International Journal on Emerging Technologies 11(2): 370-377(2020)

ISSN No. (Print): 0975-8364
ISSN No. (Online): 2249-3255

A Fast Computation of Betweenness Centrality in Large-Scale Unweighted Graphs

P.H. Du[1], N.S. Duong[2], N.C. Nguyen[3] and N.H. Nguyen[4]
[1]Ph.D. Department of Information Systems, VNU University of Engineering and Technology, Vietnam National University, Hanoi, Viet Nam.
[2]Ph.D. Student, Technical Department of Security, Vietnam Ministry of Public Security, Ha Noi, Viet Nam.
[3]Ph.D. Department of Information Technology, Vietnam Ministry of Public Security, Ha Noi, Viet Nam.
[4]Associate Professor, Department of Information Systems, VNU University of Engineering and Technology, Vietnam National University, Hanoi, Viet Nam.

(Corresponding author: N.H Nguyen)
(Received 18 December 2019, Revised 18 February 2020, Accepted 20 February 2020)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: Betweenness Centrality is one of the widely used metrics in graph analysis to identify influential nodes. This metric has been applied in various fields such as in social networks, social commerce, malware detection, transportation, detecting terrorist risk. However, computing this metric also takes much time in case of having a large number of vertices/edges, even using the Brandes's faster algorithm. A variety of methods have been proposed to improve this algorithm, such as GPU usage, cluster usage, approximation, etc. However, taking advantage of the computational resources of conventional computing systems with parallel computing on multi-core, multi-CPU, and optimizing data structures to improve the cache performance has not been focused on previous researches. Thus, we present in this paper an efficient algorithm to enhance the performance of computing the Betweenness Centrality of all vertices in a large-scale unweighted graph. This proposed algorithm is based on the main ideas: (i) using the appropriate data structure to enhance the data localization and then less reference time of graph data, and (ii) reducing the Brandes's faster algorithm execution time with parallel computing. To evaluate its performance, we conducted experiments of our method, called bigGraph, and the two other popular toolkits: TeexGraph and NetworKit, with six different real-world social networks. Experimental results figure out that bigGraph is faster than TeexGraph and NetworKit 1.11-1.35 and 1.61-2.44 times, respectively.

Keywords: Betweenness Centrality, Breadth-First Search, Multi-threaded Parallel Computing, Social Network Analysis.

Abbreviations: BC, betweenness centrality; CC, closeness centrality; BFS, breadth-first search; SSSP, single source shortest path; APSP, all pair shortest path.

I. INTRODUCTION

The graph theory is now widely used in many different areas, from problems in network planning, Internet analysis, social networks, social commerce, bioinformatics, malware detection, transportation. One of the typical applications of graph theory is analyzing large-scale social networks. For this domain, the direct relationship between two members (if exists) is represented by an edge connected directly between two vertices, one for a member. Facebook, WhatsApp, Twitter, and YouTube are now popular social networks and they are considered the efficient ways to connect people in our networked society. The scale of these networks is very large: as the statistic provided by The Statistics Portalin July 2019, the number of active users of Facebook is 2.32 billion; YouTube is 1.9 billion and WhatsApp surpassed 1.6 billion [1]. The growth of these social networks has also encouraged the social commerce research in both shaping the purchases and information sharing intentions [2], consumer behavior analysis [3], social marketing strategy, finding micro-influencers [4].

To deal with the large-scale graphs such as those mentioned networks, many graph analysis methods have been proposed and shaped a popular field called "Social Network Analysis - SNA". It composes, in general, the process of investigating social structures through the use of networks and graph theory [5]. To analyze the graphs, the centrality is the most important and widely used metrics, aiming to find the most "important" or the most influential vertices to the others in the graph. In other words, we should find and identify which node has the most effect on the other [6], or which node representing the most important users [7]. When applying this concept to different fields, we can find the main nodes on the Internet or the vertices that spread the disease when modeling the plague spread problem. The concept of "importance" is defined in different ways when analyzing graphs. From that point, many centrality metrics are proposed to clarify the "important" properties such as the betweenness centrality, closeness centrality, PageRank centrality [8].

