Efficient and Accurate Triangle Count Estimation in Large Networks

Sophia Hubscher

Robert and Donna Manning College of Information and Computer Sciences

University of Massachusetts Amherst

shubscher@umass.edu

Committee Chair: Professor Cameron Musco cmusco@cs.umass.edu

Second Committee Member: Professor Ghazaleh Parvini gparvini@cs.umass.edu

Research Type: Thesis

Abstract

Counting triangles in large networks is a fundamental problem in graph theory with significant applications in social networks, bioinformatics, and beyond. However, exact enumeration methods are computationally expensive for large-scale graphs. This thesis explores randomized algorithms for efficient and accurate triangle count estimation, focusing on sampling-based techniques such as uniform sampling, importance sampling, variance reduction, and a hybrid approach. These methods are evaluated on diverse real-world and synthetic datasets, including social networks, collaboration networks, and web graphs. Results demonstrate that the hybrid method achieves the lowest error rates, while importance sampling offers the best balance between accuracy and runtime. Theoretical analyses of variance as well as simulated tests of different shapes data could take help to explain importance sampling doing so well. The findings highlight the potential of randomized algorithms to enable scalable and precise triangle counting in large networks.

Contents

1 Introduction

Counting triangles is a fundamental problem in graph theory with widespread applications in social networks, bioinformatics, and more [13]. Triangles, are formed by three mutually connected nodes, as shown in Figure 1, which shows a graph containing three triangles. While these triangles appear simple, they are a powerful structural motif that can reveal important insights the networks they are found in.

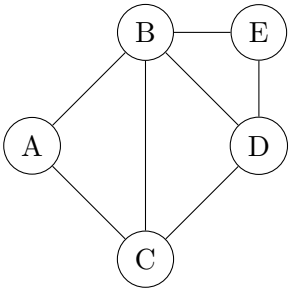


Figure 1: Graph with triangles formed between vertices (A, B, C), (B, C, D) and (B, D, E).

In social network graphs, for example, they can represent closed friendships or tightly-knit groups, signaling levels of local connectivity in a network. This, in turn, can reflect greater patterns and structures within a network. For example, in social media platforms, triangles are used to model relationships between users, where closed triangles indicate strong communities or mutual interests. A study analyzing the effect of recommender systems on X (formerly Twitter) demonstrated how an increase in closed triangles following the introduction of a "Who to Follow" friend-to-friend recommendation algorithm served as evidence of the algorithm's efficacy [20].

Additionally, triangles can be used to understand relationships within biological networks. For example, a study on yeast protein interaction networks used analysis of triangles to find transitive relationships between genes and proteins [24]. The researchers constructed graphs called "genetic congruence networks," connecting genes that shared similar interaction partners. These networks showed a higher-than-expected occurrence of triangles, indicating a strong correlation between genetic congruence and protein interactions. This suggests that triangles can capture important

structural patterns, such as proteins that function within the same biological pathway or complex.

While the utility of triangle-based metrics is well-documented, counting triangles efficiently in large graphs remains computationally challenging. Direct enumeration methods involve inspecting all possible triples of nodes in the graph, a process with a worst-case time complexity of $O(n^3)$ where n is the number of nodes [1]. With sparse graphs, where the number of edges is much smaller compared to the number of possible edges (as illustrated in Figures 2 and 3), efficiently counting these triangles can be done more efficiently. In these cases, an alternate exact counting algorithm runs in $O(nd_{max}^2)$ time, where d_{max} is the maximum node degree [17].

On smaller graphs, these runtimes may be acceptable, but as graphs grow larger and more complex, direct methods for counting triangles become increasingly time-consuming, making it difficult to handle graphs of practical size in real-world applications. This issue is particularly relevant in the era of big data, where networks of millions or even billions of nodes and edges are common, and computational efficiency is critical.

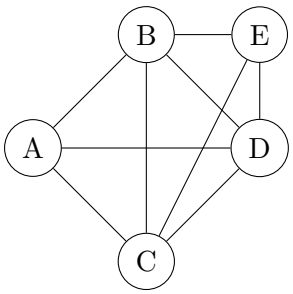


Figure 2: Dense Graph with many edges relative to the number of nodes.

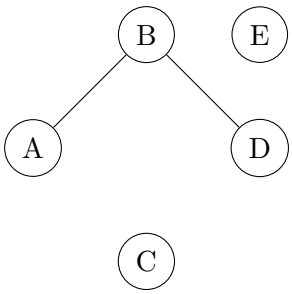


Figure 3: Sparse Graph with few edges relative to the number of nodes.

To address these challenges, researchers have developed a variety of approaches to count triangles efficiently. Some deterministic methods outlined in more detail in Section 3 decrease the time it takes to compute exact triangle counts [19]. However, these methods still face scalability issues. As a result, randomized algorithms [18, 21, 22] have emerged as a promising alternative. By leveraging probabilistic techniques, these algorithms provide approximate triangle counts with significant reductions in runtime while maintaining a high degree of accuracy.

This thesis aims to introduce new variations of randomized algorithms for triangle counting that

improve upon existing methods in both accuracy and efficiency, with a particular emphasis on scalability to large, real-world networks.

2 Notation

We first introduce the notation that will be used throughout this work.

Table 1: List of notation used.

Symbol	Description
$G(V, E)$	Graph with V vertices and E edges.
$n = \|V\|$	Number of vertices in graph G.
$m = \|E\|$	Number of edges in graph G.
A	The adjacency matrix for the graph G.
Δ_i	Number of triangles node i participates in.
Δ	Total number of triangles in G.
d_i	Degree of node i.
s	Number of nodes sampled.

In more detail, throughout, we consider a graph $G = (V, E)$ where:

- G is an undirected, unweighted, simple graph (no self-loops or multiple edges).
- $n = |V|$ is the number of vertices in G.
- $m = |E|$ is the number of edges in G.
- The adjacency matrix $A \in \{0, 1\}^{n \times n}$ is defined such that

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

- The degree of node i is denoted d_i, where $d_i = \sum_{j=1}^{n} A_{ij}$.
- The total degree sum satisfies $\sum_{i=1}^{n} d_i = 2m$, since each edge contributes to the degree of both its endpoints.
- The number of triangles node i participates in is denoted Δ_i.

- The total number of triangles in G is denoted Δ, and satisfies

$$\Delta = \frac{1}{3} \sum_{i=1}^{n} \Delta_i,$$

  since each triangle is counted once at each of its three vertices.

- s is the number of nodes sampled in G.

3 Background

This section reviews fundamental concepts in graph theory, classical and modern methods for triangle counting, and algorithmic strategies to enhance efficiency, setting the stage for our exploration of scalable estimation techniques.

3.1 Types of Graphs

In graph theory, graphs are classified as either directed or undirected. An *undirected graph* is one in which edges have no specific direction, so the relationship between connected nodes is mutual: If u connects to v, then v connects to u. In contrast, a *directed graph*, or digraph, has edges with a defined direction—u may point to v without v pointing to u. This directional property is particularly relevant when calculating triangle counts, as a triangle in a directed graph can follow a specific directional sequence. In this discussion, we will only discuss undirected graphs, although the methods described can be extended to directed graphs as well.

3.2 Graph Motifs

Graph motifs are subgraphs that occur frequently within a larger graph and carry significant structural information. One such motif is the *clique*, a subset of vertices such that every pair of vertices is connected by an edge. Triangles, for example, are a clique, as they consist of 3 mutually-connected nodes. Another common example of a clique is the *4-clique*, denoted as K_4, which consists of four vertices with edges connecting each pair of vertices.

Formally, we define a 4-clique, K_4, as the complete graph on four vertices, meaning every pair of vertices is connected by an edge. The set of vertices V and edges E for K_4 are given by $K_4 = (V, E)$ where $V = \{v_1, v_2, v_3, v_4\}, E = \{(v_i, v_j) \mid 1 \leq i < j \leq 4\}$.

Figure 4 illustrates the structure of K_4, where each vertex is connected to every other vertex, representing the maximal level of connectivity between four vertices.

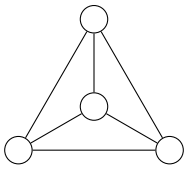


Figure 4: The complete graph K_4 on four vertices.

While this thesis focuses on triangle counting, analysis of other motifs can aide us in seeing how well methods generalize.

3.3 Methods for Triangle Counting

Triangle counting can be approached in a variety of ways, each with its own advantages and disadvantages. One of the simplest methods is the brute force technique, where all distinct sets of three vertices u, v, w are enumerated and checked for the existence of a triangle. This involves examining every possible combination of vertices (u, v, w) in the graph and testing whether all three edges (u, v), (v, w), and (w, u) exist.

Assuming optimal conditions with edges stored in a hash table, where edge retrieval takes $O(1)$ time, the time complexity of this brute force approach is $\Theta(n^3)$. This complexity stems from the fact that $\binom{n}{k} = \Theta(n^k)$, and thus, $\binom{n}{3} = \Theta(n^3)$ [1], so there are $O(n^3)$ checks and each takes $O(1)$ time.

While this method is straightforward, it is inefficient for large graphs due to its high computational cost. Additionally, this method is no more efficient on sparse graphs (those with relatively few edges compared to the maximum number of edges possible) than on dense ones.

To improve on this, researchers have developed more efficient direct counting techniques that take advantage of sparsity. Specifically, the Node-Iterator [17] algorithm takes each nodes and examines which pairs of its neighbors are connected. This method thus runs in $O(nd_{max}^2)$ time where n is the number of nodes in the graph G and d_{max} is the maximum node degree [17]. Thus, for graphs with generally low degree, this algorithm performs quickly.

still, with large n, this algorithm may perform too slowly for practical applications. Thus, researchers have turned to alternative triangle counting and estimation methods.

3.3.1 Sampling Methods

One of the most effective ways to estimate triangle counts in large, sparse graphs is through sampling methods. These methods rely on randomly selecting edges or vertices and then inspecting their local neighborhoods for the presence of triangles. Sampling-based techniques are particularly useful in scenarios where calculating the exact triangle count is computationally expensive or unnecessary.

Additionally, sampling algorithms often provide tunable accuracy, allowing for a trade-off between precision and performance, making them ideal for processing large-scale networks. We detail some of the main sampling approaches below.

Edge Sampling In edge sampling, we randomly sample a subset of edges from the graph, count the number of triangles in the subgraph, and scale up to reach our estimate.

One key edge sampling algorithm is Doulion [22], in which each edge in our graph G is sampled with probability p. As all triangles consist of three edges, this means that all triangles in G have probability p^3 of being counted. Thus, the number of triangles counted is scaled by $\frac{1}{p^3}$ to achieve a final estimate.

Other algorithms extend this even further. For example, a parallel implementation of Doulion [3], where each processor independently sparsifies its assigned partition of the graph, can improve speed.

In all of these algorithms though, the key piece of their efficiency and efficacy is the sampling of edges to get a good picture of the graph's structure without counting every triangle individually.

Wedge Sampling Wedge sampling [18] focuses on wedges—triplets of nodes that form two edges but not necessarily a triangle. A wedge is defined by three vertices (u, v, w) where u is adjacent to both v and w, but v and w may or may not be adjacent (see Figure 5).

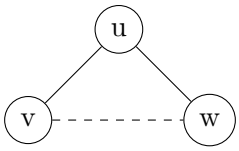


Figure 5: Wedge formed by vertices u, v, and w. Nodes v and w may or may not be connected.

First, the algorithm counts the total number of wedges in the graph. To count these wedges, only one pass over all nodes is required, as at each node, every unique pair of outgoing edges from the node is counted as a single wedge. There are $\binom{d_i}{2}$ such wedges for each node n_i, and thus the total number of wedges is $\sum_{i=1}^{n} \binom{d_i}{2}$. Thus, this operation takes $O(m)$ time where m is the number of edges in G. Once wedges are sampled, the algorithm checks how many of them are closed (i.e., form triangles). The number of triangles can then be estimated by multiplying the number of total wedges by the fraction of all wedges that were closed in the sample. This is equivalent to the Node-Iterator method mentioned in Section 3.3. Thus, this method, as we saw earlier, also runs in $O(nd_{max}^2)$ time, and is an appropriate option for sparse graphs with low maximum degrees.

3.3.2 Linear Algebraic Methods

Along with sampling, we can employ linear algebraic techniques to increase the speed of triangle counting.

Graphs can be conveniently represented using adjacency matrices, which, in social network analysis, are typically referred to as *sociomatrices* [5]. In these matrices, each row and column represents a node, and edges between nodes are represented as 1s in the corresponding matrix entry.

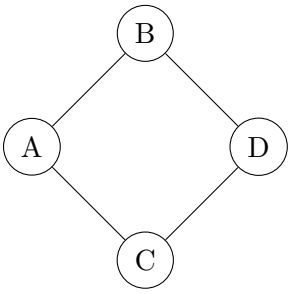


Figure 6: Graph representation of vertices A, B, C, and D.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 7: Adjacency matrix corresponding to the graph.

By using these adjacency matrices and leveraging linear algebra techniques, we can calculate triangle counts more efficiently. One simple method using the adjacency matrix is to use the following formula where A is the adjacency matrix corresponding to the graph G and Δ is the global triangle count in G:

$$\Delta = \frac{1}{6}\text{trace}(A^3)$$

This formula is derived from the fact that the diagonal elements of A^3 count the number of length-three paths (i.e. triangles) from each vertex to back to itself. Each triangle can be formed from six of these length-three paths, as each triangle can be drawn starting at any of its three nodes and moving either clockwise or counter-clockwise, as illustrated in Figure 8. Thus, the trace of A^3 is divided by six to scale down to the global triangle count.

