Title

Sophia Hubscher
Manning College of Information and Computer Sciences
shubscher@umass.edu

Committee Chair: Cameron Musco cmusco@cs.umass.edu
Second Committee Member: Ghazaleh Parvini gparvini@cs.umass.edu
Research Type: Thesis

1 Introduction

What are you investigating and why?

Provide a general description of your Honors Thesis topic (about one page)

State the scientific question or creative endeavor (at least one page): describe objectives, hypotheses, and/or other discipline-specific inquiry.

Explain the significance of the research question or the creative endeavor (at least one page): relate the research question or creative endeavor to the key literature. explain the importance of your topic to the advancement of knowledge in the discipline.

Here's a draft for your introduction based on your project focus. I've integrated some key details on objectives, hypotheses, and significance, with room for you to add any specific theoretical or experimental details you'd like to highlight.

2 Notation

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.

Table 1: List of notation used.

3 Background

3.1 Introduction

Counting triangles is a fundamental problem in graph theory with widespread applications in social networks, bioinformatics, and more [10]. These triangles, are formed by three mutually connected nodes, as shown in Figure 1, which contains three triangles.

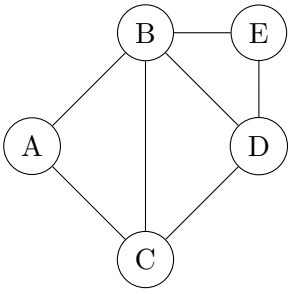

Figure 1: Graph with triangles formed between vertices (A, B, C), (B, C, D) and (B, D, E).

These triangles can, in social network graphs, represent closed friendships, indicating a high level of local connectivity, which can give insight into the network as a whole. For example, in an analysis of the effect of recommender systems on the social network Twitter, an increase in the percentage of edges that closed triangles following the introduction of a "Who to Follow" friend-to-friend recommendation algorithm was used as evidence of the algorithm's efficacy.

Unfortunately, for large graphs, especially sparse ones, where the number of edges is much smaller compared to the number of possible edges, efficiently counting these triangles poses significant computational challenges.

3.2 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 generally refer to undirected graphs unless otherwise specified, although the methods described can be extended to directed graphs as well.

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 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].

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, which is another area for improvement. 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.

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 [17], 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 [2], where each processor independently sparsifies its assigned partition of the graph, can improve accuracy.

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 [14] focuses on estimating 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 2).

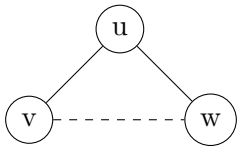

Figure 2: Wedge formed by vertices u, v, and w. Nodes v and w may or may not be connected.

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. Thus, this operation takes $O(n)$ time where n is the number of nodes 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. Wedge sampling tends to work well in graphs with a large number of high-degree vertices, where it becomes easier to sample many wedges at once, but unlike edge sampling, it cannot be efficiently done using data structures like adjacency matrices or adjacency lists. Thus, wedge sampling comes with an additional preprocessing step that adds to runtime.

3.3.2 Linear Algebraic Methods

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

Graphs can be conveniently represented using adjacency matrices, which, in social network analysis, are typically referred to as *sociomatrices* [4]. In these matrices, each row and column represents a node, and edges between nodes are represented as 1s in the corresponding matrix entry.

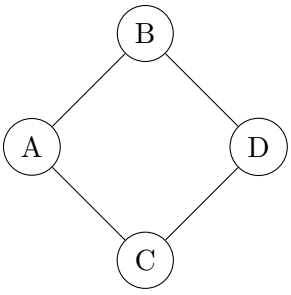

Figure 3: Graph representation of vertices A, B, C, and D.

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

Figure 4: 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) that each vertex participates in. 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 5. Thus, the trace of A^3 is divided by six to scale down to the global triangle count.

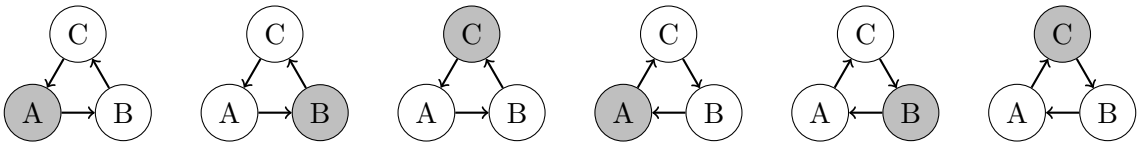

Figure 5: Six different ways to arrive at a length-three path in a triangle.

To compute A^3, we first need to calculate A^2 (which takes $O(n^3)$ 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 $O(n)$, 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.

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 [15], 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 $O(n^{\log_2 7})$. 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 this algorithm relies on the use of extremely large constants and is thus rarely used in real-world applications [18].

EigenTriangle Algorithm

Another significant approach in triangle counting is the use of spectral methods. One such method is the EigenTriangle algorithm [16], which estimates the triangle count Δ by considering the spectral decomposition of the adjacency matrix A.

