Vietnam Journal of Science and Technology ***57*** *(3) (2019) 344-355*

doi:10.15625/2525-2518/57/2/13166

A METHOD TO IMPROVE THE TIME OF COMPUTING BETWEENNESS CENTRALITY IN SOCIAL NETWORK GRAPH

Nguyen Xuan Dung[1, *], Doan Van Ban[2], Do Thi Bich Ngoc[3]

[1]*Hanoi Open University, B101 Nguyen Hien Str., Hai Ba Trung Dist., Ha Noi*

[2]*Viet Nam Academy of Science and Technology, 18 Hoang Quoc Viet, Cau Giay Dist., Ha Noi*

[3]*Posts and Telecommunications Institute of Technology, 122 Hoang Quoc Viet Str., Cau Giay Dist., Ha Noi*

[*]Email: *nguyenxuandung@hou.edu.vn*

Received: 14 August 2018; Accepted for publication: 5 December 2018

Abstract. The Betweenness centrality is an important metric in the graph theory and can be applied in the analyzing social network. The main researches about Betweenness centrality often focus on reducing the complexity. Nowadays, the number of users in the social networks is huge. Thus, improving the computing time of Betweenness centrality to apply in the social network is neccessary. In this paper, we propose the algorithm of computing Betweenness centrality by reduce the similar nodes in the graph in order to reducing computing time. Our experiments with graph networks result shows that the computing time of the proposed algorithm is less than Brandes algorithm. The proposed algorithm is compared with the Brandes algorithm in term of execution time.

Keywords: Betweenness centrality, mining graph data, analyzing the social network.

Classification numbers: 4.10.2.

1. INTRODUCTION

In analyzing social network, Betweenness is often used for analyzing, monitoring or finding subgroup or community on the social network graph. Betweenness plays an important role in spreading information in the network. The higher Betweenness of an object is, the more important it is.

The researches of analyzing social network [1, 2] proposed some measures to analyze some structured form of community and structure of the social network. Measure Betweenness centrality is often used. In 1977, Freeman [3, 4] proposed a definition about Betweenness centrality. In 2001, Brandes [1] studied about the time of computing Betweenness centrality for a graph. It is $O(nm+ n^2 \log n)$ for weight graph with n nodes and m edges; là $O(nm)$ for unweighting graph with n nodes and m edges. There are several researches about Betweenness centrality [2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. However, there researches focused on calculating Betweenness centrality in a graph. The problem of minimum the number of edges and nodes are not considered much.

Besides, nowadays the number of users in the social network increase quickly. Thus, the researches mentioned above cannot satisfy the computing time of Betweenness centrality of huge social networks (e.g., Facebook with billions of users). Betweeness centrality of edges on the graph has wide applications in social network analysis problems. The paper proposes the reduction of the number of vertices, the number of edges of the graph based on the selection of representative vertices with the same Betweenness centrality. Based on the graph reduction technique, the paper proposes an algorithm to calculate Betweenness centrality of the edges on the graph to reduce the execution time.

The structure of the paper is: Section 2 represents the materials and methods; Section 3 shows the results and discussion; The last section is a conclusion.

2. MATERIALS AND METHODS

2.1. Background

We can model the relationships in the social network as a graph G = (V, E), with V is set of node, E is a set of edges. Set V represents for members of the social network; E represents the relationships between members. Based on graph theory, the structure of the social network can be considered as an adjacency matrix $A = (a_{ij}) \in \{0, 1\}^{n \times n}$, with n = |V| and $a_{ij} = 1$ if 2 nodes i and j have an edge, if not $a_{ij} = 0$. Then the Betweenness of edges in a graph is defined as follows:

Definition 1. [16] For Graph G = (V, E) with e∈E is an edge of graph. With 2 nodes s, t ∈V, assuming that σ_{st} is the number of sorties path from s to t, and $\sigma_{st}(e)$ *is the number of* sorties path from s to t and visit e. Then, the Betweenness centrality of edge *e*, called $C_B(e)$, is defined as follows:

$$C_B(e) = \sum_{s \neq t} \frac{\delta_{st}(e)}{\delta_{st}}$$

We combine the methods of Girvan-Newman [16] and Brandes [1, 2] to traverse G = (V, E) using BFS(Breadth-First Search) in order to compute Betweenness centrality of edges in G. The algorithm traverses using BFS to find the shortest paths from source node to all nodes in G. The edges connecting between levels of nodes during BFS from source node X will be built a direct graph, no circle, called DAG_X (Directed Acyclic Graph).

