbrella of 18 baseline algorithms and discussing how to extract the highest performance out of each.
Throughout the paper, we assume that adjacency lists in graphs are sorted ascending by node ID, which corresponds to left-to-right orientation in figures.

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 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 options are random (e.g., hash-based) [15], [24], descending-degree [3], [23], [34], and ascending-degree [35]. 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 $y \rightarrow x$ 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 [29], [35], requires sorting $n$ items using $\mathcal{O}$ and rewriting the labels of all edges in $\mathcal{E}$. The second step, known as orientation [24], [34], 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 $\theta_n : V \rightarrow V$ 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, the out-list of $i$ 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 $\theta_n$ 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.

2.2 Vertex Iterator

Vertex iterator is a classical algorithm [19], [23], [34], [37] 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 $y \rightarrow x$, 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 $(y,x)$, per-node cost of $T_1$ in Figure 1(a) is

$$c_n(T_1, \theta_n) = \frac{1}{n} \sum_{i=1}^{n} X_i(\theta_n) Y_i(\theta_n).$$

The second search order, which we call $T_2$, treats the current node as being neither the smallest, nor the largest in the triangle. The corresponding technique, shown in Figure 1(b) verifies existence of edges $z \rightarrow x$, where pairs $(z,x)$ sweep all possible combinations of $y$’s in and out neighbors. The number of such tuples is given by

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

As sketched in Figure 1(c), method $T_3$ starts from the smallest node $x$ and examines presence of edges $z \rightarrow y$ between all pairs of in-neighbors such that $y < z$. Its overhead is identical to that of $T_1$, but with $X_i(\theta_n)$ and $Y_i(\theta_n)$ trading places, i.e.,

$$c_n(T_3, \theta_n) = \frac{1}{n} \sum_{i=1}^{n} Y_i(\theta_n) X_i(\theta_n).$$

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. Define reverse permutation $\theta_n' : V \rightarrow V$ to 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 $Y_i(\theta_n)$.

**Proof.** Observe that $\theta_n'(i) < \theta_n'(j)$ if and only if $\theta_n(i) > \theta_n(j)$. Therefore, the reverse permutation exchanges in and out neighbors at each node, which means that $X_i(\theta_n') = Y_i(\theta_n)$ and similarly $Y_i(\theta_n') = X_i(\theta_n)$.

Since $c_n(T_1, \theta_n) = c_n(T_3, \theta_n')$, we conclude that $T_1$ and $T_3$ are equivalent to each other, i.e., have the same complexity and speed of elementary operations. Similarly, $T_2$ and $T_5$ reverse into each other and thus form a standalone equivalence class. This is schematically shown in Figure 2. From this point on, we dismiss $T_4$-$T_6$ and keep only $T_1$-$T_2$ under consideration.
local in list intersection. We split this overhead into remote and \( c \) section at \( z \) list using a grid pattern and draw nodes in ascending order triangle \( \triangle \). intersect the sets of neighbors at both incident vertices \([15]\), \([29]\), \([34]\). A scanning edge iterator (SEI) sequentially rolls the full out-list, the remote overhead becomes later, it is possible to traverse each directed edge and in-

### 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 \([15]\), \([29]\), \([34]\). 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 \( \triangle_{xyz} \). We show the intersected portions of each list using a grid pattern and draw nodes in ascending order along the vertical dimension of the picture. Note that intersection at \( z \) does not involve all of its out-neighbors; only those smaller than \( y \). This is a consequence of the \( x < y < z \) relationship and transitivity of \( \bigcirc \).

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 \( c_o(E_1, \theta_n) = c_o(T_1, \theta_n) + c_o(T_2, \theta_n) \), where the former term is local overhead and the latter is remote.

**Proof.** The local cost is trivially

\[
\sum_{i=1}^{n} X_i(\theta_n)(X_i(\theta_n) - 1),
\]

which equals \( c_o(T_1, \theta_n) \). For remote cost, we shift the perspective to node \( y \) in Figure 3(a) and observe that the number of visits to this node equals its in-degree. Since each visit scans the full out-list, the remote overhead becomes

\[
\sum_{i=1}^{n} X_i(\theta_n)Y_i(\theta_n),
\]

which is the same as \( c_o(T_2, \theta_n) \).

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, except it hashes the local neighbor list of the first visited node and performs lookups against it for all remote nodes \([15]\). 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 the second row of Table 1. Since LEI can be reduced to vertex iterator (both in terms of operation speed and cost), there is no need to consider it separately in the remainder of the paper.

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.

Since intersection works faster scanning forward than backward, we can dismiss \( E_5 \) because it requires a binary search in \( x \)’s in-neighbors to find the starting location \( y \) for the intersection. For similar reasons, \( E_6 \) is not competitive against \( E_4 \). Next, reversal of \( \theta_n \) 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 \([15]\). 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 \([14]\), which proposes a cumbersome version of \( E_3 \) where acyclic orientation holds only for the fraction of the graph that has been processed so far. Incomplete orientation is remedied in \([29]\), which uses a dynamically growing set of vectors to implement \( E_2 \) under the name Forward. The vector constraint is removed in Compact Forward \([29]\). In more recent studies, \( T_1 \) appears in \([23]\), \([24]\), \([37]\). \( E_1 \) in \([3]\), \([22]\), \([34]\), \( E_3 \) in \([15]\), and \( E_5 \) in \([33]\). At least six methods, i.e., \( T_1-T_3 \), \( E_1 \), \( E_3 \), \( E_4 \), are identified in \([33]\), but no analysis is offered. The majority of literature \([3]\), \([22]\), \([23]\), \([24]\), \([31]\), \([33]\), \([37]\) omits relabeling and performs only orientation. In such cases, the split entails evaluating \( \bigcirc \) for each edge \((x, y)\) to decide whether \( y \) belongs to \( N^+(x) \) or \( N^-(x) \). After \( G(\theta_n) \)

![Figure 3: Search order in SEI.](image1)

![Figure 4: SEI equivalence classes.](image2)

<table>
<thead>
<tr>
<th>( E_1 )</th>
<th>( E_2 )</th>
<th>( E_3 )</th>
<th>( E_4 )</th>
<th>( E_5 )</th>
<th>( E_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>( T_1 )</td>
<td>( T_2 )</td>
<td>( T_3 )</td>
<td>( T_1 )</td>
<td>( T_2 )</td>
</tr>
<tr>
<td>Remote</td>
<td>( T_2 )</td>
<td>( T_1 )</td>
<td>( T_3 )</td>
<td>( T_2 )</td>
<td>( T_1 )</td>
</tr>
</tbody>
</table>

