Internet Mathematics, 11:403–420, 2015

ISSN: 1542-7951 print/1944-9488 online
DOI: 10.1080/15427951.2014.982311

Taylor & Francis
Taylor & Francis Group

A FASTER ALGORITHM TO UPDATE BETWEENNESS CENTRALITY AFTER NODE ALTERATION

Rishi Ranjan Singh,[1] Keshav Goel,[2] S. R. S. Iyengar,[1] and Sukrit Gupta[3]

[1]*Department of Computer Science and Engineering, Indian Institute of Technology, Ropar, Punjab, India*

[2]*Department of Computer Engineering, National Institute of Technology, Kurukshetra, India*

[3]*Department of Computer Science and Engineering, PEC University of Technology, Chandigarh, India*

Abstract Betweenness centrality is widely used as a centrality measure, with applications across several disciplines. It is a measure that quantifies the importance of a vertex based on the vertex's occurrence on shortest paths in a graph. This is a global measure, and in order to find the betweenness centrality of a node, one is supposed to have complete information about the graph. Most of the algorithms that are used to find betweenness centrality assume the constancy of the graph and are not efficient for *dynamic networks*. We propose a technique to update betweenness centrality of a graph when nodes are added or deleted. Observed experimentally, for real graphs, our algorithm speeds up the calculation of betweenness centrality from 7 to 412 times in comparison to the currently best-known techniques.

1. INTRODUCTION

Network centrality measures are used to quantify the intuitive notion of nodes' importance in a network. There are several application-centric definitions of network centrality measures, the popular ones being *degree centrality, closeness centrality, eigenvector centrality*, and *betweenness centrality*. For background and description of centrality measures, one can refer to [30, 18, 5].

There are a number of centrality indices based on the shortest path lengths: closeness centrality [33], graph centrality [14]; and the number of shortest paths: stress centrality [36], betweenness centrality [10, 1] in a graph. Each centrality measure signifies a particular characteristic that a node exhibits. *Closeness centrality* of a vertex indicates the distance of a vertex from other vertices. *Graph centrality* denotes the difference between closeness centrality of the vertex under consideration and the vertex with the highest closeness centrality. *Stress centrality* simply denotes the total number of shortest paths passing through a vertex.

The idea of betweenness centrality was proposed in [1, 10]. Betweenness centrality of a node v is defined as $BC(v) = \sum_{s \neq t \neq v \epsilon V} \frac{\sigma_{st}(v)}{\sigma_{st}}$, where σ_{st} is the total number of shortest

Addresss corrrespondence to Sudarshan Iyengar, Department of Computer Science and Engineering, Indian Institute of Technology, Ropar, Nangal Road, Rupnagar Punjab, India – 140001. E-mail: sudarshan@iitrpr.ac.in

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/uinm.

paths from vertex s to vertex t, and $\sigma_{st}(v)$ is the total number of shortest paths from vertex s to vertex t passing through vertex v.

Betweenness centrality insinuates a more *global* characteristic, unlike the degree centrality, which takes into consideration the number of links originating from a node (also called the *degree* of a node), which is clearly a local characteristic. Betweenness centrality has found many important applications across different disciplines. It has been used in the identification of sensitive nodes in biological networks [28]. Betweenness scores an play important role in public transit system networks [34, 8], gas pipeline networks [7], and waste-water disposal system networks [24]. In protein-protein interaction (PPI) networks, essential proteins can be identified by their high betweenness centrality [19]. This characteristic of proteins can be used to select suitable drug targets [41] for various ailments including cancer [16], tuberculosis [35], zoonotic cutaneous leishmaniasis [9], etc. Recently, Nagata et al. [27] proposed a new load-balancing approach that reduces the blocking probability of request in wavelength-division multiplexing (WDM) networks. In their algorithm, they used betweenness centrality of nodes for adjusting the link costs.

Betweenness centrality is also used to identify nodes that are crucial for information flow in a brain network [17], where different regions of the brain represent nodes in the network and white matter represents the links. With recent advances in Electrical & Electronic Systems (EES systems such as Electronic Control Units used in vehicles) the mechanism of fault isolation and fault detection is of great importance. In [25], it was observed that the betweenness centrality score of a node is a good basis for ranking the fault tolerance monitoring points and that it outperforms the degree centrality measure.

Similarly, in supply chain networks [40] it is reported that for a lower level of tolerance (load carrying capacity), more harm is created in the case when a node with high load is deleted from a network opposed to when a node with high degree is deleted.

[3] An algorithm to calculate betweenness centrality that reduced the time complexity from $O(|V|^3)$ to $O(|V||E|)$ for unweighted graphs was suggested in [3]. Since was suggested in .the real-world networks tend to be large and transient, such algorithms are observedly impractical if one requires computing the betweenness centrality of nodes in a dynamic network. In [40], experiments were performed to report that the betweenness ranking order of vertices before and after being updated in a graph can be significantly different. Most of the real-world networks are dynamic in nature, which calls for designing an algorithm that can update the betweenness centrality of nodes faster than the algorithms designed for static networks. There are several algorithms proposed in [23, 12, 20] to find betweenness centrality for updating edges in a graph. Most of the currently available literature considers only the case of updating edges, i.e., these algorithms assume that alteration of a node from a graph is equivalent to modifying one or more edges incident on that node. We propose an algorithm that is $deg(v)$-times faster than the afore mentioned algorithms in the case of a deletion/addition of a node v.

1.1. Motivation

It is sometimes necessary to calculate betweenness centrality for a network at every stage of transition. With a large network and the current algorithms in use, recalculation becomes difficult. Some examples of such networks follow.

Complex communication networks are continuously growing and evolving. Each node in a communication network has a maximum capacity for carrying load,[1] after which the node shuts down and its load is distributed among the remaining nodes. Because of increased load, other nodes might shut down and the network might become disconnected. This phenomenon is commonly known as *cascading failure*. Betweenness centrality of a node corresponds to its load. In scenarios where a node failure is taking place, we need to rapidly calculate load of a node and compare it to its load-carrying capacity to determine whether this node will be able to sustain the extra load that was added to it due to the failure of the previous node. This helps us determine whether a cascading failure will take place. This has applications not only in communication networks but in transport networks and in EES networks, too.

It has been found through experiments conducted [28] that breakdown of nodes with higher betweenness centrality causes greater harm. In such networks, we can compute a sequence of nodes as follows: We start with the given network. At each step, we delete the node with the highest betweenness centrality, add that node to the sequence and then repeat this process until the network becomes disconnected. This sequence can be used to decide the order in which security should be provided to the nodes in the network and to ensure that if a node in the present network fails, the node with the highest betweenness centrality in the resulting network has enough security and resources. This requires repetitive calculation of betweenness centrality which when done with the conventional algorithm [3], will be highly inefficient. Similarly, points that have excess load in power grid systems and computer networks can be provided with more resources; stations with excess traffic in public transit systems can be provided with more measures to redistribute traffic, and sewer lines with higher betweenness centrality can be provided with more frequent maintenance to prevent blockades. This exercise can also be done after the failure of some random node in a graph, and appropriate actions on the nodes in the network may be taken thereafter.

