Literature Review: Parallel Algorithms for Centrality in Dynamic Graphs

Nathan Bowness
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
nbown088@uottawa.ca

October 2nd, 2020

Introduction

As the 21st Century progresses, humans are becoming more dependant on technology; whether that is a smart-phone, DNA sequencer, or even a bank machine they all lead to the creation of data. The mass abundance of data has led to tremendous research efforts into the fields of Data Science and Big Data. Through this research it has become apparent that sequential computing is insufficient for performing computational tasks on massive data sets. As a result, Parallel Computing has been adopted to speed-up the computation process. Parallel computing allows for programmers to push past the limitations of sequential computing, which are often restricted by hardware, by splitting sections of a large task into independent steps that can be executed simultaneously[1]. Parallel computing allows these independent steps to be run on multiple processors at the same time. Once the individual steps are completed the output can be interpreted by the main process, it will combine the different outputs defining a single solution to the large task. This paralysation will reduce the overall compute time for the task. Parallel computing cannot be used for all applications, particularly applications that require steps to be completed in a specific sequence, but where it is application users will see large gains in performance.

Graphs have become an instrumental tool for modelling relationships in applications such as biological, social, and transportation networks. The applications mentioned, and many others, encompass massive data sets that require parallel computing for graph analysis within a reasonable period. One core metric for graph analysis is to evaluate the centrality of all nodes. Depending on the information desired centrality can be measured in multiple ways including degree-, closeness-, betweenness-, Eigenvector-centrality and many others each offering a different overview into the data [2]. In this paper, the focus will be solely on betweenness centrality. Betweenness centrality for a node *v* in a graph *G* can be defined as “the fraction of the shortest paths between all pairs of nodes that pass-through *v*”[3],[4]. Practically, betweenness centrality values in a social network measures how influential a person is in connecting the network around them; someone with a high value will have the most influence.

There are many different algorithms for calculating betweenness centrality, differing based on graph size and if updates are expected. This paper's focus will be directed towards re-computing centrality in Dynamic graphs, graphs that are change over time by removing/adding edges. These graphs are better representations for real-world applications that are continuously changing. For example, in a transportation network routes are constantly being added/removed, or a social media network where people may be friending and unfriending multiple people every minute. In these scenarios algorithms are required to recompute existing values quickly, rather than recomputing all values again wasting long sections of time. The goal of this paper is to evaluate a cutting-edge parallel algorithm produced by Shukla et al. [5] for computing betweenness centrality on a dynamic graphs; and evaluate it's performance on additional datasets to verify the massive performance gain claimed.

Literature Review

Betweenness centrality can be computed using many algorithms, the "best" algorithm for a certain case generally depends on two factors; whether the graph is expected to change and the size of it. Using those factors these algorithms can be grouped into three main subsections: static graphs, massive graphs (100s of millions to billions of nodes/edges) and dynamic graphs. This paper will briefly touch on the first 2 sections with a focus on state-of-the-art algorithms for dynamic graphs.

Betweenness centrality for a node *v* in a Graph *G* can be defined as [6]:

$$BC_G[v] = \sum_{\substack{s,t \in V, \\ s \neq t \neq v}} \frac{\sigma_{st}(v)}{\sigma_{st}} = \sum_{\substack{s,t \in V, \\ s \neq t \neq v}} \frac{\delta_{st}(v)}{\delta_{st}} \qquad (1)$$

Where:

- *s, t* – are also nodes in the Graph G
- $\sigma_{st}(v)$ – number of shortest paths from s to t that pass-through *v*
- σ_{st} – the number of shortest paths from s to t

Betweenness Centrality in Static Graphs:

Computing the betweenness centrality for each node in a static graph is required as a preliminary step for most dynamic approaches. The dynamic algorithms leverage information such as: all-pairs shortest paths and number of shortest paths that are stored during the preliminary run to speed up the calculations for an update. The algorithms for static graphs are often used for comparison to see how much a dynamic algorithm improves performance, rather than having to recomputing the values for the entire graph again. The fastest known algorithm for computing betweenness centrality in a static graph was found by Brandes [6] and has a runtime of $\vartheta(|V||E|)$. Recently there have been papers [7], [8] trying to increase the performance of Brandes algorithm, but both papers were shown to only improve in some situations, theoretically the algorithms do not offer any computation advantage. Additionally there has been lots of work in computing betweenness centrality for static graphs in parallel, this work is outlined in many papers including [9], [10], [11], [12]. These parallel algorithms offer a decrease in computational time, with a downside of requiring a large amount of memory, so as graphs grow, they surpass the memory requirements of some machines. That is why approximation algorithms are used for massive graphs, to bypass the required memory needed for exact computations.