**Table 1: Complexity of SEI.**
is produced, the directed neighbors of each node are not ordered in any particular way against each other, which doubles the cost of all terms that depend 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 \cite{29, 33, 35} 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 $\zeta = \sum_{i=1}^n \log d_i$ random memory accesses. Scanning edge iterators $E_1 / E_2$ use the same $\zeta$ extra jumps, while $E_3 / E_4$ and $E_3 / E_6$ take a larger performance hit -- one binary search per edge. This adds up to $\sum_{i=1}^n X_i \log d_i$ or $\sum_{i=1}^n Y_i \log d_i$ unnecessary memory lookups depending on the method. Using backwards-sorted lists, this overhead can be reduced to $\zeta$ for $E_3 / E_5$, but not $E_4 / E_6$. 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 algorithms, while imposing no restrictions on adjacency lists. All introduced algorithms are explained in detail and benchmarked in \cite{13}, whose results are summarized in Table 2. Even though SEI executes faster, Table 1 shows that it requires more operations. Assuming $w_n$ is the ratio of lowest cost achievable by SEI to that of the other two families of algorithms, it follows that SEI has better runtime on modern Intel CPUs iff $w_n < 95$. Since $w_n$ is a function of $G_n$, 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 $G_n$.

#### 3.1 Degree Growth

Recall that $F(x)$ is some fixed CDF, $t_n \to \infty$ is a monotonic function, and $F(x) = F(x)/F(t_n)$ is a truncated distribution. Suppose $D \sim F(x)$ represents the non-truncated degree and let $D_n = (D_1, \ldots, D_m)$ be an iid degree sequence drawn from $F(x)$. Note that $t_n \leq n - 1$ is required for $D_n$ to be graphic (i.e., realizable by a graph). 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 \cite{10, 31}, in which this metric is proportional to the product of corresponding degrees \cite{11, 16}.

\[ p_{ij}(\theta_n) \approx \frac{d_i(\theta_n) d_j(\theta_n)}{\sum_{k=1}^n d_k}. \] (12)

For this to be a probability, the numerator must not exceed the denominator, which we introduce more formally next. Note that graphic Pareto-based degree sequences \cite{5} require $E[D] < \infty$, i.e., scale parameter $\alpha > 1$, which means that $\sum_k d_k$ must be linear in $n$.

**Definition 1.** Suppose $L_n = \max \{D_n\}$ 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 (12) is exact in the limit. One option for satisfying this condition is to use distributions $F(x)$ with finite variance, 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 $E[D^{1/c}] < \infty$.

**Proof.** Suppose $D \sim F(x)$, $Z = D^{1/c}$, and $H(x) = P(Z < x)$. Let $\bar{H}(x) = 1 - H(x)$ be the complementary CDF of $Z$. Then, define $Q_n = Z \cdot 1_{Z > n}$ and notice that

\[ n\bar{H}(n) = n \int_n^\infty dH(y) \leq \int_n^\infty y dH(y), \] (13)

implies that $n\bar{H}(n) \leq E[Q_n]$. Since $Q_n \to 0$ as $n \to \infty$ and $Q_n \leq Z$, where $E[Z] < \infty$, it follows from the dominated convergence theorem that $E[Q_n] \to 0$ and therefore $n\bar{H}(n) \to 0$. Using Bernoulli inequality

\[ H^n(n) \approx (1 - n\bar{H}(n))^n \geq 1 - n\bar{H}(n) \] (14)

and recalling that $H(n) \leq 1$, we get that $H^n(n) \to 1$ as $n \to \infty$. Finally,

\[ P(L_n > n^c) = 1 - F_n(n^c) \leq 1 - F^n(n^c) = 1 - H^n(n) \to 0, \] (15)

which uses the fact that $F_n(x) \geq F(x)$.

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 $L_n \leq \sqrt{n}$, which keeps $p_{ij}(\theta_n) \leq 1$. 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 \( M \) is a particular triangle-listing method. Throughout the paper, we develop two sets of results—those that rely on \( F_0(x) \) to establish \( E[c_n(M, \theta_n)|D_n] \) and those that use \( F(x) \) to arrive at the corresponding limit \( c(M) \) as \( n \to \infty \). The former models are accurate for finite \( n \) 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, models of \( c(M) \) 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

\[
E[X_i(\theta_n)|D_n] \approx d_i(\theta_n) \frac{\sum_{j=1}^{i-1} d_j(\theta_n)}{\sum_{k=1}^{i-1} d_k - d_i(\theta_n)}.
\]

While (15) is asymptotically precise, it may exhibit errors for finite \( n \) in unconstrained graphs. This occurs 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 (16) by applying some positive and monotonically non-decreasing function \( w(x) \) to the degree of potential neighbors, i.e.,

\[
E[X_i(\theta_n)|D_n] \approx d_i(\theta_n) \frac{\sum_{j=1}^{i-1} w(d_j(\theta_n))}{\sum_{k=1}^{i-1} w(d_k) - w(d_i(\theta_n))}.
\]

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 (17) 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

\[
q_i(\theta_n) = \frac{E[X_i(\theta_n)|D_n]}{d_i(\theta_n)},
\]

and consider the next result.

**Proposition 4.** In asymptotically large AMRC graphs, all four triangle-listing techniques in Figure 3 are covered by one formula

\[
E[c_n(M, \theta_n)|D_n] \approx \frac{1}{n} \sum_{i=1}^{n} g(d_i(\theta_n)) h(q_i(\theta_n)).
\]

where \( g(x) = x^2 - x \) and \( h(x) \) is given by Table 3.

