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 changing 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 (BC) 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 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 recompute 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 large 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:

Re-computing betweenness centrality for each node in a dynamic graph is a common scenario when modelling real-world networks. Lots of research has been done to improve processing speed, reducing space required to store graph information and parallelizing the algorithms. Some key papers on the topic include [18], [19], [20], [21], [22]; most notably the paper by Jamour et al. introduced iCENTRAL [22] an algorithm that offered a large speed improvement, without a large space requirement and was parallelizable. iCENTRAL was based on a few key concepts to reduce run time: limiting Breadth-first search (BFS), using Biconnected Components (BCC), as well as identifying Redundant Nodes. Shukla et al. [5] have improved on the iCENTRAL algorithm by introducing the concept of a batch update to address its main flaw of being limited to sequential updates.

Calculating Betweenness Centrality for Dynamic Graphs

The rest of this paper will consider a Graph G defined with a set of edges E and nodes V. Using that notion, in general the betweenness centrality (BC) for a node *v* in a graph 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

The ideal algorithm proposed by Brandes [6] which is leveraged by dynamic graphs algorithms uses pair and source dependencies denoted by $\delta_{st}(v)$ and $\delta_{s\bullet}(v)$ respectively, to calculate the betweenness centrality.

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad (2) \qquad \delta_{s\bullet}(v) = \sum_{t\in V, t\neq v} \delta_{st}(v) \qquad (3)$$

The algorithm implements a BFS from node s to compute both σ_{sw} and $P_s(w)$ for all nodes w∈V with s≠w. The second step uses a reverse-BFS to find the source dependencies, $\delta_{s\bullet}(v)$, using Equation (5).

$$\delta_{s\bullet}(v) = \sum_{v\in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \left(1 + \delta_{s\bullet}(w)\right) \qquad (4)$$

Where:

- σ_{ij} - is the number of shortest paths from i to j
- $P_s(w)$ - is the list of parents of w in the BFS of s

$$BC_G[v] = \sum_{s \epsilon V, s \neq v} \delta_{s\bullet}(v) \qquad (5)$$

Brandes algorithms[6] allows for the computation of betweenness centrality by only performing a BFS and reverse-BFS, this combined with other techniques offers a large speed improvement for dynamic graph algorithms. Brandes also keeps properties about the BFS traversal from each node, so the properties do not need to be re-computed from scratch each time. The information kept with regards to a source node, s, includes: S_s the order of nodes visited from the source node, D_s the distance from the source node, σ_s number of paths from s and P_s the parents nodes of s.

Biconnected Components:

Large graphs can usually be sectioned into their biconnected components (BCC). A BCC of a graph "is a maximal biconnected subgraph"[22]. The BBCs within a graph are only connected by articulation points – a node that would disconnect the graph if removed. The BBC sections allow for the graph to be logically split ensuring the following claims: a node can be a part of multiple BCCs, but an edge can only be part of one BCC. Figure 1 below shows a graph split into different BCCs and the articulation points connecting them.

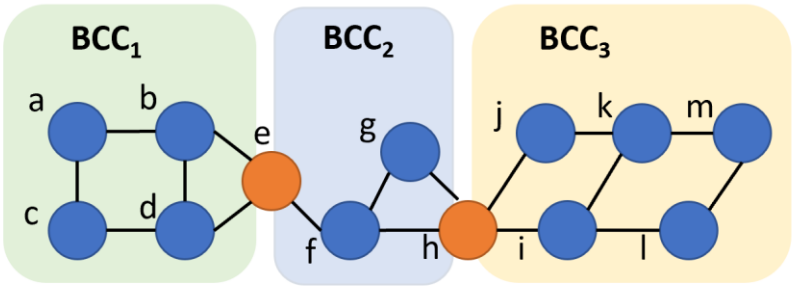

Figure 1: Graph displaying different biconnected components (BCC), and articulation points e, h in orange.

Biconnected components are a critical piece of dynamic BC algorithms as they limit the scope of the BFS required by Brandes algorithm to compute BC. Jamour et al. [22] shows that the betweenness centrality can be computed completely within the BCC of the affected edge, reducing the number of nodes the BFS must be run from drastically and offering a large speed improvement.