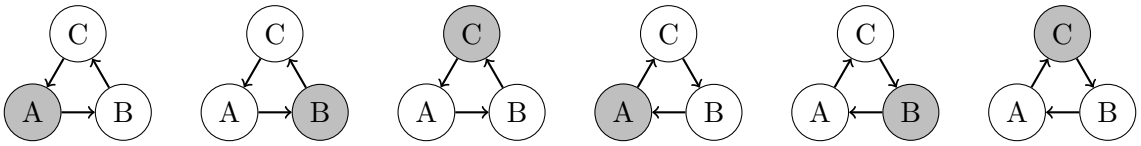


Figure 8: Six different ways to arrive at a length-three path in a triangle.

To compute A^3, we first need to calculate A^2 (which takes $O(n^3)$ time for an $n \times n$ matrix, n thus also being the number of nodes in our graph G) and then multiply A^2 by A (also $O(n^3)$). Thus, the total complexity for computing A^3 is $O(n^3)$. After computing A^3, calculating the trace takes $O(n)$ time, as we need to iterate over the n diagonal elements. Thus, the overall runtime complexity for the operation is $O(n^3)$. While this is not a direct improvement over the runtime of the naive algorithm, this strategy forms the basis of many faster methods, which we discuss below.

Strassen's Algorithm This runtime analysis above assumes that matrix multiplication is performed using the standard algorithm. However, more sophisticated techniques, such as Strassen's algorithm [19], can reduce matrix multiplication time. In this algorithm, that is used on large, square matrices, such as undirected sociomatrices, each matrix is divided into smaller submatrices on which a series additions and multiplications are performed.

Specifically, Strassen's algorithm reduces the complexity of multiplying two $n \times n$ matrices to approximately $O(n^{\log_2 7})$, which is about $O(n^{2.81})$. Computing A^2 using Strassen's algorithm will take $O(n^{\log_2 7})$. Then, multiplying A^2 by A again takes $O(n^{\log_2 7})$ time. Therefore, the total complexity for computing A^3 with Strassen's algorithm is $O(n^{\log_2 7}) + O(n^{\log_2 7}) = O(n^{\log_2 7})$, or roughly $O(n^{2.81})$.

To contextualize this, on a 2×2 matrix, the n^3 multiplications required for the naive method would mean we would complete $2^3 = 8$ multiplications. With Strassen's method and its $n^{\log_2 7}$ multiplications, there would instead only be $2^{\log_2 7} = 7$ multiplications computed. On larger matrices, this leads to a significant speedup.

There are matrix multiplication algorithms that are even faster, such as one with a $O(n^{2.371552})$ runtime, but these algorithms rely on the use of extremely large constants and are thus rarely used in real-world applications [23].

EigenTriangle Another significant approach in triangle counting is the use of spectral methods. One such method is the EigenTriangle algorithm [21], which estimates the triangle count Δ by considering the spectral decomposition of the adjacency matrix A. In particular, the adjacency matrix A can be decomposed as:

$$A = U \Lambda U^T,$$

where U is a matrix whose columns are the orthonormal eigenvectors of A, and Λ is a diagonal matrix containing the corresponding eigenvalues.

Once the decomposition is performed, the number of triangles can be computed exactly using $\Delta = \frac{1}{6}\text{trace}(A^3) = \frac{1}{6}\sum_{i=1}^{n} \lambda_i(A^3) = \frac{1}{6}\sum_{i=1}^{n} \lambda_i^3$, and can be estimated using:

$$\Delta \approx \frac{1}{6}\sum_{i=1}^{k} \lambda_i^3,$$

where $\lambda_1 \dots \lambda_k$ are the k eigenvalues of largest magnitude of the adjacency matrix A. Thus, we see that EigenTriangle approximates trace(A^3). Given this, it makes sense that this runtime would see a substantial improvement over the complexity of computing trace(A^3) directly.

The runtime of EigenTriangle is dominated by the cost of approximating the top k eigenvalues and eigenvectors of A, which, using the Lanszos method [7], can be done in roughly $O(km)$ time, where m is the number of edges since k is typically set much smaller than the number of nodes n, this is a significant improvement over the runtimes of direct methods.

TraceTriangle The TraceTriangle algorithm [4] relies on the previously mentioned property: $\Delta = \frac{1}{6}\text{trace}(A^3)$, where A is the adjacency matrix of the graph and Δ is the number of triangles. However, rather than computing the full matrix multiplication A^3, which is computationally expensive for large graphs, the TraceTriangle algorithm uses a randomized approach to approximate this trace, significantly reducing computation time.

This randomized approach is based on Hutchinson's method [11], which is a technique for estimating the trace of a matrix by randomly sampling vectors. In this case, this significantly reduces computation time by approximating trace(A^3) through randomized sampling rather than explicit computation.

The TraceTriangle algorithm is a sampling algorithm, and thus, its runtime depends on the desired accuracy of output, as more or fewer samples can be taken depending on the application. Generally though, experiments demonstrate that typically $O(\log^2 n)$, where n is the number of vertices in G, samples are required to get good approximations on real-world graphs, and regardless of application, the runtime for taking each sample is $O(m)$, where m is the number of edges in G [4].

Comparing TraceTriangle to the EigenTriangle algorithm, TraceTriangle achieves higher accuracy across multiple types of graphs [4]. Despite this accuracy advantage, EigenTriangle tends to run more quickly than TraceTriangle on large graphs. That said, one advantage of TraceTriangle is its potential for parallelization. This allows TraceTriangle to scale effectively with the size of the graph, ultimately reducing the speed advantage of EigenTriangle in larger computations.

3.4 General Algorithmic Strategies

Beyond specific algorithms for triangle counting, various general techniques from statistics and theoretical computer science have been adapted for this problem, particularly in designing faster algorithms.

3.4.1 Learning-Augmented Algorithms

A learning-augmented algorithm [14] is an algorithm that uses a prediction to boost its performance. Whereas most algorithms take only the problem their input, learning-augmented algorithms also accept an extra piece of information—usually a prediction about some part of the solution. The algorithm then uses this prediction to run faster or produce better results.

An example of a learning-augmented algorithm is its use in the maximum weight matching problem. The maximum weight matching problem [9] is the problem of finding a matching in which the sum of weights is maximized in a weighted graph. The typical solution for this problem, the Hungarian algorithm, runs in $O(m\sqrt{n})$ time.

When a learning-augmented approach [8] is applied however, where machine-learned predictions are used to "warm-start" the algorithm—that is, feed the algorithm a pre-computed starting state that is designed to help the algorithm reach a solution more quickly—that runtime is significantly reduced when the predictions are accurate. When the predictions are inaccurate, the runtime is simply the same as in the Hungarian algorithm.

This technique can be applied to triangle counting too. For example, Tonic [6], a learning-augmented algorithm for counting triangles in graph streams, leverages predictions about edge "heaviness" (i.e., the number of triangles they are involved in) to improve the accuracy and speed of triangle counting. Tonic combines these predictions with sampling methods to keep track of the most relevant edges. This allows the algorithm to focus on the edges that are most likely to contribute to the triangle count.

Notably, Tonic provides unbiased estimates of triangle counts regardless of the accuracy of the

predictor. However, when the predictor provides useful information on heavy edges, the algorithm produces estimates with reduced variance compared to state-of-the-art alternatives.

In general, these methods can be highly effective, as accurate predictions can significantly enhance algorithms' efficiency or result quality. The next two techniques we will discuss—variance reduction and importance sampling—are both examples of learning augmented algorithms.

3.4.2 Variance Reduction

Variance reduction methods [16] aim to reduce the spread (or variance) of estimations, leading to more reliable results even with fewer samples. This is particularly important in large-scale graphs, where taking a high number of samples may be computationally infeasible.

In terms of triangle counting, this method can be applied by finding a fast way of estimating the global triangle count, and then using sampling to estimate the error on that count. In our case, our way of estimating the global triangle count will be to leverage the relationship between node degree and triangle count. Intuitively, we would expect a relationship between these two quantities, as the higher a node's degree, the more triangles it could possibly have. For example, if a node has only two edges, it can have at most one triangle. If that node had far more edges, it could clearly also have far more triangles.

We can begin by quantifying the relationship between the degree of nodes and the number of triangles they are involved in by graphing. Specifically, we can plot nodes' degrees (d_i) versus triangle counts (Δ_i) on a log-log plot (shown later in Figure 9), finding a line of best fit, and then exponentiating as follows:

$$\log(\Delta_i) \approx \alpha \cdot \log(d_i) + \beta$$

$$\Delta_i \approx d_i^{\alpha} \cdot e^{\beta} = m_i.$$

Now, using this equation, we can estimate the overall triangle count Δ by applying this line of best

fit relationship to all nodes in the graph and summing our results:

$$3\Delta \approx M = \sum_{i=1}^{n} m_i.$$

Next, we estimate the error on our global triangle count (written E).

$$3\Delta = M + (3\Delta - M)$$

$$E \approx 3\Delta - M.$$

To arrive at this value E, we sample our graph to get s nodes, with s being our sample size. For each of these s sampled nodes, we count the number of triangles they are involved in, Δ_i, and find the difference between those actual triangle counts and their estimated triangle counts obtained when using the line of best fit. We then take the sum of these errors and scale them up to get E. Mathematically, this can be expressed as follows:

$$E = \left(\sum_{i=1}^{s} \Delta_i - m_i \right) \cdot \frac{n}{s}.$$

Lastly, we take the sum of our estimate and our sampled error, and divide this sum by three to avoid triple-counting triangles, as each triangle has three nodes it is involved in:

$$\Delta \approx \frac{M + E}{3}.$$

Thus, by applying this variance reduction technique, we arrive at an estimate for the triangle count Δ.

It is also important to note that effectiveness of this variance reduction method depends on how well the estimate m_i approximates the true triangle count Δ_i for each node. The closer m_i is to Δ_i, the smaller the variance of the estimator will be, since the quantities $\Delta_i - m_i$ will tend to

be small on average. A more detailed theoretical analysis of this variance behavior is provided in Section 7.2.3.

3.4.3 Importance Sampling

An alternative to the variance reduction approach is *importance sampling*. When estimating a metric relating to a large population using uniform sampling, where all edges/nodes/wedges/etc. are sampled with the same probability, often a very large number of samples is required to ensure a good relative approximation [13]. This is because in uniform sampling, nodes with high triangle counts are just as likely to be sampled as nodes with low triangle counts. This means, if you want to have high odds of selecting high-impact nodes, you need a very large sample size s. Consequently, the computational cost can be high for achieving a desired accuracy level in many cases.

When using importance sampling [15], the process is improved by sampling higher-interest nodes with higher probability, focusing computational effort on the most "important" parts of a graph. The key idea behind importance sampling is to bias the sampling distribution towards more informative areas of the graph. For instance, in a graph where certain nodes are highly connected or play a critical role in the overall structure, importance sampling would prioritize these nodes to reduce the variance of the estimates.

Importance sampling can also be applied to triangle counts. For example, we can prioritize high-degree nodes as the most "important." The weight of this importance is decided by some power α greater than 0 (which is equivalent to uniform sampling). This α can be tuned to indicate different strengths of relationships between the degree and triangle counts of nodes.

To get the optimal value for α, we can plot nodes' degrees versus triangle count as described in Section 3.4.2 and use the value of the slope (α) obtained when calculating the line of best fit of this plot.

Once α has been selected, we use it to ascribe each node a probability p_i to each node based in its degree d_i:

$$D = \sum_{i=1}^{n} d_i^{\alpha} \tag{1}$$

$$p_i = \frac{d_i^{\alpha}}{D} \tag{2}$$

As a note, we drop the e^{β} term because, given its a constant, it would simply get canceled out regardless.

Next, we independently sample s nodes from the distribution defined by the probabilities $p_1, \ldots p_n$ with replacement. For example if $p_1 = 0.01$ and $p_2 = 0.1$, we are 10 times more likely to sample node 2 than node 1.

Next, for each sampled node we count the number of triangles it is a part of, and then scale that count by $\frac{1}{s \cdot p_i}$. Intuitively, we do this because, while we want to make sure we sample high-degree nodes, we also want to make sure not to overemphasize them in our final counts. Thus, we scaling by the inverse of their probabilities. The sum of all these counts, scaled down by three (as to avoid triple-counting triangles), is our estimate for the global triangle count Δ.

After this, we arrive at our final estimate for Δ:

$$\tilde{\Delta} = \frac{1}{3s} \sum_{j=1}^{s} \frac{\Delta_{ij}}{p_{ij}}.$$

4 Methods

To evaluate different triangle count estimation methods, we implemented them in Python and compared their accuracies, runtimes, and sample sizes on a diverse set of networks. The primary methods implemented were uniform sampling, importance sampling, a variance reduction method, and a hybrid method that combines elements of these approaches.

4.1 Implementation

Algorithms were implemented in Python using NetworkX [10], and experiments were run on consistent hardware (8 GM of memory, and an Apple M1 chip), with multiple trials to ensure reliable comparisons.

4.2 Datasets

The datasets used in this study include both synthetic networks and real-world graphs from the Stanford Network Analysis Platform (SNAP) library [12], covering various domains such as social networks, collaboration networks, and web graphs. Specifically, methods were evaluated on these three networks:

- Social Network (Facebook): Social circles (or 'friends lists') are represented in this graph, where each user is a node, and each edge is a friendship between users. Contains 4039 nodes, 88234 edges, and 1612010 triangles.
- Collaboration Network: This graph represents a network of co-authorships from the General Relativity and Quantum Cosmology collaboration network, where nodes represent authors and edges indicate co-authored papers. Contains 5242 nodes, 14496 edges, and 48260 triangles.
- Wikipedia Article Network: This graph represents a network of Wikipedia articles relating to crocodiles, where nodes represent articles and edges represent links between them. Contains

11631 nodes, 170918 edges, and 630879 triangles.