Algorithm EBC (Edge- Betweenness centrality) compute Betweenness centrality of two edges in DAG (using BFS) [16]

Input:DAG_x – a graph for BFS of G

Output: $C_B(e)$, with all e in DAG_x

// Bottom up for DAG_x

- Find all node t is a leaf of DAG_x

// Node t has no sorted path visits it.

for v adjacent with t do {//∀v ∈ N(t){

e = (v, t); //edge connect v and t, e ∈ E

if (deg(t) == 1) then //t is a leaf node in G

$C_B(e) = W_1(t)$; // $W_1(t)$ is number of similar leaf nodes

else $C_B(e) = w_v / w_t$; }

- Start from edges which are farthest from x: let e = (i, j) in DAG_x - i is parent of j

$$C_B(e) = \left(\sum_{e'=(j,k),k\in N(j),d_k=d_j+1} C_B(e') + 1 \right) * \frac{w_i}{w_j}$$

- Redo step 2 until i is x.

2.2. Nodes with similar Betweenness centrality in a graph

2.2.1. Class of similar leaf nodes

Social network graphs are often complicated with a huge number of edges and nodes. Thus, computing Betweenness centrality is time- consuming. As we known, the problem of finding the shortest path between nodes in graph is NP-hard. Next part will study node class with a similar structure [17, 18]. These nodes will be combined in order to reduce the number of nodes and edges in the graph. Then, the performance of the algorithm for computing Betweenness centrality, the algorithm for analyzing structure community of graph are increasing.

In social network graph, many nodes are similar structure, they create similar classes and can be combined together to be a single node which stands for class of node to reduce the number of nodes and edges in the graph. From now, we change the original graph G = (V, E)to graph G = (V, E, W), with W is weight function for nodes, at first W(v) = 1 for all v ∈ V.

Definition 2. [18] For graph G = (V, E, W), node v ∈ V, is called leaf not if degree of it is 1 (deg(v) = 1).

Property 1. If v is a leaf node of graph G and e = (v, w) ∈ E then:

$C_B(e) = (|V| - 1)$

Proof.

G is connected, thus all v' ∈ V - {v} have paths to v. That means, there is shortest path from v to v'. Because v is leaf node, all shortest paths between v, v' must visit e. From Definition 2, the Betweenness centrality of edge is $C_B(e) = (|V| - 1)$.

Property 2. If v is a leaf node of G, and w is adjacent with v, (v, w) ∈ E then DAG_V and DAG_W have same subgraph $G_1 = DAG_V \cap DAG_W$. Graph G_1 is a subgraph of DAG_V (or DAG_W) obtained by remove leaf nodes connected with w and remove the connected edge.

Corollary 1. All leaf nodes connect with a node have the same subsystem G_1. Thus, when doing BFS, we can skip the start nodes are leaves.

Definition 3. [18] For undirected connected graph G = (V, E), u, w ∈ V are two leaf nodes, , u is similar level 1 with w, called $u \approx_1 w$ if and only if they adjective with v (N(u) = N(w) = {v}).

It is easy to see that relationship $\approx_1$ is the equivalent relationship.

Based on Property 1, all edges connect with leaf nodes have Betweenness centrality |V|-1.

Denote that V1 is a set of leaf nodes of G,

$V1 = \{ u \in V \mid deg(u) = 1\}$

$V1/\approx_1$ will create classes of similar leaf nodes, $V1/\approx_1 = \{C_1, C_2, \ldots, C_k\}$. Similarly, leaf nodes will connect with a node and have the same Betweenness centrality.