Redundant Nodes:

Identifying redundant nodes in a graph and excluding them from computation can save significant time as it eliminates nodes BFS is run from. Shukla et al. [5] mentions two cases of redundancy that are common in networks for nodes with degree 3 and degree 4. Redundancy in a node *v* with degree 3, denoted by R_3, occurs when the 3 neighbours of v create a cycle of length 3 (they are all neighbours of each other). The same applies to redundant nodes of degree 4, denoted by R_4, if the neighbours create a cycle of length 4 the node is redundant. An example of R_3 and R_4 is found below in Figure 2.

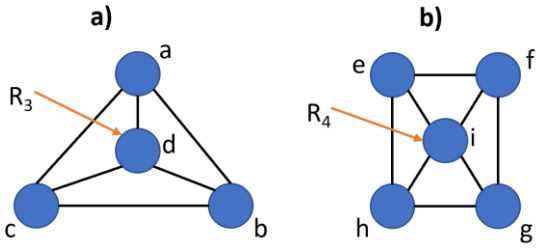

Figure 2: (a) A graph showing a R_3 redundant node d, (b) A graph showing a R_4 redundant node i.

Another consideration when finding nodes affected by an edge being added/deleted is identifying if the shortest path has been modified. If an edge is added and it has no affect on the node's shortest paths, that node is not affected. As show in [22] and [23] a node *v* is only affected if an edge, *st,* is added to the graph and $d(v,s) \neq d(v,t)$.

Batch Update for Betweenness Centrality:

The previous approaches by [19], [22], [20], [24] to updating betweenness centrality were restricted to processing only one update at a time. The disadvantage to that approach is if multiple updates occur during a short period; it causes consecutive updates to all nodes. As well, if multiple updates affect a single node *v*, recomputing the betweenness centrality for each incremental update is wasteful. Equation (6) below shows the formula used by *i*CENTRAL[22] to update the betweenness centrality by removing and adding source dependencies for each added/removed edge one by one. Further information about the *i*CENTRAL algorithm can be found in the appendix.

$$BC_{G'}[v] = BC_{G'}[v] - \sum_{s \epsilon Q, s \neq v} \delta_{s\bullet}(v) + \sum_{s \epsilon Q, s \neq v} \delta'_{s\bullet}(v) \qquad (6)$$

Where:

- $BC_{G'}$ – updated betweenness centrality value for a node *v* in Graph G
- Q – set of all nodes for where $\delta_{s\bullet}(v)$has changed after the insertion of edge e

Shukla et al. [5] improved upon *i*CENTRAL to allow for batch updates, circumventing the disadvantage of current approaches and improving performance dramatically. Shukla et al. propose removing all old source old dependencies, then adding all new source dependencies for the entire batch of updated edges. Figure 1 below shows the improvements of calculating the betweenness centrality in a batch rather than sequentially for each edge.

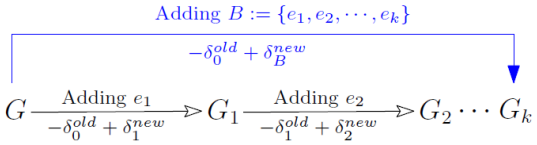

Figure 1: Diagram taken from Shukla et al. [5] showing their batch update approach in blue, and comparing it to *i*CENTRAL's [22] approach of sequential updates in black.

The main goal of this paper is to implement the algorithm designed by Shukla et al. [5] and test the performance of their batch update model on additional datasets. Additionally, their paper did not outline performance on a small numbers of edge updates, they only show results on batch sizes of 25 or more. Depending on the graph's application it is reasonable to assume only a small number edge updates will come in over a period. Another goal will be to see if their algorithm is always faster than *i*CENTRAL or if there is additional overhead on small batch sizes.