**Proof.** It is not difficult to see that both (7)–(8) depend on the second moment of out-degree \( E[X_i^2(\theta_n)|D_n] \). Modeling this term for small \( n \) is complicated due to the same dependency issues; however, as \( n \) becomes large, the Chernoff theorem [4] ensures that \( X_i(\theta_n) \) behaves as a sum of \( d_i(\theta_n) \) iid variables. As a result, its distribution is approximately binomial with parameters \( d_i(\theta_n) \) and \( q_i(\theta_n) \). Applying this insight, we get

\[
E[X_i^2(\theta_n)|D_n] \approx d_i(\theta_n) q_i(\theta_n)(1 - q_i(\theta_n)) + d_i^2(\theta_n) q_i^2(\theta_n),
\]

which after usage in (7)–(8) yields (19). \( \square \)

### 4. CONVERGENCE OF COST

We next examine the limit of (13) 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

Suppose \( D_n \) is sorted in ascending order to produce a new sequence \( A_n = (A_{n1}, \ldots, A_{nn}) \), where \( A_{ni} \leq A_{n(i+1)} \). 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

\[
\lim_{n \to \infty} \int_0^t \phi_n(u) du = \int_0^t \phi(u) du, \quad (20)
\]

where \( \phi(u) \) is some integrable function. Then, given a sufficiently smooth function \( g(x) \), results in the field of L-estimators [38], [41] show that

\[
\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} g(A_{ni}) \phi_n(i/n) = \int_0^1 g(F^{-1}(u)) \phi(u) du. \quad (21)
\]

Limits in the form of (21) are known as Glivenko-Cantelli results for functions of order statistics. Letting \( U \) be uniform in \([0,1]\), the integral in (21) 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 (21) to cover partial sums.

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

\[
\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{\lfloor nu \rfloor} g(A_{ni}) = \int_0^u g(F^{-1}(x)) dx. \quad (22)
\]

**Proof.** Using \( \phi_n(x) = 1_{nx \leq \lfloor nu \rfloor} \), notice that

\[
\frac{1}{n} \sum_{i=1}^{\lfloor nu \rfloor} g(A_{ni}) = \frac{1}{n} \sum_{i=1}^{\lfloor nu \rfloor} g(A_{ni}) \phi_n(i/n),
\]

which yields (22) after applying (21). \( \square \)

Our next step is to generalize (21) to handle summations that involve \( q_i(\theta_n) \). This metric specifies the fraction of degree covered by the nodes preceding \( i \).

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

\[
J(x) := \frac{1}{E[w(D)]} \int_0^x w(y) dF(y). \quad (24)
\]

**Proof.** Since \( E[w(D)] < \infty \), we can ignore the term that precludes self-loops in the denominator of (13) and write

\[
q_{\lfloor nu \rfloor}(\theta_A) \approx \frac{1}{n} \sum_{i=1}^{\lfloor nu \rfloor} w(A_{ni}),
\]

where

\[
\begin{array}{cccc}
T_1 & T_2 & E_1 & E_2 \\
\frac{2}{3} & x(1-x) & \frac{2(2-x)}{3} & \frac{2x-1-x^2}{3}
\end{array}
\]

Table 3: Function \( h(x) \).
We can now invoke Lemma 1 for the numerator and denominator of \( \frac{25}{x} \) separately. Setting \( F^{-1}(x) = y \), notice that the former converges to

\[
\int_0^u w(F^{-1}(x))dx = \int_0^{F^{-1}(u)} w(y)dF(y),
\]

and the latter to

\[
\int_0^1 w(F^{-1}(x))dx = \int_0^\infty w(y)dF(y) = E[w(D)].
\]

Dividing the two, we get \( J(F^{-1}(u)) \).

Assuming \( E[w(D)] < \infty \), (23) defines a CDF. We study its properties next.

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

**Proof.** Observe that the distribution of \( S_n \) is

\[
J_n(x) := P(S_n \leq x) = E\left[ \frac{\sum_{i=1}^{n} w(D_{ni}) \mathbb{1}_{D_{ni} \leq x}}{\sum_{i=1}^{n} w(D_{ni})} \right].
\]

From the law of large numbers,

\[
\frac{1}{n} \sum_{i=1}^{n} w(D_{ni}) \mathbb{1}_{D_{ni} \leq x} \to E[w(D) 1_{D \leq x}], \tag{29}
\]

where \( D \sim F(x) \). Therefore,

\[
\frac{\sum_{i=1}^{n} w(D_{ni}) \mathbb{1}_{D_{ni} \leq x}}{\sum_{i=1}^{n} w(D_{ni})} \to \frac{E[w(D) 1_{D \leq x}]}{E[w(D)]} = J(x). \tag{30}
\]

From the dominated convergence theorem, \( J_n(x) \) has the same limit. \( \square \)

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 \([0, \sum_{i=1}^{n} w(d_i)]\). Then, the length of the interval that the point hits follows the spread distribution. The selected intervals are biased towards larger \( w(d_i) \), which is known as the **inspection paradox** \( 33 \). 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 \( S \sim \text{Erlang}(2) \) and Pareto(\( \alpha, \beta \)) yields

\[
J(x) = 1 - \frac{\beta + \alpha x}{\beta} \left(1 + \frac{x}{\beta}\right)^{-\alpha}, \tag{31}
\]

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 two monotonic permutations (i.e., ascending/descending).

**Theorem 1.** The ascending permutation produces

\[
\lim_{n \to \infty} E[c_n(M, \theta_A)|D_n] = E[g(D)h(J(D))]. \tag{32}
\]

**Proof.** First recall that

\[
E[c_n(M, \theta_A)|D_n] \approx \frac{1}{n} \sum_{i=1}^{n} g(A_{ni}) h(q_i(\theta_A)). \tag{33}
\]

From Lemma 2 term \( q_i(\theta_A) \) is asymptotically close to \( J(F^{-1}(i/n)) \). Defining \( \phi_n(x) = h(J(F^{-1}(x))) \) and again applying (21), we obtain

\[
\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} g(A_{ni}) \phi_n(i/n) = \int_0^{1} h(J(F^{-1}(u))) g(F^{-1}(u))du.
\]

Using \( D = F^{-1}(U) \), this equals \( E[h(J(D))g(D)] \).