Betweenness Centrality Approximation in Massive Graphs:

Calculating the exact betweenness centrality of a graph with hundreds-of-millions to billions of nodes and edges is slow and can be resource intensive. To increase the speed of computation, users can sacrifice accuracy to get quick results using approximation algorithms. These are especially useful for applications where some chance of error is acceptable, but results are wanted quickly. Similar to calculating the exact betweenness, approximation research has been split into approximating static graphs and dynamic graphs. Approximating the betweenness centrality if static graph has been researched in depth for social networks, some key papers include [13], [14]. More recent research has turned toward approximating the betweenness centrality values in dynamic graphs [15], [16], [17]. The approximation gives an exponential speed up over the exact calculations for a dynamic graph. To give an example, Hayashi et. al [16] mentioned their algorithm can "reflect a graph change in less than a millisecond on an average large-scale web graph with 106M vertices and 3.7B edges". To contrast, the exact algorithm by Shukla et al. [5] for a graph with 325 thousand nodes and 1.082 million edges could process a batch update of 25 nodes in approximately 700 seconds.

Betweenness Centrality in Dynamic Graphs:

- Section is a work in progress

Biconnected Component Decomposition:

Batch Update of Edges:

Chains and Redundant Chains:

Redundant Nodes:

References:

[1] S. Rastogi and H. Zaheer, “Significance of Parallel Computation over Serial Computation Using OpenMP, MPI, and CUDA,” *ResearchGate*, Oct. 2018. https://www.researchgate.net/publication/320213267_Significance_of_Parallel_Computation_over_Serial_Computation_Using_OpenMP_MPI_and_CUDA (accessed Oct. 03, 2020).

[2] E. Yan and Y. Ding, “Applying centrality measures to impact analysis: A coauthorship network analysis,” *Journal of the American Society for Information Science and Technology*, vol. 60, no. 10, pp. 2107–2118, 2009, doi: 10.1002/asi.21128.

[3] L. C. Freeman, “A Set of Measures of Centrality Based on Betweenness,” *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977, doi: 10.2307/3033543.

[4] J. M. Anthonisse, “The rush in a directed graph,” Art. no. BN 9/71, Jan. 1971, Accessed: Oct. 09, 2020. [Online]. Available: https://ir.cwi.nl/pub/9791.

[5] K. Shukla, S. C. Regunta, S. H. Tondomker, and K. Kothapalli, “Efficient parallel algorithms for betweenness- and closeness-centrality in dynamic graphs,” in *Proceedings of the 34th ACM International Conference on Supercomputing*, New York, NY, USA, Jun. 2020, pp. 1–12, doi: 10.1145/3392717.3392743.

[6] U. Brandes, “A faster algorithm for betweenness centrality,” *The Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, Jun. 2001, doi: 10.1080/0022250X.2001.9990249.

[7] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, *Heuristics for Speeding Up Betweenness Centrality Computation*. 2012, p. 311.

[8] D. Erdos, V. Ishakian, A. Bestavros, and E. Terzi, “A Divide-and-Conquer Algorithm for Betweenness Centrality,” Jun. 2014, doi: 10.1137/1.9781611974010.49.

[9] D. A. Bader and K. Madduri, “Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks,” in *2006 International Conference on Parallel Processing (ICPP’06)*, Aug. 2006, pp. 539–550, doi: 10.1109/ICPP.2006.57.

[10] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. Chavarria-Miranda, “A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets,” in *2009 IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1–8, doi: 10.1109/IPDPS.2009.5161100.

[11] G. Tan, D. Tu, and N. Sun, “A Parallel Algorithm for Computing Betweenness Centrality,” in *2009 International Conference on Parallel Processing*, Sep. 2009, pp. 340–347, doi: 10.1109/ICPP.2009.53.

[12] N. Edmonds, T. Hoefler, and A. Lumsdaine, “A space-efficient parallel algorithm for computing betweenness centrality in distributed memory,” in *2010 International Conference on High Performance Computing*, Dec. 2010, pp. 1–10, doi: 10.1109/HIPC.2010.5713180.

[13] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail, “Approximating Betweenness Centrality,” in *Algorithms and Models for the Web-Graph*, Berlin, Heidelberg, 2007, pp. 124–137, doi: 10.1007/978-3-540-77004-6_10.

[14] R. Geisberger, P. Sanders, and D. Schultes, “Better Approximation of Betweenness Centrality,” in *2008 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 0 vols., Society for Industrial and Applied Mathematics, 2008, pp. 90–100.

[15] E. Bergamini, H. Meyerhenke, and C. L. Staudt, “Approximating Betweenness Centrality in Large Evolving Networks,” in *2015 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, 0 vols., Society for Industrial and Applied Mathematics, 2014, pp. 133–146.

[16] T. Hayashi, T. Akiba, and Y. Yoshida, “Fully dynamic betweenness centrality maintenance on massive networks,” *Proc. VLDB Endow.*, vol. 9, no. 2, pp. 48–59, Oct. 2015, doi: 10.14778/2850578.2850580.

[17] S. K. Maurya, X. Liu, and T. Murata, “Fast Approximations of Betweenness Centrality with Graph Neural Networks,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, New York, NY, USA, Nov. 2019, pp. 2149–2152, doi: 10.1145/3357384.3358080.