Two sequential failure strategies, random and usage based, are being used [25] for fault mitigation analysis. In usage-based failure strategy, it is assumed that usage is proportional to its betweenness centrality. Thus, the node with the highest betweenness centrality score is most likely to fail in case of usage-based failure strategy. After every failure, we need to calculate which node has the highest betweenness centrality and check if we have reached a preset completion criterion (maybe a threshold value, after which the network is so fragmented that it is not usable, or a percentage of nodes) and if it does, we can check the number of failures that were required to reach there from the first failure. The node with the minimum number of failures required should be monitored more rigorously. This recursive calculation of betweenness centrality is useful in fault mitigation analysis.

In social networking websites such as Twitter and Facebook, betweenness centrality of a node denotes the number of heterogeneous groups of nodes that the node under consideration links [37]. Since these nodes are involved in passing information between heterogeneous groups of nodes, they are more important than a node with just a higher degree.[2] Also, we may want to determine the next important actor in case the current social

[1]The amount of information flowing through a node in a communication network is called its load.

[2]In the study [15] conducted on the uprising in Egypt, where the social networking site Twitter played an important role in the formation of public opinion against Mr. Hosni Mubarak (the then President of Egypt), it was found that these nodes played an important role in shaping public opinion.

network is altered. Such networks are highly dynamic due to the continuous addition and removal of actors.

In section 2, we present some basic definitions and concepts used in this article. Section 3 contains the algorithm with explanation. Implementation details and results are presented in Section 4. For synthetic graphs, we tested our algorithm by generating Erdös Rényi graphs with different probabilities. We took both cases of node addition and node deletion into consideration and achieved speedups ranging from 1.14 to 255. It is important to mention that the speedups achieved for synthetic graphs here are graph dependent and can be further varied by changing various elements while constructing synthetic graphs. We also considered various real-world graphs in our experiments and observed average speedups ranging from 7 to 412. We discuss related works in Section 5.

2. DEFINITIONS AND PRELIMINARIES

In this section, we define some terms that have been used throughout the article. We also explain the basic concepts which provide basis for developing the algorithm in Section 3.

We use following terms interchangeably throughout the article; node or vertex and graph or network. A (simple) *path* in a graph is a sequence of edges connecting a sequence of vertices without any repetition of vertices. Thus, a path between two vertices v_i and v_j (called terminal vertices) can be denoted as a sequence of vertices, $\{v_i, ..., v_j\}$ such that $v_i \neq v_j$ and no vertices in the sequence are repeated. The *length* of a path is the sum of the weights of edges in the path (edge weight is taken as one for unweighted graphs). A *shortest path* between two vertices is the smallest length path between them. An *end vertex* is a vertex with degree one. A graph is said to be connected if there exists a path between each pair of vertices. An articulation vertex is a vertex whose deletion will leave the graph disconnected. A *biconnected graph* is a connected graph having no articulation vertex. A *cycle* in a graph is a path having the same terminal vertices. A set of cycles is called linearly independent when each cycle contains atleast one edge that doesn't belong to any other cycle. A *cycle basis* of a graph is defined as a maximal set of linearly independent cycles. Weight of a cycle basis is the sum of the lengths of all cycles in the cycle basis. A cycle basis of minimum total weight is called the *minimum cycle basis*(MCB).

Repetitive merging (taking union) of all the elements of the MCB that have at least one vertex in common, gives us a set called the *Minimum Union Cycle* set (MUCset). Each element of an MUCset is termed as the *Minimum Union Cycle* (MUC). Due to the way an MUCset is formed, two MUCs can not have any vertex in common. A *connection vertex c* in an MUC (say, MUC_i) is an articulation vertex such that it is adjacent to a vertex that does not belong to MUC_i. On removal of the connection vertex c, the graph will become disconnected and the components that are disconnected from MUC_i are together termed as *disconnected subgraph* G_c. For a more detailed description of MCB and MUC, readers are referred to [21]. Details about MUCs and their importance in updatingz of betweenness centrality can be understood from [23].

2.1. Methodology and Observations

We first understand the case of vertex deletion in undirected unweighted connected graphs. Then we derive similar observations for a vertex addition case. We have conducted experiments and have shown results for both vertex addition and deletion. On the basis of

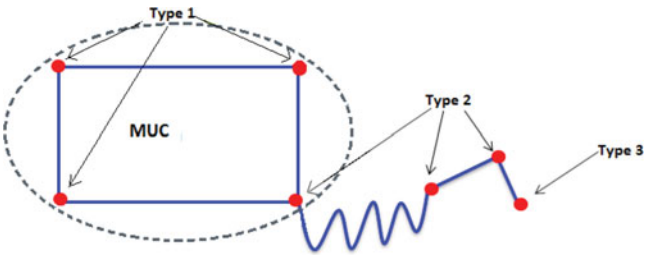

Figure 1 Type 1: Vertex belongs to an MUC but is not an articulation vertex. Type 2: Vertex is an articulation vertex. Type 3: Vertex does not belong to an MUC and is an end vertex.

the method used for updating of betweenness centrality after deletion of a vertex, we can categorize the vertices of the graph into three groups as in Figure 1.

In this article, we explain the updating process after deletion of vertices of Type 1. Deletion of a vertex of Type 2 will leave the graph disconnected and the concept of betweenness centrality will then be limited to only the component of the disconnected graph. We do not explain this case in detail. On deletion of these types of nodes, a variation of the approach given in this article for handling Type 1 node deletion can be used for each disconnected graph to update the betweenness centrality. After deletion of vertices of Type 3, we can use a procedure similar to Algorithm 2 to update the centrality scores. Now, we define a few more terminologies and give lemmas. We establish a theorem that provides a basis to develop our algorithm to update betweenness centrality after vertex deletion.

Pair dependency of a pair of vertices (s, t) on a vertex v is defined as: $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$, where σ_{st} is the number of shortest paths from vertex s to vertex t, and $\sigma_{st}(v)$ is the number of shortest paths from vertex s to vertex t passing through vertex v. Betweenness centrality of a vertex v can be defined in terms of pair dependency as: $\mathrm{BC}(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v)$. Let BFT_r denote the *breadth-first traversal* (BFT) of the graph rooted on vertex r. *Dependency* of a vertex s on a vertex v is defined as: $\delta_{s\bullet}(v) = \sum_{t \in V \setminus \{s,v\}} \delta_{st}(v)$ [3]. Let us define a set $P^s(w) = \{v : v \in V,\ w$ isasuccessor of v *in* $\mathrm{BFT}_s\}$. In [3], the author proved that

$$\delta_{s\bullet}(v) = \sum_{w:v \in P^s(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_{s\bullet}(w)). \tag{2.1}$$

Let $SP(v_i, v_j)$ be the set of all shortest paths from vertex v_i to v_j. Let MUC_U be the MUC where alteration has been made. Let G_i be the subgraph made of the components that will be disconnected from MUC_U after removal of connection vertex $c_i \in \mathrm{MUC}_U$. Let $V(G_i)$ denote the set of vertices in subgraph G_i. Then, we can establish the following lemmas and theorem.