A synthetic network was also generated using the NetworkX library. Specifically, the library was used to create a Barabási–Albert [2] graph. Barabási–Albert graphs are designed to be highly simplified models of social networks, and model preferential attachment, exhibiting power-law degree distributions, making them interesting graphs when using degree as a predictor for other graph features.

To assess accuracy, ground-truth triangle counts were computed using exact algorithms and compared against the approximate counts produced by each estimation method. The evaluation metrics included accuracy, runtime efficiency, and the sample size required to achieve a specified error margin. By examining performance across different graph characteristics, the study identifies the strengths and weaknesses of each method in diverse scenarios.

4.3 Sampling Methods Evaluated

Four sampling methods—uniform sampling, importance sampling, variance reduction, and a hybrid method, were implemented and tested with all datasets.

Uniform Sampling

This baseline method involves randomly sampling nodes and scaling up the observed triangle counts proportionally. While simple, uniform sampling often struggles in graphs with skewed degree distributions.

Code for this method is given below, where A is the adjacency matrix of our graph G and s is the number of nodes of G being sampled.

```
import numpy as np
import random

# Generate a sorted list of s random integers between 0 and n-1
def gen_s_ints(s, n):
    choice_arr = sorted(random.choices(range(n), k=s))
    return choice_arr

```

```
# Estimate the number of triangles in a graph using uniform sampling
def estimate_uniformly_per_node_method(A, s):
    n = len(A)  # Total number of nodes in the graph
    sampled_nodes = gen_s_ints(s, n)  # Sample s nodes uniformly at random

    triangle_count = 0
    for i in sampled_nodes:
        triangle_count += count_node_triangles(A, i)  # Count triangles involving
    node i

    # Scale the sample count up to estimate the total number of triangles
    # n/s scales the sample to the full node set
    # Divide by 3 because each triangle is counted once at each vertex
    return triangle_count * (n / s) // 3
```

Importance Sampling

As detailed in Section 3.4.3, for importance sampling, we attempt to prioritize the sampling of nodes that are likely to form more triangles. This is done by sampling proportionally to the degree of nodes. This method aims for higher accuracy with fewer samples.

After we select a value α to indicate the strength of the relationship between the degree and triangle count of nodes, we use it to compute p_i for each node based on its degree d_i, as shown in Equations Equation (1) and Equation (2).

We can then sample each node n_i with probability p_i. The code for this sampling and estimation process is given below where, like before A is the adjacency matrix of G and s is our sample size. The parameter α is represented by the variable "power" in the code snippet.

```
# Sample s nodes from the graph, with probability proportional to degree^power
def sample_by_degree(A, n, s, power):
    degrees = np.sum(A, axis=1)  # Compute degree of each node
    degrees_to_power = np.power(degrees, power)  # Raise degrees to the given
    power

    sum_of_degrees_to_power = np.sum(degrees_to_power)  # Sum of all degrees^power
    probabilities = degrees_to_power / sum_of_degrees_to_power  # Normalize to get
     probabilities

    # Sample s nodes according to the computed probabilities
    sampled_nodes = random.choices(range(n), weights=probabilities, k=s)

    return probabilities, sampled_nodes

# Estimate the total number of triangles using importance sampling based on degree
def importance_estimate_per_node_method(A, s, power):
    n = len(A)  # Total number of nodes in the graph
    probabilities, sampled_nodes = sample_by_degree(A, n, s, power)  # Sample
```

```
    nodes with importance weights

    estimate = 0
    for i in sampled_nodes:
        triangle_count = count_node_triangles(A, i)  # Count triangles involving node i
        estimate += triangle_count * (1 / (s * probabilities[i]))  # Scale by inverse sampling probability

    # Divide by 3 because each triangle is counted once at each vertex
    return estimate // 3
```

To select the ideal value for the power α, as described in Section 3.4.3, we plot nodes' log degrees verses log triangle counts and set α to be the line of best fit of this plot.

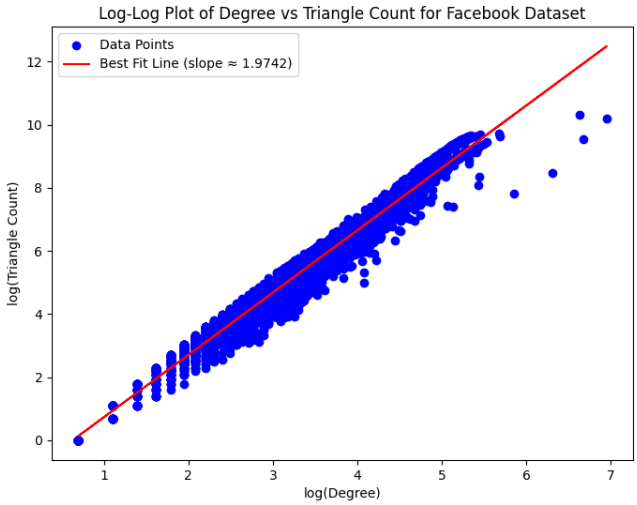
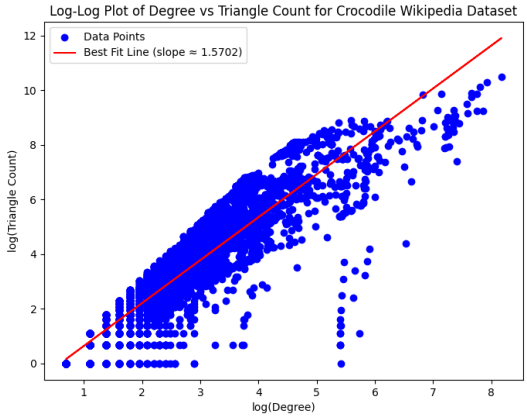
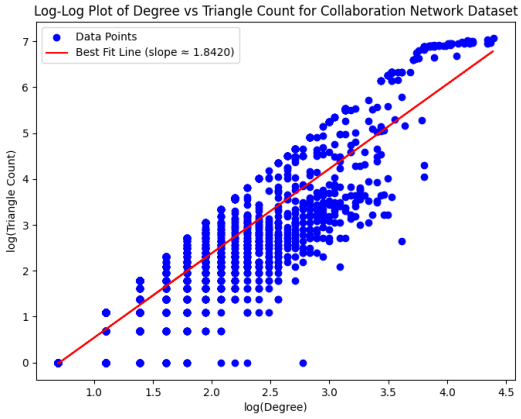


Figure 9: Degree vs. triangle count for the Facebook dataset. The slope $\alpha \approx 1.97$.

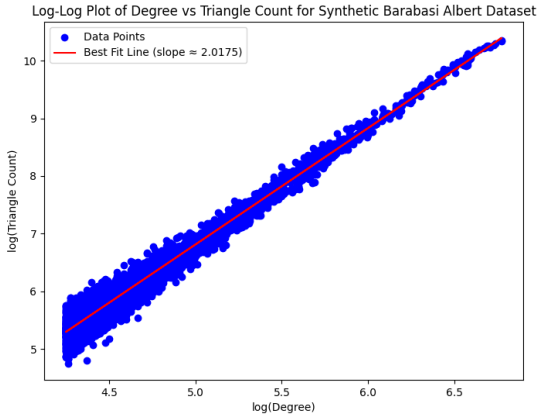
The slope α in Figure 9 is 1.97. Below, we give the degree vs triangle count plots for the other three datasets used:



(a) Wikipedia



(b) Collaboration



(c) Barabási–Albert

Figure 10: Degree vs. triangle count across datasets. For the Wikipedia dataset $\alpha \approx 1.57$, for the collaboration network $\alpha \approx 1.84$, and for the Barabási–Albert graph $\alpha \approx 2.02$.

Given the spread of α values obtained, we set our default value of α to 2, but also tested values 1 and 1.5 for all datasets. In addition, for each dataset, importance sampling was run with the "optimal" α found in its scatter plot.

Variance Reduction

Our next method is variance reduction, which aims to minimize the variability of our estimates. The high variance in uniform or importance sampling methods can lead to inaccurate triangle count estimations, particularly in graphs with skewed degree distributions. To address this, we leverage the relationship between node degrees and triangle counts as with importance sampling but use it

differently and introduce an error correction term.

Like with importance sampling, we first perform a linear regression in log-log space, obtaining a line of best fit that estimates triangle counts as a function of node degree. Then, instead of sampling, we use our line of best fit to arrive an an estimate of our triangle count by plugging in the degree of every node into the formula $\Delta_i \approx d_i^{\alpha} \cdot e^{\beta} = m_i$ from Section 3.4.2 and summing our results.

This number, however, may either over- or under-count the number of triangles depending on the relationship between degree and triangle count. Thus, we sample s additional nodes and calculate the difference between their actual triangle counts and those predicted by our line of best fit. Then, scaling this up, we can combine our predicted triangle count and predicted error to arrive at a final estimate of the triangle count Δ.

The specific math for this method is given in Section 3.4.2 and the code for it is given below.

```
# Find the slope and intercept of the best fit line for log(degree) vs log(triangle count)
def get_line_of_best_fit(A):
    n = len(A)
    degrees = np.sum(A, axis=1)  # Compute the degree of each node
    triangles = np.zeros(n)  # Initialize triangle counts

    for i in range(n):
        triangles[i] = count_node_triangles(A, i)  # Count triangles for each node

    # Only keep nodes with degree > 0 and triangle count > 0
    valid_indices = (degrees > 0) & (triangles > 0)
    filtered_degrees = degrees[valid_indices]
    filtered_triangles = triangles[valid_indices]

    # Take logs of degrees and triangle counts
    log_degrees = np.log(filtered_degrees)
    log_triangles = np.log(filtered_triangles)

    # Fit a line: log(triangle count) ~= slope * log(degree) + intercept
    slope, intercept = np.polyfit(log_degrees, log_triangles, 1)

    return slope, intercept

# Estimate the number of triangles using the variance reduction technique
def estimate_variance_reduction_method(A, s, power):
    n = len(A)

    slope, intercept = get_line_of_best_fit(A)  # Fit line between degree and triangle count

    # Estimate number of triangles for each node using the fitted line
    degree_array = np.sum(A, axis=1)
```

```
    approx_triangles = np.power(degree_array, slope) * np.exp(intercept)
    M = np.sum(approx_triangles)  # Initial estimate: sum of approximated triangle counts

    if s == 0:
        return M // 3  # If no samples, just return scaled M

    # Sample s nodes uniformly at random
    sampled_nodes = gen_s_ints(s, n)
    sampled_node_triangles = np.array([count_node_triangles(A, i) for i in sampled_nodes]) # Actual triangle counts

    sampled_m_i_vals = np.array([approx_triangles[i] for i in sampled_nodes]) # Approximated triangle counts

    # Compute correction term D
    D = np.sum(sampled_node_triangles - sampled_m_i_vals) * (n / s)

    # Final estimate: corrected M, divided by 3 (because each triangle is counted 3 times)
    return (M + D) // 3
```

Hybrid Approach

The hybrid approach combines elements from both importance sampling and variance reduction techniques. It functions by running the variance reduction method as before, but using importance sampling instead of uniform sampling to generate the s nodes used in correcting our triangle count estimate.

The differences between the following code snippet and that in variance reduction begins on line 13.

```
# Estimate the number of triangles using the hybrid method
def estimate_importance_variance_reduction_method(A, s, power):
    n = len(A)  # Total number of nodes in the graph

    slope, intercept = get_line_of_best_fit(A)  # Fit line between degree and triangle count

    # Approximate the number of triangles for each node using the fitted line
    degree_array = np.sum(A, axis=1)
    approx_triangles = np.power(degree_array, slope) * np.exp(intercept)
    M = np.sum(approx_triangles)  # Initial total estimate

    if s == 0:
        return M // 3  # If no samples, return the approximation

    # Sample nodes with importance sampling (probability proportional to degree^power)
    probabilities, sampled_nodes = sample_by_degree(A, n, s, power)

```

```
    # Get the probability for each sampled node
    sampled_node_probabilities = np.array([probabilities[i] for i in sampled_nodes])
    # Count actual triangles for each sampled node
    sampled_node_triangles = np.array([count_node_triangles(A, i) for i in sampled_nodes])
    # Get the approximated triangle counts for each sampled node
    sampled_m_i_vals = np.array([approx_triangles[i] for i in sampled_nodes])

    # Compute the correction term D, scaling each difference by 1/(s * probability)
    D = np.sum((sampled_node_triangles - sampled_m_i_vals) * (1 / (s * sampled_node_probabilities)))

    # Final corrected estimate, divided by 3 (each triangle is counted 3 times)
    return (M + D) // 3
```

Values for α for this hybrid method were selected the same way they were for importance sampling and variance reduction.

4.4 Evaluation Metrics

To assess the performance of the triangle count estimation methods, the following evaluation metrics were measured on real-world and synthetic networks:

- Accuracy: The difference between the estimated and true triangle counts, calculated as the relative error ($|\frac{\tilde{\Delta}-\Delta}{\Delta}|$ where Δ is the true triangle count and $\tilde{\Delta}$ is the estimated triangle count).
- Runtime: The computational time required to generate triangle count estimates.
- Sample size: The number of nodes sampled to produce an estimate.
- Variance: The variance in triangle count estimates across multiple independent runs of the same method.