This means that all sequences \( D_n \) lead to the same result as \( n \to \infty \). The descending case is handled similarly since \( q_i(\theta_D) = 1 - q_i(\theta_A) \). This means \( q[i/n] \) converges to \( 1 - J(F^{-1}(u)) \) and consequently

\[
\lim_{n \to \infty} E[c_n(M, \theta_D)|D_n] = E[g(D)h(1 - J(D))]. \tag{34}
\]

Applying these observations and recalling Table 3

\[
\lim_{n \to \infty} E[c_n(T_1, \theta_A)|D_n] = \frac{E[g(D)J^2(D)]}{2}, \tag{35}
\]

\[
\lim_{n \to \infty} E[c_n(T_1, \theta_D)|D_n] = \frac{E[g(D)(1 - J(D))^2]}{2} \tag{36}
\]

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

\[
\lim_{n \to \infty} E[c_n(T_2, \theta_D)|D_n] = E[g(D)J(D)(1 - J(D))]. \tag{37}
\]

Interestingly, \( T_1 \) under \( \theta_A \) is finite if \( \alpha > 2 \), which is shown by expanding the integral in (35) using \( 1 - J(x) \sim x^{1-\alpha} \) from (31). This implies that asymptotically \( \theta_A \) offers little benefit over omitting orientation, where the cost is \( E[D^2 - D]/2 \). Next, \( T_2 \) in (37) is finite if \( \alpha > 1.5 \), which is a noticeable improvement over (35). But this is eclipsed by \( T_1 \) under \( \theta_D \), 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 (32) and (34) hold regardless of truncation. For finite \( n \), however, accurate results are guaranteed only for AMRC cases. In these cases, Theorem 1 yields the following approximation to the exact cost across all degree sequences drawn from \( F_n(x) \)

\[
E[c_n(M, \theta_A)] \approx E[g(D_n)h(J(D_n))], \tag{38}
\]

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 modeling 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 \( \sigma \)-algebra \( \Sigma \), function \( \nu : \Sigma \to \mathbb{R} \) is called a measure if a) \( \nu(C) \geq 0 \) for all elements \( C \in \Sigma; b) \nu(\emptyset) = 0; c) \nu(\cup_{i=1}^\infty C_i) = \sum_{i=1}^\infty \nu(C_i) \) for pairwise disjoint \( C_i \in \Sigma \).

It is sufficient for us to consider just two cases. For finite sets, the \( \sigma \)-algebra consists of all subsets of \( S \), in which case \( \nu(C) = |C| \). For continuum intervals, we use the Lebesgue measure \( \nu([a,b]) = b - a \).
**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 \(K(\cdot \mid u) = 0\), 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 \(\nu\) and \(U\) be a uniformly random variable in \(S\). Kernel \(K(v; u)\) is called measure-preserving if it satisfies for all \(v \in \mathbb{R}\)
\[
E[K(v; U)] = \frac{\nu\{s \in S : s \leq v\}}{\nu(S)},
\]
in which case the corresponding variable \(\xi(u) \sim K(v; u)\) is a random map.

For \(S = [0, 1]\), which we often use below, this definition says that the set of points that map to the interval \([0, v]\) has measure exactly \(v\). Note that any deterministic permutation over finite sets (i.e., a bijection) has a measure-preserving kernel.

Suppose a sequence of functions \(\theta_n : [1, n] \to [1, n]\) specifies a relationship between the position in the ascending-order vector \(A_n\) and that in the permuted sequence. If these functions are deterministic, \(j = \theta_n(i)\) means that \(d_j(\theta_n) = A_{ni}\). Since each \(\theta_n\) is a bijection, its inverse exists and satisfies \(\theta_n^{-1}(\theta_n(i)) = i\). For more general cases, where specifying a deterministic relationship is inconvenient, \(\theta_n\) may be random. If so, \(\theta_n^{-1}(j)\) has the following marginal probability mass function (PMF)
\[
P(\theta_n^{-1}(j) = i) = \frac{P(\theta_n(i) = j)}{\sum_k P(\theta_n(k) = j)}.
\]

Either way, there must exist a measure-preserving kernel \(M_n(j; i)\) such that \(P(\theta_n(i) \leq j) = M_n(j; i)\). 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.,
\[
K_n(v; u) := \frac{1}{2k(n) + 1} \sum_{i = -k(n)}^{k(n)} M_n(nu; nu + i),
\]
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 \(\theta_B\) for even), most reasonable permutation sequences are admissible. For such cases, the limiting behavior of \(\theta_n\) is a random process \(\{\xi(u)\}\) with distribution \(K(\xi(u) \leq v)\). Note that if \(\{\xi(u)\}\) 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 \([21]\) beyond monotonic \(\theta_n\).

**Lemma 3.** For an admissible sequence \(\{\theta_n\}\),
\[
\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} g(d_i(\theta_n)) h \left( i/n \right) = E \left[ g(F^{-1}(U)) h(\xi(U)) \right].
\]

**Proof.** For brevity, we only consider deterministic \(\theta_n\). The random case is proven similarly. Transforming the degree sequence back to ascending, we have
\[
\frac{1}{n} \sum_{k=1}^{n} g(d_k(\theta_n)) h(k/n) = \frac{1}{n} \sum_{i=1}^{n} g(A_n, \theta_n^{-1}(k)) h(k/n)
\]
\[
= \frac{1}{n} \sum_{i=1}^{n} g(A_n) h(\theta_n(i)/n)
\]
\[
= \frac{1}{n} \sum_{i=1}^{n} g(A_n) \phi_n(i/n),
\]
where \(\phi_n(u) = h(\theta_n([nu])/n)\). Since \(\{\xi(u)\}\) converges, it is not difficult to show that for all \(t\)
\[
\lim_{n \to \infty} \int_{0}^{t} \phi_n(u) du = \int_{0}^{t} E[h(\xi(u))] du,
\]
which in the context of \(\{\mu\}\) means \(\phi(u) = E[h(\xi(u))]\). Applying \(\{\xi(n)\}\) yields convergence of \(\{\xi(u)\}\) to
\[
\int_{0}^{t} g(F^{-1}(u)) h(\xi(u))] du,
\]
which is the same as \(\{\xi(u)\}\).

Re-writing \(\{\mu\}\) as \(E[g(D)h(\xi(F(D)))\]) and using the same logic as in Theorem \(\{\xi(n)\}\) we get the following.

**Theorem 2.** For an admissible sequence \(\{\theta_n\}\),
\[
c(M, \xi) := \lim_{n \to \infty} E[c_n(M, \theta_n)D_n]
\]
\[
= E[g(D)h(\xi(J(D)))].
\]

For AMRC graphs, we can use the limiting map \(\xi(u)\) to achieve accurate cost analysis even under finite \(n\)
\[
E[c_n(M, \theta_n)] = E[g(D_n)h(\xi(J_n(D_n)))].
\]

Not surprisingly, the ascending map \(\xi(u) = u\) and descending \(\xi(u) = 1-u\) produce in \(\{\xi(n)\}\) the already-established \(\{\xi(n)\}\). Additional cases are covered next.