Among these metrics, by the first formal definition of Freeman [9], the Betweenness Centrality (BC) of a vertex is a metric that is determined based on shortest paths: for every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) are minimized. The BC for each vertex is the number of

these shortest paths that pass through the vertex. It is also considered as one of the most widely used metrics in graph analyses [8]. This is the main idea that encourages us to focus on this metric in our work.

By its definition, determining the BC metric for every node in a graph/network has to figure out the all pairs shortest path problem. It means that we need to perform a complete breadth-first search (BFS) for an unweighted graph or a complete execution of Dijkstra's algorithm for a weighted graph. In general, for the large-scale real-world social networks, computing the BC for all members takes remarkably a complex duration time [10], even using the Brandes's faster algorithm [17]. A variety of methods have been proposed to improve this algorithm, such as GPU usage, cluster usage, approximation, etc [39]. However, taking advantage of the computational resources of conventional computing systems with parallel computing on multi-core, multi-CPU, and optimizing data structures to improve the cache performance has not been focused on previous researches.

In this paper, we focus on proposing a method to enhance the performance of computing the Betweenness Centrality for all vertices of a large-scale unweighted graph. Our proposed method is based on two main ideas. First, we proposed an appropriate data structure to improve the data localization, thereby increasing the hit cache rate in the shared memory model. Secondly, we improve the faster Brandes's algorithm by paralleling the BFS for all vertices with the threading programming model. The proposed method was implemented in a graph analyzing tool, namely bigGraph, and freely published on the GitHub.

The rest of this paper is organized into the following sections: In Section II, centrality metrics and an algorithm for BC computation are given; the related work is also mentioned in this section. Our proposed method for improving the BC computation for all vertices is explained in Section III. Section IV gives experimental results to validate our approach with different social networks and other similar toolkits. Finally, we summarize the paper in Section V and show our future scope in Section VI.

II. PRELIMINARIES AND RELATED WORK

A. Notations

In this article, we focus only on unweighted graph *G*(*V, E*), where $\boldsymbol{V}$ is the set of all vertices and $\boldsymbol{E} = \{(\boldsymbol{v_i}, \boldsymbol{v_j}) | \boldsymbol{v_i}, \boldsymbol{v_j} \in \boldsymbol{V}\}$ represents the set of all edges ($\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ are connected with a single unweighted link). The total number of edges to (incoming) and from (outgoing) a vertex $\boldsymbol{v_i}$ is called the degree of $\boldsymbol{v_i}$ and is represented as $deg(v_i)$.

Two nodes $u, v \in V$ are connected if there exists a path between $u$and v. If all vertex pairs in G are connected, we say that G is connected. Otherwise, it is disconnected, and each maximal connected subgraph of $\boldsymbol{G}$ is a connected component, or a component, of *G*.

In our work, we use $dst(u, v)$ to denote the length of the shortest path between two vertices u, v in a graph G. Ifu and$\boldsymbol{v}$ are identical then $dst(u, v) = 0$. Moreover, if $\boldsymbol{u}$ and $\boldsymbol{v}$ are disconnected then $dst(u, v) = \infty$.

In graph analysis, the centrality of a node allows identifying the most important one. Centrality concepts are also applied in other problems such as essential nodes on the Internet and super-spreaders of disease. There are four indicators of centrality defined as follows:

– **Degree Centrality** is defined as the number of links incident upon a node. It is measured by the following formula:

$$DC = \deg(v) : v \in V \quad (1)$$

– **Closeness Centrality** is the indicator computed by the average length of the shortest path between the node and all other nodes in the network. Thus, the more central a node is, the closer it is to all of the other nodes. Closeness Centrality is computed by the following formula:

$$CC(v) = \frac{1}{\sum_{u \in V} dst(u,v)} \quad (2)$$

where $dst(u, v)$ is the shortest distance between node $\boldsymbol{u}$ and node $\boldsymbol{v}$.

In order to avoid the value ∞ when computing the shortest distance of a disconnected graph G, the *CC* of a node v is computed for the largest-componentΓ_G of G. Moreover, if a node u cannot reach any other node in G, then *CC(u) = 0*.

