Fast Algorithms on Big Data

CSE 592 Social Networks

Murali Krishna Somanur Ramasamy
Department of Computer Science
Stony Brook University
New York 11790
msomanurrama@cs.stonybrook.edu

Sriganesh Navaneethakrishnan
Department of Computer Science
Stony Brook University
New York, 11790
srnavaneetha@cs.stonybrook.edu

ABSTRACT

Betweenness centrality is a fundamental measure in social network analysis, expressing the importance or influence of individual vertices (or edges) in a network in terms of the fraction of shortest paths that pass through them. Algorithms which calculate the betweenness centrality of a node in a huge graph have to be fast and efficient. The existing Brandes' algorithm runs in quadratic time of *O(nm)* for unweighted graphs and $O(nm + n^2 logn)$ for weighted graphs. Hence, improving the performance of these algorithms becomes a necessity. In this project we have studied, analyzed the existing algorithms and improved the efficiency of such algorithms using two techniques namely ***approximation*** and ***parallelization***. However, the increase in the performance of these algorithms comes with a cost. Our techniques use ***adaptive sampling***. It is a sampling algorithm in that it estimates the centrality by sampling a subset of vertices and performing SSSP computations from these vertices. It is termed adaptive, because the number of samples required varies with the information obtained from each sample. As a part of parallelization mechanism, we have tuned a tool called SNAP to compute betweenness centrality score with high concurrency.

Keywords: Betweenness centrality, approximation, parallelization, adaptive sampling, random sampling

1. INTRODUCTION

One of the most basic problems in analysis of any social network is to figure out the importance of a particular vertex or edge in that network. This is also known as *centrality* value of the vertex. Centrality is an important measure because of its various applications in several different fields like identifying the most influential person(s) in a social network, determining the key infrastructure nodes in urban networks, figuring out the major spreaders of a disease, identifying key actors in a terrorist network, analysis of an organizational behavior, study about supply chain management process, etc. In order to compute the centrality, there are well known several metrics known which include *closeness*, *stress* and *betweenness*. Out of these measures, betweenness has been used in recent years extensively for analysis of huge complex networks and real world massive social interaction networks. Betweenness is used as the primary routine in several well known algorithms for clustering and community detection in real-world huge social networks. For example, the Girvan-Newman identifies the nodes with high betweenness centrality and segregates the huge network iteratively based on these values.

Since the real-world social graphs are gigantic with several millions of nodes and edges (for instance, facebook graph), it is an implicit requirement that all the algorithms that manipulate these graphs should run in linear or sub-linear time. This new challenge of figuring out faster algorithms on a Big Data is an interesting area. In this project we came up with several ways to increase the efficiency of the existing algorithms by combining with tools which offer high performance. Further, in order to determine the betweenness centrality of a node in a graph, the existing algorithms run in quadratic time. We have improved the performance of such algorithms using few algorithmic techniques and trade-offs so that the betweenness centrality score is calculated with reasonable accuracy.

We presented details about the data we used in Section 2. Section 3 gives background knowledge about Betweeness Centrality. Methodology and the parallelization tools we used are explained in Section 4. Section 5 talks about experimental setup and results obtained. Section 6 acts as summary of this whole project.

2. DATA

For our project, we started off with fetching our Facebook friends network data using *Give Me My Data*. It is an application through which Facebook friends network can be exported in various formats for further analysis. This preliminary data that we had consisted of few thousand nodes and edges. The existing algorithms were able to manipulate the data within a minute. This data perfectly represented the real world social network, however, it is too small. The results obtained from this data couldn't be used to measure the performance of the algorithms.

Hence, we had to look for real-world huge graph to appreciate the efficiency of the existing algorithms. After some search, we found *Friendster* website, which is a gaming site with millions of users. This The data obtained from this site was huge (which was around 10 GB). The dataset contained the friends lists of 103,750,348 users. The friends lists of an additional 14,001,031 users had been marked private. In total, the dataset contains 2,586,147,869 friend connections. In graph terms: the graph contains 117,751,379 nodes and 2,586,147,869 edges. Since it was a sparse graph and our machines' computational power was relatively low to process the whole data, we extracted a subset of data and refined it.