Lemma 3.1. *If v lies on all shortest paths between s and t, where $s \neq t \neq v \in V$, then*

$$\sigma_{st} = \sigma_{sv}.\sigma_{vt}.$$

Lemma 3.2. *For $u \in V(G_k)$ and $v \in MUC_U$, every element of $SP(u, v)$ must contain c_k.*

Proof. Since c_k (connection vertex) is the only vertex that links G_k with MUC_U, every path between vertices in G_k and MUC_U must pass through c_k. □

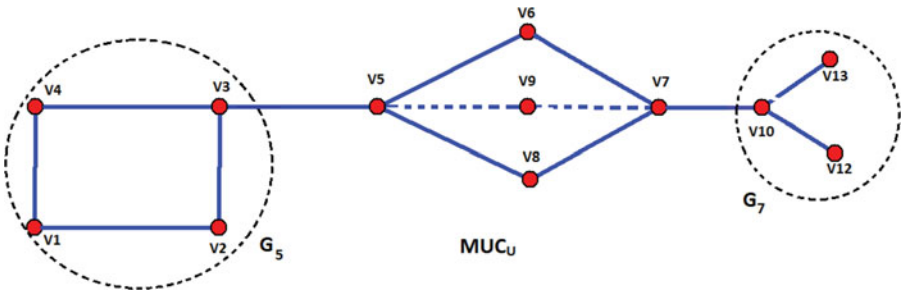

Figure 2 After deletion of vertex $V9$, the shortest paths starting and ending in G_5 or G_7 will not be affected. Other shortest paths may be altered.

Lemma 3.3. *Betweenness centrality of a vertex v can be changed only because of the shortest paths that had the altered vertex as one of their terminal vertices where $v \in V \setminus MUC_U$.*

Proof. Assume $s, t \in V$ and $s \neq t \neq v$. Betweenness centrality of vertex v, BC(v), will be influenced by following types of shortest paths:

1. Shortest paths that do not pass through the MUC_U. They start and end outside MUC_U, without passing through it. Naturally, when an alteration is made in a graph, these paths remain unchanged. An example is shown in Figure 2.
2. Shortest paths that have one terminal vertex in one disconnected subgraph G_i and the other in a disconnected subgraph G_j: These shortest paths may change. For one such shortest path, suppose the terminal vertices s and t are in different disconnected subgraphs, which pass through connection vertices c_1 and c_2, respectively (v lies in the disconnected subgraph where s lies). According to Lemma 3.1 and Lemma 3.2, $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}} = \frac{\sigma_{sc_1}(v).\sigma_{c_1c_2}.\sigma_{c_2t}}{\sigma_{sc_1}.\sigma_{c_1c_2}.\sigma_{c_2t}} = \frac{\sigma_{sc_1}(v)}{\sigma_{sc_1}}$. We can observe that the shortest paths from s to c_1 remain the same after node deletion, and so this factor doesn't change.
3. Shortest paths that have one terminal vertex in one disconnected subgraph G_i and the other in MUC_U: Out of these shortest paths, the paths where deleted vertex is not a terminal vertex, we can get a relation similar to that obtained above. When the deleted vertex is a terminal vertex (i.e., either s or t is deletion vertex), a factor of $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$ should be deleted from the betweenness centrality score of vertex v. This is because existing shortest paths from s to t are nonexistent now.

Thus, only one type of shortest paths (with altered vertex as one of the terminal vertices) can change the betweenness centrality of the vertex v. □

Theorem 3.4. *Let v_d be the vertex to be deleted. Let $BC(v)$ be the betweenness centrality of the vertex v and dependency of the vertex v_d on the vertex $v \in V \setminus MUC_U$ is $\delta_{v_d\bullet}(v)$. Then, the updated betweenness centrality of the vertex v after deletion of the vertex v_d can be calculated as:*

$$BC'(v) = BC(v) - 2\delta_{v_d\bullet}(v)$$

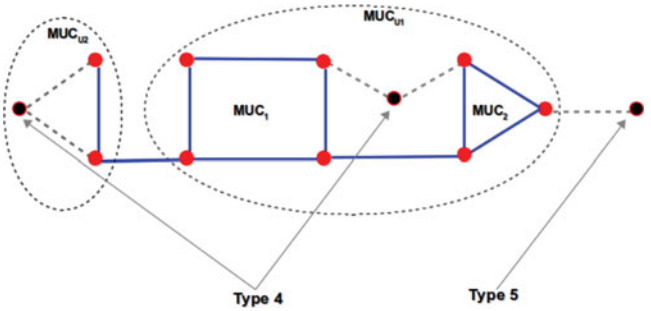

Figure 3 Addition of Type 4 vertex forms a new MUC ($MUC_U 2$) or forms a large MUC ($MUC_U 1$) by joining several MUCs. Vertex of Type 5 is added as an end vertex.

Proof. By the definition of dependency, $\delta_{v_d\bullet}(v)$ gives the effect of all shortest paths starting at vertex v_d in the betweenness centrality of node v. According to Lemma 3.3, shortest paths with v_d as terminal vertex (start vertex or end vertex on the path) are affecting the change in centrality only of vertices outside the MUC_U. After deletion of v_d, all such shortest paths will be deleted. Because the graph is undirected, $\sigma_{v_d t} = \sigma_{t v_d}$, so, we will subtract the dependency $\delta_{v_d\bullet}(v)$ twice. □

Now, we will describe the vertex addition case in brief. On the basis of the method used to update the betweenness centrality after addition of a vertex, we can categorize the vertices of the graph into two groups as in Figure 3. The process of updating the betweenness centrality after addition of Type 4 vertices is very much similar to the updating process used after deletion of Type 1 vertices. A slight modification in Theorem 3.4 gives a theorem for vertex addition, as stated following.

Theorem 3.5. *Let v_a be the vertex that has been added. Let $BC(v)$ be the betweenness centrality of the vertex v before the addition of vertex v_a. After addition, let the dependency of the vertex v_a on the vertex $v \in V \setminus MUC_U$ be $\delta_{v_a\bullet}(v)$. Then, the updated betweenness centrality of the vertex v after addition of the vertex v_a can be calculated as:*

$$BC'(v) = BC(v) + 2\delta_{v_a\bullet}(v)$$

The proof of Theorem 3.5 is similar to the proof of Theorem 3.4. In the next section, we explain algorithms to update the betweenness centrality for the case in which a vertex (Type 1) is being deleted. Algorithms for updating of betweenness centrality after addition of vertices are similar to the algorithms in Section 4, with a slight modification based on Theorem 3.5. The Updating process of the betweenness centrality scores after addition of Type 5 vertices can use the Algorithm 2 with just a slight modification based on Theorem 3.5.