**5.3 Non-Monotonic Permutations**

Besides \(\theta_A\) and \(\theta_D\), another previously used option is uniform \(\theta^x\), where the direction of edges in the acyclic orientation is based on original (e.g., hashed) node IDs. In this case, \(\xi(u)\) is a uniform variable in \([0, 1]\), independent of the starting position \(u\). This yields
\[
c(M, \xi) := 5E[g(D)h(U)] = 5E[D^2 - D] E[h(U)],
\]
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 \(q_i(\theta_n)(1 - q_i(\theta_n))\). This leads to a new permutation we call Round-Robin (RR)

\[
\theta_n(i) = \begin{cases} 
\frac{i + nu}{2} & \text{if } i \text{ is odd} \\
\frac{i - nu}{2} + 1 & \text{if } i \text{ is even}
\end{cases}
\]

(49)

whose map is derived next.

**Proposition 6.** Permutation (49) converges to a random map \(\xi_{RR}(u)\) that equals \((1 - u)/2\) or \((1 + u)/2\), each with probability 1/2.

**Proof.** Since \(\theta_n\) is deterministic, its kernel \(M_n(j; i) = 1_{\theta_n(i) \leq j}\). Fix \(u \in [0, 1]\) and consider

\[
\begin{align*}
\theta_n([nu]) & = \frac{1}{n} \left\lfloor \frac{i + nu}{2} \right\rfloor + 1 \quad \text{if } [nu] \text{ is even} \\
\theta_n([nu]) & = \frac{1}{n} \left\lfloor \frac{i - nu}{2} \right\rfloor + 1 \quad \text{if } [nu] \text{ is odd}
\end{align*}
\]

(50)

In the \(k(n)\)-neighborhood of \([nu]\), half the normalized values in \([nu]\) are mapped to \((1 + u)/2 + o(1)\) and half to \((1 - u)/2 + o(1)\). As \(n \to \infty\), this permutation converges to a probability kernel \(K(u)\) that defines random variables \(\xi(u)\) equal to \((1 \pm u)/2\), each with probability 1/2. □

Re-writing (49), we have a general result

\[
c(M, \xi_{RR}) = E[g(D)\left(h\left(\frac{1 - J(D)}{2}\right) + h\left(\frac{1 + J(D)}{2}\right)\right)].
\]

(51)

Applying this to \(T_2\) yields

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

(52)

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 (56) and (57) leads to

\[
c(E_1, \xi_{RR}) = \frac{E[g(D)(1 - J^2(D))]}{2}
\]

(53)

and expanding (51) using \(h(x) = x(2 - x)/2\) produces

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

(54)

While (53) is finite iff \(\alpha > 1.5\), this cannot be said about (54), 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 \(\theta_n\), 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 (46), 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 to create a framework for minimizing (46) over all measure-preserving maps \(\{\xi(u)\}\).

**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. This implies that \(J(S)\), where \(S \sim J(x)\), is a uniform variable in \([0, 1]\). For heavy-tailed distributions, such as Pareto, this assumption holds iff \(E[w(D)] < \infty\). For the results that follow, 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 \(g(x)/w(x)\).

**Lemma 4.** Model (15) can be written as

\[
c(M, \xi) = E[w(D)]E[r(U)h(\xi(U))],
\]

(55)

where \(U\) is uniform in \([0,1]\).

**Proof.** Rewriting (46),

\[
c(M, \xi) = E[w(D)] \int_0^\infty \frac{g(x)}{w(x)} E[h(\xi(J(x)))]dJ(x)
\]

\[
= E[w(D)] \int_0^1 \frac{g(J^{-1}(u))}{w(J^{-1}(u))} E[h(\xi(u))]du.
\]

(56)

Define \(r(u) = g(J^{-1}(u))/w(J^{-1}(u))\). Then, cost becomes

\[
c(M, \xi) = E[w(D)] \int_0^1 r(y)E[h(\xi(y))]dy,
\]

(57)

which is the same as (55). □

Note that (55) 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 fixed method \(M\) and constant \(r(x)\), all permutations yield the same complexity

\[
c(M, \xi) = E[g(D)]E[h(U)].
\]

(58)

**Proof.** Suppose \(r(x) = b\) is some constant. Then,

\[
c(M, \xi) = bE[w(D)]E[h(\xi(U))] = E[g(D)]E[h(\xi(U))].
\]

Since \(\xi\) is measure-preserving and maps \([0, 1] \to [0, 1]\), it immediately follows that \(E[h(\xi(U))] = E[h(U)]\), which leads to (55). □

Interestingly, this is exactly the same overhead as under the random permutation in (48). When \(r(x)\) deviates from
Algorithm 1: Construction of optimal permutations

1. Function OPT (h)
2. for (i = 1; i ≤ n; i++) do
3.   z[i].key = h(i/n)
4.   z[i].index = i
5. if r(x) is an increasing function then
6.   sort array z descending by key
7. else
8.   sort array z ascending by key
9. for (i = 1; i ≤ n; i++) do
10.  theta[i] = z[i].index
11. return theta

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. It creates a sequence

\[ z := (h(1/n), h(2/n), \ldots, h(1)), \quad (59) \]

which is then sorted in the opposite order of monotonicity of r(x). Suppose z transforms into h(i1/n), h(i2/n), \ldots, h(in/n). Then, the algorithm assigns \( \theta_n(j) = i_j \). When there are ties, they are broken arbitrarily.

**Theorem 3.** When r(x) is monotonic, Algorithm 1 builds permutations that minimize \( c_{\text{OPT}} \).

**Proof.** Assume the result of OPT is permutation \( \theta_n \). We now show that no other permutation \( \gamma_n \) can have lower cost. Assume that \( \gamma_n \) places \( A_{n1}, \ldots, A_{nk-1} \), where \( k \geq 1 \), in the same position as \( \theta_n \). This implies that \( A_{nk} \) is the smallest degree in \( \gamma_n \) that violates order \( \theta_n \). Define \( j = \gamma_n^{-1}(\theta_n(k)) \) to be the node that is currently occupying \( A_{nk} \)’s position in \( \theta_n \). Note that \( j > k \). If \( r(x) \) is increasing,

\[ h(\gamma_n(k)/n) \leq h(\theta_n(k)/n) = h(\gamma_n(j)/n); \quad (60) \]

otherwise, the inequality is reversed.