– **Betweenness Centrality** is defined as a centrality measure of a node within a network that quantifies the number of times a node acts as a bridge along the shortest path between two other nodes. It was introduced as a measure for quantifying the control of a human on the communication between other humans in a social network by Linton Freeman [9]. In his work, vertices that have a high probability to occur on a randomly chosen shortest path between two randomly chosen vertices will have high betweenness centrality value. Betweenness Centrality (BC) is computed by the following formula:

$$BC(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st} v}{\sigma_{st}} \quad (3)$$

where σ_{st} is the total number of shortest paths from node $\boldsymbol{s}$ to node $\boldsymbol{t}$; $\sigma_{st} v$ is the number of those paths that pass through $\boldsymbol{v}$.

– **Eigenvector Centrality** is an indicator to measure the influence of a respective node in a network. Relative scores are assigned to all influence nodes based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than the equal connections to low-scoring nodes [11]. Examples of variants of Eigenvector Centrality are Katz Centrality and Google's Page Rank.

The adjacency matrix is used to compute the Eigenvector Centrality. Let $A = (a_{u,v})$ be the adjacency matrix of $\boldsymbol{G}$: $a_{u,v} = 1$ if node u is linked to node v and $a_{u,v} = 0$ otherwise. The Eigenvector Centrality x of node $\boldsymbol{v}$ can be defined as:

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t \quad (4)$$

where $M(v)$ is a set of the neighbors of v and λ is a constant. In matrix form we have: $\lambda x = xA$.

B. Related Work

The Betweenness Centrality, which was proposed by Freeman in 1977, is one of the metrics that is widely used in general graph analysis to identify important vertices. This measure has been applied to graph/network analysis in many different areas such as transportation [12], biology - health [13], social network analysis for community detection [14], discovering the risk of terrorism [15], control power flow pattern throughout the whole power grid [39].

To calculate this indicator for all vertices V in graph $G = (V, E)$, we need to solve the problem of finding the

shortest path on all the vertex pairs in G (All Pair Shortest Path - APSP problem). Obviously, when the number of vertices and edges of G is large, the implementation of the APSP calculation process will have a considerable time if the usual calculation method using the Floyd-Warshall algorithm is used (the calculation complexity is $O(|V|^3)$) or Johnson algorithm (with complexity $O(|V|^2 log(|V|) + |V||E|)$). Up to now, the most effective algorithm to accurately calculate the BC with all vertices in G is proposed by Brandes i 2001, which has the time complexity of $O(|V|.|E|)$ o unweighted graphs and $O(|V|.|E| + |V|^2.log(|E|))$ on weighted graphs [16].

Brandes designated by $\delta(s,t)$ the number of shortest (s,t)-paths, and by $\sigma(s,t|v)$ the number of shortest (s,t)-paths passing through some vertex v other than s,t. His algorithm is based on the essential idea that the cubic number of *pair-wise dependencies* $\delta(s,t|v) = \frac{\sigma(s,t|v)}{\sigma(s,t)}$ can be aggregated without calculating all of them explicitly. He also denoted the *one-sided dependencies* $\delta(s|v) = \sum_{t\in V} \delta(s,t|v)$ [17]. The pseudo code of the Brandes's algorithm with an unweighted graph is illustrated as in Algorithm 1.

Algorithm 1: Brandes's Algorithm to Compute the Betweenness Centrality

Input: $G = (V, E)$, is organized as a two-dimensional vector $Edges[][]$
Data: an empty queue Q, stack S able to contain $|V|$ vertices ;
$dist[v]$: to save the distance from the source vertex to v ;
$Pred[v]$: to store the list all the vertices on the shortest path from the source vertex to v ;
$\sigma[v]$: the number of shortest paths from the source vertex to v ;
$\delta[v]$: the dependence of the source via/through v ;
Output: $BC[.]$ for any $v \in V$
foreach $s \in V$ **do**
  /* Phase 1. Examine the SSSP problem */
  foreach $v \in V$ **do** $Pred[v] \leftarrow$ empty list ;
  foreach $v \in V$ **do** $dist[v] \leftarrow \infty; \sigma[v] \leftarrow 0$;
  $dist[s] \leftarrow 0; \sigma[s] \leftarrow 1; Q.push(s)$;
  while Q *not empty* **do**
    $v \leftarrow Q.pop(); S.push(v)$;
    foreach $w \in Edges[v]$ **do**
      /* w has not been examined */
      if $dist[w] == \infty$ **then**
        $dist[w] \leftarrow dist[v] + 1; Q.push(w)$;
      /* (v,w) on the shortest path */
      if $dist[w] == dist[v] + 1$ **then** $\sigma[w] \leftarrow \sigma[w] + \sigma[v]; Pred[w].push\_back(v)$;
    end
  end
  /* Phase 2: Accumulation (back-propagation of dependencies) */
  foreach $v \in V$ **do** $\delta[v] \leftarrow 0$;
  while S *not empty* **do**
    $w \leftarrow S.pop()$;
    for $v \in Pred[w]$ **do**
      $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}.(1 + \delta[w])$;
    if $w \neq s$ **then** $BC[w] \leftarrow BC[w] + \delta[w]$;
  end
end
return $BC[.]$;

