1 Introduction

1.1 Motivation

This project aims to evaluate algorithms for analyzing graphs. Graphs (also known as networks) represent entities (nodes) and the connections between them (edges). Their generality lends them to a vast number of applications, and the analysis of large graphs has lead to interesting results in disease modeling, sociology, supply chain management, and biology [15][14][13][2].

One method of analyzing graphs is through the use of graph statistics. In contrast to *metrics*, which are single numbers that describe the entire graph (such as average degree, connectivity, or diameter), *statistics* assign a value to each node. One widely used and important statistic is betweenness centrality, which measures the importance of a node. While there are several other measures of importance, betweenness centrality is by far the most frequently used [10] TODO: mention applications. Further, algorithms for computing betweenness centrality can be trivially extended to compute several other centrality statistics, including closeness centrality and stress centrality [6].

The betweenness centrality of a node v is defined as the number of shortest paths from any node to any other node which pass through v. Formally, we can write:

$$C(v) := \sum_{s \neq v \neq t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{1}$$

where $\sigma(s,t|v)$ is the number of shortest paths from s to t that pass through v, and $\sigma(s,t)$ is the total number of shortest paths from s to t.

Betweenness centrality was first defined in 1977 by Linton Freeman [9]. Naïve implementations can compute betweenness centrality for all nodes in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space on a graph of n nodes and m edges by solving the all-pairs shortest paths problem. In 2001, Brandes improved this to $\mathcal{O}(nm)$ time and $\mathcal{O}(n+m)$ space for unweighted graphs, and $\mathcal{O}(nm + n^2 \log(n))$ time and $\mathcal{O}(n+m)$ space for weighted graphs [6]. No asymptotic improvement has been found since, and $\mathcal{O}(nm)$ time is still too high to compute betweenness centrality for large graphs, which may have millions of nodes and edges.

Despite the lack of asymptotic improvements, there have been dozens of proposed algorithms to compute betweenness centrality faster or to use statistical methods to rapidly compute approximations to it. However, papers proposing new algorithms have just compared them to the algorithm by Brandes, and direct comparisons between them have been limited (see *Related Work*). In this project, I will examine and evaluate five promising algorithms for computing betweenness centrality, detailed in *Descriptions of Selected Algorithms*.

1.2 Related Work

Despite the importance of efficient betweenness centrality algorithms, relatively little work has been done to compare the various existing algorithms.

Of the examined algorithms, no author evaluates their algorithm in a comparable way. Bader et al. does not compare the performance of their algorithm to any other [3]. Borassi and Natale [5] compare the performance of their algorithm to the `RK`, `ABRA-Aut`, and `ABRA-1.2` algorithms. Geisberger et al. [10] and Erdős et al. [8] compare their algorithms with `Brandes`. Brandes [6], in turn, compares his algorithm to the now-obsolete `Floyd-Warshall` algorithm. This is copied from my proposal, is that alright?

There are two major studies which attempt to compare the performance of multiple betweenness centrality algorithms. Al-Ghamdi et al. [1] create a system for benchmarking betweenness centrality algorithms on a supercomputer. Their paper focuses primarily on the creation of the benchmark system and the process of benchmarking, and the resulting comparison is treated as a corollary. They compare the performance of the algorithms by Bader et al. and Geisberger et al., as well as others algorithms not discussed in this project.

The other study, by Matt et al. [12] compares a smaller number of algorithms by using them to solve two "real world" problems – clustering a graph and iteratively removing the most important nodes. They run the tests on more typical hardware than Al-Ghamdi et al. but concluded that by running their tests on a computer with a more powerful GPU, the relative rankings of the algorithms changes. They evaluated the algorithms by Borassi and Natale, Brandes, and Geisberger et al., among others.

1.3 Project Aims

In this project, I will create efficient implementations of five algorithms (see Descriptions of Selected Algorithms) and a testing harness to evaluate their performance. I will instrument each of these algorithms to compute performance metrics (time per node, total time, memory usage, the number of graph reads). I will run each of these on large graphs using a high performance computing server and determine the accuracy and efficiency trade-offs each algorithm offers.