The similar leaf nodes can combine together to be a single node to reduce the leaf nodes in the graph. After reduce the similar leaf nodes of class C_i, i = 1..k, to be a node C'_i (is also a leaf), we obtain the graph $G_1 = (V_1, E_1, W_1)$, in that:

+ $V_1 = V - V1 \cup V_C$, với $V_C = \{C'_1, C'_2, \ldots, C'_k\}$, $V1 = \{ u \in V \mid deg(u) = 1\}$

+ $E_1 = E - \{(u, v) \mid u \in V1, v = N(u)\} \cup \{(v, C'_i) \mid i = 1..k, v = N(u), u \in C_i \}$

+ $W_1(v) = 1$, where $v \in V - V1$, $W_1(v) = |C_i|$, where $v \in V_C$.

2.2.2. Class of similar side nodes

Definition 4. [18] For undirected, connected graph G = (V, E, W), $u \in V$ is called side node of G if subgraph generated by set of adjacent nodes N(u) are clique (complete subgraph).

In here, we only consider side node with $|N(u)| \geq 2$, because if $|N(u)| = 1$ then u is leaf node we already consider before.

Property 3. If u is side node of graph G = (V, E, W), then u is either root or leaf in graph DAG with BFS.

Proof.

Assuming that u is neither root nor leaf node in graph DAG with BFS. Because u is a node in DAG,

thus, we have the shortest path visit u. All shortest paths from the root which visit u, must visit two adjacents v, w of u. If u is side node, from Definition 4, N(u) is a clique, it means that $(v, w) \in E$, for all v, w which are adjacent of u. Thus, the path visit u is not the shortest path in DAG, this conflicts with the property all paths in DAG are shortest paths.

Definition 5. [18] For aundirrected connected graph graph G = (V, E, W), $u, v \in V$, the relationship $\approx_2$: $u \approx_2 v$ if u, v are side nodes of G and N(u) = N(v).

Property 4. Assuming that the set of similar side node, $S = \{v_1, v_2, \ldots, v_h\}$ and $N(S) = N(v_i)$, i = 1..h, then the Betweenness centralities of edges connect side node with similar adjacent nodes is are the same: $C_B((v_i, v)) = C_B((v_j, v))$, for all $v_i, v_j \in S$, $v \in N(S)$.

Proof.

From assumption, the nodes v_i, i = 1..h are similar to side nodes, thus, they have the same set of adjacent nodes $N(S) = N(v_j) = N(v_i)$, $i,j \in [1,h]$. Then, for all $v \in N(S)$, $e_i = (v_i, v) \in E$, for all i = 1..h. Besides, a side node can be either root or a leaf in non-circle graphs (obtained from BFS) [18], thus $C_B(e_i) = C_B(e_j)$ for all i,j = 1..h. The similar side nodes can be combine to be a node to reduce the number of similar side nodes in the graph.

Assuming that G=(V,E,W) has classes of similar side node (each class has at least 2 side nodes). Each class is replaced by a node, we obtain graph $G_{1=}(V_1, E_1, W_1)$:

+ $V_1 = V - V1 \cup \{S'_1, S'_2, \ldots, S'_h\}$,

with $V1 = S_1 \cup S_2 \cup \ldots \cup S_h$.

+ $E_1 = E - \{(u, v) \mid u \in V1, v \in N(u)\} \cup \{(v, S'_i) \mid i = 1..h, v \in N(u) \text{ with } u \in V1\}$

+ $W_1(v) = W(v), v \in V - V1$; $W_1(S'_i) = |S_i|, i = 1..h$

To compute Betweenness centrality of edges in G_2 which is also Betweenness centrality of nodes in G_1, we use the following properties.

Property 5. Assuming that S is a set of similar side nodes, $S = \{v_1, v_2, \ldots, v_h\}$ if a side node v_i, $i = 1..h$, is selected to be root to do BFS, then h-1 remain nodes are leaves and the length from root is 2, the Betweenness centrality of adjacent edges of side node and adjacent nodes of DAG_{vi} are the same, $C_B((v, v_j)) = 1/ |N(S)|$, for all $j \neq i$, $v \in N(S)$.

Proof.