4.5 Additional Explorations

Along with running the triangle count estimation methods, we extended our work in two additional ways: running our methods on 4-cliques to see how well they generalize and using a simulated

version of our methods to study how different types of noise impact performance.

4.5.1 4-Clique Variant

To evaluate how well these methods generalize, the code for each algorithm was adapted to count 4-cliques. To adapt the previous methods to count 4-cliques, we simply replace the `count_node_triangles()` method with a `count_node_4_cliques()` method, and scale final results by 4 instead of by 3.

To find the optimal values for α like before, we would plot degree versus 4-clique count and calculate a line of best fit. Likely, the optimal values would be close to 3. However, in this thesis, we only tested on values of α of 1 and 1.5 as this variant is intended as a preliminary extension to demonstrate feasibility rather than to optimize performance More extensive tuning of α and evaluation across a wider range of graphs would be a valuable direction for future work.

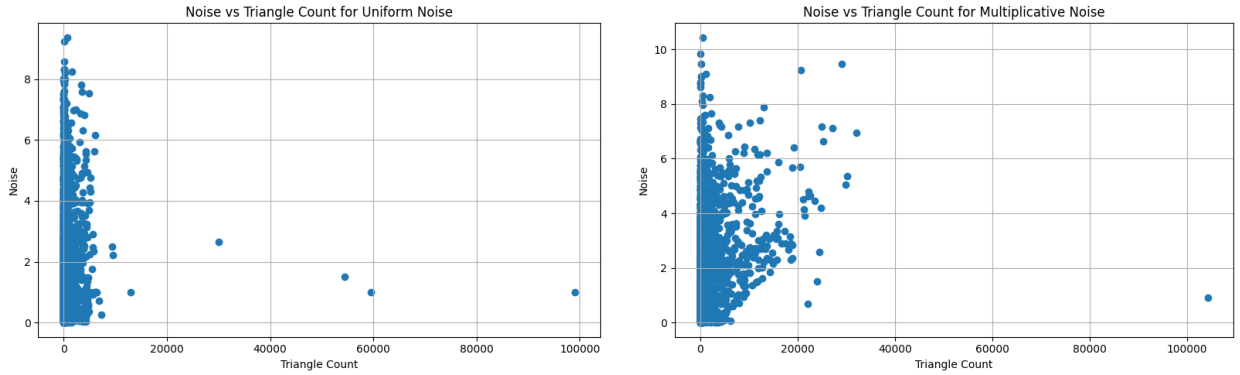
4.5.2 Simulated Method Tests

In order to learn more about when importance sampling and variance reduction perform best, we ran these algorithms on manually defined degree-triangle relationships.

The methods were tested on two different degree-triangle relationships. First, we set a variable `degrees` to be the degree distribution of a real-world dataset. I also defined `noise` as `noise = np.random.normal(0, noise_scale, triangles.shape)`. This noise is normally distributed with a mean of zero and a standard deviation of `noise_scale`, allowing for random fluctuations around the expected triangle count and maintaining a symmetric distribution.

The two relationships defined using these degrees and noise values are the **Uniform noise** and **multiplicative noise** relationships, illustrated in Figure 11. For the first, artificial triangle counts for the degree distributions were set to `np.power(degrees, slope) * np.exp(intercept) + noise` where the slope and intercept are the outputs from the `get_line_of_best()` function defined in Section 4.3. That is, the slope will be equivalent to our power value α discussed in Sections 3.4.2 and 3.4.3. Thus, in this relationship, noise does not grow with degree.

For the second relationship, the **multiplicative noise** relationship, the artificial triangle counts were set to `np.power(degrees, slope) * np.exp(intercept) * (1 + noise)` where the slope, intercept, and noise are defined as before. In this relationship, noise grows as the degree does too, distinguishing this degree-triangle relationship from the former.



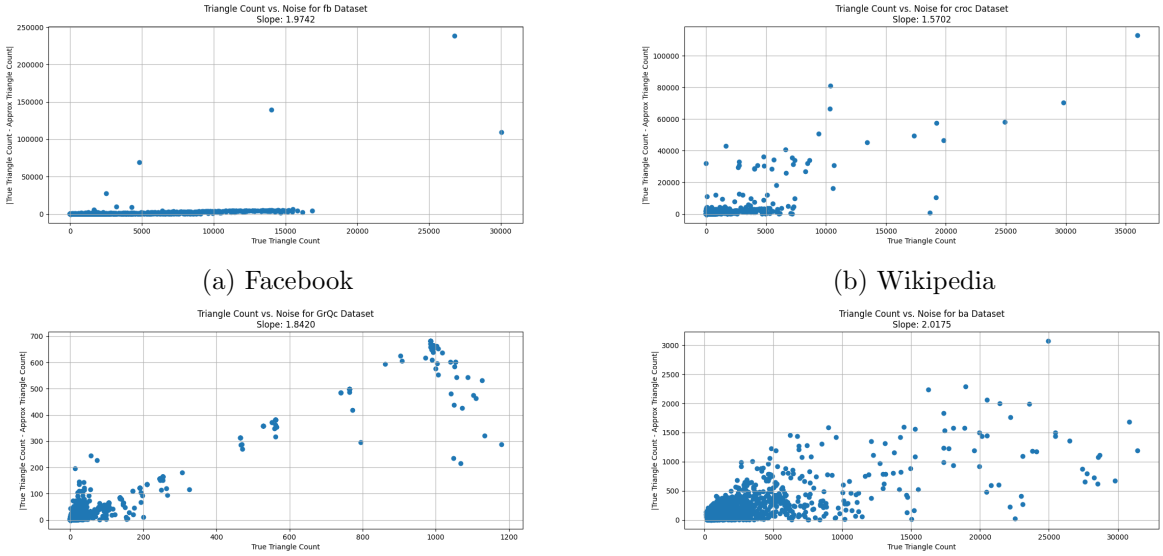
(a) Uniform

(b) Multiplicative

Figure 11: Triangle Count vs Noise for Uniform and Multiplicative Noise

These noise models allow us to analyze how different types of randomness impact the performance of variance reduction and importance sampling techniques.

To see which model most closely matches our datasets, we plot true triangle counts vs. 'noise' (i.e. the absolute value of the true minus approximate triangle counts) for each dataset. These plots are given below in Figure 12.



(a) Facebook

(b) Wikipedia

(c) Collaboration

(d) Barabási–Albert

Figure 12: Triangle counts vs. noise across all datasets. Nodes with higher triangle counts often have higher noise.

In these plots, we see that as the triangle counts increase, so does the noise. This indicates that our datasets more closely fit the multiplicative model than the uniform one.

5 Results and Discussion

In this section, we evaluate the accuracy and efficiency of various sampling methods, demonstrating that while the hybrid method consistently yields the lowest percent error, importance sampling offers the best trade-off between accuracy and runtime.

5.1 Triangle Counting

Tables 2 and 3 summarize the average percent error for different sampling methods across various datasets with sample sizes of 100 and 4000, respectively. The most accurate method is highlighted in bold for each dataset.

Table 2: Percent Errors for Different Methods on Various Datasets (Sample Size = 100, $\alpha = 2$)

	Avg Percent Error			
Dataset	**Unif. Sampling**	**Imp. Sampling**	**Var. Reduction**	**Hybrid**
Social Network (FB)	0.17471	0.03741	0.18911	**0.03267**
Collaboration Network	0.39456	**0.04186**	0.22566	0.04470
Wikipedia Article Network	0.36941	0.25732	0.85303	**0.18180**
Barabási–Albert	0.17220	0.01243	0.01377	**0.00791**

Table 3: Percent Errors for Different Methods on Various Datasets (Sample Size = 4000, $\alpha = 2$)

	Avg Percent Error			
Dataset	**Unif. Sampling**	**Imp. Sampling**	**Var. Reduction**	**Hybrid**
Social Network (FB)	0.02551	0.00608	0.05014	**0.00482**
Collaboration Network	0.03359	**0.00932**	0.02316	0.00944
Wikipedia Article Network	0.06177	0.04821	0.14815	**0.03789**
Barabási–Albert	0.03299	0.00163	0.00280	**0.00132**

In Tables 2 and 3, we see that for all datasets tested on except for the collaboration network dataset, the hybrid method tends to achieve the lowest percent error at both small and large sample sizes (specially, the table displays data for sample sizes of 100 and 4000).

Shown in Figures 13 to 16, while the hybrid method almost always achieves the lowest percent error overall, by sample size, importance sampling and the hybrid method are very comparable.

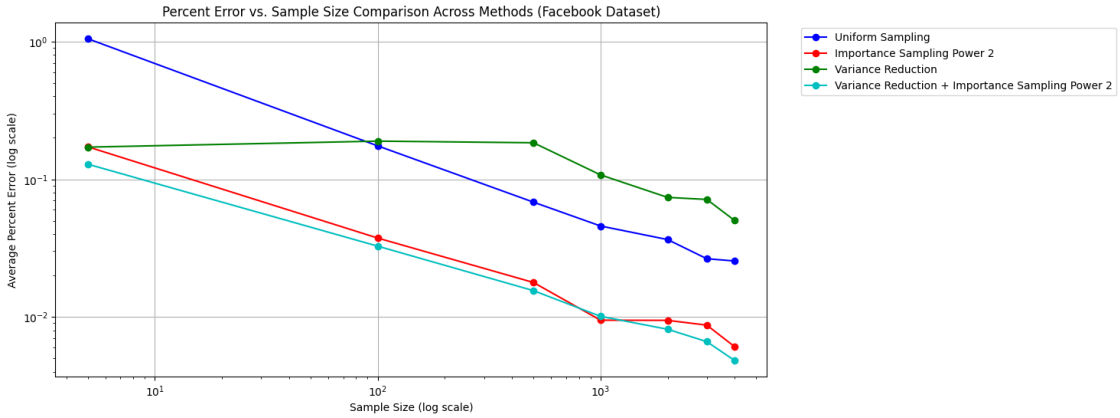


Figure 13: Percent Error vs. Sample Size for Facebook Dataset. Importance sampling and the hybrid approach perform best, with the hybrid method performing slightly better.

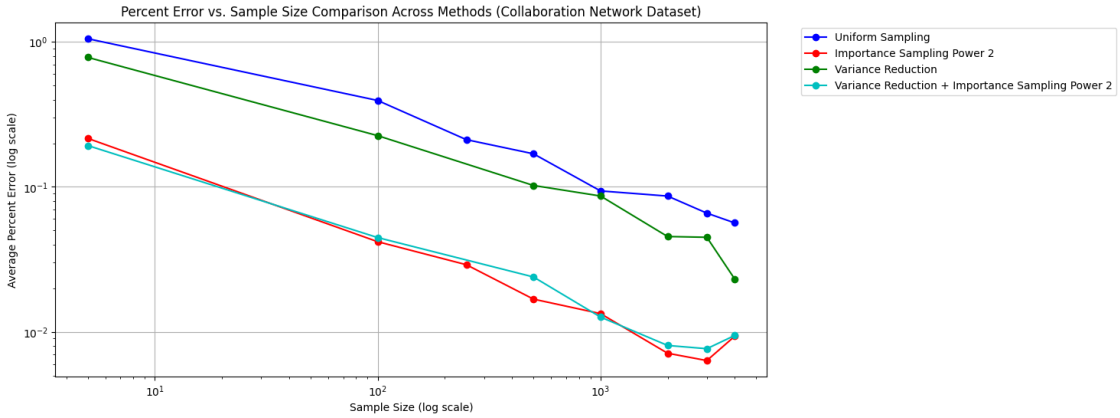


Figure 14: Percent Error vs. Sample Size for Collaboration Network Dataset. Importance sampling and the hybrid approach perform best.

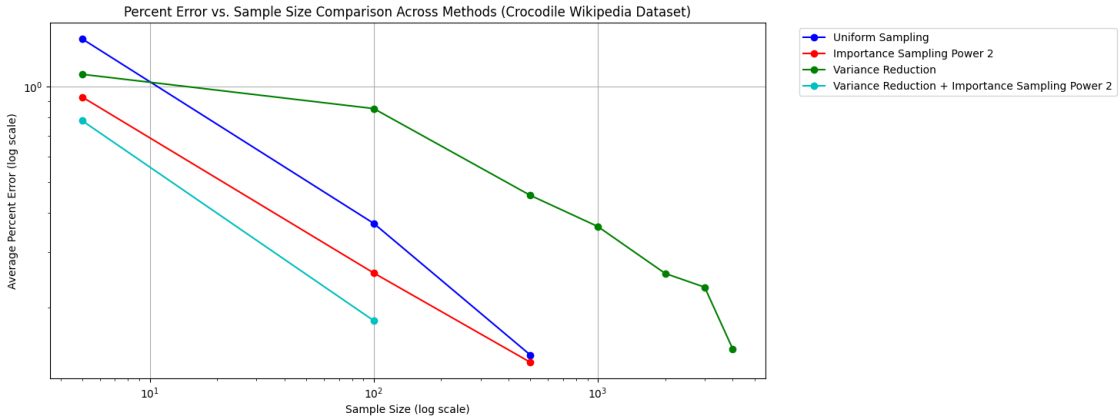


Figure 15: Percent Error vs. Sample Size for Crocodile Wikipedia Dataset. The hybrid method performs best.