3. OUR ALGORITHM

After deletion of a vertex that belonged to an MUC and was not an articulation vertex, we will update the betweenness centrality in different ways for the two types of vertices: vertices outside MUC_U and vertices in MUC_U. We use Theorem 3.4 to update betweenness centrality for vertices outside MUC_U, and the algorithm is explained in detail in

Section 3.2. When vertices in MUC_U are considered, we observe that several shortest paths that were passing through an altered vertex changed after deletion. So we recompute betweenness centrality using the idea given in [23], which is explained in brief in Section 3.3. We assume that the betweenness centrality score of all vertices is available before proceeding with the preprocessing step of our algorithm.

3.1. Preprocessing Step

Every time a change is made in the graph, updating the MUCset becomes necessary. We can do it in two ways, either by updating the MUCset (approach used in [23]) or by recalculation of the MUCset. The approach for updatind the MUCset takes longer than recalculation. Instead of updating the MUCset, we recalculate it using the output of Tarjan's biconnected components algorithm [38] (commonly known as Tarjan's Algorithm). Tarjan's biconnected algorithm uses the depth-first traversal of a graph for calculating biconnected components. The input to and output from Tarjan's algorithm can be understood with the help of Figures 4(a) and 4(b). The process of recalculation of the MUCset is given in Algorithm 1. The time complexity for calculating biconnected components is $O(|V| + |E|)$ due to the bound on depth-first search. Thus, the time complexity for recalculation of the MUCset (Algorithm 1) is $O(|V| + |E|)$. Following, we explain the procedure for calculating the MUCset using biconnected components.

Algorithm 1 Preprocessing step: Calculating MUCs in the Graph

```
1: Use Tarjan's Algorithm to calculate a set of biconnected components, C.
2: for each C_i ∈ C do
3:    if |C_i| = 2 then
4:       Remove C_i from C.
5:    end if
6: end for
7: while ∃ C_i, C_j ∈ C where C_i and C_j have at least one common vertex do
8:    Remove C_i and C_j from C.
9:    Insert C_i ∪ C_j in C.
10: end while
11: MUCset ← C
12: for each MUC_j ∈ MUCset do
13:    Find all the connection vertices and corresponding disconnected subgraphs.
14: end for
```

Every graph can be decomposed into a set of biconnected components, C, in which the elements of C are denoted by C_i (using Tarjan's algorithm). Let $|C_i|$ denote the number of vertices in the biconnected component C_i. Each biconnected component contains at least one edge (two vertices) and may share vertices (articulation vertex) with other biconnected components. We remove components that contain only one edge because a single edge cannot form an MUC. Because the elements of an MUCset are disjoint, we take repetitive union of the components that have at least one vertex in common. We form the MUCset in this fashion. The MUCset generated by applying Algorithm 1 on the biconnected components (Figure 4(b)), which are calculated after applying Tarjan's algorithm on the graph in Figure 4(a), are shown in Figure 4(c). After calculating the MUCset, for each MUC, we

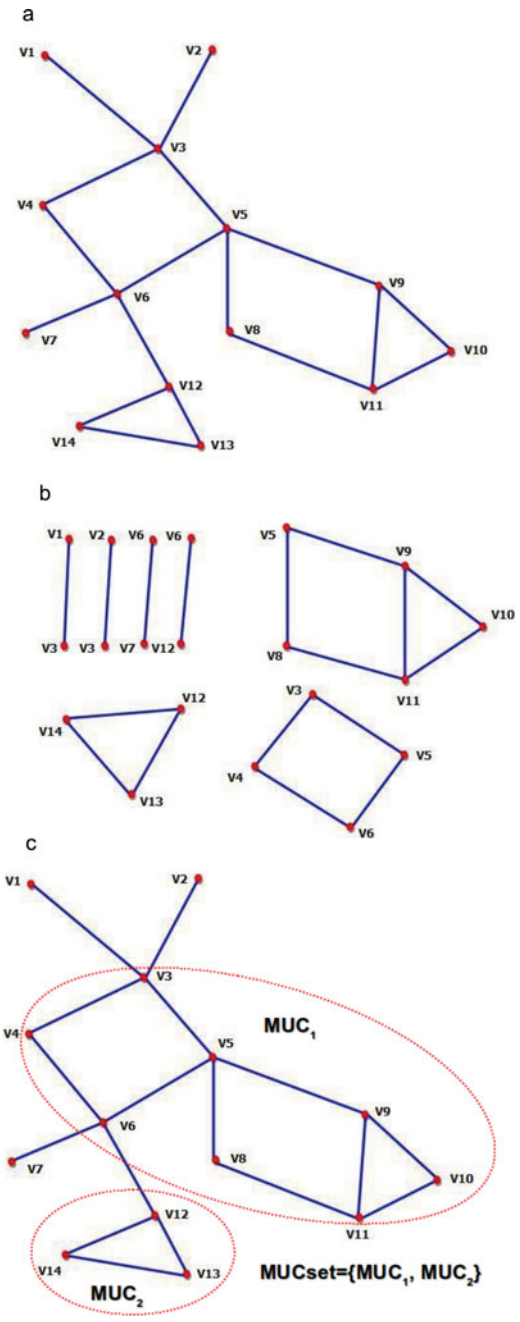

Figure 4 (a)Graph before applying Tarjan's algorithm. (b) Output graph after applying Tarjan's algorithm on the graph given in 4(a). (c) MUCset achieved after applying Algorithm 1 on the graph given in 4(a).

calculate connection vertices and disconnected subgraph(s) associated with each connection vertex.

3.2. Calculating Changes in Betweenness Centrality for Vertices Outside MUC_U

The Effect of the altered vertex on betweenness centrality of vertices outside MUC_U can be found by forming BFT for the vertex that was deleted. The BFT can be calculated with a time complexity of $O(|E|)$. We then calculate the dependency of each vertex with respect to the deleted vertex, starting from the vertices in the bottom level and recursively calculating the dependency for vertices in subsequent higher levels using (3.1). Then, we use Theorem 1 to update the centrality values. The complete procedure is shown in Algorithm 2. In the case of vertex addition, we add the dependency to the betweenness centrality scores of each vertex outside MUC_U, as stated in Theorem 3.5.