From the assumption, the nodes v_i, $i = 1..h$, are similar side nodes, thus they have the same set of similar adjacent nodes $N(S) = N(v_i)$, $i = 1..h$. When an adjacent v_i is selected to do BFS, then we have $|N(S)|$ adjacent nodes of v_i are all level 1 (the distance to the root is 1). Besides, $v \in N(S)$ is the parent of all remain similar side nodes v_j (level 2), that means v_j has $|N(S)|$ parent nodes. Thus, the ratio the shortest path from v_i (root node) to other similar side nodes in DAG_{vi} and visit edge (v, v_j) is $1 / |N(S)|$.

Property 6. Assuming that S is set of similar side node, $S = \{v_1, v_2, \ldots, v_h\}$ and $N(S) = N(v_i)$, $i = 1..h$, the betweenness centralities of edges that connect side node with similar adjacent nodes are the same: $C_B((v_i, v)) = C_B((v_j, v))$, for all $v_i, v_j \in S$, $v \in N(S)$.

Proof.

From the assumption, the nodes v_i, $i = 1..h$ are similar to side nodes, thus, they have the same set of adjacent nodes $N(S) = N(v_i)$, $i = 1..h$. Then, for all $v \in N(S)$, edge $e_i = (v_i, v) \in E$, for all $i = 1..h$. From Property 3, all side node can be either root or node of DAG, then $C_B(e_i) = C_B(e_j)$, for all $i, j = 1..h$.

2.2.3. Algorithm of combining similar nodes

The main task of analyzing social network, finding the structure of community is clarifying the metrics to evaluate the role of edges, or the Betweenness centrality of edges in a big graph. Combining the similar nodes to be a node will reduce the number of edges and reduce the task of commuting Betweenness, thus, clarifying the task of entities in the network will be faster.

Graph G_1 obtained from G by algorithm RED (Reduce Equivalence Degree) as follow: remove similar leaf node, side node with their adjacent edges, and replace them by a represent node with name is name of class and an adjacent edge for each represent node, the weights of leaf node, side node are the size of the classes that they represent.

Algorithm RED (Reduce Equivalence Degree)

Algorithm combine similar nodes

Input: G = (V, E, W)

Output: + $G_1 = (V_1, E_1, W_1)$ – obtained graph after combine similar nodes

+ V_C – set of leaf nodes represent for similar classes
+ V_S - set of side nodes represent for similar classes
$V_1 = V$;
$E_1 = E$;
P1 = ∅; //Stack keeps pair (leaf, Adjacent node)
P2 = ∅; //Stack keeps pair (side node, set of adjacent nodes)
for u ∈ V_1 do{
N[u] = Neighbor(G, u);//find adjacent node of u
if(deg(u) == 1) then {
v = N[u]; // N[u] is an adjacent node of u
P1.push(u, v);
$V_1 = V_1 – \{u\}$; $E_1 = E_1 – \{(u, v)\}$;
}// if(deg(u) == 1)
//find all side nodes
if(Clique(N[u]) then {
$V_1 = V_1 – \{u\}$;
for v ∈ N[u] do{
$E_1 = E_1 – \{(u, v)\}$;
P2.push(u, N[u]);
}
}//for u ∈ V do
//find similar nodes of side nodes
k = 1;
(u, v) = P1.pop();
C[k] = {u}
N[k] = v;
while(P1 != ∅) do {
(u, v) = P1.pop();
j = 1;
loop = true;
while (j <= k && loop) do
if (N[j] == v) then {
C[j] = C[j] ∪ {u};
loop = false;
}else j = j + 1;
if (loop) then {

```
                                k = k + 1;          //find next class
        C[k] = {u};
        N[k] = v;
                            }
          }// while( P1 != ∅)
          //combine similar nodes in class C[j] to be a leaf node C'j
          VC = ∅;
          for j = 1 to k do {
//k similar classes of side nodes
                    VC = VC∪ {C'j};
                    W1(C'j) = |C[j]|;
                    E1 = E1∪ {(C'j, N[j])}
       V1 = V1∪ VC;
          }// for j = 1 to k
          //find similar classes of side nodes
          h = 1;
          (u, M) = P2.pop();
          S[h] = {u}
          N[h] = M;
          while( P2 != ∅) do {
                    (u, M) = P2.pop();
                    j = 1;
                    loop = true;
while (j <= h && loop) do
                        if(N[j] == M) then
                              S[j] = S[j] ∪ {u};
                              loop = false;
                        }else j = j +1;
                    if (loop) then {
                              h = h + 1;
    S[h] = {u};
    N[h] = M;
}
          }
          //combine similar nodes in class S[j] tobe a side node Sj
          VS = ∅;
        for j = 1 to h do {
```

$V_S = V_S \cup \{S'_j\}$;
$W_1(S'_j) = |S[j]|$;
for $v \in N[j]$ do
$E_1 = E_1 \cup \{(S'_j, v)\}$
}
for $u \in V_1$ do
$W_1(u) = W(u)$;
$V_1 = V_1 \cup V_S$;

Algorithm Neighbor (G, u) [19]: find adjacent node of u in graph G.

Input: Graph G = (V, E, W) and node $u \in V$

Output: N – set of adjacent nodes of u in G

$N = \varnothing$;
for $v \in V$ do
if $((u, v) \in E)$ then
$N = N \cup \{v\}$;
return N;

Algorithm Clique (G, N) [20]: check if subgraph generated by a set of node N in G is a clique or not.

Input: Graph G = (V, E, W) and set of node $N \subseteq V$

Output: true if subgraph generated by a set of node N in graph G_1 is a clique, false in otherwise

for $u \in N$ do {
for $v \in N - \{u\}$ do
if$((u, v) \notin E)$ then
return false;
}
return true;

2.3. Fast algorithm for computing Betweenness centrality

Based on algorithm computing Betweenness centrality C_B using dependency accumulation technique of Brandes [1, 2] and using above features, the algorithm FBC (Fast algorithm for Betweenness centrality) for computing Betweenness centrality C_B of edges in the graph.

Algorithm FBC (Fast algorithm for Betweenness centrality)

Compute C_B of edges in social network graph

Input: + G = (V, E, W) social network graph

Output: + $C_B(e)$, $e \in E$

Algorithm **FBC** include 4 main phases:

Phase 1. Execute algorithm RED (G) to reduce similar leaf node, side nodes, change graph G = (V, E, W) to graph $G_2 = (V_1, E_1, W_1)$.

Phase 2. Initial the value of array $C_B[e] = 0$, $e \in E_2$, stack S = ∅, queue Q = ∅, for array Pr_x, Po_x, δ and d. Array δ is a number of the shortest path from root x to every node in DAG_x, array d is a distance of nodes from root x. At first, the distance between nodes and root is -1.

Array Pr_x, is a list of parent nodes link with each node v, $Po_s[v]$ contain children node after *v* in the next visit of BFS from *x*. V_C is a set of leaf node and V_S is a set of side nodes of G_1.

Phase 3. BFS travel from root x to find the shortest path to every node. In this phase, each element is pushed to a queue when it is found. During BFS, the distance from x to node v will be computed. For each node v found in next visit of BFS will have two lists of parents and children. δ[t] is shortest path from *x* to *t*.

Phase 4. Compute Betweenness centrality C_B of edges based on the accumulation technique of Brandes [1, 2]. For each DAG_x, $x \in V2$, compute Betweenness centrality of each edge in DAG_x, then compute the sum of them to be Betweenness centrality of the graph.

$$C_B(e) = \sum_{s \neq t} \frac{\delta_{st}(e)}{\delta_{st}}.$$

3. RESULTS AND DISCUSSION

3.1. *Evaluate the complexity of algorithm FBC*

The size of the memory stack, queue and array σ and d is O(|V2|), mean the size of the limit by the number of node V2 of graph. The complexity of BFS is $O(|V2| + |E_1|)$ and accumulation is about $O(|V2| + |E_1|)$, the maximum number steps of parent nodes $O(|E_1|)$, the corresponding children node is O(|V2|). Thus, the complexity of the algorithm is $O(|V2|^2 + |V2| * |E_1|)$. In case $|E_1| > |V2|$, the complexity of the algorithm is $O(|V2| * |E_1|)$. The complexity of Algorithm Brandes [1] is $O(|V|^2 + |V| * |E|)$, so the algorithm has better complexity, because we often have $|V2| < |V|$ and $|E_1| < |E|$.