At the end of all processing, we had a graph which had around 85,000 vertices and 120,000 edges. Since there was no information about the characteristics of the graph, we assumed an equal weight of '1' on all the edges, and hence

Table 1: Graph data used for the exprimentation

Network	Nodes	Edges	Type
Friendster	85,000	120,000	Undirected
Twitter	81,306	1,768,149	Directed

the final input graph was `undirected` and `unweighted`.[1]

For directed graphs, we have procured dataset from *Stanford Network Analysis Project (SNAP)* twitter ego network consisting of 81,306 nodes and 1,768,149 edges. Similarly, we assumed an equal weight of '1' on all edges. So, this constitutes our second final graph which is `directed` and `unweighted`.[2]

3. BETWEENESS CENTRALITY

Betweenness Centrality is one of the primary measures that is used in the analysis of graphs and networks. Analysis of graphs and networks have become a hot subject mainly because social media boom. Social Networks which provide a digital map of how people interact and move around. This has been the main source of such huge network of graphs. These networks are huge in size mainly because it deals with real world scenarios. The interactions between the nodes in the graph are also complex. This means analyzing these graphs are pretty expensive. BC is one of the major measure in the analysis of such huge graphs. BC can also help in finding other related measures which could potentially provide some important properties of the underlying graph.

Betweenness is a widely used centrality metric that is determined by the all pairs shortest-path calculation. In a graph *G = (V, E)*, where V is set of all the vertices representing actors and E is set of all edges representing the relationships between the actors or edges. The number of actors and relationships are denoted by n and m respectively. The graphs can be directed or undirected. Lets assume that each edge $e \in E$ has a positive integer weight *w(e)*. For unweighted graphs, we use *w(e) = 1*. A *path* from vertex s to t is defined as a sequence of edges $[u_i, u_{i+1}], 0 \leq i < l$, where $u_0 = s$ and $u_1 = t$. The *length* of a path is the sum of the weights of edges. We use *d(s, t)* to denote the distance between vertices s and t (the minimum length of any path connecting s and t in G). Let us denote the total number of shortest paths between vertices s and t by λ_{st}, and the number passing through vertex v by $\lambda_{st}(v)$. Let $\delta_{st}(v)$ denote the *fraction* of shortest paths between s and t that pass through a particular vertex v i.e, $\delta_{st}(v) = \frac{\lambda_{st}(v)}{\lambda_{st}}$. We call $\delta_{st}(v)$ the *pair dependency* of *s,t* on *v*.

Betweenness centrality of the vertex v is defined as

$$BC(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v)$$

Currently, the fastest known algorithm for exactly calculating betweenness of all the vertices, designed by Brandes, requires atleast O(mn) time for unweighted graphs and $O(nm + n^2 \log n)$ time for weighted graphs, where n is the number of vertices and m is the number of edges. Thus, for large scale graphs, exact centrality computation on current workstations is not practically viable.

[1]https://archive.org/details/friendster-dataset-201107
[2]https://snap.stanford.edu/data/egonets-Twitter.html

3.1 Brandes' Algorithm

Let us define the *dependency* of a source vertex $s \in V$ on a vertex $v \in V$ as $\delta_{s*}(v) = \sum_{t \neq s \neq v \in V} \delta_{st}(v)$. Then the betweenness score of v can be then expressed as

$$BC(v) = \sum_{s \neq v \in V} \delta_{s*}(v)$$

. Also, let $P_s(v)$ denote the set of *predecessors* of a vertex v on shortest paths from s: $P_s(v) = \{u \in V : (u, v) \in E, d(s, v) = d(s, u) + w(u, v)\}$. Brandes shows that the dependencies satisfy the following recursive relation, which is the most crucial step in the algorithm analysis.

The dependency of $s \in V$ on any $v \in V$ obeys

$$\delta_{s*}(v) = \sum_{w : v \in P_s(w)} \frac{\lambda_{sv}}{\lambda_{sw}} (1 + \delta_{s*}(w))$$

First, n SSSP computations are done, one for each $s \in V$. The predecessor sets $P_s(v)$ are maintained during these computations. Next, for every $s \in V$, using the information from the shortest paths tree and predecessor sets along the paths, compute the dependencies $\delta_{s*}(v)$ for all other $v \in V$. To compute the centrality value of a vertex v, we finally compute the sum of all dependency values. The $O(n^2)$ space requirements can be reduced to $O(m + n)$ by maintaining a *running centrality score.*

4. METHODOLOGY

4.1 Parallelization

Parallelization is an integral part of computing whenever Big Data is concerned. Utilizing full efficiency of the hardware and software capabilities for processing is absolutely necessary to aid the algorithms running on huge graphs. There are several parallel frameworks available (for instance, Hadoop, OpenMP) which process the tasks concurrently.

Open Multi-Processing is an api which maximizes the performance of any algorithm by means of multithreading. Multiple threads are run concurrently each performing individual tasks with a master thread managing them, thus ensuring the maximum throughput.

We have used a tool called *Small-world Network Analysis and Partitioning (SNAP)* which internally uses OpenMP for parallelization of tasks. Since computing the betweenness centrality of a node demands calculation of All Pair Shortest Paths through that node, parallellization becomes an absolute necessity.