The EigenTriangle algorithm is based on the observation that the number of triangles in a graph is closely related to the spectrum of its adjacency matrix. In particular, the adjacency matrix A is decomposed as:

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

where U is a matrix whose columns are the 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 $\Delta = \frac{1}{6} \sum_{i=1}^{n} \lambda_i^3$, and can be estimated using:

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

where λ_i are the k eigenvalues of largest magnitude of the adjacency matrix A. The runtime of EigenTriangle is dominated by the cost of approximating the top k eigenvalues and eigenvectors of A, which, using the Lanszos method [6], can be done in roughly $O(km)$ time, where m is the number of edges and k is typically much smaller than the number of nodes n. This is a significant improvement over the runtimes of direct methods.

Specifically, for the direct method in which was compute the trace of A^3, we know $trace(A^3) = \sum_{i=1}^{n} \lambda_i(A^3) = \sum_{i=1}^{n} \lambda_i^3$. Thus, we see that EigenTriangle approximates $trace(A^3)$. Given this,

it makes sense that this runtime is a substantial improvement over the complexity of computing trace(A^3) directly.

TraceTriangle Algorithm

The TraceTriangle algorithm [3] is a randomized algorithm designed for efficient triangle estimation in large graphs. It leverages trace-based techniques, which compute the trace of matrix powers to approximate the number of triangles. Specifically, the algorithm 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 [9], 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, it's 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.

Comparing TraceTriangle to the EigenTriangle algorithm, TraceTriangle achieves higher accuracy across multiple types of graphs [3]. 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 theoretical computer science have been adapted for this problem, particularly in designing faster algorithms.

3.4.1 Variance Reduction

Variance reduction [13] is another general technique that can be applied to triangle estimation, improving accuracy without increasing the number of samples needed.

Variance reduction methods 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. Specifically, we can begin by finding a relationship between the degree of nodes and the number of triangles they are involved in. This can be done by plotting nodes' degrees (d_i) versus triangle counts (Δ_i) on a log-log plot, 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 M by applying this line of best fit relationship to all nodes in the graph:

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

Next, 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 (written Δ_i) and find the difference between those actual triangle counts and their estimated triangle counts using the line of best fit relationship. We then take the sum of these errors and scale them up to estimate the error on our global triangle count (written 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 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.

Importance Sampling

One example of a variance reduction method 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 [10]. This is because uniform sampling does not prioritize areas of the graph that may have a disproportionately large impact on the estimate. Consequently, the computational cost can be high for achieving a desired accuracy level in many cases.

When using importance sampling [12], 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 1 (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.

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:

$$D = \sum_{i=1}^{n} d_i^{\alpha}$$
$$p_i = \frac{d_i^{\alpha}}{D}.$$

Next, we sample s nodes based on their probabilities p_i. 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 $\frac{1}{s \cdot p_i}$. 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.

3.4.2 Learning-Augmented Algorithms

A learning-augmented algorithm [11] 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 [8] 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 [7] is applied however, where machine-learned predictions are used to "warm-start" the algorithm, 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 [5], 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, this method can be highly effective, as accurate predictions can significantly enhance algorithms' efficiency or result quality.

4 Methods

How are you conducting your research or creative endeavor?

Detail the procedures or techniques you are using to conduct your research or produce your artifact.

Describe the resources or materials you are using in your research.

If specialized training was required for your research manuscript or creative portfolio (e.g., lab safety certification or human/animal testing) describe the training you received.

5 Evaluation

How will your work be reviewed and graded?

List the measurable goals the Thesis Committee expects you to accomplish during the semester. Indicate how your Thesis Committee will provide feedback regarding your progress. Indicate how your Thesis Committee will assess the viability of your research or creative endeavor in both the written and presentation formats. If you are registering for the Portfolio option, be specific about the artifact that you will produce in addition to the manuscript, such as a performance, musical score, architectural blueprint, engineering invention, screenplay, business case study, collection of original poetry, or art exhibition.

6 Communication

Meetings with Committee Chair: I will meet with my Committee Chair for half an hour once a week throughout the semester. In each meeting, I'll bring an informal report of my progress that covers the experiments and research I've completed over the week. During each meeting, my Committee Chair and I will discuss experiments that can be run in the upcoming week. In addition, I will continuously update my thesis report, and will go over any questions relating to it at our weekly meetings.

Meetings with Full Committee: Full committee meetings, where both of my committee mem-

bers and I are present, will happen monthly. In these meetings we will discuss my progress toward the project milestones and brainstorm new approaches I could try.

Weekly Time Commitment: Outside of these meetings, I will be committing 6-10 hours each week of work to my thesis. The specific hours worked will vary depending on the goals set at each weekly meeting.

7 Timeline

- Week of 02/10/25: Milestone 1
- Week of 03/03/25: Milestone 2
- Week of 03/24/25: Milestone 3
- Week of 04/07/25: 1st Draft of thesis submitted Committee Chair
- Week of 04/21/25: 2nd Draft of Thesis submitted to CHC PATHS
- Week of 05/05/25: Oral Defense
- Week of 05/10/25: Final Submission to CHC (No later than last day of classes)

References

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

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

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

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

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

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

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

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

[9] 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.

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

[11] 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.

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

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

[14] 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.

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

[16] 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.

[17] 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.

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