In this algorithm, the computation of BC is repeated with all vertices $s \in V$, each time calculating $\delta(s|v)$ (one-sided dependencies) for every $v \in V$ in two phases. The first phase is a SSSP (Single Source Shortest Path) calculating to determine the distance and shortest path from s to the other vertices in V. The second phase is a back-traversing on the approved vertices in the first phase in order to calculate the dependencies of vertices according to the following equation [17], beginning with the furthest vertex from *s:*

$$\delta(s|v) = \sum_{w:\,(v,w)\in E \text{ and } dist(s,w)=dist(s,v)+1} \frac{\delta(s,v)}{\delta(s,w)}.\big(1 + \delta(s,w)\big)$$

Brandes's algorithm execution takes a lot of time if the number of vertices/edges of the graph is too large. For example, with a graph having 100 million vertices and 100 million edges, the Brandes's execution time is measured absolutely complex: about $|V|.|E| = 10^8.10^8 = 10^{16}$ or 10 quadrillion arithmetic operations.

In order to improve the speed of computing the Betweenness Centrality, many solutions have been proposed to compute approximately this metric. For example, research works [18-21], proposed the idea of quickly computing the approximate Betweenness Centrality based on sampling techniques. In addition to the approximation approach, research works [8, 10, 16, 22, 23] applied heuristics in building the topology manipulation, such as pre-computation for 1-degree vertices, graph partitioning, considering 2-degree vertices to re-use the results of the two subtrees of each one to reduce the calculation time of Brandes's accumulative phase.

In addition to the above methods, the parallel computing approach is also applied to improve the performance of BC computation. Recent results on this approach can include parallelizing the calculation of BC in the NetworKit toolkit [24], or the TeexGraph toolkit [25] for analyzing large-scale social networks. These toolkits all use a share-memory based parallel model and use the *OpenMP* library to parallelize the process of computing this BC. In addition, to compute the BC for fast evolving graphs, Jamour *et al.* proposed the *iCENTRAL* incremental algorithm based on decomposing the graph into biconnected components [26].

For static graphs, there has been a lot of researches in the parallel computation of BC using GPU (Graphics Processing Unit) processors. In 2016, Bernaschi and colleagues built the MGBC (Multi-GPU Betweenness Centrality) solution combining the use of multiple GPUs and parallel models using MPI distributed memory to improve the speed of BC computing [27]. Exploiting GPU to parallelize this measure mostly uses the classic algorithm of Brandes [28, 29].

Several solutions have also been proposed for performing the BC computation on distributed high-performance computing systems. For example, GraphLab [30] or Apache Giraph [31] are able to manipulate networks having trillions of edges [32]. These toolkits are primarily designed to analyze networks on a very large scale and must use complex computational infrastructures such as cluster systems or supercomputers [30, 40]. With such computational infrastructure requirements, these solutions are not really effective for the problem of computing the BC with social networks not so big as Facebook.

In this paper, we only focus on improving the performance of computing precisely the BC by structuring graph data appropriately and emphasizing in

parallel techniques for computing the Brandes's algorithm with the shared memory model (combining with GPU will be oriented research in the future).
Because most social networks use the concept of weightless relations, we are only interested in unweighted graphs that can be directed or undirected.

III. A FAST ALGORITHM OF BETWEENNESS CENTRALITY COMPUTATION

Based on the above analysis, we proposed a method to improve the performance of BC computation on unweighted graphs. This method is constituted of the following ideas:
– Use the appropriate data structure to improve the data localization, thereby increasing the hit cache rate in the shared memory model. That empowers us to decrease the reference time of the graph data in the main memory.
– Parallel SSSP computations of Brandes's algorithm for all vertices with the threading programming model using the Intel CilkPlus [33] library.