The new parallel algorithm presented in the paper Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks by Bader et al. for computing betweenness on low-diameter graphs provides the same work complexity as Brandes algorithm. This is the algorithm that is being used in the SNAP packages with OpenMP support. The understanding of the algorithm is presented below.

The algorithm comprises of two main steps in each iteration of the fine-grained parallelization that could be achieved while computing the measure. Starting from the source vertex s, the frontier of the visited vertices is expanded and breadth-first graph traversal is augmented (referred as level-synchronous graph traversal) to count the number of shortest paths passing through each vertex. Further, a multiset of predecessors for each vertex is maintained as well. A vertex v belongs to the predecessor multiset of w if $\langle v, w \rangle \in E$ and

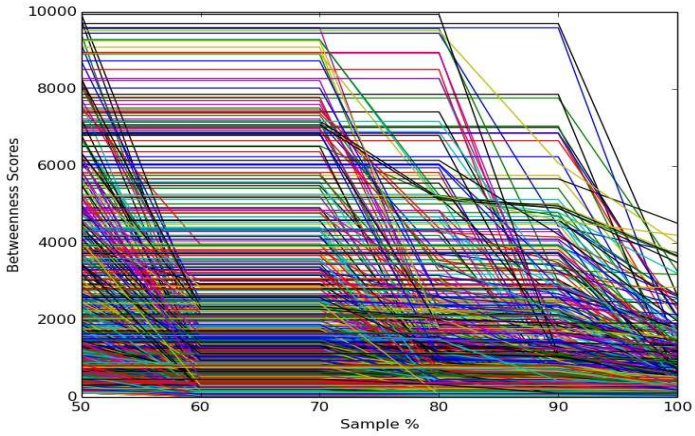

Figure 1: Betweenness Centrality Scores for Directed Graphs with different Sampling

d(s, w) = d(s, v) + 1. The size of a predecessor multiset for a vertex is bounded by its in-degree. The predecessor information is used in the dependency accumulation step. The steps in the algorithm that are amenable to parallel execution. However, the accesses to the shared data structures (such as the predecessor multisets and the stack) and updates to the distance and path counts are protected with appropriate synchronization constructs.

4.2 Adaptive Sampling

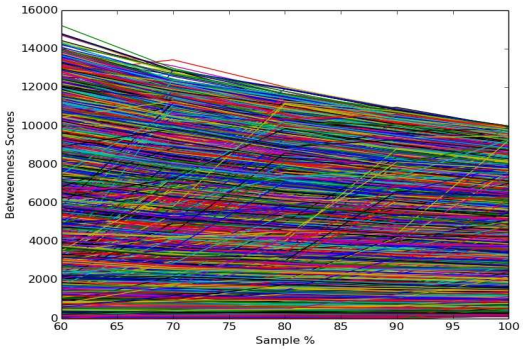

Figure 2: Betweenness Centrality measures for all nodes in a dense directed graph with different sample rate(Twitter)

Fast centrality estimation is an important problem, and a good approximation would be an acceptable alternative to exact scores. The fastest exact algorithms for shortest path enumeration-based metrics require n shortest path computations. It is possible to estimate centrality by extrapolating scores from a fewer number of path computations. Using a random sampling technique, Eppstein and Wang show that the closeness centrality of all vertices in a undirected graph can be approximated with high probability in $O(\frac{\log n}{\epsilon^2}(n \log n + m))$ time, and an additive error of at most $\epsilon\Delta G$(ϵ is a fixed constant, and ΔG is the vertex diameter of the graph). However, betweenness centrality

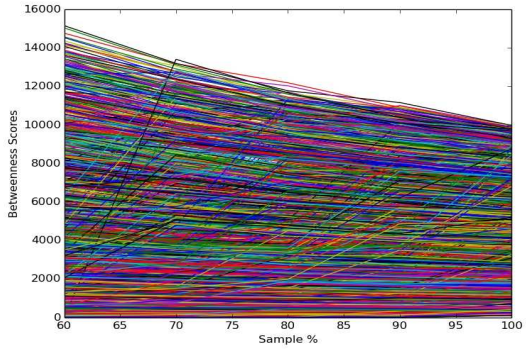

Figure 3: Betweenness Centrality measures for all nodes in a directed graph with different sample rate(Twitter)

scores are harder to estimate, and the quality of approximation is found to be dependent on the vertices from which the shortest path computations are initiated from. Brandes and Pich presented centrality estimation heuristics, where they experimented with different strategies for selecting the source vertices. They observe that a random selection of source vertices is superior to deterministic strategies. In addition to exact parallel algorithms, we also discussed parallel techniques to compute approximate betweenness centrality, using a random source selection strategy.