Define \( \tilde{\gamma}_n \) to be a permutation that equals \( \gamma_n \), except it swaps the positions of nodes \( k \) and \( j \). We intend to prove that \( \tilde{\gamma}_n \) produces lower cost. For convenience of argument, \( \tilde{\gamma}_n \) can be represented as the limit of

\[ c_{\tilde{n}}(M, \tilde{\gamma}_n) = E[w(D)] \frac{1}{n} \sum_{i=1}^{n} r\left( \frac{i}{n} \right) h\left( \frac{\gamma_n(i)}{n} \right). \quad (61) \]

Let the cost difference under the two permutations be

\[ \pi_n = n \frac{c_{\tilde{n}}(M, \gamma_n) - c_{\tilde{n}}(M, \tilde{\gamma}_n)}{E[w(D)]}. \quad (62) \]

Using \( \tilde{\gamma}_n(k) = \gamma(j) \) and \( \gamma_n(j) = \gamma(k) \), the swap produces

\[ \pi_n = \left[ r\left( \frac{k}{n} \right) h\left( \frac{\gamma_n(k)}{n} \right) + r\left( \frac{j}{n} \right) h\left( \frac{\gamma_n(j)}{n} \right) - r\left( \frac{j}{n} \right) h\left( \frac{\gamma_n(j)}{n} \right) - r\left( \frac{k}{n} \right) h\left( \frac{\gamma_n(k)}{n} \right) \right] \]

\[ = \left[ r\left( \frac{j}{n} \right) - r\left( \frac{k}{n} \right) \right] h\left( \frac{\gamma_n(j)}{n} \right) - h\left( \frac{\gamma_n(k)}{n} \right). \quad (63) \]

When \( r(x) \) is increasing, both terms are non-negative using \( j > k \) and \( (63) \). When \( r(x) \) is decreasing, both are non-positive. Either way, we get \( \pi_n \geq 0 \). Therefore, \( \tilde{\gamma}_n \) is a better permutation than \( \gamma_n \). Repeating this argument creates a sequence of progressively better permutations that converge to \( \theta_n \) constructed earlier.

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 3 as well as the various relationships in Figures 2 and 4, it follows that \( \theta_D \) is optimal for \( T_1/E_1/E_2 \) and \( \theta_A \) 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 \( T_2 \) is indeed optimized by RR and \( E_4/E_5 \) by CRR. While generally prior papers use experimentation to correctly identify the best permutation for their technique, there are exceptions. As discussed in the introduction, \( [14] \) uses a variation of \( E_3 \) under \( \theta_D \). Based on the next result, \( \theta_D \) is actually the worst possible permutation for \( [14] \).

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

This might explain why \( [23] \) changed the orientation to \( \theta_A \) while implementing the algorithm from \( [14] \).

6.2 Comparison

We now select which vertex iterator is better under their respective optimal permutations. Re-write \( (59) \) and \( (62) \) as

\[ c(T_3, \xi_D) = \frac{E[r(U)(1 - U)^2]}{2}, \quad (64) \]

\[ c(T_2, \xi_{RR}) = \frac{E[r(U)(1 - U)^2]}{4}, \quad (65) \]

and consider the next result.

**Theorem 4.** If \( r(x) \) is increasing, \( (64) \) is smaller than \( (65) \). If \( r(x) \) is decreasing, the inequality is reversed. If \( r(x) \) is a constant, they produce the same cost.

**Proof.** Notice that

\[ c(T_2, \xi_{RR}) - c(T_1, \xi_D) = \frac{E[r(U)(4U - 1 - 3U^2)]}{4}. \]

Function \( \eta(x) = 4x - 1 - 3x^2 \) is negative in \([0, 1/3]\) and positive in \((1/3, 1) \). Since \( E[\eta(U)] = 0 \), we get

\[ - \int_{0}^{1/3} \eta(x)dx = \int_{1/3}^{1} \eta(x)dx. \quad (66) \]

An increasing \( r(x) \) amplifies the negative portion by a smaller amount than the positive. Consequently,

\[ - \int_{0}^{1/3} r(x)\eta(x)dx \leq \int_{1/3}^{1} r(x)\eta(x)dx \quad (67) \]

and \( E[r(U)\eta(U)] \geq 0 \), i.e., \( T_1 \) is better than \( T_2 \). Similar reasoning applies to the other cases of \( r(x) \).
For \( w(x) = \min(x, \alpha) \), we already know that \( r(x) \) is increasing, which implies that \( T_1 \) is faster than \( T_2 \). For edge iterator, we have

\[
c(E_1, \xi_D) = \frac{E[r(U)(1-U^2)]}{2}, \\
c(E_4, \xi_{CRR}) = \frac{E[r(U)(U^2 - 2U + 2)]}{4},
\]

with the corresponding comparison next.

**Theorem 5.** If \( r(x) \) is increasing, (68) is smaller than (3). If \( r(x) \) is decreasing, the opposite holds. If \( r(x) \) is a constant, they are identical.

**Proof.** Observe that

\[
c(E_4, \xi_{CRR}) - c(E_1, \xi_D) = \frac{E[r(U)U(3U - 2)]}{4}. \tag{70}
\]

Define \( \eta(x) = x(3x - 2) \) and notice that it stays negative in \((0, 2/3)\) and positive in \((2/3, 1)\). Since \( E[\eta(U)] = 0 \), we obtain

\[
-\int_0^{2/3} \eta(x)dx = \int_{2/3}^1 \eta(x)dx
\]

where the rest of the argument is identical to that in Theorem 3.

**6.3 Asymptotics**

Our investigation revealed that the two best methods, regardless of graph size or degree distribution, are \( T_1 \) and \( E_1 \), whose optimal cost is

\[
c(T_1, \xi_D) = \frac{E[(D^2 - D)(1 - J(D))^2]}{2}, \tag{72}
\]

\[
c(E_1, \xi_D) = \frac{E[(D^2 - D)(1 - J^2(D))^2]}{2} \tag{73}
\]

Using Pareto \( F(x) \), it was established earlier that (2) is finite iff \( \alpha > 4/3 \) and (72) iff \( \alpha > 1.5 \). This means that as \( n \to \infty \), vertex iterator is provably faster than edge iterator in all graphs with \( \alpha \in (4/3, 1.5] \), regardless of 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 in Table 3.