3.2. Experimental result

We do an experiment and evaluate the algorithm FBC and compare with an algorithm of Brandes [1, 2] about execution time.

The program is implemented by R programming language and does an experiment in a computer with CPU Intel Core i5 4200U, RAM 4G. Because of the limitation of the computer, the experiment is done in Graph with maximum 1000 nodes and 4000 edges.

We do the experiment for 40 random graphs with 100 - 1000 nodes. For each node, 4 undirected graphs are created with the ratio from 2- 4 (each node connect with 2 - 4 other nodes).

(Unit: seconds)

Edge	Ratio	FBC	Brandes
100	2	3.59	4.06
200	2	16.85	17.78
300	2	41.94	49.02
400	2	82.12	100.00
500	2	174.17	201.27
600	2	259.49	292.85
700	2	389.83	446.71
800	2	538.60	601.57
900	2	682.55	746.00
1000	2	862.83	958.28
100	3	4.62	4.96
200	3	21.72	23.00
300	3	58.11	60.44
400	3	112.85	125.43
500	3	229.49	238.86
600	3	356.36	368.44
700	3	525.42	548.23
800	3	682.10	705.83
900	3	865.02	886.84
1000	3	1078.68	1099.30
100	4	5.52	5.84
200	4	23.16	23.41
300	4	62.32	67.05
400	4	146.19	159.29
500	4	282.92	295.28
600	4	441.20	449.95
700	4	619.00	630.71
800	4	786.12	796.84
900	4	992.18	1003.36
1000	4	1271.21	1279.61

Figure 1. The computing time of Betweenness centrality of algorithms FBC and Brandes.

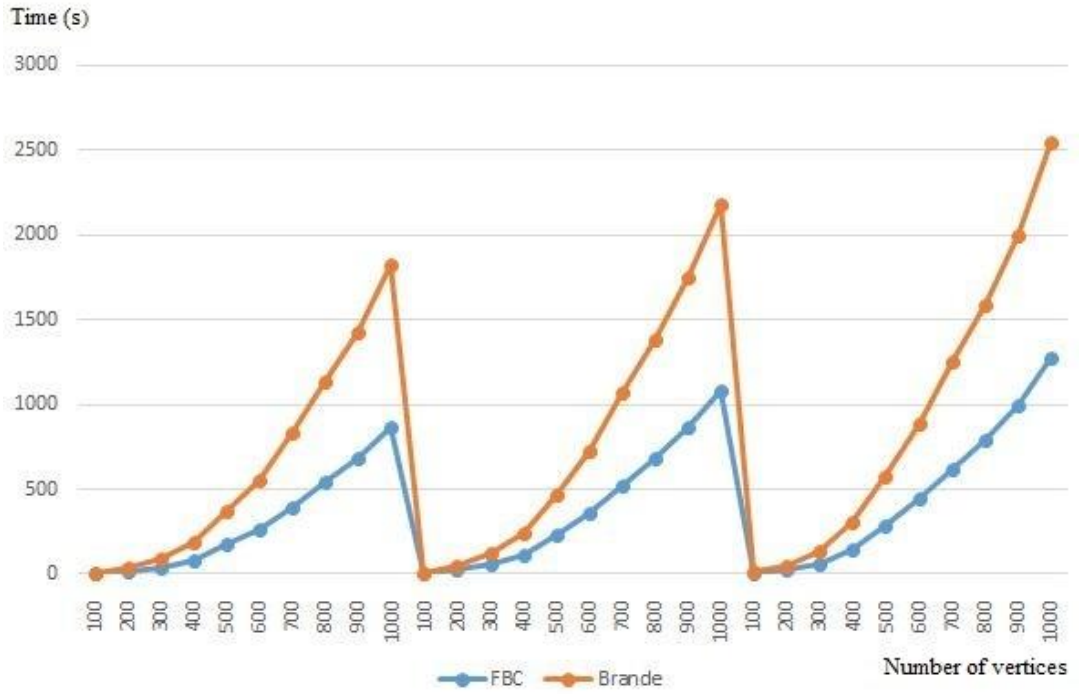

Figure 2. The computing time of Betweenness centrality of algorithms FBC and Brandes.

The results of our experiments can be found in figure 1. The run time values are measured in seconds.

Figure 2 shows running times for Betweenness centrality on random graphs with 100 to 1000 vertices.

The experiments results show that the proposed algorithm FBC is faster than the Brandes algorithm [1, 2].

4. CONCLUSIONS

In this paper, we proposed a method to increase the speed of computing exactly Betweenness centrality of edges in the social network graph. The proposed algorithm FBC allow us to compute Betweenness centralities of edges in social network faster. For future work, we can apply FBC to find the structure of the social network community faster. We implement this method in R, and experiments showed that the computing time of the proposed algorithm is less than Brandes algorithm.

REFERENCES

1. Brandes U. - A faster algorithm for Betweenness centrality, Journal of Mathematical Sociology **25** (2) (2001) 163-177.
2. Brandes U. and Pich C. - Centrality estimation in large networks, International Journal of Bifurcation and Chaos (2007).

3. Freeman L. C. - A set of measures of centrality based on Betweenness centrality, Sociometry **40** (1) (1977) 35-41.
4. Freeman L. C. - Centrality in social networks: Conceptual clarification, Social Networks **1** (1978) 215-239.
5. Bader D. A., Kintali S., Madduri K., and Mihail M. - Approximating Betweenness centrality. In WAW, 2007.
6. Bader D. A. and Madduri K. - Parrallel algorithm for evaluating centrality indices in real-world networks, In ICPP, 2006.
7. Edmonds N., Hoefler T., and Lumsdaine A. - A space efficient parallel algorithm for computing betweenness centrality in distributed memory, In HiPC, 2010.
8. Dora Erdos, Vatche Ishakian, Azer Bestavros, Evimaria Terzi - A divide and Conquer Algorithm for Betweenness Centrality, 2015.
9. Steven Fleuren - Using preprocessing to speed up Brandes' betweenness centrality algorithm, 2018.
10. Geisberger R., Sanders P., and Schultes D. - Better approximation of betweenness centrality, In ALENEX, 2008.
11. Newman M. - A measure of Betweenness centrality centrality based on random walks, Social Networks **27** (1) (2005) 39-54.
12. Mudduri K., Ediger D., Jiang K., Bader D. A., and Chavarria-Miranda D. G. - A faster paralel algorithm and effcient multithreaded implementations for evaluating Betweenness centrality on massive datasets. In IPDPS, (2009).
13. Riondato M. and Kornaropoulos E. M. - Fast approximation of Betweenness centrality through sampling WSDM'14, (2014) 413-422.
14. Sariyuce A. E., Saule E., Kaya K., and Catalyurek U. V. - Shattering and compressing networks for Betweenness centrality, In SDM, 2013.
15. Tan G., Tu D. and Sum N. - A parallel algorithm for computing Betweenness centrality, In ICPP, 2009.
16. Newman M. E. J. and Girvan M. - Finding and evaluating community structure in networks, Physical Review E **69** (2004) 026113.
17. Mikhail Chernoskutov, Yves Ineichen, Costas Bekas - Heuristic Algorithm for Approximation Betweenness centrality Using Graph Coarsening, 4th International Young Scientists Conference on Computational Science, Elsevier B.V **66** (2015) 83-92.
18. Rami Puzis, Polina Zilberman, Yuval Elovici, Shlomi Dolev and Ulrik Brandes - Heuristics for Speeding up Betweenness centrality Computation, ASE/IEEE International Conference on Privacy, Security, Risk and Trust, 2012.
19. Zhang H., Berg A. C., Maire M., and Malik J. - "SVM-KNN: Discriminative nearest neighbor classification for visual category recognition", In IEEE Computer Society Conference on Computer Vision and Pattern Recognition **2** (2006) 2126-2136.
20. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. - Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, SIGMOD'(98).