The below is the theorem presented from [3]. This theorem provides the basis for the approximation algorithm used in the project.

Theorem : For $0 < \epsilon < 0.5$, if the centrality of a vertex v is n^2/t for some constant t ≥ 1, then with probability ≥ 1 âĹŠ 2ϵ its centrality can be estimated to within a factor of $1/\epsilon$ with ϵt samples of source vertices.

5. EXPERIMENTS AND EVALUATION

Our experiments involved working on the data sets mentioned in table 1. The experiments were conducted by extracting the graph data from these text files. The graph

data should be converted into one of the following formats to feed into the SNAP tool.

- snap (.gr)
- dimacs (.dim)
- GML (.gml)
- rand (.rnd)
- rmat (.rmat)

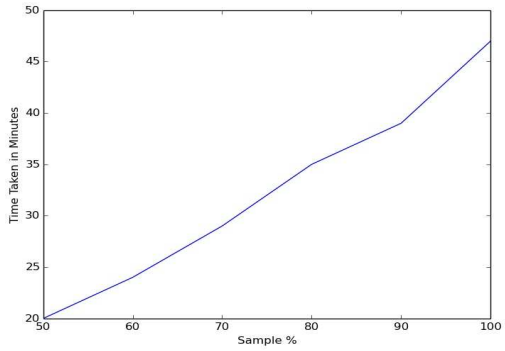

Figure 4: Time plot for measuring BC with different sample rates (UNDIRECTED)

We use the dimacs as the input format for the computation. The *eval_vertex_betweenness_centrality* computes the vertex similarity of all the nodes in the graph. The output of this file is then parsed and the betweenness centrality of each of the vertex is normalized. The time for execution is noted so that the plot can be used to find a good approximation and computation trade-off.

Table 2: Axis for the plots.

X-Axis	Y-Axis
Sampling Percentage 0->100	Normalized Measure $\langle 0, 1 \rangle$

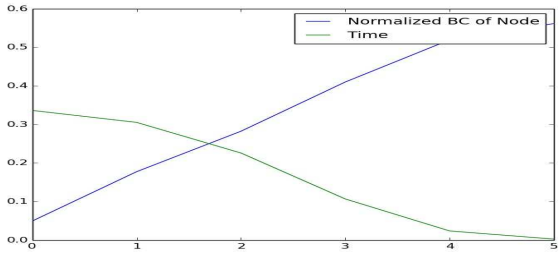

Figure 5: Betweenness Centrality variation for a node with very high indegree and outdegree across sampling rates

The sampling is done for 90, 80, 70, 60 percentages of the same graph. We work on both directed and undirected graphs. The computation trade of between nodes which have

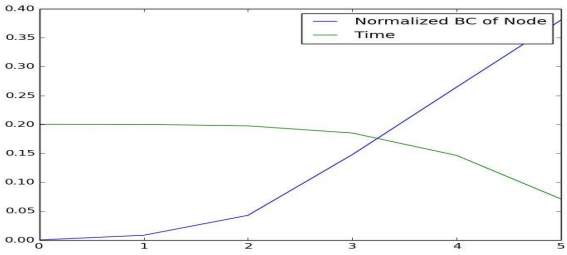

Figure 6: Betweenness Centrality variation for a node with very less indegree and outdegree across sampling rates

a high in degree and out degree and the less dense counterpart is shown in the plots.

The values are normalized so that time and betweenness centrality are represented within the range of $\langle 0, 1 \rangle$

We also provide results on the variation of the betweenness centrality over different percentages of sampling on the directed as well as undirected graph. We see that the betweenness centrality score vary very little on different sampling measures. They seem to increase logarithmically over the sampling percentage. This implies that the percentage of smapling could be reduced so that the computation costs could be lowered while yielding an almost exact betweenness scores.

6. CONCLUSIONS

In the project, we concentrate on the measure of betweenness centrality. We provide a good motivation on why measuring betweenness centrality is important and discuss about the different techniques of measuring it. We provide information on the existing methods to compute the measure. We extend the implementation by introducing two techniques, namely approximation and parallelization. We analyze different ways of using these techniques by providing theorems based on node properties. We provide results to support our claim. This study will help the reader to understand how the approximation could be used to measure such import properties in huge data with reduced computational complexity.

7. REFERENCES

[1] Ulrik Brandes *A Faster Algorithm for Betweenness Centrality*

[2] Matteo Riondato and Evgenios M. Kornaropoulos *Fast Approximation of Betweenness Centrality through Sampling*

[3] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail *Approximating Betweenness Centrality*

[4] David A. Bader Kamesh Madduri *Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks*

[5] Kamesh Madduri, David Ediger Karl Jiang David A. Bader, Daniel Chavarria-Miranda *A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets*