Chapter 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 1.1: List of notation used.

Chapter 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 subsections:
      - 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. 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 insite 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 done in a number of ways. Most trivially, we an use a brute force technique in which we enumerate all distinct sets of three vertices u, v, w and check if they form 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, optimally, that we have edges stored in a hash table where retrival takes $O(1)$ time, the time complexity of this brute force approach is $\Theta(n^3)$, since the number of such three-vertex sets grows cubically with the number of vertices [1]. This method is straightforward but inefficient for large graphs due to its high computational cost.

While for smaller graphs, this runtime is acceptable, when dealing with larger graphs, such as those representing large social netwworks, a $\Theta(n^3)$ runtime becomes impractical. Thus, we turn toward alternative triangle counting and estimation methods.

2.3.1 Linear Algebraic Methods

Graphs can be easily represented with adjacency matrices, where each row/column represents a node, and edges between those nodes are stored as 1s in the matrix.

Using this adjaceny matrix, we can leverage linear algebra techniques to calculate triangle counts more quickly. The simplest method of this is to analyze the trace of the matrix. Specifically, we can use $\Delta = \frac{1}{6}\text{trace}(A^3)$.

2.4 Recycling Bin DELETE LATER

To address this, we turn toward randomized algorithms. These algorithms rely on probabilistic techniques to sample subsets of the graph's nodes, estimating the global triangle count based on local observations. By doing this, these algorithms significantly reduce the runtime compared to exact methods, particularly in sparse graphs where only a fraction of potential edges exist. Additionally, randomized algorithms often provide tunable accuracy, allowing for a trade-off between precision and performance, making them ideal for processing large-scale networks.

Bibliography

[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.