When \( \alpha \) drops below the corresponding finitness thresholds, the scaling rate of cost is determined by the tail of the spread distribution

\[
1 - J_n(x) \sim \begin{cases} x^{1-\alpha} & \alpha > 1 \\ 1 - \log(x)/\log(t_n) & \alpha = 1 \\ 1 - x^{1-\alpha}/t_n^{1-\alpha} & 0 < \alpha < 1 \end{cases}, \tag{74}
\]

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

\[
a_n = \begin{cases} \log n & \alpha = 4/3 \\ n^{2-1.5\alpha} & 1 < \alpha < 4/3 \\ \sqrt{n}/\log^2 n & \alpha = 1 \\ n^{1-\alpha/2} & 0 < \alpha < 1 \end{cases}. \tag{75}
\]

Figure 6: Comparison of scaling rates.

![Figure 6](image.png)

and \( E[c_n(E_1, \theta_{CRR})]/b_n \to 1 \), where

\[
b_n = \begin{cases} \log n & \alpha = 1.5 \\ n^{1-\alpha} & 1 < \alpha < 1.5 \\ \sqrt{n}/\log n & \alpha = 1 \\ n^{1-\alpha/2} & 0 < \alpha < 1 \end{cases}. \tag{76}
\]

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 unconstrained graphs, which is beyond the scope of our investigation here. In the mean time, (73)-(76) can be used to assess the crudeness of arboricity analysis [14, 29], which places the cost of both vertex/edge iterators at \( O(m^{1.5}/n) \). Under root truncation, simple manipulation produces

\[
\frac{m^{1.5}}{n} \sim \begin{cases} \sqrt{n} & \alpha > 1 \\ \sqrt{n}\log^{-1.5} n & \alpha = 1 \\ n^{5/4-3\alpha/4} & 0 < \alpha < 1 \end{cases} \tag{77}
\]

To visualize the result, suppose the power-law exponent of a function \( f(n) \) is defined as

\[
\lim_{n \to \infty} \frac{\log f(n)}{\log n}.
\]

Then, Figure 6 shows the relationship between the exponents of (66)-(67). Notice that the upper bound is power-law tight at \( \alpha = 1 \) (i.e., differs from the actual rate by a poly-log factor) and pretty loose otherwise. The largest overestimation occurs at \( \alpha > 4/3 \) in \( T_1 \) and \( \alpha > 1.5 \) in \( E_1 \), where the true cost is finite, while (77) grows proportional to \( \sqrt{n} \).

**7. EVALUATION**

We now use finite \( n \), where the models are only approximate, to examine the impact of \( \alpha \), 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 \( F(x) = 1 - (1 + [x/\beta])^{-\alpha} \) defined on natural numbers, which we employ in the construction of random graphs and comparison against the model. In general, (77) expands into a double Lebesgue-Stieltjes integral

\[
\int_0^{\tau n} g(x) \tilde{h}(\xi(x)) \left( \int_0^{\tau n} w(y) dF_n(y) \right) dF_n(x), \tag{79}
\]
Algorithm 2: Quick computation of \( F(x) \)

1. **Function DiscreteModel** (Fn, g, w, x; tn, eps)
2. \( E_n = J = \text{cost} = 0 \)
3. for \((i=1; i \leq t_n; i += \text{jump})\) do \( \triangleright E[D_n] \)
4. \( E_D n += w(i) * (F_n(i + \text{jump}) - F_n(i - 1)) \)
5. for \((i=1; i \leq t_n; i += \text{jump})\) do \( \triangleright E[cw(M, \theta_n)] \)
6. \( \text{jump} = \text{ceil}(\text{eps} * i) \)
7. \( p = F_n(i + \text{jump}) - F_n(i - 1) \)
8. \( J += w(i) * p / E_D n \)
9. \( \text{cost} += w(i) * h(x\{J\}) * p \)
10. return cost

<table>
<thead>
<tr>
<th>n</th>
<th>( F^*(x) ) in ([82])</th>
<th>( F(x) ) in ([82])</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^3)</td>
<td>353.92</td>
<td>7.8</td>
<td>346.92</td>
</tr>
<tr>
<td>(10^4)</td>
<td>359.85</td>
<td>9.0</td>
<td>352.73</td>
</tr>
<tr>
<td>(10^5)</td>
<td>362.08</td>
<td>7.9</td>
<td>354.94</td>
</tr>
<tr>
<td>(10^6)</td>
<td>363.06</td>
<td>8.0</td>
<td>355.79</td>
</tr>
<tr>
<td>(10^7)</td>
<td>363.56</td>
<td>8.2</td>
<td>356.26</td>
</tr>
<tr>
<td>(10^8)</td>
<td>363.57</td>
<td>8.2</td>
<td>356.28</td>
</tr>
<tr>
<td>(10^9)</td>
<td>363.57</td>
<td>8.3</td>
<td>356.28</td>
</tr>
</tbody>
</table>

Table 4: Model results and computation time (in seconds) for \( T_1 \) under descending order (\( \alpha = 1.5, \epsilon = 10^{-5}, \) linear truncation).

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

\[
\sum_{i=1}^{t_n} g(i) h(\xi((\sum_{k=1}^{j} w(j)p_j)/p_i)) p_i, \tag{80}
\]

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

Even though \([82]\) contains two nested sums and appears to exhibit quadratic complexity, it can be computed in linear time and \( O(1) \) space. However, when \( F(x) \) decays to zero very slowly as \( x \to \infty \) and \( t_n = n - 1 \), linear complexity may be too high for usage in practice, especially when the limiting cost is of interest. For such cases, the runtime can be improved to \( O((1 + \log(\epsilon t_n))/\epsilon) \) by compressing all summation terms from large intervals \([i, (1 + \epsilon) i] \) into a single term, where \( 1/t_n \leq \epsilon < 1 \) is a chosen parameter. This is demonstrated in Algorithm 2 where \( \epsilon = 1/t_n \) 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. First, using two decimal digits of precision, notice that the continuous model does not converge until \( n = 10^{14} \). Calculation for this value of \( n \) would require an extrapolated four months in the exact model \([82]\). Second, observe that Algorithm 2 with \( \epsilon = 10^{-5} \) produces results within 0.003\% of the exact summation, but in a fraction of a second due to its log \( n \) complexity. Finally, the table shows that the continuous model overestimates the discrete version by non-negligible amounts (i.e., 1.5 - 2\%).