Algorithm 2 Calculating BFT and updating the vertices outside MUC_U accordingly

v_d: Vertex to be deleted.
Input: BC[v] of each vertex of original graph ($v \in V$).
$S \leftarrow$ Empty Stack
$P[w] \leftarrow$Empty List, $w \in V$
$\sigma[t] \leftarrow 0, t \in V, \sigma[v_d] = 1$
$d[t] \leftarrow -1, t \in V, d[v_d] = 0$
$Q \leftarrow$ Empty Queue
Enqueue $v_d \rightarrow Q$
while Q not empty **do**
  Dequeue $v \leftarrow Q$
  push $v \rightarrow S$
  for each neighbur of v **do**
    if $d[w] < 0$ **then**
      enqueue $w \rightarrow Q$
      $d[w] \leftarrow d[v] + 1$
    end if
    if $d[w] = d[v] + 1$ **then**
      $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
      append $v \rightarrow P[w]$
    end if
  end for
end while
$\delta[v] \leftarrow 0, v \in V$
while S not empty **do**
  pop $w \leftarrow S$
  for $v \in P[w]$ **do**
    $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}(1 + \delta[w])$
  end for
end while
for $v \in V \setminus \text{MUC}_U$ **do**
  $\text{BC}[v] \leftarrow \text{BC}[v] - 2.\delta[v]$
end for

3.3. Calculating Betweenness Centrality for Vertices in MUC_U

This section briefly describes the idea suggested in [23] for recomputation of betweenness centrality for vertices in MUC_U. In the disconnected subgraph G_j, let $V(G_j)$ denote the vertex set and let $|V(G_j)|$ denote the number of vertices. Let $|SP(u, v)|$ denote the number of shortest paths between vertex u and vertex v. Here, we will explain the basic steps of the algorithm, in brief. For detailed concept and the algorithm, used please refer to the QUBE algorithm [23]. Let betweenness centrality of vertex v, BC(v) for all $v \in MUC_U$ be initialized with 0. Let c_j be a connection vertex of MUC_U and G_j be the corresponding disconnected subgraph.

Now we calculate the betweenness centrality of vertex v by calculating and adding the effect of the following types of shortest paths (shown in Figure 5) on vertex v:

1. The shortest paths with both source and destination in MUC_U (Figure 5(a)): For counting the effect of these paths, we use the algorithm suggested by [3] for only the vertices in MUC_U and compute local betweenness centrality $BC_0^{MUC_U}(v)$ for all vertices $v \in MUC_U$.
2. The shortest paths with either source or destination (but not both) in MUC_U (Figure 5(b)): Let $< s, .., t >$ be a shortest path from $s \in V(G_j)$ to $t \in MUC_U$.

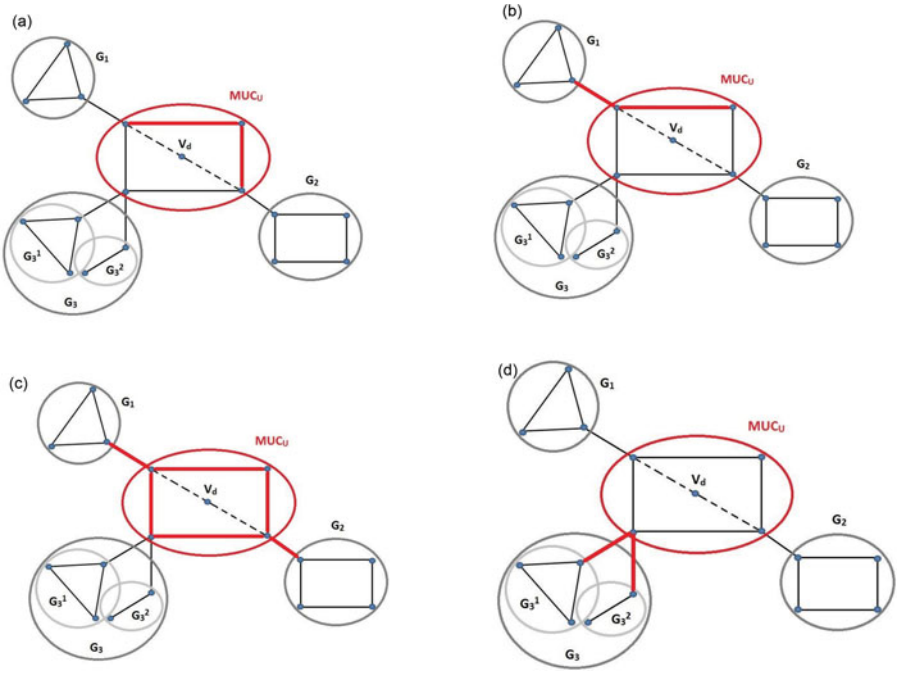

Figure 5 Types of the shortest paths considered for calculating betweenness centrality for vertices in MUC_U. (a) The shortest paths with both source and destination in MUC_U; (b) The shortest paths with either source or destination (but not both) in MUC_U; (c) The shortest path between the nodes from two different subgraphs connected to MUC_U, i.e., the shortest paths with neither source nor destination in MUC_U; (d) The shortest path between the nodes from two different components of a single subgraph connected to MUC_U.

In this case, $\frac{\sigma_{st}(v)}{\sigma_{st}} = \frac{\sigma_{c_j t}(v)}{\sigma_{c_j t}}$. So, to calculate the total effect of such paths, for each shortest path $< c_j, ..., t >$, we add the following factor to BC(v):

$$\mathrm{BC}_1^{<c_j,...,t>}(v) = \begin{cases} \dfrac{|V(G_j)|}{|SP(c_j,t)|}, & \text{if } v \in < c_j, ..., v_t > \backslash \{v_t\} \\ 0, & \text{otherwise.} \end{cases}$$

3. The shortest paths with neither source nor destination in MUC_U (Figures 5(c) and 5(d)): Let $< s, .., t >$ be a shortest path from $s \in V(G_j)$ to $t \in V(G_k)$ where $j \neq k$. In this case, $\frac{\sigma_{st}(v)}{\sigma_{st}} = \frac{\sigma_{c_j c_k}(v)}{\sigma_{c_j c_k}}$. So, to calculate the total effect of such paths, for each shortest path $< c_j, ..., c_k >$, we add the following factor to $BC(v)$:

$$\mathrm{BC}_2^{<c_j,...,c_k>}(v) = \begin{cases} \dfrac{|V(G_j)||V(G_k)|}{|SP(c_j,c_k)|}, & \text{if } v \in < c_j, ..., c_k > \\ 0, & \text{otherwise.} \end{cases}$$

When either of the subgraphs is disconnected, an additional factor:

$$\mathrm{BC}_3^i(c_i) = \begin{cases} |V(G_i)|^2 - \sum_{l=1}^{x}(|V(G_i^l)|^2), & \text{if } G_i \text{ is disconnected} \\ 0, & \text{otherwise.} \end{cases}$$

is added to the betweenness centrality calculations (c_i is a connection vertex), where ${G_j}^l$ is the lth component of G_i and x is the number of connected components in G_i.

So, we have the following formula to calculate the betweenness centrality score of a vertex in MUC_U:

$$\mathrm{BC}(v) = \mathrm{BC}_0^{\mathrm{MUC}_U}(v) + 2\sum_{G_j,t}\sum_{x \in SP(c_j,t)} \mathrm{BC}_1^x(v) + \sum_{G_j,G_k(j\neq k)}\sum_{y \in SP(c_j,c_k)} \mathrm{BC}_2^y(v)$$
$$(+\ \mathrm{BC}_3^i(c_i) \quad \text{if } v = c_i).$$

Let m be the number of edges and let n be the number of nodes in the given connected, undirected, unweighted graph with $m > n$. The time complexity of the proposed algorithm is $O(m)$ when the altered vertex is an end vertex (Type 3, Type 5). In this case, we just have to run Algorithm 2. In the other case when the altered node is of Type 1 or Type 4, the time complexity is $O(m'n' + m)$, where m' is the number of edges and n' is the number of nodes in MUC_U; $O(m'n')$ is the complexity due to the recomputation of betweenness scores of the nodes in MUC_U by the traditional Brandes' Algorithm. The additional m term in the complexity is due to the Algorithm 2 for updating the centrality score of vertices outside MUC_U.