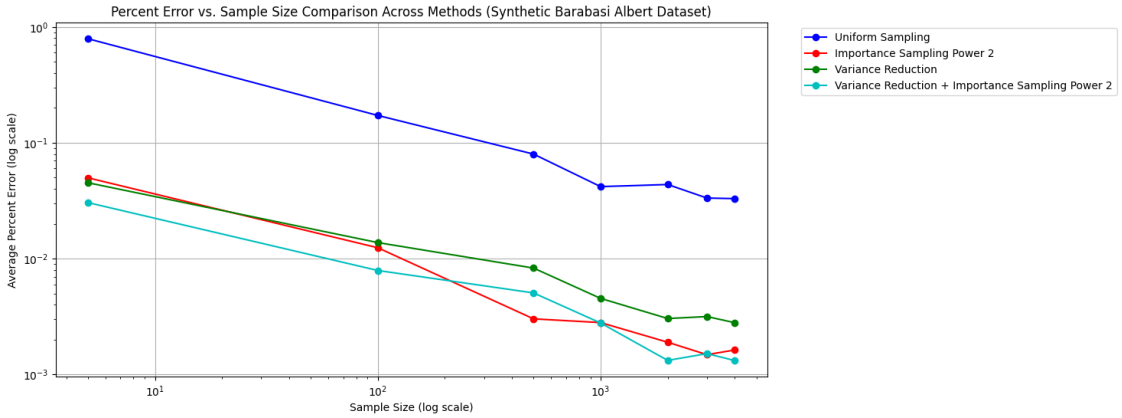


Figure 16: Percent Error vs. Sample Size for Synthetic Barabási–Albert Dataset. Importance sampling, variance reduction, and the hybrid method perform best, with importance sampling and the hybrid method performing best.

When we also consider runtime, as in Figures 17 to 20, we see importance sampling begin to outperform the hybrid method, especially on smaller sample sizes. Importance sampling generally has lower computational overhead than the hybrid method, and at smaller sample sizes, the hybrid method's added complexity doesn't pay off in terms of improved accuracy, so importance sampling achieves comparable or better results more efficiently. As a result, importance sampling outperforms the hybrid method in runtime-constrained settings.

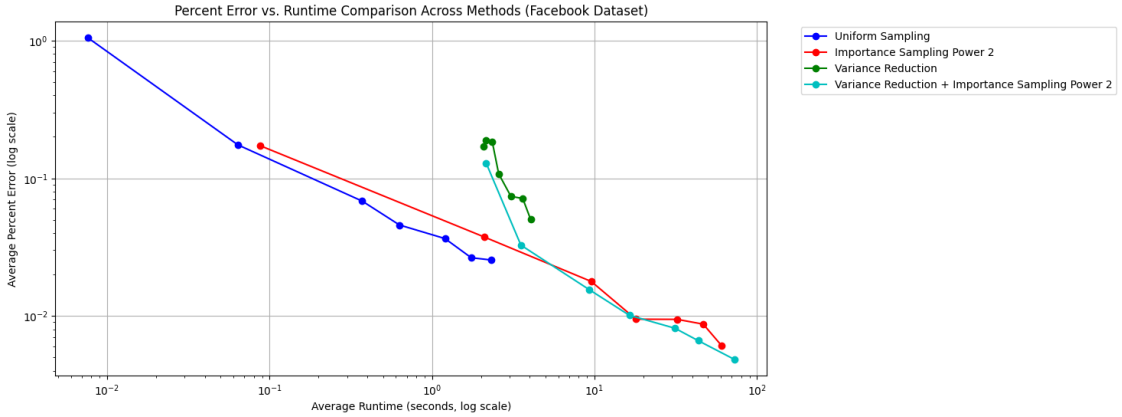


Figure 17: Percent Error vs. Runtime for Facebook Dataset. Uniform sampling, importance sampling, and the hybrid method have similar errors by runtime, but importance sampling and the hybrid method achieve the lowest errors overall.

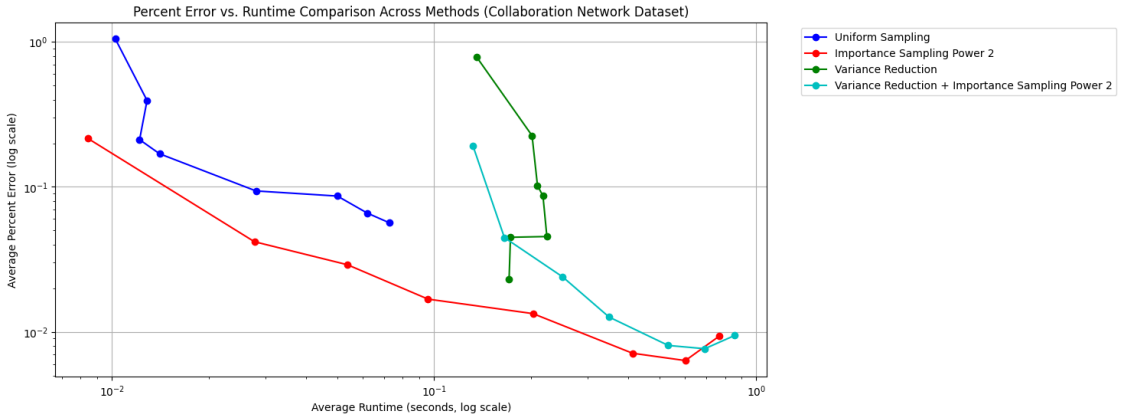


Figure 18: Percent Error vs. Runtime for Collaboration Network Dataset. Importance sampling outperforms other methods.

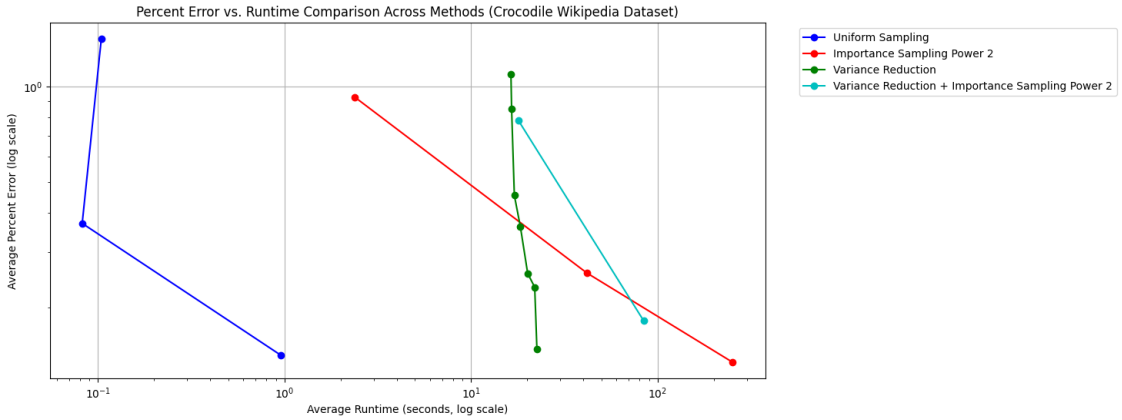


Figure 19: Percent Error vs. Runtime for Crocodile Wikipedia Dataset. Uniform sampling has a far shorter runtime than the other methods with comparable accuracy for large samples.

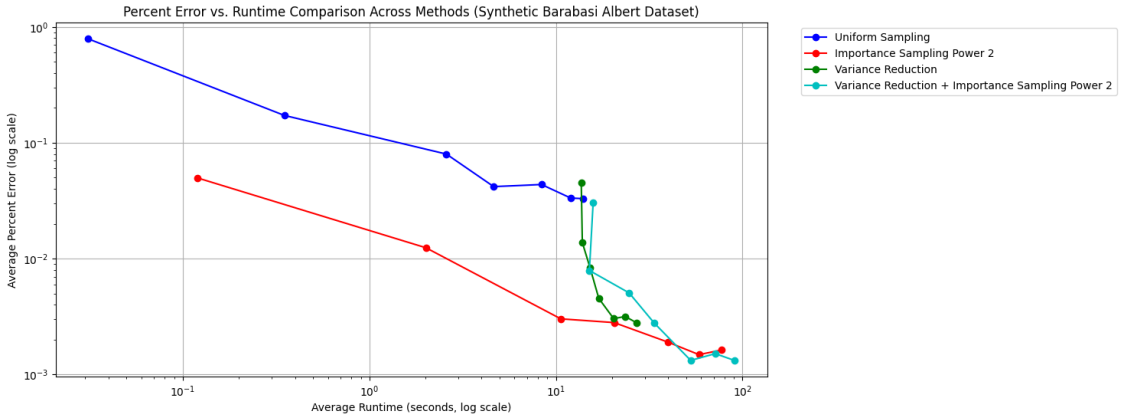


Figure 20: Percent Error vs. Runtime for Synthetic Barabási–Albert Dataset. The hybrid approach yields the lowest overall error, but for shorter runtimes, importance sampling performs best.

5.1.1 α Tradeoff

Also important to keep in mind is that in all previously referenced figures and tables, we use a default power, or α of 2. However, as illustrated in Figures 21 and 22, which present data from the Facebook dataset, larger powers lead to longer runtimes. This is because, for larger powers, we are more likely to sample high-degree (and therefore high-triangle) nodes. Thus, we will have more triangles to count overall, resulting in longer runtimes.

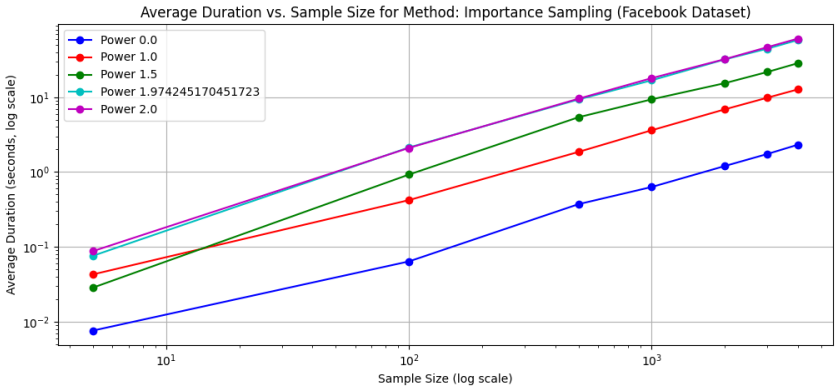


Figure 21: Average Duration - Importance Sampling. Runtime increases with sample size and power.

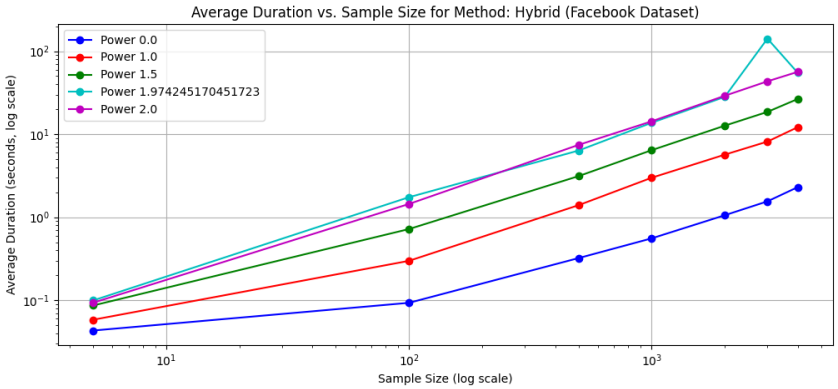


Figure 22: Average Duration - Hybrid Method. Runtime increases with sample size and power.

In Figures 23 and 24, we also find that powers closest to the optimal value α, as described in Section 3.4.3 have lowest percent error. In the case of the Social Network (Facebook) dataset, this value is $\alpha \approx 1.97$.

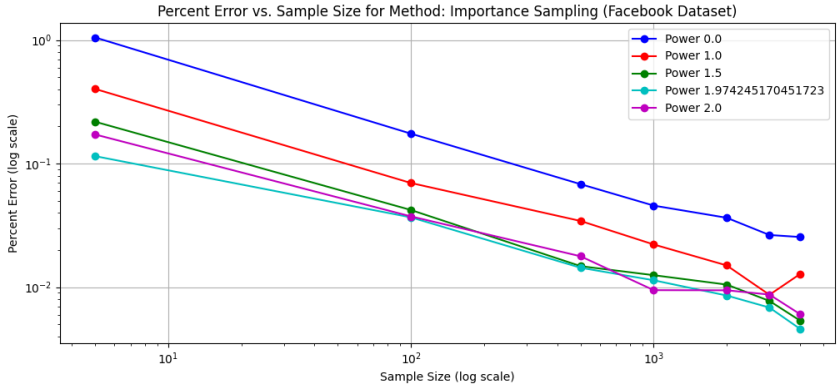


Figure 23: Percent Error - Importance Sampling. Percent error decreases as sample size increases. Powers closest to 1.97 perform best.

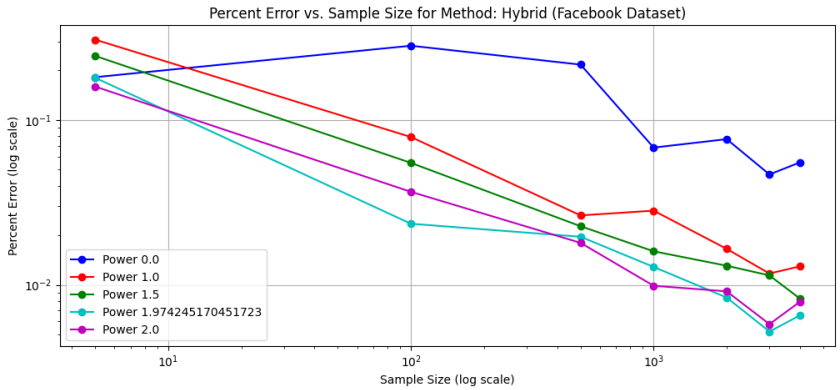


Figure 24: Percent Error - Hybrid Method. Percent error decreases as sample size increases. Powers closest to 1.97 perform best.