Aimed with these results, we next show how to efficiently compute \([2]\) from \([11]\). On the surface, this model requires a triple nested summation

\[
\frac{1}{2E^2[D_n]} \sum_{i=1}^{t_n} \sum_{j=i+1}^{t_n} \sum_{k=i+1}^{t_n} (i^2 - i)jkp_j p_k, \tag{81}
\]

which is \( \Theta(n^3) \) complexity. However, the next result shows that after rearranging the terms, this formula is identical to our result for \( T_1 \) under \( \theta_D \). Therefore, Algorithm 2 applies to \([31]\) as well.

**Proposition 9.** For both discrete and continuous \( F(x) \), model \([2]\) is the same as \([31]\) under \( w(x) = x \).

**Proof.** We only consider the discrete case, omitting the continuous version due to similarity. Using \( w(x) = x \), first rewrite \([24]\) as

\[
J_n(i) = \frac{1}{E[D_n]} \sum_{j=1}^{i} j p_j, \tag{82}
\]

where \( E[D_n] = \sum_{i=1}^{t_n} j p_j \), and represent the finite version of \([30]\) as

\[
\rho := \frac{1}{2} \sum_{i=1}^{t_n} g(i)(1 - J_n(i))^2 p_i. \tag{83}
\]

Then,

\[
\begin{align*}
\rho &= \frac{1}{2} \sum_{i=1}^{t_n} g(i) p_i \left( 1 - \frac{1}{E[D_n]} \sum_{j=1}^{i} j p_j \right)^2 \\
&= \frac{1}{2} \sum_{i=1}^{t_n} g(i) p_i \left( \sum_{j=i+1}^{t_n} j p_j - \sum_{i=1}^{j} j p_j \right)^2 \\
&= \frac{1}{2E^2[D_n]} \sum_{i=1}^{t_n} g(i)p_i \left( \sum_{j=i+1}^{t_n} j p_j \right)^2 \\
&= \frac{1}{2E^2[D_n]} \sum_{i=1}^{t_n} \sum_{j=i+1}^{t_n} \sum_{k=i+1}^{t_n} g(i) j k p_j p_k
end{align*}
\]

Recalling that \( g(x) = x^2 - x \), we get \([31]\). □

### 7.2 Random Graph Generation

Traditional methods \([10, 31]\) 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. However, this 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 arbitrary values in the range \([1, d_i]\), simulations will not match theoretical predictions of \( E[X_i(\theta_D)] \), particularly under permutations with many small \( X_i(\theta_n) \).

To overcome this problem, we employ a variation of the method from [13] that picks neighbors in proportion to their residual degree and excludes the already-attached neighbors 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. Due to limited space, we omit further details, but note that, 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 \( D_n \) 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 \( D_n \), each with 100 random graphs \( G_n \) (i.e., 10K graph instances total). We keep Pareto \( \beta = 30(\alpha - 1) \), which yields \( E[D] = 30.5 \) after discretization. We use \( \delta \) for finite \( n \), while the corresponding limits as \( n \to \infty \) are provided by Algorithm 2. Unless mentioned otherwise, we use \( w(x) = x \).

Tables 6 and 7 examine two cases of \( \alpha < 2 \) under root truncation. Since these graphs deterministically limit the maximum degree to \( \sqrt{n} \), \( \delta \) is accurate even for tiny \( n \). Note that simulation results averaged over 10K iterations are still pretty noisy, which explains the random fluctuation of error once it drops below 1\%. Table 8 shows another AMRC scenario, where \( \alpha = 2.1 \) and truncation is linear. Because these graphs are only asymptotically constrained, some discrepancy for small \( n \) was expected. However, even the slowest-converging method studied here (i.e., \( T_2 + \theta_{RR} \)) exhibits less than 1\% error for \( n \geq 1M \) nodes.

### 7.4 Unconstrained Degree

We now transition to more challenging cases. Table 8 revisits the data in Table 5 under linear truncation. Both permutations now produce larger cost and quicker convergence towards their respective limits, which is especially noticeable under \( \theta_D \). A similar scenario for \( T_2 \) is shown in Table 9, where the error is much larger than previously in Table 6, 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., \( c(M, \xi) = \infty \). This arises from the difference in the rates at which simulations and \( \delta \) 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 \( w_2(x) = \min(x, \sqrt{\pi}) \).

Table 6 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.

While both functions \( w_1, w_2 \) have the same limit as \( n \to \infty \), which can be shown using root-truncated \( F_2(x) \), the convergence speed is clearly different. As discussed earlier, finding \( w(x) \) that keeps \( \delta \) 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.

### 7.5 Real Graphs

We now examine our conclusions and model applicability to Twitter [28], which is a standard graph in this field with \( n = 41M \) nodes and \( m = 1.2B \) edges (9.3 GB). Its degree distribution is Pareto with \( \alpha = 1.27 \) and \( \beta = 15.2 \), while the average degree is \( E[D_n] = 57 \). In addition to the five main permutations, i.e., ascending \( \theta_A \), descending \( \theta_D \), round-robin \( \theta_{RR} \), complementary round-robin \( \theta_{CRRR} \), and uniform \( \theta_U \), we also consider the degenerate option, built using the algorithm from [30]. Because this orientation re-
pared to the scenario in Table 12, there is an additional
result using its empirical degree distribution
within random graphs. However, all other cases can be accurately predicted from
simulations and that directly computed from Twitter. Re-
elected would have doubled the cost of T
layer of randomness because (80) considers all possible de-
gree sequences drawn from F_m(x). Table 13 demonstrates
that w_1(x) significantly overestimates the interesting cases,
which is similar to the α = 1.2 outcome in Table 13. The un-
constrained nature of degree in Twitter is confirmed by the
presence of many values (i.e., 14K) exceeding √n. The good
news is that w_2(x) can be again leveraged to substantially
reduce this deviation, as shown in Table 14.

8. CONCLUSION
Our efforts produced the first accurate result on the asymp-
totics of triangle listing under arbitrary permutations. We
proved that expected cost could be optimized using four dif-
ferent 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 edge-iterator. In many cases
the winner will be determined by the edge-existence veri-
fication speed of these algorithms; however, we discovered
degree sequences for which the former would always outper-
form the latter as n → ∞.

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.

9. REFERENCES
Model for Massive Graphs,” in Proc. ACM STOC, May