1 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.
d_i	Degree of node i.

Table 1: List of notation used.

2 Literature Review

2.1 Outline DELETE LATER

- Why do we care about triangles? (Motivation)
- If we don't care about runtime, how do we calculate them? (Exact algorithms)
- Look, it's slow.
- Methods for fast triangle counting:
  - Talk about different methods:
    - Split into subsubsections:
      - Linear algebraic methods (e.g. Eigentriangle, Hutchinson's estimator paper by Avron)
      - Sampling methods
- Our techniques that aren't specific to triangles:
  - Importance sampling
  - Variance reduction
  - Learning-augmented algorithms
    - If I have predictions about my output, how can I use them to augment my algorithms?
  - Talk about these techniques and where they've been used

2.2 Introduction

Counting triangles is a fundamental problem in graph theory with widespread applications in social networks, bioinformatics, and more [3]. These triangles, formed by three mutually connected nodes, can, in social network graphs, represent closed friendships, indicating a high level of local connectivity, which can give great insight into the network as a whole.

However, 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.

2.3 Methods for Triangle Counting

Triangle counting can be approached in a variety of ways, each with its advantages and drawbacks. 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 cubic complexity arises because the number of combinations of three vertices grows cubically with the total number of vertices [1].

While this method is straightforward, it is inefficient for large graphs due to its high computational cost. In smaller graphs, this runtime may be acceptable. However, when scaling up to larger graphs—such as those representing extensive social networks, a $\Theta(n^3)$ runtime quickly becomes impractical.

Thus, researchers have turned to alternative triangle counting and estimation methods.

2.3.1 Linear Algebraic Methods

Graphs can be conveniently represented using adjacency matrices, which in social network analysis are typically referred to as *sociomatrices* [2]. In these matrices, each row and column represents a node, while edges between nodes are represented as 1s in the corresponding matrix entry.

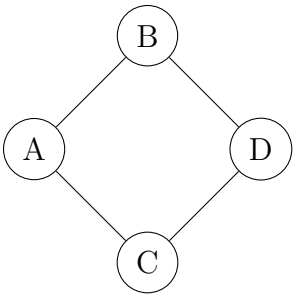

Figure 1: Graph representation of vertices A, B, C, and D.

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

Figure 2: Adjacency matrix corresponding to the graph.

By using these adjacency matrices and leveraging linear algebra techniques, we can calculate triangle counts more efficiently. The trace of the matrix offers a simple and effective approach. Specifically, we can use the formula:

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

This formula is derived from the fact that the diagonal elements of A^3 count the number of triangles that each vertex participates in, effectively summing these counts across all vertices.

To compute A^3, you first need to calculate A^2 (which takes $O(n^3)$ for an $n \times n$ matrix) 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 you need to iterate over the n diagonal elements. Thus, the overall runtime complexity for the operation is $O(n^3)$.

Strassen's Algorithm

This previous approach assumes that matrix multiplication is performed using the standard algorithm. However, more sophisticated techniques, such as Strassen's algorithm [5], can reduce matrix multiplication time. Strassen's algorithm divides each matrix into smaller submatrices and performs a series of additions and multiplications that require only seven multiplications of these smaller matrices instead of eight, which is typical in standard matrix multiplication.

For large, square matrices, such as sociomatrices, 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 $O(n^{2.81})$. 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 $O(n^{2.81})$.

EigenTriangle Algorithm

Another significant approach in triangle counting is the use of spectral methods. One such method is the EigenTriangle algorithm [6], 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. The cube of the adjacency matrix A^3 can be approximated by considering the largest eigenvalues and their corresponding eigenvectors, significantly reducing the computational complexity when compared to the direct calculation of A^3.

Once the decomposition is performed, the number of triangles can be estimated using:

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

where λ_i are the k top eigenvalues of the adjacency matrix. The runtime of EigenTriangle is dominated by the cost of computing the top k eigenvalues and eigenvectors of A, which can be done in $O(km)$, where m is the number of edges and k is typically much smaller than the number of nodes n. This is a substantial improvement over the complexity of direct methods like trace(A^3).

2.3.2 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 [7], where each edge in 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.

Wedge Sampling

Wedge sampling [4] focuses on counting 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. 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 closed 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.

References

[1] Mohammad Al Hasan and Vachik S. Dave. Triangle counting in large networks: a review. *WIREs Data Mining and Knowledge Discovery*, 8(2):e1226, March 2018.

[2] Corlin O. Beum and Everett G. Brundage. A Method for Analyzing the Sociomatrix. *Sociometry*, 13(2):141–145, 1950.

[3] Laaszlo Lovaasz. *Large networks and graph limits*. Number volume 60 in American Mathematical Society colloquium publications. American Mathematical Society, Providence, Rhode Island, 2012.

[4] 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, May 2013. arXiv:1202.5230 [cs].

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

[6] 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, December 2008. ISSN: 2374-8486.

[7] 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, New York, NY, USA, June 2009. Association for Computing Machinery.