1.4 Overview of selected algorithms

Table 1: Selected Algorithms

Algorithm	Type	Brief overview
`Brandes` [6]	Exact	The most commonly used betweenness centrality algorithm [8]. It is used as a baseline to represent the current state of the world.
`Brandes++` [8]	Exact	Uses a divide and conquer approach to run the `Brandes` algorithm on smaller subgraphs and collate the results. Claims significant speedups to `Brandes`, but never before tested.
`BP2008` [7]	Approximate	Randomly selects source nodes to calculate betweenness centrality. It is a simple modification to `Brandes` and is what Geisberger et al. build off of.
Geisberger et al. [10] (`Linear`, `Bisection`, `Bisection Sampling`)	Approximate	Three different algorithms that are extensions of `BP2008` with a new weighting strategy. `Linear` was the most accurate approximate algorithm tested by Matta et al. [12].
Bader et al. [3]	Approximate	This algorithm uses a different sampling strategy. TODO: what is it? On a supercomputer, it had significantly better performance than any other algorithm [1].
`KADABRA` [5]	Approximate	Uses advanced statistical methods TODO: which? to select random shortest paths to compute betweenness centrality. It has a guaranteed lower bound on accuracy and was the fastest algorithm in tests by Matta et al. [12].

1.5 Descriptions of Selected Algorithms

1.6 Brandes

When betweenness centrality was first described by Freeman in 1977, there were no known approaches to calculate it other than the naïve approach of calculating all shortest paths and doing the summation in equation 1. By using the `Floyd-Warshall` algorithm, it is possible to do this in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space for a graph with n nodes.

The first major improvement to this was described by Ulrik Brandes in his 2001 paper "A Faster Algorithm for Betweenness Centrality" [6]. Brandes introduces *pair-dependency*, representing the proportion of shortest paths between s and t that pass throubh v, defined as

$$\delta(s,t|v) = \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{2}$$

Further, he defines the *dependency* of s on v as

$$\delta(s|v) = \sum_{t \in V} \delta(s, t|V) \tag{3}$$

From equation 1, can see that

$$C(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)} = \sum_{s \neq v \neq t \in V} \delta(s, t|V) = \sum_{s \neq v \in V} \delta(s|v) \tag{4}$$

The crucial observation in Brandes's paper is that $\delta(s|v)$ follows the following recursive relation:

$$\delta(s|v) = \sum_{w : v \in pred(w)} \frac{\sigma(s, v)}{\sigma(s, w)} \cdot (1 + \delta(s|w)) \tag{5}$$

Where $pred(v)$ is the set of immediate predecessors of v on all shortest paths from s to any node t that pass through v.

With this, Brandes proved that we can calculate $\delta(s|v)$ in two phases. First, compute the solution to the single-source shortest paths (SSSP) problem for s. That is, compute all shortest paths from s to any node t, storing $pred(v)$ for all $v \in V$ and a list of nodes in non-ascending order of distance from s. This can be done by running a breadth-first search (BFS) (for unweighted graphs) or Djikstra's algorithm (for weighted graphs) starting at s. While exploring the graph, add the immediate predecessor of each explored node v to $pred(v)$ and add the node to a stack. This operation takes $\mathcal{O}(n)$ time for unweighted graphs, $\mathcal{O}(m + n \log(v))$ time for weighted graphs, and takes $\mathcal{O}(n + m)$ space.

Additionally, augment the SSSP to also calculate $\sigma(s, v)$ by adding $\sigma(s, w)$ to $\sigma(s, v)$ when exploring the edge (w, v).

Next, accumulate dependencies by iteratively popping elements w off of the stack and incrementing each $v \in pred(w)$ by $\frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta(s|w))$. If all $\delta(s|v)$ are initialized to 0, then each δ will be the value prescribed by equation 5 once the stack is empty. This step takes $\mathcal{O}(m)$ time and $\mathcal{O}(n + m)$ space.

Finally, increment $C(v)$ by $\delta(s|v)$ for all $v \in V$. By iterating over all $s \in V$, this will compute $C(v)$ for all $v \in V$.

Overall, this algorithm takes $\mathcal{O}(nm)$ time for unweighted graphs, $\mathcal{O}(nm + n^2 \log(n))$ time for weighted graphs, and $\mathcal{O}(n + m)$ space in either case.

This algorithm, denoted here as `Brandes`, is the de facto standard algorithm uses to compute betweenness centrality. In fact, it has been proven that the complexity of computing the betweenness centrality of a single vertex is at least $\mathcal{O}(n^2)$ if the Strong Exponential Time Hypothesis holds [4]. Therefore if the graph is sparse (that is, $m \sim n$), then the `Brandes` algorithm has optimal asymptotic performance, even for computing the centrality of a single node.

