

# LITERATURE REVIEW: Parallel Betweenness Centrality Measurement in Large-Scale Social Networks

Geoffrey Foster  
School of Computer Science  
Carleton University  
Ottawa, Canada K1S 5B6  
*gfoster@scs.carleton.ca*  
<http://g-0ff.net/school/comp5704>

October 23, 2007

## 1 Introduction

The concept of Betweenness Centrality was first introduced by Freeman[4] as a measure of the importance of a node (actor) in a network based upon how much traffic flows through them.

For a given graph  $G = (V, E)$  that has  $n$  vertices, the betweenness centrality ( $C_B(v)$ ) is measured for a vertex,  $v$  by the following:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where  $\sigma_{st}$  is the number of shortest paths from  $s$  to  $t$  and  $\sigma_{st}(v)$  is the number of shortest paths from  $s$  to  $t$  that pass through the vertex  $v$ .

In the worst case scenario the graph is a complete graph with  $\frac{n(n-1)}{2}$  edges. Typical sequential betweenness centrality algorithms use a breadth-first-search which has a runtime of  $\mathcal{O}(n+m)$ , where  $n = |V|$ ,  $m = |E|$ , and thus an overall runtime of  $\mathcal{O}(n^3)$  since a breadth-first-search must be done for each vertex.

The aim of this project is to develop and test a parallel implementation for computing betweenness centrality on a real-world, large-scale social network data set.

## 2 Literature Review

In [2] Brandes provides a sequential algorithm for computing the betweenness centrality of a graph.

Describes the traditional method of computing betweenness centrality as a two step process. The first step involves computing the length and number of shortest paths between all pairs of vertices. The second step involves computing the sum of all pair-dependencies.

Brandes claims a running time of  $\mathcal{O}(nm)$  for the presented algorithm. At first this was misleading since the algorithm performed  $n$  iterations where, in each iteration, a breadth-first-search was performed which has a runtime of  $\mathcal{O}(n+m)$ . Although not stated I can

only assume that the  $n$  value was dropped from the breadth-first-search runtime analysis because it would typically be much smaller than  $m$ .

The algorithm essentially does a breadth first search for each vertex in the graph but contains a few extra steps in order to save some time in tallying the results.

A randomized approximation approach is presented in [3] by Eppstein and Wang. It is a sequential algorithm for computing the closeness centrality of the vertices in a graph. Closeness centrality is similar to betweenness centrality in that it requires the computation of single source shortest path. The presented algorithm performs a set number of iterations and for each generation uniformly at random selects a vertex from the graph and solves the single source shortest path problem for it. After completing the set number of iterations an estimate is obtained where each estimated value should probabilistically fall within an error range.

Bader and Madduri[1] demonstrate that the betweenness centrality can be computed for all actors in a network in parallel on an SMP system. The method they chose involves executing the algorithm presented by Brandes[2] in parallel. Because of the additional memory requirements of computing  $p$  breadth first searches in parallel the algorithm was tuned for the particular systems that the algorithm was being tested on, the Cray MTA-2 and IBM p570. The actual tuning that was done for each system is only briefly mentioned and is described as doing “a coarse-grained partitioning of work”.

In [6], Yang and Lonardi present an MPI implementation of a clustering algorithm that utilizes betweenness centrality for detecting clusters in protein-protein interaction networks. The betweenness clustering algorithm requires multiple iterations, each of which computes the betweenness centrality for the entire graph. The running time for this is  $\mathcal{O}(nm^2)$ , but, for large scale connected networks this could be  $\mathcal{O}(n^5)$ . Once again, the algorithm presented by Brandes[2] is used because of its  $\mathcal{O}(nm)$  runtime. Each of the centrality computations (breadth first search, summing of pair dependencies) for each vertex is done in parallel. In order to accomplish this each of the processors receives their own copy of the graph. During each iteration of the clustering algorithm, after the betweenness centrality is computed, each processor removes the same set of edges in order to stay in synch.

A parallel and distributed approach that is anytime/anywhere, that is, “anytime refers to providing results at any given time and refining the results through time” and “anywhere refers to incorporating new information wherever it happens and propagating it through the network” is presented in [5]. It consists of three separate phases. The initial phase, referred to as Domain Decomposition, consists of taking the original social network and breaking it into subgraphs that are then given to separate processors. An Initial Approximation is then performed on each processor using the available subgraph. A Recombination phase then occurs where each processor communicates with the others and generates more accurate solutions based upon their results.

During the first phase the algorithm attempts to adhere to a set of constraints. These are: the subgraphs should be small enough to meet the limits of the processor, all subgraphs should be approximately the same size and finally, each subgraph should be isolated from the others as much as possible. The most important property that is looked at in creating these subgraphs is the cut size, that is, the number of edges that have to be removed to produce a subgraph. This property should be minimized in order to reduce the work done

in the recombination phase.

The initial approximation phase is then done using traditional social network analysis techniques (i.e. breadth-first-search based method for computing betweenness centrality) independently on each processor.

Finally, in the recombination phase the algorithm uses ego-centric betweenness centrality measurements to estimate the betweenness centrality of each node.

## References

- [1] David A. Bader and Kamesh Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In *Proceedings of the 2006 International Conference on Parallel Processing (ICPP)*. IEEE Comp. Soc. Dig. Library, 2006.
- [2] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.
- [3] David Eppstein and Joseph Wang. Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, 8(1):39–45, 2004.
- [4] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.
- [5] Eunice E. Santos, Long Pan, Dustin Arendt, and Morgan Pittkin. An effective anytime anywhere parallel approach for centrality measurements in social network analysis. *IEEE International Conference on Systems, Man, and Cybernetics*, 6:4693–4698, 2006.
- [6] Q. Yang and S. Lonardi. A parallel algorithm for clustering protein-protein interaction networks. *Computational Systems Bioinformatics Conference, 2005. Workshops and Poster Abstracts. IEEE*, pages 174–177, 2005.