In these figures, we also observe that after a certain point, the decrease in percent error by power is very slight. For both importance sampling and the hybrid method, powers of 1.5, 2, and 1.97 all perform about the same.

Thus, while a higher power may yield the lowest error, in practical applications, it may make more sense to take a slightly lower power to decrease runtime. Using a smaller default value for α, such as 1.5, may thus impact the performance of the algorithms, particularly when looking at runtime vs. percent error, making it an interesting potential direction for future work.

To summarize, the hybrid method achieves the overall lowest percent error across almost all datasets, but typically, with runtime in mind, importance sampling performs best, and that per-

formance may be improved even further with more thoughtful selection of the α value.

5.2 4-Clique Counting

To see if these properties found with triangle counting generalize out to other graph motifs, we look at the results from 4-clique counting. Specifically, we compare uniform sampling, importance sampling, and variance reduction, and illustrate how the power in importance sampling impacts percent error and runtime in 4-clique counting. All shown results are from the Collaboration Network dataset.

As with triangle counting, we find that importance sampling yields better results than uniform sampling and variance reduction on our dataset, as shown in Figures 25 and 26.

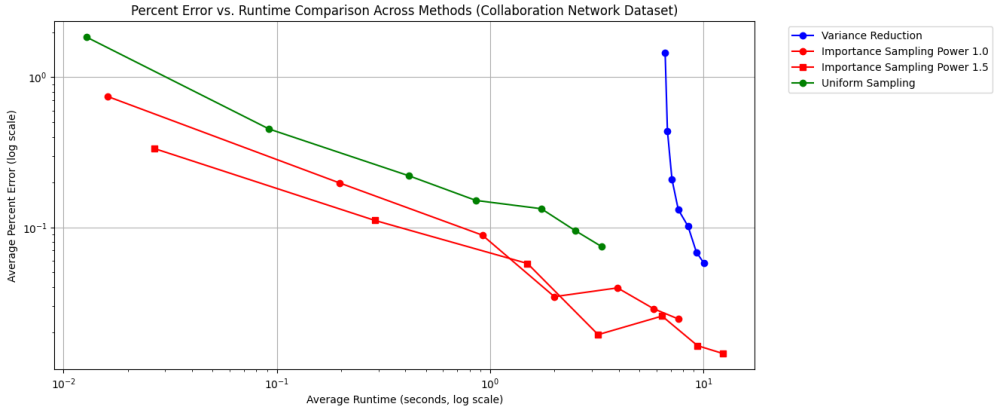


Figure 25: Comparison of percent error versus runtime for different methods in 4-clique counting. Importance sampling with higher powers outperforms variance reduction.

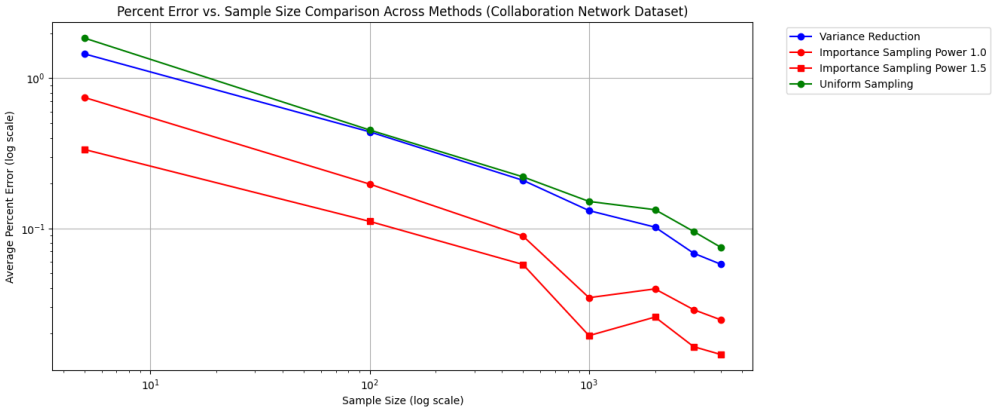


Figure 26: Comparison of percent error versus sample size for different 4-clique counting methods. Importance sampling with higher powers outperforms variance reduction.

Additionally, as made clear in Figures 27 and 28, the same relationships between power and error and power and runtime are present with 4-clique counting as they are with triangle counting.

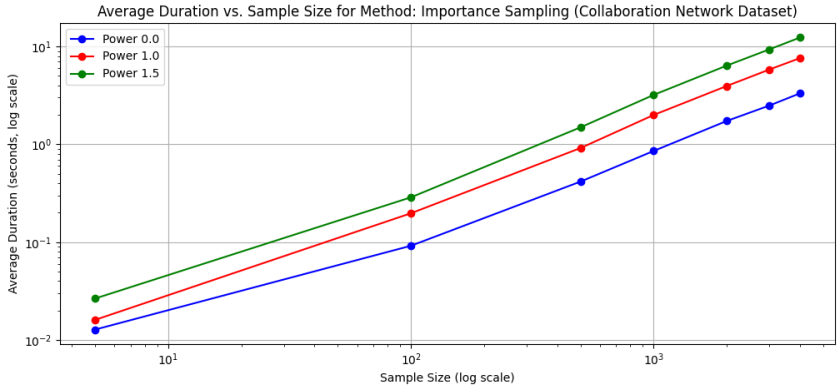


Figure 27: Average duration of the importance sampling method across powers for 4-clique counting. Runtime increases with sample size and power.

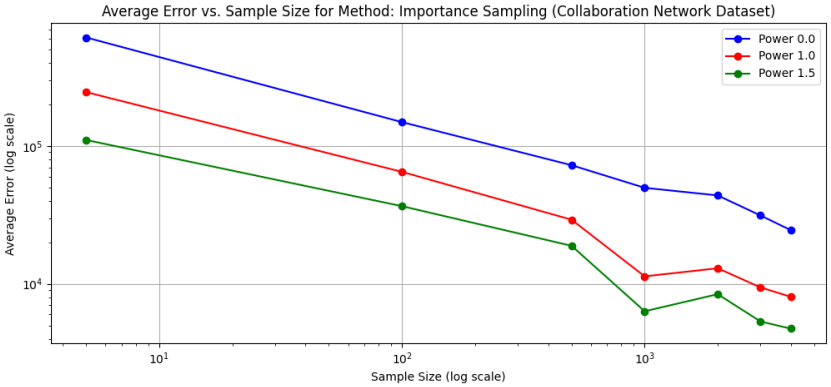


Figure 28: Average error of the importance sampling method across powers for 4-clique counting. Average error decreases as sample size increases. The power of 1.5 performs best.

These comparisons are not very robust, as due to 4-clique counting's runtime, fewer combinations of method, dataset, sample size, and power were run, but they do provide evidence that the patterns observed in the triangle counting experiments are likely generalizable to other motifs.

5.3 Understanding the Benefits of Importance Sampling vs. Variance Reduction

To gather insight into why specifically importance sampling outperformed the other methods, we turn to a simulated test of the methods. Specifically as described in Section 4.5.2, we tested variance reduction and importance sampling under two different degree-triangle relationships: uniform noise and multiplicative noise.

5.3.1 Uniform Noise Relationship

When running the algorithms on data simulated to have the following noise:

$$\text{triangle counts} = \text{np.power(degrees, slope)} \times \text{np.exp(intercept)} + \text{noise},$$

i.e noise that is normally distributed with mean zero and standard deviation defined by two scales:

10 and 100, the average error of variance reduction is lower than that of importance sampling, given in Figures 29 and 30. These noise scales were chosen arbitrarily so that we could compare the results between smaller and larger scales. We see too, that as the noise increases, the difference in performance between variance reduction and importance sampling increases too. In Figure 29, the difference in average errors between the methods is far smaller than the difference in Figure 30 with noise that is 10 times the size as before.

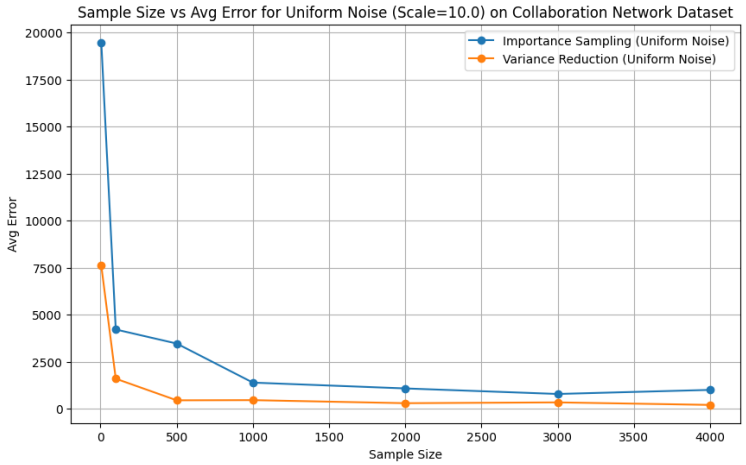


Figure 29: Sample size vs. average error with uniform noise (scale = 10). Variance reduction outperforms importance sampling.

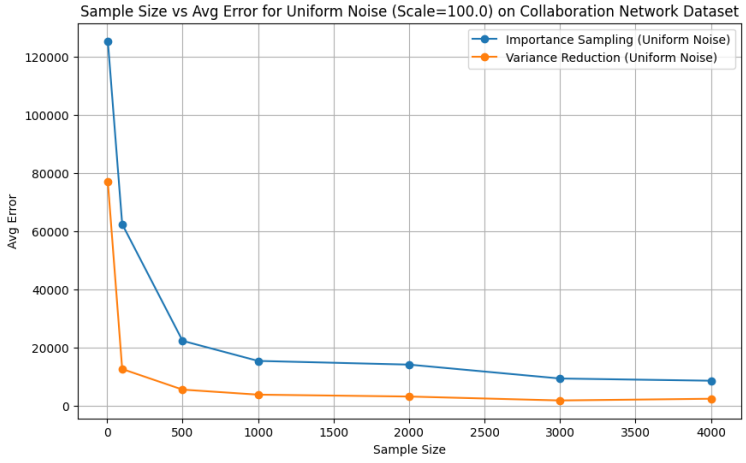


Figure 30: Sample size vs. average error with uniform noise (scale = 100). Variance reduction outperforms importance sampling.

5.3.2 Multiplicative Noise Relationship

Next, we consider the performance of the algorithms with multiplicative noise. The triangle counts were generated using the following formula:

$$\text{triangle counts} = \text{np.power(degrees, slope)} \times \text{np.exp(intercept)} \times (1 + \text{noise}),$$

where noise is again normally distributed and grows with the degree, and the noise scale was tested at two values: 0.01 and 0.1.

Unlike before, with this type of noise, as given in Figures 31 and 32, importance sampling outperforms variance reduction. Similarly to before, we also see the difference in performance between the two methods grow as the noise grows, meaning that, as the noise grows larger, the difference in average error between importance sampling and variance reduction grows larger too.

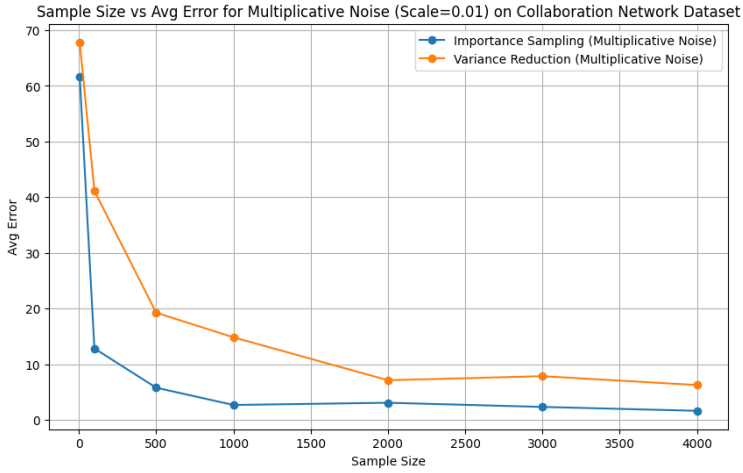


Figure 31: Sample size vs. average error with multiplicative noise (scale = 0.01). Importance sampling outperforms variance reduction.

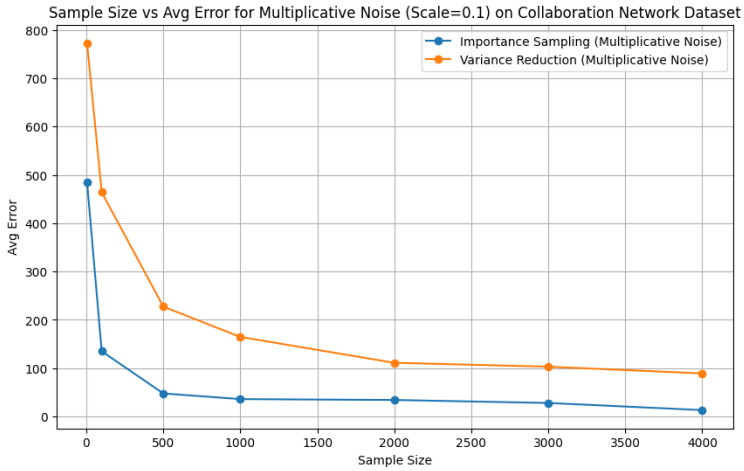


Figure 32: Sample size vs. average error with multiplicative noise (scale = 0.1). Importance sampling outperforms variance reduction.

Additionally, as derived in the appendix in Section 7.2.4, we have the following theorem:

Theorem 1 *In the case of multiplicative noise,* $\mathrm{Var}(\tilde{\Delta}_{IS}) \leq \mathrm{Var}(\tilde{\Delta}_{VR})$.

This theorem supports the previous findings. For the multiplicative case, the variance of importance sampling is lower than that of variance reduction, aligning with the finding that, for that case, the average error of importance sampling is lower than that of variance reduction too.

Given this, if our datasets also experience noise similar to the multiplicative noise model, that would explain the high performance of importance sampling for them. As shown Figure 12, that is the case for all four of our datasets. Thus, our simulated tests support our findings that importance sampling leads to lower error on our datasets. We can also imagine that in a dataset that aligns more closely with the uniform model, we may see the comparative performance of variance reduction improve.

6 Conclusion

In this thesis, we aimed to find ways to estimate triangles quickly and accurately using randomized algorithms. Through experiments and theoretical analysis, we found that, for many networks, particularly those with noise matching the multiplicative model, importance sampling offers the best trade-off between accuracy and efficiency, particularly for smaller sample sizes, but the hybrid method achieves the lowest overall error.

Research into the case where, according to the simulated tests and theoretical analysis, variance reduction outperforms importance sampling, i.e., the case where noise is uniform instead of multiplicative, would be an interesting direction for future work.

We also found a trade-off between power and runtime, finding that powers closest to the optimum for achieving low error did not necessarily achieve the best balance between accuracy and runtime. Thus, another further research direction would be investigating, and perhaps quantifying, that tradeoff.

Lastly, we extended our analysis out to other motifs, specifically, 4-cliques. There are many more graph motifs with interesting associated quantities that could be estimated using these techniques. Additionally, these techniques could be extended beyond graphs to estimate any number of other sums.

References

[1] Mohammad Al Hasan and Vachik S. Dave. Triangle counting in large networks: a review. *WIREs Data Mining and Knowledge Discovery*, 2018.

[2] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.

[3] SM Arifuzzaman, Maleq Khan, and Madhav Marathe. Parallel Algorithms for Counting Triangles and Computing Clustering Coefficients. pages 1448–1449, 2012.

[4] Haim Avron. Counting Triangles in Large Graphs using Randomized Matrix Trace Estimation. In *Proceedings of Kdd-Ldmta'10*, 2010.

[5] Corlin O. Beum and Everett G. Brundage. A Method for Analyzing the Sociomatrix. *Sociometry*, pages 141–145, 1950.

[6] Cristian Boldrin and Fabio Vandin. Fast and Accurate Triangle Counting in Graph Streams Using Predictions, 2024.

[7] Jane K. Cullum and Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol. I: Theory*. Society for Industrial and Applied Mathematics, 2002.

[8] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster Matchings via Learned Duals, 2021.

[9] Ran Duan and Seth Pettie. Linear-Time Approximation for Maximum Weight Matching. *J. ACM*, pages 1:1–1:23, 2014.

[10] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. Exploring network structure, dynamics, and function using NetworkX. Technical report, 2008.

[11] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, pages 433–450, 1990.

[12] Jure Leskovec and Rok Sosič. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology*, 8(1):1–20, 2017.

[13] Laszlo Lovasz. *Large networks and graph limits*. American Mathematical Society colloquium publications. American Mathematical Society, Providence, Rhode Island, 2012.

[14] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with Predictions. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.

[15] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, 1995.

[16] P. Prescott, J. M. Hammersley, and D. C. Handscomb. Monte Carlo Methods. *Applied Statistics*, 14(2/3):211, 1965.

[17] Thomas Schank and Dorothea Wagner. Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study. *Lecture Notes in Computer Science*, 3503, 2005.

[18] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Triadic Measures on Graphs: The Power of Wedge Sampling. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 10–18, 2013.

[19] V. Strassen. Gaussian Elimination is not Optimal. *Numerische Mathematik*, 13:354–356, 1969.

[20] Jessica Su, Aneesh Sharma, and Sharad Goel. The Effect of Recommendations on Network Structure. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 1157–1167, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[21] Charalampos E. Tsourakakis. Fast Counting of Triangles in Large Real Networks without Counting: Algorithms and Laws. In *2008 Eighth IEEE International Conference on Data Mining*, pages 608–617, 2008.

[22] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD*

international conference on Knowledge discovery and data mining, KDD '09, pages 837–846. Association for Computing Machinery, 2009.

[23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New Bounds for Matrix Multiplication: from Alpha to Omega, 2023.

[24] Ping Ye, Brian D. Peyser, Forrest A. Spencer, and Joel S. Bader. Commensurate distances and similar motifs in genetic congruence and protein interaction networks in yeast. *BMC Bioinformatics*, 6:270, 2005.

7 Appendix

In the appendix, we give links to code and extended results as well as a theoretical analysis of the variances of algorithms implemented.

7.1 Links to Code and Extended Results

The code used for this thesis is available on GitHub at the following URL:

`https://github.com/sophia-hubscher/triangle-counting`

Below is a list of the most relevant folders files with descriptions:

- Triangle counting code: Contains implementations of triangle counting algorithms, including the exact counting algorithm and the approximation algorithms. This file also contains code for the simulated runs.
- 4-Clique counting code: Includes similar code to thesis.py, but for the 4-clique implementations.
- Results folder: Contains CSVs of all results from this thesis. Results are separated by motif (triangle vs. 4-clique) and dataset (Facebook, Collaboration Network, etc.).
- All plots Contains all plots generated throughout the thesis, including various plots not included in the thesis report such as histograms showing the difference between approximate and true triangle counts across all datasets and whisker plots of the estimates for each method on each dataset. Many folders contain date-labeled subfolders. The most recently-dated folder will contain the most up-to-date plots.

7.2 Theoretical Analysis of Variances

Understanding the variance of the algorithms tested is crucial for evaluating their reliability. Thus, in this section, we derive expressions for the variance of the estimated triangle count across methods.

7.2.1 Uniform Sampling

For uniform sampling, the global triangle count Δ is estimated as $\tilde{\Delta}$ using the following formula:

$$\tilde{\Delta} = \frac{n}{3s} \sum_{i=1}^{s} \Delta_i,$$

where n is the number of nodes in our graph G, s is our sample size, and $\sum_{i=1}^{s} \Delta_i$ is the sum of all sampled triangle counts. Using this, we can find the variance of $\tilde{\Delta}$ in terms of n, s, and $\mathrm{Var}\left(\Delta_i\right)$.

$$\begin{aligned}
\mathrm{Var}\left(\tilde{\Delta}\right) &= \mathrm{Var}\left(\frac{n}{3s} \sum_{i=1}^{s} \Delta_i\right) \\
&= \frac{n^2}{9s^2} \mathrm{Var}\left(\sum_{i=1}^{s} \Delta_i\right) \\
&= \frac{n^2}{9s^2} \sum_{i=1}^{s} \mathrm{Var}\left(\Delta_i\right).
\end{aligned}$$

Next, we find $\mathrm{Var}\left(\Delta_i\right)$.

$$\mathrm{Var}\left(\Delta_i\right) = \mathbb{E}\left[\left(\Delta_i - \mathbb{E}[\Delta_i]\right)^2\right].$$

We use the fact that the probability of sampling each node is $\frac{1}{n}$ to arrive at the following step:

$$\mathbb{E}\left[\Delta_i\right] = 3\left(\frac{1}{n}\Delta_1 + \frac{1}{n}\Delta_2 + \ldots + \frac{1}{n}\Delta_n\right) = \frac{3\Delta}{n}.$$

$$\begin{aligned}
\mathbb{E}\left[(\Delta_i - \mathbb{E}\left[\Delta_i\right])^2\right] &= \mathbb{E}\left[\left(\Delta_i - \frac{3\Delta}{n}\right)^2\right] \\
&= \sum_{i=1}^{n} \frac{1}{n}\left(\Delta_i - \frac{3\Delta}{n}\right)^2 \\
&= \sum_{i=1}^{n} \frac{1}{n}\left(\Delta_i^2 - 6\Delta_i\frac{\Delta}{n} + \frac{9\Delta^2}{n^2}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{6}{n}\sum_{i=1}^{n}\Delta_i\frac{\Delta}{n} + \frac{1}{n}\sum_{i=1}^{n}\frac{9\Delta^2}{n^2} \\
&= \frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{6}{n}\sum_{i=1}^{n}\Delta_i\frac{\Delta}{n} + \frac{9\Delta^2}{n^2} \\
&= \frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{18\Delta^2}{n^2} + \frac{9\Delta^2}{n^2} \\
&= \frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{9\Delta^2}{n^2}.
\end{aligned}$$

Now, we can plug our value of $\mathrm{Var}\left(\Delta_i\right)$ into $\mathrm{Var}\left(\tilde{\Delta}\right) = \frac{n^2}{9s^2}\sum_{i=1}^{s}\mathrm{Var}\left(\Delta_i\right)$.

$$\begin{aligned}
\mathrm{Var}\left(\tilde{\Delta}\right) &= \frac{n^2}{9s^2}\sum_{i=1}^{s}\left[\frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{9\Delta^2}{n^2}\right] \\
&= \frac{n^2}{9s^2}s\left[\frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{9\Delta^2}{n^2}\right] \\
&= \frac{n^2}{9s}\left[\frac{1}{n}\sum_{i=1}^{n}\Delta_i^2 - \frac{9\Delta^2}{n^2}\right] \\
&= \frac{n}{s}\left[\frac{1}{9}\sum_{i=1}^{n}\Delta_i^2 - \frac{\Delta^2}{n}\right] \\
&= \frac{n}{9s}\sum_{i=1}^{n}\Delta_i^2 - \frac{\Delta^2}{s}.
\end{aligned}$$

Altogether, we get

$$\mathrm{Var}\left(\tilde{\Delta}_{US}\right) = \frac{n}{9s}\sum_{i=1}^{n}\Delta_i^2 - \frac{\Delta^2}{s}. \tag{3}$$

7.2.2 Importance Sampling

For importance sampling, the global triangle count Δ is estimated as $\tilde{\Delta}$ using the following formula:

$$\tilde{\Delta} = \frac{1}{3s} \sum_{j=1}^{s} \frac{\Delta_{ij}}{p_{ij}},$$

where $p_i = \frac{m_i}{\sum_{j=1}^{n} m_{ij}}$ with $m_j = \Delta_j + n_j$ where $n_i \in [-\sigma\Delta_i, \sigma\Delta_i]$ and $\sigma < 1$. In this model of importance sampling, we will assume multiplicative noise, hence, we see the scale of n_i increase as Δ_i increases.

$$
\begin{aligned}
\operatorname{Var}\left(\tilde{\Delta}\right) &= \operatorname{Var}\left(\frac{1}{3s}\sum_{j=1}^{s}\frac{\Delta_{ij}}{p_{ij}}\right) \\
&= \frac{1}{9s^2}\operatorname{Var}\left(\sum_{j=1}^{s}\frac{\Delta_{ij}}{p_{ij}}\right) \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\operatorname{Var}\left(\frac{\Delta_{ij}}{p_{ij}}\right) \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\mathbb{E}\left[\left(\frac{\Delta_{ij}}{p_{ij}}\right)^2\right] - \left(\mathbb{E}\left[\frac{\Delta_{ij}}{p_{ij}}\right]\right)^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\mathbb{E}\left[\left(\frac{\Delta_{ij}}{p_{ij}}\right)^2\right] - \left(\sum_{n}^{j=1}\left(p_j\frac{\Delta_j}{p_j}\right)\right)^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\mathbb{E}\left[\left(\frac{\Delta_{ij}}{p_{ij}}\right)^2\right] - \left(\sum_{n}^{j=1}\Delta_j\right)^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\mathbb{E}\left[\left(\frac{\Delta_{ij}}{p_{ij}}\right)^2\right] - \Delta^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\sum_{j=1}^{n}\left(p_j\frac{\Delta_j^2}{p_j^2}\right) - \Delta^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\sum_{j=1}^{n}\left(\frac{\Delta_j^2}{p_j}\right) - \Delta^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\sum_{j=1}^{n}\frac{\Delta_j^2}{m_j}\sum_{i=1}^{n}m_i - \Delta^2\right] \\
&= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\sum_{i=1}^{n}m_i\sum_{j=1}^{n}\frac{\Delta_j^2}{m_j} - \Delta^2\right].
\end{aligned}
$$

We can set an upper bound on $\sum_{i=1}^{n} m_i$ as follows:

$$
\begin{aligned}
\sum_{i=1}^{n} m_i &\leq \sum_{i=1}^{n}\left(\Delta_i + \sigma\Delta_i\right) \\
&\leq \left(1+\sigma\right)\Delta.
\end{aligned}
$$

We can similarly set an upper bound on $\sum_{j=1}^{n}\frac{\Delta_j^2}{m_j}$ as follows:

$$\begin{aligned}\sum_{j=1}^{n}\frac{\Delta_j^2}{m_j} &\leq \sum_{j=1}^{n}\frac{\Delta_j^2}{(1-\sigma)\,\Delta_j}\\ &= \sum_{j=1}^{n}\frac{\Delta_j}{1-\sigma}\\ &= \frac{1}{1-\sigma}\sum_{j=1}^{n}\Delta_j\\ &= \frac{1}{1-\sigma}\Delta.\end{aligned}$$