Appendix:

iCENTRAL:

Jamour et al. [22] proposed an algorithm *i*CENTRAL that computes updated betweenness centrality values after an edge insertion/deletion. Although it requires each update to be processed sequentially, their algorithm was a large improvement over [24]. *i*Central allows for updating betweenness centrality values to be computed incrementally by only considering the affected biconnected components, rather than the entire graph. This decreases the amount of BFS and reverse-BFS required to compute betweenness centrality as its limited to only the affect BCC. The algorithm uses equation **Error! Reference source not found.** to sequentially remove and then add source dependencies as each edge is added one by one, calculating betweenness centrality by using Brandes algorithms. Equation (7) below shows a more in-depth formula for the calculation used in the iCENTRAL algorithm. For further information on the proof and theorem please refer to [22].

$$BC_{G'}[v] = BC_G[v] - A[v] + A'[v] - B[v] + B'[v] - C[v] + C'[v] \quad (7)$$

Where:

A[v], A'[v] – contribution of nodes s and t that are in the affected BCC, to the source dependencies of v
B[v], B'[v] – contribution of node s that is in the affected BCC and t that is not in the affected BCC, to the source dependencies of v
C[v], C'[v] – contribution of nodes s and t that are not in the affected BCC, to the source dependencies of v

Full derivations and definitions of the equations for A[v], A'[v], B[v], B'[v], C[v], C'[v] can be found in [22].

$$A[v] = \sum_{\substack{s,t \in B'_e \\ s \neq t \neq v}} \delta_{st}(v) \quad , \; A'[v] = \sum_{\substack{s,t \in B'_e \\ s \neq t \neq v}} \delta'_{st}(v) \qquad (8), (9)$$

$$B[v] = \sum_{\substack{s \in G_i, t \in B'_e \\ s \neq t \neq v \\ i=1 \ldots k}} \delta_{st}(v) \quad , \; B'[v] = \sum_{\substack{s \in G_i, t \in B'_e \\ s \neq t \neq v \\ i=1 \ldots k}} \delta'_{st}(v) \qquad (10), (11)$$

$$C[v] = \sum_{\substack{s \in G_i, ts \in G_j \\ s \neq t \neq v, i=1 \ldots k \\ j=1 \ldots k, i \neq j}} \delta_{st}(v) \quad , \; C'[v] = \sum_{\substack{s \in G_i, ts \in G_j \\ s \neq t \neq v, i=1 \ldots k \\ j=1 \ldots k, i \neq j}} \delta'_{st}(v) \qquad (12), (13)$$

Where:

G' – graph constructed by added an edge *e* to graph *G*
B'_e – biconnected component of G' that edge *e* belongs to
$a_1, \ldots, a_k$ – articulation points of BCC B'_e
$G_1, \ldots, G_k$ – subgraphs of G' connected to B'_e through $a_1, \ldots, a_k$ respectively

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.

[18] M.-J. Lee, J. Lee, J. Park, R. Choi, and C.-W. Chung, “QUBE: A quick algorithm for updating betweenness centrality,” *WWW’12 - Proceedings of the 21st Annual Conference on World Wide Web*, Apr. 2012, doi: 10.1145/2187836.2187884.

[19] O. Green, R. McColl, and D. A. Bader, “A Fast Algorithm for Streaming Betweenness Centrality,” in *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, Sep. 2012, pp. 11–20, doi: 10.1109/SocialCom-PASSAT.2012.37.

[20] M. Pontecorvi and V. Ramachandran, “A Faster Algorithm for Fully Dynamic Betweenness Centrality,” Jun. 2015, Accessed: Oct. 17, 2020. [Online]. Available: https://arxiv.org/abs/1506.05783v3.

[21] N. Kourtellis, G. De Francisci Morales, and F. Bonchi, “Scalable online betweenness centrality in evolving graphs,” in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, May 2016, pp. 1580–1581, doi: 10.1109/ICDE.2016.7498421.

[22] F. Jamour, S. Skiadopoulos, and P. Kalnis, “Parallel Algorithm for Incremental Betweenness Centrality on Large Graphs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 659–672, Mar. 2018, doi: 10.1109/TPDS.2017.2763951.

[23] A. E. Sariyüce, E. Saule, K. Kaya, and Ü. V. Çatalyürek, “STREAMER: A distributed framework for incremental closeness centrality computation,” in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2013, pp. 1–8, doi: 10.1109/CLUSTER.2013.6702680.

[24] M.-J. Lee, S. Choi, and C.-W. Chung, “Efficient algorithms for updating betweenness centrality in fully dynamic graphs,” *Information Sciences*, vol. 326, pp. 278–296, Jan. 2016, doi: 10.1016/j.ins.2015.07.053.