1.7 Brandes++

1.7.1 Target Set Betweenness Centrality

In "A Divide-and-Conquer Algorithm for Betweenness Centrality" [8] Erdős et al. describe a slightly different metric - target set betweenness centrality. It is defined as follows: For a set $S \subset V$ such that $2 \leq |S| \leq |V|$, the target set betweenness centrality is defined as the betweenness centrality considering only shortest paths between nodes in the target set, so

$$C^S(v) = \sum_{s \neq v \neq t \in S} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{6}$$

Observe that if $S = V$ then $\forall v \in V$, $C^S(v) = C(v)$.

Target set betweenness centrality can be calculated with simple modifications to the `Brandes` algorithm: SSSP is only run from each $s \in S$ and equation 5 becomes

$$\delta(s|v) = \sum_{w:v \in pred(w)} \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (I_{w \in S} + \delta(s|w)) \tag{7}$$

where $I_{w \in S}$ is an indicator that is 1 if $w \in S$ and 0 otherwise.

Erdős et al. aim to speed up this calculation by using techniques similar to those used in network routing - splitting the graph into clusters, running a computations on each cluster, and aggregating the results.

1.7.2 The Algorithm

Erdős et al. detail a new algorithm for efficiently computing target set betweenness centrality. Their algorithm, which they call `Brandes++` consists of five main steps: clustering the graph, adjusting the clustering, constructing a 'skeleton graph', calculating the betweenness centrality of nodes in the skeleton graph, and finally calculating the betweenness centrality of nodes not in the skeleton graph.

We or I? I will describe each step individually

1. **Clustering**

   First, we split the overall graph G into a set of sub-graphs $\{G_1, G_2 \dots G_k\}$ in a way that minimizes the number of edges between sub-graphs. Erdős et al. evaluate the performance of several graph partitioning algorithms and find that using the `METIS` software package results in the best better performance of `Brandes++`. Erdős et al. encounter the issue that the speed of the highly optimized `METIS` package can't be compared to their Python implementation of `Brandes++`. In order to keep the clustering algorithm's performance comparable to the other algorithms, I re-implement the paper that `METIS` is based off of. "A fast and high quality multilevel scheme for partitioning irregular graphs"[11] describes the algorithm used by `METIS`, a multilevel graph partition algorithm we will refer to as `MLGP`.

   MLGP diagram `MLGP` recursively splits a graph into two partitions by using a three-step process. The algorithm aims to minimize the edge cut, which is the sum of the weights of edges between nodes in different partitions.

1. First, the graph is 'coarsened' a pre-determined k times by combining nodes. Each node is combined with with the neighbor connected by the heaviest edge. This step reduces the size of the graph to approximately $\frac{|V|}{2^k}$, significantly speeding up the next step. Note that `MLGP` interprets weights as closeness while `Brandes++` interprets them as distance, so all weights must be inverted before running `MLGP`.
2. Next, the graph is actually partitioned. Of the various partitioning algorithms tested by Karypis and Kumar, `GGGP` consistently achieves the best performance and results [11].

   `GGGP` begins by adding a random vertex to a set T. Then for each neighbor, `GGGP` calculates the change in the edge cut if that node were to be added to T. This is called the gain. GGGP then iteratively adds the neighbor with the lowest gain to T, and updates the gain of that node's neighbors. This iteration continues until half of the graph has been added. Then, we define the partitions as the two sets T and $V \setminus T$.

   Since the starting node greatly affects the quality of the partition, `MLGP` runs `GGGP` multiple times and selects the partitioning with the lowest edge cut.
3. Finally, the graph is uncoarsened and refined. For each of the k times the graph was coarsened we do the following:
   i. Create a new graph where nodes that were merged in the j^{th} iteration are unmerged
   ii. Refine the graph: `MLGP` uses a modification of the Kernighan–Lin algorithm, where we calculate the gains (as described above) of all nodes with neighbors in the opposite partition. Then, two nodes in different partitions which would reduce the edge cut the most when swapped are swapped. This process is repeated until no progress has been made in a constant number of swaps or if there are no more nodes to be swapped. Then the iteration with the best edge-cut (computed adding the gains of executed swaps) is selected and output as the refined partition.

       This is made efficient by using a specialized data structure to store the gains and only selecting from all combinations of the best 3 nodes from each partition.

2. Adjustment of Partitions

Step 4 requires that each target node $s \in S$ is a frontier node (has a neighbor in another partition). To satisfy this, we iterate over all $s \in S$ and if s is not a frontier node, we remove it from its partition and create a new partition containing only s. If $S = V$ then every node will be a frontier node and step 3 will take the same time as `Brandes`, but if S is small and the clustering is good, not every node will be a frontier node.