4. IMPLEMENTATION AND RESULTS

We have implemented the algorithm for both addition and deletion of vertices. The algorithm can work faster for updating betweenness centrality, because it forms a subset of vertices of the graph, MUC_U, for which recalculation of betweenness centrality is to be done. For the rest of the vertices, Algorithm 2 updates the betweenness centrality in negligible

time compared to the recalculation step. The recalculation procedure includes the local Brandes algorithm. So, it is directly proportional to the number of vertices inside MUC_U. We have compared our results with the Brandes algorithm [3] because that is the best-known algorithm, according to our knowledge, for calculation of betweenness centrality after vertex updating. The experiments were performed on an Intel i5-2450M CPU with 2.5 GHz clock speed and 4 GB main memory.

We use a similar measure used by authors in [23] termed *proportion* to compare the algorithms. Proportion can be calculated as:

$$\left(\frac{\text{Number of vertices in MUC}_\text{U}}{\text{Total number of vertices in the graph}}\right).100$$

Proportion is a direct function of the number of vertices in MUC_U and thus speedup achieved by our algorithm is directly affected by the proportion. So, a smaller proportion would mean that betweenness centrality for a lesser number of nodes will have to be recomputed, and this should achieve greater speedup, which we achieved in our experimental results. We considered the following strategy to compute the *average proportion* of a graph. We start randomly deleting vertices from the graph untill either the graph becomes disconnected or k vertices are deleted. Then we take the average of proportion values for each deletion. The *average speedup* is calculated in a similar way. We consider $k = 500$ for real networks. In general, average speedup on a graph or average proportion of a graph depends on the number of MUCs formed by biconnected components and the fraction of the total number of vertices belonging to these MUCs. If a graph consists of most of the MUCs having small numbers of vertices with respect to the total number of vertices in the graph, the average proportion will be small, and thus, average speedup on that graph will be large.

4.1. Results for Synthetic Graphs

For experiments related to synthetic graphs we generated random Erdös Rényi graphs with 1000, 2000, and 3000 nodes. We considered addition and deletion of nodes for different proportions of x, for each $x \in A$, where $A = \{10; 20; 30; 40; 50; 60; 70; 80\}$ in the case of deletion and $A = \{10; 20; 30; 40; 50; 60; 70; 80; 90\}$ for addition. For the experiments in which we considered node deletion, we used a probability of 0.05 for all generated graphs, and for experiments in which we considered node addition, we used a probability of 0.08 for graphs with 1000 and 2000 nodes and 0.07 for graphs with 3000 nodes.

We initially calculated the betweenness centrality of each vertex using the Brandes algorithm. Then we ran the preprocessing step (modified Algorithm 1 on the basis of Theorem 3.5 for node addition and Algorithm 2 for deletion) to calculate the MUCs, connection vertices, and disconnected subgraphs for our graphs. For the case of node deletion, 50 vertices were continuously removed from each x proportion graph of each group and average update times were calculated for different proportions. In the case of node addition, average updation times were calculated over 100 iterations, where, in each iteration, a vertex was added randomly to each x proportion graph of each group.

The results are plotted with average update time (in ms) at the y-axis and the proportion of the graphs at the x-axis for each group and are shown in Figure 6 for the vertex deletion case and Figure 7 for the vertex addition case. We get different speedups for an different proportion of graphs in each group. In the case of deletion, for synthetic graphs

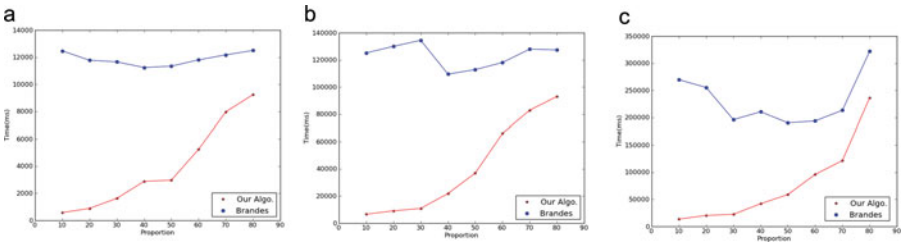

Figure 6 Plots for comparison of ours and the result in on synthetic graphs with 1000, 2000, and 3000 vertices, respectively, in the case of vertex deletion. (**a**) 1000 vertices; (**b**) 2000 vertices; (**c**) 3000 vertices.

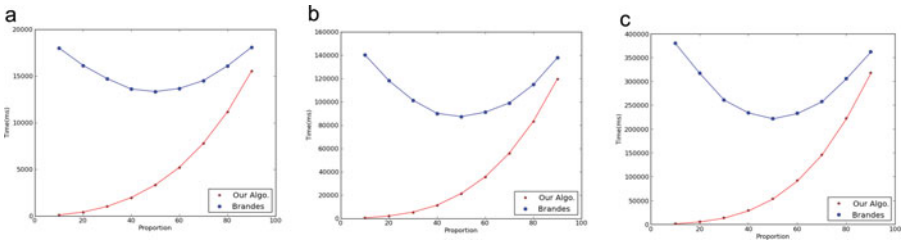

Figure 7 Plots for comparison of ours and the result in on synthetic graphs with 1000, 2000, and 3000 vertices, respectively, in the case of vertex addition. (**a**) 1000 vertices; (**b**) 2000 vertices; (**c**) 3000 vertices.

with 1000 nodes, we achieve a speedup of 13.26, 3.90, and 2.25 for graphs with proportion 20, 40, and 60, respectively; for synthetic graphs with 2000 nodes, we achieve a speedup of 14.27, 5.01, and 1.78 for graphs with proportion 20, 40, and 60, respectively; for synthetic graphs with 3000 nodes, we achieve a speedup of 12.54, 5.02 and 2.02 for graphs with proportion 20, 40, and 60, respectively as shown in Figure 6. In the case of addition, for synthetic graphs with 1000 nodes, we achieve a speedup of 38.40, 6.89, and 2.62 for graph with proportion 20, 40, and 60, respectively; for synthetic graphs with 2000 nodes, we achieve a speedup of 54.36, 7.87, and 2.55 for graphs with proportion 20, 40, and 60, respectively; for synthetic graphs with 3000 nodes, we achieve a speedup of 59.63, 8.11, and 2.54 for graphs with proportion 20, 40, and 60, respectively, as shown in Figure 7.