A. Appropriate Data Structure

Similar to [18], data of large-scale graph $G = (V, E)$ will be organized by the adjacent vertex lists: each vertex is assigned an identifier from 0 to $|V| - 1$. For edge data, sorted vertex vectors will be used to represent the graph edges. From that point, the edge data will be represented in the vectors array $Edges[u]\ \forall u \in V$. This is a data organization method which provides the ability to have the highest hit rate when referring to graph data [35].
The vertices traversed by BFS method will be traced in a vector, where each bit in the vector indicates the traversed or non-traversed state of the vertex corresponding to that bit position.
We structure the queue Q to store both the vertices v that will be traversed and the distance from the source vertex s to the target vertex v. This structure allows us to always get the distance value from s to v in the cache memory when considering vertex v, and the cache hit rate will be increased.

B. Parallel Algorithm to Compute the Betweenness Centrality

As Brandes's algorithm illustrated in Algorithm 1, computing the BC for all vertices is entirely based on the BFS method. To exploit the advantages of multi-core processors as well as multi-processor computing systems, we will compute in parallel the BC of all vertices. Our parallel approach is based on parallel computing BC on different vertices rather than doing parallel BFS from one vertex to all the remaining ones (SSSP). This approach allows SSSP traverse to be performed in each own thread, which helps to improve the hit cache rate by using the adjacent data in the cache.
According to Leist and Gilman (2014) CilkPlus is the best parallel programming paradigm [36]. Moreover, we also conducted evaluation tests with Intel CilkPlus [33], OpenMP [42] and pThread [43] libraries, the results showed that CilkPlus's performance is better than the others. This is the reason we decide to use CilkPlus library to perform the BC computation in parallel.
To implement Brandes's algorithm in parallel, we recognize that its *Phase 1* (SSSP) can be dependently computed for each vertex $v \in V$. However, its *Phase 2* (accumulation), the value $BC[w]$ is calculated gradually over the related vertices. Therefore, Phase 1 can entirely be performed in parallel with our proposed appropriate data structure. Notwithstanding, Phase 2 requires the concurrency control technique to handle the data races when calculating the accumulator variable $BC[w]$ from all related vertices. With Intel CilkPlus runtime system, the data races from updates to an accumulator variable may be able to be performed in parallel by using a **reducer**. Technically, a **reducer** allows to create a private accumulator variable for each thread and to combine the thread-private accumulator variable results in the correct order as the threads finish. Because reducers do not need locks, they can be very efficient [33].
Thence, the parallel computation of BC will be illustrated in Algorithm 2.

Algorithm 2: Parallel Algorithm to Compute the Betweenness Centrality

Input: $G = (V, E)$, is organized as $Edges[][]$ vector field
Data: queue $Q \leftarrow$, stack S create the hollow (empty) and its able to contain $|V|$ vertices ;
$dist[v]$: to save the distance from the source vertex to v ;
$Pred[v]$: to store the list all the vertices on the shortest path from the source vertex to v ;
$\sigma[v]$: the number of shortest paths from the source vertex to v ;
$\delta[v]$: the dependence of the source via/through v ;
$reducerBC[v]$: vector contains the BC values of all vertices v and allows the concurrency update in parallel with CilkPlus library;
Output: $BC[.]$ for any $v \in V$

```
/* Execute in parallel using CilkPlus
   library                                   */
for s = 0 to Edges.size() do
    /* Phase 1. Examine the SSSP
       problem                               */
    foreach v ∈ V do Pred[v] ← empty list ;
    foreach v ∈ V do dist[v] ← ∞; σ[v] ← 0 ;
    dist[s] ← 0; σ[s] ← 1; Q.push(s) ;
    while Q not empty do
        v ← Q.pop(); S.push(v) ;
        foreach w ∈ Edges[v] do
            /* w has not been examined   */
            if dist[w] == ∞ then
              dist[w] ← dist[v] + 1; Q.push(w) ;
            /* (v,w) on the shortest path
               */
            if dist[w] == dist[v] + 1 then σ[w] ←
              σ[w] + σ[v]; Pred[w].push_back(v);
        end
    end
    /* Phase 2: Accumulation
       (back-propagation of
       dependencies)                         */
    foreach v ∈ V do δ[v] ← 0;
    while S not empty do
        w ← S.pop() ;
        for v ∈ Pred[w] do
          δ[v] ← δ[v] + σ[v]/σ[w] .(1 + δ[w]);
        /* to ensure the accuracy of
           concurrency execute               */
        if w ≠ s then
          reducerBC[w] ← reducerBC[w] + δ[w];
    end
end
reducerBC.move_out(BC) ; /* to return the
 results to vector BC */
return BC[.] ;
```