3. Skeleton Graph

Once the graph has been partitioned, we construct a simplified representation of the graph, called SKELETON, with the following properties:

If u and v are frontier nodes in different partitions and the edge $(u, v) \in G$ with weight w, then $(u, v) \in$ SKELETON with weight w.

If u and v are frontier nodes in the same partition P, and there exists a shortest path from u to v of length l only going through non-frontier nodes (to avoid double-counting), then $(u, v) \in$ SKELETON with weight l. We can calculate this by doing an SSSP from each frontier node u, skipping any $v \notin P$ and not adding the neighbors of any other frontier node $v \in P$.

We also associate a value θ with each edge, defined as the number of shortest paths that the edge represents. If u and v are in different partitions, then $\omega(u, v) = 1$. If u and v are in the same partition, we calculate θ when we run the SSSP to determine l.

4. **BRANDES_SK**

   The target-set `Brandes` (described in Target Set Betweenness Centrality) is run on SKELETON, with a few modifications to account for the fact that paths between frontier nodes in the same partition can represent multiple paths (as given by θ). The modifications are as follows.

   Rather than incrementing $\sigma(s, v)$ by $\sigma(s, w)$ when exploring the edge (w, v), we instead increment it by $\sigma(s, w) \cdot \theta(w, v)$

   We must also account for the multiple paths when accumulating delta, so use the following instead of equation 5:

$$\delta(s|v) = \sum_{w} \sigma(v, w) \frac{\sigma(s, v)}{\sigma(s, w)} \cdot (I_{w \in S} + \delta(s|w)) \tag{8}$$

   Note that if $|\text{SKELETON}| = |V|$ then this takes exactly as much time as `Brandes`, illustrating that we must have a target set S such that $|S| < |V|$ for the algorithm to be effective, and we must have a good clustering that minimizes the number of frontier nodes.

5. **CENTRALITY**

   Thus far, we have calculated the betweenness centrality of every node on the skeleton graph, which includes every node in the target set S. However, we have not yet computed the betweenness centrality of nodes not in the skeleton graph (those which are not frontier nodes).

   To do so, Erdős et al. describe one final algorithm.

   I have managed to get into contact with one of the authors of the paper, and will write this section after that discussion

Brandes and Pich(2008)

While I didn't originally propose to implement the algorithm detailed by Brandes and Pich in 2008 [7], the algorithm by Geisberger et al. is a direct extension of it, making this algorithm useful both for comparing to other approximate algorithms and for understanding them.

Their algorithm (here called `BP2008`) is a simple extension of the `Brandes` algorithm. Rather than iterate over every source $s \in V$, we compute $n \leq |V|$ samples by randomly selecting source nodes and doing the same computation as in `Brandes`, accumulating the centrality. At the end, we multiply each centrality by $\frac{|V|}{n}$ to extrapolate from these n samples.

While Brandes and Pich test several different methods for randomly selecting sources, they find that simply selecting a random source with uniform probability $\frac{1}{|V|}$ results in the highest accuracy estimator.

1.8 Geisberger et al.

Geisberger et al. detail three different algorithms that use the same framework.

I can't think of a more concise way to describe their framework than theirs, how much duplication is okay (equations etc.)

1.8.1 Linear Scaling

The first algorithm described by Geisberger et al. uses $f(x) = x$ as the scaling function, which is why it is called the Linear Scaling algorithm.

It is a simple modification to `BP2008`, we can simply use the following instead of equation 5:

$$\delta(s|v) = \sum_{w:v \in pred(w)} \frac{\mu(s,v)}{\mu(s,w)} \cdot \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta(s|w)) \tag{9}$$

Where $\mu(s,v)$ is the distance from s to v.

1.8.2 Bisection Scaling

The second algorithm uses

$$f(x) = \begin{cases} 0 & \text{for } x \in [0, 1/2) \\ 1 & \text{for } x \in [1/2, 1] \end{cases} \tag{10}$$

Implementing this scaling function is somewhat more complicated - each node t only contributes to the δ of nodes more than halfway on the path from s to t. The way Gesiberger et al. address this is by modifying the SSSP algorithm to store successors rather than predecessors. Then, they do a depth-first traversal of this shortest-paths directed acyclic graph (DAG). When a node $v$is visited, we call `Decrement_Half` (described below). Then, we continue the depth-first traversal and when all children have been explored, we can do $\delta(s|v) = \delta(s|v) + \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta(s|w))$ for all children $w \in succ(v)$.