4.2. Results for Real Graphs

We also tested our algorithm on real-world networks. We chose different types of real datasets to depict the flexibility of our algorithm. We converted directed graphs into simple undirected graphs without self-loops and parallel edges. We picked the networks mentioned Following and performed simulations on them in July 2013. Collaboration networks generally depict the collaborations an author has made while writing research papers. Interaction networks are networks in which nodes are connected because of their features. An ownership network suggests transfer of resources between two nodes. A trust network is a network of individuals with kindred interests and connections. Citation networks depict are a which author has cited which author in his/her research work. Web subgraphs collection of webpages that match a certain query of a search engine. The yeast protein-protein interaction network's (YeastL and YeastS) were also taken as an input to perform

Name of Dataset	Type	|V|	|E|	Avg. Proportion	Avg. Speed-up
YeastL	Interaction	2361	6646	31.16	64.88
YeastS	Interaction	2361	6646	28.05	72.76
Geom	Collaboration	7343	11898	17.08	7.63
Erdos02	Collaboration	6927	11850	7.13	323.75
Edros972	Collaboration	5488	8972	9.49	411.835
ODLIS	Dictionary data	2909	16377	67.68	18.52
Wiki-vote	Trust	7115	100762	38.452	28.472
California	Web Subgraphs	9664	15969	25.54	77.658
EPA	Web Subgraphs	4772	8909	21.55	150.456
Lederberg	Citation	8843	41532	58.86	45.159
SciMet	Citation	3084	10399	62.42	17.55
US Power Grid	Electronic Transport	4941	6594	38.03	35.42

Table I Simulation results for real-world datasets.

simulations.[3] We picked Geom, Erdos02, Erdos972, California, EPA, Lederberg, SciMet and US Power Grid networks.[4] ODLIS network.[5] weblink. We also extracted Wiki-vote network.[6] For the real networks mentioned, information about networks, average proportion, and average speed-ups over the Brandes algorithm are summarized in Table I.

5. RELATED WORK

The idea of betweenness centrality was first introduced in articles by [1, 10]. In article [31], another measure, which considered random walks on any arbitrary length rather than just shortest paths between two vertices, was defined. In [4], other types of betweenness centrality such as, edge betweenness and group betweenness and algorithms to compute them efficiently were considered. Betweenness centrality was earlier calculated by finding the number and length of shortest paths between two vertices and then adding pair dependencies for all pairs. An algorithm was suggested in [3], which introduces a recursive way to sum the dependencies in graphs. Although the algorithm proposed in [3] was faster than that used previously, it was still too costly for large graphs. Recently, [32] gave an approach that could reduce the betweennesss computation based on some special structures in graphs such as several small biconnected components and several isomorphic nodes (nodes with the same neighburhood). Several approximation algorithms were proposed [2, 6, 11, 26]. Real-world networks tend to be large and transient. Work has been done in [23, 12] to find betweenness centrality after updating in a graph. The algorithm suggested in [23] selects a subset of vertices whose betweenness centrality is updated. However, it works only in the case of edge removal and addition. The algorithm

[3]The data for YeastL and Yeast S is available at *http://vlado.fmf.unilj.si/pub/networks/data/bio/Yeast/ Yeast.htm* website

[4]*http://www.cise.ufl.edu/research/sparse/matrices/Pajek/*

[5]*http://vlado.fmf.uni-lj.si/pub/networks/data/dic/odlis/Odlis.htm*

[6]*http://snap.stanford.edu/data/wiki-Vote.html*

suggested in [12] takes into consideration different instances that might arise due to edge addition in a graph and speeds up the algorithm in these cases. This algorithm works only for streaming graphs, i.e., only in the case of edge additions. Recently, in an article [29], an incremental algorithm for updating edges using Single Source Shortest Path Directed Acyclic Graphs for each vertex was used recursively to update vertices. They have reported the time complexity for updating to $O(m'n + n^2)$, where m' is the maximum number of edges that can lie on a shortest path. Another incremental algorithm was given in [20], which extends the incremental algorithm for finding the all pairs shortest path problem. Recently, several other algorithms [39, 13, 22] have been proposed that speedup the updating process of betweenness centrality and [39]. algorithms for updating closeness centrality . Those algorithms work in two steps: convergence and aggregation. In the convergence step, the authors calculate the difference in the number of shortest paths or the difference in the length's of shortest paths before and after updating. Then, they aggregate the changes in the existing centrality scores to get the updated centrality in the aggregation step. In another study [22], an online algorithm is given that can keep the betweennesss of nodes and edges up to date. [13] A distributed approach for updating the betweenness score in the distributed online social networks is given.

6. CONCLUSION

In this article, we formulated an algorithm that efficiently calculates betweenness centrality when vertices in a graph are updated. We did not consider the traditional way of updating betweenness centrality after node alteration, which considers a node alteration event as a series of edge alteration event's. We achieved the speedups by calculating two sets of vertices; one for which we need to update betweenness scores and the other for which we need to recompute the betweenness score. We achieve an average speedup of 6.13 for a proportion of 40 for considered synthetic graphs compared to the Brandes algorithm [3]. For real graphs, we get an average speedup of around 133 for a proportion of 29. The speedup will increase further when proportion decreases.

REFERENCES

[1] J. M. Anthonisse "The Rush in a Directed Graph." *Stichting Mathematisch Centrum. Mathematische Besliskunde* BN 9/71(1971):1–10.

[2] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. "Approximating Betweenness Centrality." In *Proceedings of the 5th International Conference on Algorithms and Models for the Web-Graph*, WAW'07, pp. 124–137. Berlin, Heidelberg: Springer-Verlag, 2007.

[3] U. Brandes. "A Faster Algorithm for Betweenness Centrality." *The Journal of Mathematical Sociology* 25:2 (2001),163–177.

[4] U. Brandes. "On Variants of Shortest-Path Betweenness Centrality and Their Generic Computation." *Social Networks* 30:2 (2008),136–145.

[5] U. Brandes and T. Erlebach (Editors). *Network Analysis: Methodological Foundations*, LNCS 3418. Berlin, Heidelberg: Springer, 2005.

[6] U. Brandes and C. Pich. "Centrality Estimation in Large Networks." *International Journal of Bifurcation and Chaos* 17:07 (2007), 2303–2318.

[7] R. Carvalho, L. Buzna, F. Bono, E. Gutiérrez, W. Just, and D. Arrowsmith. "Robustness of Trans-European Gas Networks." *Physical review E* 80:1 (2009), 016106.

[8] S. Derrible. "Network Centrality of Metro Systems." *PloS One* 7:7 (2012), e40575.

[9] A. Flrez, D. Park, J. Bhak, B.-C. Kim, A. Kuchinsky, J. Morris, J. Espinosa, and C. Muskus. "Protein Network Prediction and Topological Analysis in Leishmania Major as a Tool for Drug Target Selection." *BMC Bioinformatics* 11:1 (2010), 1–9.

[10] L. C. Freeman. "A Set of Measures of Centrality Based on Betweenness." *Sociometry* 40:1 (1977), 35–41.

[11] R. Geisberger, P. Sanders, and D. Schultes. "Better Approximation of Betweenness Centrality." In *Proceedings of 10th Workshop on ALENEX*, pp. 90–100. SIAM, 2008.