The computational complexity of Algorithm 2 is clearly similar to Algorithm 1: $O(|V| * |E|)$ in the case of $1 - thread$ computation (that means the sequential computation). In the case of computing in $t - threads$ parallel, the computational complexity of Algorithm 2 is proportionally reduced t times and is only $O(\frac{|V|*|E|}{t})$.

IV. EXPERIMENT AND EVALUATION

The experiments were performed in a machine having 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), 128GB for the main memory, CentOS Linux release 7.4.1708, gcc 7.2.0. This computing system was configured with a maximum of 36-threads in parallel without hyperthreading. The dataset we used in our experiment and the obtained results will be detailed in the next sections.

A. Datasets

To evaluate our method for computing the BC of all vertex on a graph, five datasets from the Stanford Large Network Dataset Collection (SNAP) [37] and one from Aminer Datasets for Social Network Analysis [38] are selected to evaluate the results.

– *gemsec-Facebook*: These datasets contain eight networks built to represent blue verified Facebook page networks. These pages are modeled using vertices, while the edges show the links between them. Due to time constraints, we choose only two big datasets in *gemsec-Facebook* for our experiment: Politician and Artist.

– *ego-Facebook*: This dataset is built from the 'friends lists' of Facebook, collected from survey participants using this Facebook app.

– *com-DBLP*: is a dataset that represents the DBLP co-authorship network.

– *com-Youtube*: This dataset is collected from the ground-truth communities in Youtube social network.

– *Flickr*: is a dataset representing a popular photo-sharing network allowing users to upload and share photos.

Among these datasets, *Flickr* is a disconnected graph, and the others are connected graphs. Descriptions of the datasets are shown in Table 1:

Table 1: Detail of the graph datasets.

Dataset	Edges	Nodes	Diameter
ego-Facebook (DS1)	88,234	4,039	8
gemsec-Facebook Politician (DS2)	41,729	5,908	14
gemsec-Facebook Artist (DS3)	819,306	50,515	11
DBLP (DS4)	1,049,866	425,975	23
Flickr (DS5)	2,987,624	1,157,828	24
Youtube (DS6)	9,114,557	214,626	10

B. Results and Evaluation

Based on the previous work on computing the closeness centrality [34], we implemented our solution, called *bigGraph*, in C++ language using the CilkPlus parallel library and published both source codes and test results on the GitHub [41].

To evaluate our solution, two recent network analysis toolkits presented in Section II were chosen to compare the performance with bigGraph: *TeexGraph* and *NetworKit*. We implemented these toolkits and bigGraph in the same platform mentioned above.

To analyze the speedup factor, we conducted the first evaluation of bigGraph solution with the number of parallel threads varied from 1 (corresponding to the sequential execution) to 36 (the maximum) in our computing system. For large-scale graph data sets such as YouTube, DBLP, and Flickr, the execution time to compute BC is generally very high (as illustrated in Table 3). Since then, the comparison between bigGraph and the other two graph analysis toolkits (TeexGraph and NetWorKit) will focus on using the first three datasets: ego-Facebook (DS1); gemsec-Facebook Politician (DS2); and gemsec-Facebook Artist (DS3). Our experimental results are aggregated based on the average execution time of 10 tests run for each solution and shown in Table 2.

Table 2: Time (in seconds) and speedup of bigGraph when computing BC.