`Decrement_Half` decrements the node halfway on the path from s to v by $\frac{1}{\sigma(s,v)}$ Since this is called once for each path going to v, this entirely eliminates the contribution of node v on the node halfway from s to v. To determine which node to decrement, we maintain a list representing the current path from s to v.

If the graph is unweighted and the stack has k elements, we decrement the node in position i, where

$$i = \begin{cases} \max(0, \lfloor (k-2)/2 \rfloor) & \text{if forward search} \\ \lfloor (k-1)/2 \rfloor & \text{else} \end{cases} \tag{11}$$

If the graph is weighted, we can store the distances calculated when solving the SSSP and do a binary search for the last node with distance less than $d(v)/2$ (if a forward search) or the first node with a distance greater than $d(v)/2$ (if a backward search).

This algorithm visits each node v $\sigma(s, v)$ times rather than once as the Linear Scaling algorithm does, but this performs well if most shortest paths are unique (there is only one way shortest way to reach a node), such as in road networks.

1.9 Bisection Sampling

To address the runtime problems of the Bisection Scaling algorithm, Geisberger et al. introduce the Bisection Sampling method. Here, they do the following procedure k times for some constant k.

1. For each node v in the shortest-paths DAG, randomly pick a parent w with probability $\frac{\sigma(s,w)}{\sigma(s,v)}$.
2. For this new tree, run the accumulation step of Bisection Scaling
3. For each $v \in V \setminus \{s\}$, increment $C(v)$ by $\delta(s, v)/k$

This reduces the DAG to a tree, ensuring we visit each node only once. Since this is sensitive to the choice of parents, we run this sampling multiple times and average the results.

1.10 Bader et al.

I want to talk to you about this algorithm - all it is is the `BP2008` algorithm where the end condition is replaced with the condition that the betweenness centrality of a pre-chosen node reaches a certain threshold. One of the reviews of betweenness centrality algorithms finds that it's the fastest, while the other points out that it also had the worst performance in the first algorithm and the faster performance was probably just because it ended early.

Also it's performance is wildly unpredictable because it depends on the choice of node. If a node with low betweenness centrality is chosen, it will run many many iterations, or it may end very quickly if the chosen node has very high betweenness centrality.

1.11 KADABRA

References

[1] Ziyad AlGhamdi, Fuad Jamour, Spiros Skiadopoulos, and Panos Kalnis. A benchmark for betweenness centrality approximation algorithms on large graphs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, SSDBM '17, New York, NY, USA, 2017. Association for Computing Machinery.

[2] Gil Amitai, Arye Shemesh, Einat Sitbon, Maxim Shklar, Dvir Netanely, Ilya Venger, and Shmuel Pietrokovski. Network analysis of protein structures identifies functional residues. *Journal of Molecular Biology*, 344(4):1135 – 1146, 2004.

[3] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In Anthony Bonato and Fan R. K. Chung, editors, *Algorithms and Models for the Web-Graph*, pages 124–137, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[4] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the Square - On the Complexity of Quadratic-Time Solvable Problems. *arXiv:1407.4972 [cs]*, July 2014. arXiv: 1407.4972.

[5] Michele Borassi and Emanuele Natale. KADABRA is an adaptive algorithm for betweenness via random approximation. *CoRR*, abs/1604.08553, 2016.

[6] Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[7] Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318, 2007.

[8] Dora Erdos, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. A divide-and-conquer algorithm for betweenness centrality, 2015.

[9] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[10] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In *Proceedings of the Meeting on Algorithm Engineering Expermiments*, page 90–100, USA, 2008. Society for Industrial and Applied Mathematics.

[11] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[12] John Matta, Gunes Ercal, and Koushik Sinha. Comparing the speed and accuracy of approaches to betweenness centrality approximation. *Computational Social Networks*, 6(1):2, Feb 2019.

[13] Philip Nuss, T.E. Graedel, Elisa Alonso, and Adam Carroll. Mapping supply chain risk by network analysis of product platforms. *Sustainable Materials and Technologies*, 10:14 – 22, 2016.

[14] Evelien Otte and Ronald Rousseau. Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science*, 28(6):441–453, 2002.

[15] Navavat Pipatsart, Charin Modchang, Wannapong Triampo, and Somkid Amornsamankul. Network based model of infectious disease transmission in macroalgae. *International Journal of Simulation: Systems, Science and Technology*, 19:11.1–11.8, 10 2018.