[12] O. Green, R. McColl, and D. Bader. "A Fast Algorithm for Streaming Betweenness Centrality." In *2012 International Conference on Privacy, Security, Risk and Trust (PASSAT), and 2012 International Conference on Social Computing (SocialCom)*, pp. 11–20. ASE/IEEE, (2012).

[13] B. Guidi, M. Conti, A. Passarella, and L. Ricci. "Distributed Protocols for Ego Betweenness Centrality Computation in DOSNs." In 2014 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 539–544. IEEE, 2014.

[14] P. Hage, and F. Harary. "Eccentricity and Centrality in Networks." Social Networks 17:1(1995), 57–63.

[15] A. Hanna. "Revolutionary Making and Self-Understanding: The Case of #Jan25 and Social Media Activism." Paper presented at meeting of the International Studies Association, San Diego , CA, April 2012.

[16] Y. J. Huang, D. Hang, L. J. Lu, L. Tong, M. B. Gerstein, and G. T. Montelione. Targeting the human cancer pathway protein interaction network by structural genomics. *Molecular & Cellular Proteomics* 7:10 (2012), 2048–2060.

[17] Y. Iturria-Medina, R. C. Sotero, E. J. Canales-Rodríguez, Y. Alemán-García, and L. Melie-Gómez. "Studying the Human Brain Anatomical Network via Diffusion-Weighted MRI And Graph Theory." *Neuroimage* 40:3 (2008), 1064–1076.

[18] M. O. Jackson. *Social and Economic Networks*. Princeton, NJ, USA: Princeton University Press, 2008.

[19] M. P. Joy, A. Brock, D. E. Ingber, and S. Huang. "High-Betweenness Proteins in the Yeast Protein Interaction Network." *BioMed Research International* 2005:2 (2005), 96–103.

[20] M. Kas, M. Wachs, K. M. Carley, and L. R. Carley. "Incremental Algorithm for Updating Betweenness Centrality in Dynamically Growing Networks." In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM '13*, pp. 33–40. New York, NY, USA. ACM, 2005.

[21] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. "A Faster Algorithm for Minimum Cycle Basis of Graphs." In *Automata, Languages and Programming*, edited by J. Daz, J. Karhumki, A. Lepist, and D. Sannella, pp. 846–857, Lecture Notes in Computer Science 3142. Berlin, Heidelberg: Springer, 2004.

[22] N. Kourtellis, G. D. F. Morales, and F. Bonchi. "Scalable Online Betweenness Centrality in Evolving Graphs." arXiv preprint arXiv:1401.6981, 2014.

[23] M.-J. Lee, J. Lee, J. Y. Park, R. H. Choi, and C.-W. Chung. "Qube: A Quick Algorithm for Updating Betweenness Centrality." In *Proceedings of the 21st International Conference on World Wide Web, WWW '12*, pp. 351–360. New York, NY, USA. ACM, 2012.

[24] J. Lienert, F. Schnetzer, and K. Ingold. "Stakeholder Analysis Combined with Social Network Analysis Provides Fine-Grained Insights into Water Infrastructure Planning Processes." *Journal of Environmental Management* 125 (2013), 134–148.

[25] T.-C. Lu, Y. Zhang, D. L. Allen, and M. A. Salman. "Design for Fault Analysis Using Multi-Partite, Multi-Attribute Betweenness Centrality Measures." In *Proceedings of the Annual Conference of the Prognostics and Health Management Society 2011*, pp. 190–197. PHM, 2011.

[26] Y. Makarychev. "Simple Linear Time Approximation Algorithm for Betweenness." *Operations Research Letters* 40:6 (2012), 450–452.

[27] Y. Nagata, K. Asatani, and H. Otsuka. "A New Load-Balancing Method Applying Link-Cost Adjustment Based on Betweenness Centrality in WDM Networks." In *Proc. of the 6th International Conference on Ubiquitous and Future Networks (ICUFN),* pp. 59–60. IEEE, 2014.

[28] S. Narayanan. "The Betweenness Centrality of Biological Networks." PhD thesis, Virginia Polytechnic Institute and State University, 2005.

[29] M. Nasre, M. Pontecorvi, and V. Ramachandran. "Betweenness Centrality–Incremental and Faster." *arXiv preprint arXiv:1311.2147*, 2013.

[30] M. Newman. *Networks: An Introduction*. New York, NY, USA: Oxford University Press, 2010.

[31] M. J. Newman "A Measure Of Betweenness Centrality Based On Random Walks." *Social Networks*, 27:1 (2005), 39–54.

[32] R. Puzis, Y. Elovici, P. Zilberman, S. Dolev, and U. Brandes. "Topology Manipulations for Speeding Betweenness Centrality Computation." *Journal of Complex NetWorks*. Available online (http://comnet.oxfordjournals.org/content/early/2014/04/29/comnet.cnu015.abstract), 2014.

[33] G. Sabidussi. "The Centrality Index of a Graph." *Psychometrika* 31:4(1966), 581–603.

[34] J. Scheurer and C. Curtis. "*Spatial Network Analysis of Multimodal Transport Systems: Developing a Strategic Planning Tool to Assess the Congruence of Movement and Urban Structure: a Case Study of Perth Before and After the Perth-to-Mandurah Railway*." GAMUT, Australasian Centre for the Governance and Management of Urban Transport, University of Melbourne, 2008.

[35] B. Shanmugham and A. Pan. "Identification and Characterization of Potential Therapeutic Candidates in Emerging Human Pathogen Mycobacterium Abscessus: A Novel Hierarchical in Silico Approach." *PloS One*, 8:3 (2013), e59126.

[36] A. Shimbel. "Structural Parameters of Communication Networks." *The Bulletin of Mathematical Biophysics* 15:4 (1953), 501–507.

[37] T. Spiliotopoulos and I. Oakley. "Applications of Social Network Analysis for User Modeling." In *Proceedings of the International Workshop on User Modeling from Social Media*. New York, NY, USA: ACM, 2012.

[38] R. Tarjan. "Depth-First Search and Linear Graph Algorithms." *SIAM Journal on Computing*, 1:2 (1972), 146–160.

[39] W. Wei and K. Carley. "Real Time Closeness and Betweenness Centrality Calculations on Streaming Network Data." Paper presented at the Proceedings of the 2014 ASE BigData/SocialCom/Cybersecurity Conference, Stanford University, May 27–31, 2014.

[40] Y. Yan, L. Xiao, and Z. Xintian. "Analyzing and Identifying of Cascading Failure in Supply Chain Networks." In *Proceedings of the 2010 International Conference on Logistics Systems and Intelligent Management* 3 (2010), pp.1292–1295.

[41] H. Yu, P. M. Kim, E. Sprecher, V.Trifonov, and M. Gerstein. "The Importance of Bottlenecks in Protein Networks: Correlation with Gene Essentiality and Expression Dynamics." *PLoS Computational Biology* 3:4 (2007), e59.