Combining these two bounds we can return to our variance expression:

$$\begin{aligned}\mathrm{Var}\left(\tilde{\Delta}\right) &= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\sum_{i=1}^{n}m_i\sum_{j=1}^{n}\frac{\Delta_j^2}{m_j}-\Delta^2\right]\\ &\leq \frac{1}{9s^2}\sum_{j=1}^{s}\left[\frac{1+\sigma}{1-\sigma}\Delta^2-\Delta^2\right]\\ &= \frac{1}{9s^2}\sum_{j=1}^{s}\left[\left(\frac{1+\sigma}{1-\sigma}-1\right)\Delta^2\right]\\ &= \frac{1}{9s}\left(\frac{1+\sigma}{1-\sigma}-1\right)\Delta^2.\end{aligned}$$

Altogether, we get

$$\mathrm{Var}\left(\tilde{\Delta}_{IS}\right) \leq \frac{1}{9s}\left(\frac{1+\sigma}{1-\sigma}-1\right)\Delta^2. \tag{4}$$

7.2.3 Variance Reduction

Across this section, we will use the fact that, when our distributions are mean 0, $\mathrm{Var}\left[X+Y\right] \leq 2\left(\mathrm{Var}\left[X\right]+\mathrm{Var}\left[Y\right]\right)$. We derive that as follows:

$$
\begin{aligned}
\mathrm{Var}(X+Y) &= \mathbb{E}[(X+Y)^2] - \mathbb{E}[X+Y]^2 \\
&= \mathbb{E}[X^2 + 2XY + Y^2] - (\mathbb{E}[X] + \mathbb{E}[Y])^2 \\
&= \mathbb{E}[X^2] + 2\mathbb{E}[XY] + \mathbb{E}[Y^2] - \mathbb{E}[X]^2 - 2\mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[Y]^2 \\
&= \underbrace{\mathbb{E}[X^2] - \mathbb{E}[X]^2}_{\mathrm{Var}(X)} + \underbrace{\mathbb{E}[Y^2] - \mathbb{E}[Y]^2}_{\mathrm{Var}(Y)} + 2(\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]) \\
&= \mathrm{Var}(X) + \mathrm{Var}(Y) + 2(\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]).
\end{aligned}
$$

Now, applying the Cauchy-Schwarz inequality:

$$
\begin{aligned}
2\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] &\leq 2\sqrt{\mathbb{E}[(X - \mathbb{E}[X])^2] \cdot \mathbb{E}[(Y - \mathbb{E}[Y])^2]} \\
&= 2\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)} \\
&\leq \mathrm{Var}(X) + \mathrm{Var}(Y) \quad \text{(by AM-GM inequality)}.
\end{aligned}
$$

Thus, we have $\mathrm{Var}\left[X+Y\right] \leq 2\left(\mathrm{Var}\left[X\right] + \mathrm{Var}\left[Y\right]\right)$.

Variance Reduction with Uniform Noise Take the estimator of Δ to be $\sum_{i=1}^{n} m_i$ where $m_i = \Delta_i + N\left(0, \sigma^2\right)$.

$$
\tilde{\Delta} = \sum_{i=1}^{n} m_i + \frac{n}{s} \sum_{i=1}^{s} \left(m_i - \Delta_i\right).
$$

This equations mirrors that seen in Section 3.4.2, where $\sum_{i=1}^{n} m_i$ gives an estimate of Δ and $\frac{n}{s} \sum_{i=1}^{s} \left(m_i - \Delta_i\right)$ serves as the error correction term.

$$
\begin{aligned}
\mathrm{Var}\left(\tilde{\Delta}\right) &= \mathrm{Var}\left(\sum_{i=1}^{n} m_i + \frac{n}{s}\sum_{i=1}^{s}\left(m_i - \Delta_i\right)\right) \\
&\leq 2\sum_{i=1}^{n}\mathrm{Var}\left(m_i\right) + 2\mathrm{Var}\left(\frac{n}{s}\sum_{i=1}^{s}\left(m_i - \Delta_i\right)\right) \\
&= 2\sum_{i=1}^{n}\mathrm{Var}\left(m_i\right) + \frac{2n^2}{s^2}\mathrm{Var}\left(\sum_{i=1}^{s}\left(m_i - \Delta_i\right)\right) \\
&= 2\sum_{i=1}^{n}\mathrm{Var}\left(m_i\right) + \frac{2n^2}{s^2}\sum_{i=1}^{s}\mathrm{Var}\left(m_i - \Delta_i\right) \\
&= 2n\sigma^2 + \frac{2n^2}{s^2}\sum_{i=1}^{s}\mathrm{Var}\left(m_i - \Delta_i\right) \\
&= 2n\sigma^2 + \frac{2n^2}{s^2}s\sigma^2 \\
&= 2n\sigma^2 + \frac{2n^2}{s}\sigma^2.
\end{aligned}
$$

Thus, we get

$$
\mathrm{Var}\left(\tilde{\Delta}_{VRu}\right) \leq 2n\sigma^2 + \frac{2n^2}{s}\sigma^2. \tag{5}
$$

To allow for better comparison methods that do not contain a variable σ (in our case, uniform sampling), assume σ^2 is the average squared error when using the trivial predictor of $m_i = 0$. In this case, $\sigma^2 = \frac{\sum_{i=1}^{n}\Delta_i^2}{n}$.

Plugging this value for σ^2 into our result, we get:

$$
\begin{aligned}
\mathrm{Var}\left(\tilde{\Delta}\right) &\leq 2n\sigma^2 + \frac{2n^2}{s}\sigma^2 \\
&= 2n\frac{\sum_{i=1}^{n}\Delta_i^2}{n} + \frac{2n^2}{s}\frac{\sum_{i=1}^{n}\Delta_i^2}{n} \\
&= 2\sum_{i=1}^{n}\Delta_i^2 + \frac{2n}{s}\sum_{i=1}^{n}\Delta_i^2 \\
&= 2\sum_{i=1}^{n}\Delta_i^2\left(1 + \frac{n}{s}\right).
\end{aligned}
$$

This is the variance expression reached with the trivial predictor, but a better predictor $\sigma^2 =$

$\frac{\sum_{i=1}^n \Delta_i^2}{n}\epsilon$ for some small value ϵ would lead to a far smaller variance of:

$$\mathrm{Var}\left(\tilde{\Delta}\right) \leq 2\sum_{i=1}^{n} \Delta_i^2 \epsilon \left(1 + \frac{n}{s}\right).$$

Variance Reduction with Multiplicative Noise Here, take the estimator of Δ to be $\sum_{i=1}^n m_i$ where $m_i = \Delta_i + n_i$ where $\mathrm{Var}(n_i) = \sigma_i^2$ and $\sigma_i^2 = \Delta_i^2\sigma^2$. i.e., as the size of Δ_i increases, so does the noise m_i.

$$\tilde{\Delta} = \sum_{i=1}^{n} m_i + \frac{n}{s}\sum_{i=1}^{s} (m_i - \Delta_i).$$

$$\begin{aligned}
\mathrm{Var}\left(\tilde{\Delta}\right) &= \mathrm{Var}\left(\sum_{i=1}^{n} m_i + \frac{n}{s}\sum_{i=1}^{s} (m_i - \Delta_i)\right) \\
&\leq 2\sum_{i=1}^{n} \mathrm{Var}\left(m_i\right) + 2\mathrm{Var}\left(\frac{n}{s}\sum_{i=1}^{s} (m_i - \Delta_i)\right) \\
&= 2\sum_{i=1}^{n} \mathrm{Var}\left(m_i\right) + \frac{2n^2}{s^2}\mathrm{Var}\left(\sum_{i=1}^{s} (m_i - \Delta_i)\right) \\
&= 2\sum_{i=1}^{n} \sigma_i^2 + \frac{2n^2}{s^2}\mathrm{Var}\left(\sum_{i=1}^{s} (m_i - \Delta_i)\right) \\
&= 2\sigma^2\sum_{i=1}^{n} \Delta_i^2 + \frac{2n^2}{s^2}\mathrm{Var}\left(\sum_{i=1}^{s} (m_i - \Delta_i)\right) \\
&= 2\sigma^2\sum_{i=1}^{n} \Delta_i^2 + \frac{2n^2}{s^2}\sum_{i=1}^{s}\left(\frac{1}{n}\sum_{i=1}^{n} \sigma^2\Delta_i^2\right) \\
&= 2\sigma^2\sum_{i=1}^{n} \Delta_i^2 + \frac{2n}{s}\sigma^2\sum_{i=1}^{n} \Delta_i^2 \\
&= 2\sigma^2\left(\sum_{i=1}^{n} \Delta_i^2 + \frac{n}{s}\sum_{i=1}^{n} \Delta_i^2\right) \\
&= 2\sigma^2\sum_{i=1}^{n} \Delta_i^2\left(1 + \frac{n}{s}\right).
\end{aligned}$$

Thus, we get

$$\text{Var}\left(\tilde{\Delta}_{VRm}\right) \leq 2\sigma^2 \sum_{i=1}^{n} \Delta_i^2 \left(1 + \frac{n}{s}\right). \tag{6}$$

As with uniform noise, we will plug in the value $\sigma^2 = \frac{\sum_{i=1}^{n} \Delta_i^2}{n}$. This yields the following:

$$\begin{aligned}
\text{Var}\left(\tilde{\Delta}\right) &\leq 2\sigma^2 \sum_{i=1}^{n} \Delta_i^2 \left(1 + \frac{n}{s}\right) \\
&= 2\frac{\sum_{i=1}^{n} \Delta_i^2}{n} \sum_{i=1}^{n} \Delta_i^2 \left(1 + \frac{n}{s}\right) \\
&= \frac{2}{n} \left(\sum_{i=1}^{n} \Delta_i^2\right)^2 \left(1 + \frac{n}{s}\right) \\
&= 2 \left(\sum_{i=1}^{n} \Delta_i^2\right)^2 \left(\frac{1}{n} + \frac{1}{s}\right).
\end{aligned}$$

As before, with a better predictor of $\sigma^2 = \frac{\sum_{i=1}^{n} \Delta_i^2}{n}\epsilon$ for some small value ϵ we can recompute the variance as:

$$\text{Var}\left(\tilde{\Delta}\right) \leq 2\epsilon \left(\sum_{i=1}^{n} \Delta_i^2\right)^2 \left(\frac{1}{n} + \frac{1}{s}\right).$$

7.2.4 Comparing Variance Reduction and Importance Sampling

In this section we will refer to variance in the case of the importance sampling method as $\text{Var}\left(\tilde{\Delta}_{IS}\right)$ and for the variance reduction method (with multiplicative noise) as $\text{Var}\left(\tilde{\Delta}_{VR}\right)$. In the previous sections, we derived expressions for $\text{Var}\left(\tilde{\Delta}_{IS}\right)$ and $\text{Var}\left(\tilde{\Delta}_{VRm}\right)$ as equations (4) and (6) respectively.

As σ approaches 1, $\frac{1+\sigma}{1-\sigma}$ approaches infinity. However, with smaller values of σ, the value of $\frac{1+\sigma}{1-\sigma}$ becomes negligible in contrast to Δ^2. Thus, we will simplify our expression by removing the $\frac{1+\sigma}{1-\sigma}$ term as well as the also small -1 term. Also, for our purposes, $\frac{1}{9s} \approx \frac{1}{s}$, thus we can now rewrite $\text{Var}\left(\tilde{\Delta}_{IS}\right)$ as:

$$\text{Var}\left(\tilde{\Delta}_{IS}\right) \leq \frac{1}{9s}\left(\frac{1+\sigma}{1-\sigma} - 1\right)\Delta^2 \approx \frac{1}{s}\Delta^2.$$

.

Similarly, for our expression of $\text{Var}\left(\tilde{\Delta}_{VRm}\right)$, we know $\frac{n}{s}$ is far greater than 1. Thus, omitting 1 from our expression will not fundamentally change it. In addition, as $\sigma < 1$, σ^2 is a very small number, and we can remove it multiplied by 2 from our expression also. Thus, we can rewrite as:

$$\text{Var}\left(\tilde{\Delta}_{VRm}\right) \leq 2\sigma^2 \sum_{i=1}^{n} \Delta_i^2 \left(1 + \frac{n}{s}\right) \approx \sum_{i=1}^{n} \Delta_i^2 \frac{n}{s}.$$

.

Now that we have our two simplified expressions for $\text{Var}\left(\tilde{\Delta}_{IS}\right)$ and $\text{Var}\left(\tilde{\Delta}_{VRm}\right)$, we can compare them.

Both expressions are multiplied by $\frac{1}{s}$, so we can remove that from both and simply compare Δ^2 to $\sum_{i=1}^{n} \Delta_i^2 n$. We will begin by rewriting Δ as follows:

$$\begin{aligned}\Delta &= \sum_{i=1}^{n} \Delta_i \\ &= \begin{bmatrix}\Delta_1 & \Delta_2 & \cdots & \Delta_n\end{bmatrix} \cdot \begin{bmatrix}1 \\ 1 \\ \vdots \\ 1\end{bmatrix}\end{aligned}$$

By the Cauchy–Schwarz inequality, we know $|\langle \mathbf{u}, \mathbf{v} \rangle| \leq \|\mathbf{u}\|\|\mathbf{v}\|$. Squaring both sides, we also get $|\langle \mathbf{u}, \mathbf{v} \rangle|^2 \leq \|\mathbf{u}\|^2\|\mathbf{v}\|^2$.

Thus, using this inequality, we know that $\Delta^2 \leq \sum_{i=1}^{n} \Delta_i^2 n$, and so, in the case of multiplicative noise, $\text{Var}\left(\tilde{\Delta}_{IS}\right) \leq \text{Var}\left(\tilde{\Delta}_{VRm}\right)$.