Threads	DS1		DS2		DS3	
	Time	Speedup	Time	Speedup	Time	Speedup
1	3.03	1.00	8.20	1.00	1129.47	1.00
2	2.52	1.20	7.44	1.10	832.76	1.36
4	1.51	2.01	4.52	1.82	556.46	2.03
8	0.92	3.29	2.61	3.15	330.64	3.42
16	0.54	5.62	1.60	5.14	196.46	5.75
32	0.28	10.79	0.89	9.19	118.92	9.50
36	0.23	13.26	0.74	11.01	99.85	11.31

Fig. 1 clearly shows the change of the bigGraph's speedup factor when the number of parallel threads changes:

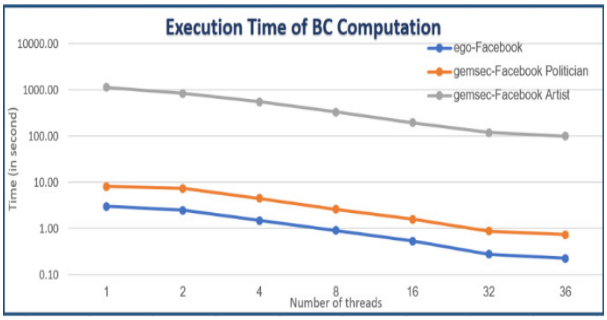

Fig. 1. Execution time to compute BC.

As illustrated in Fig. 2, the more parallel threads we can perform, the shorter execution time for computing BC. So, we decided to choose the number of 36 parallel threads (the maximum number of threads on our computing system) when evaluating the performance between bigGraph and NetWorKit, TeexGraph.

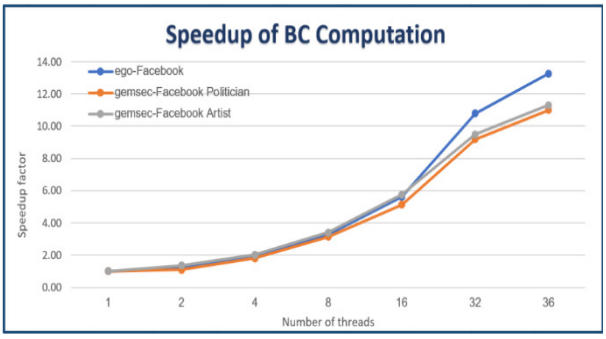

Fig. 2. Speed up Factor when Computing BC of bigGraph.

Table 3 below illustrates the average time of 10 executions for computing BC of all three solutions mentioned above:

Table 3: Time (in seconds) for computing BC.

Dataset	bigGraph	TeeGraph	NetworKit
ego-Facebook	**0.23**	0.31	0.56
gemsec-Facebook Politician	**0.84**	0.84	1.70
gemsec-Facebook Artist	**99.85**	110.58	234.12
DBLP	**2,345.62**	2,694.78	4,823.47
Flickr	**3,506.34**	4,447.93	7,694.61
Youtube	**56,071.60**	68,744.80	90,522.30

The experimental results in Table 3 demonstrate that our proposed algorithm for computing the BC has a shorter execution time than the two other solutions of TeexGraph and NetworKit. Detailed illustration of the execution time of all three toolkits is shown in Fig. 3 below:

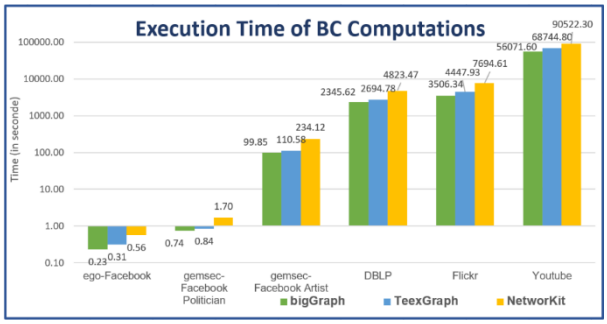

Fig. 3. Execution time to compute BC for different toolkits

Besides, we also compared and evaluated the values of BC obtained from all three solutions. The results showed that all three solutions returned the same BC values with all six data sets mentioned above.

The obtained experimental results allow us to confirm our parallel solution to compute the BC of all vertices in *bigGraph* with better performance than the other graph analysis toolkits. Table 4 shows that the *bigGraph*'s speedup factor for all six datasets is faster from 1.11 to 1.35 and from 2.06 to 2.44 times compared to TeexGraph and NetworKit, respectively.

Table 4: Speedup of BigGraph in Comparison with TeexGraph and NetworKit.

Dataset	TeeGraph/ bigGraph	NetworKit / bigGraph
ego-Facebook	1.35	2.44
gemsec-Facebook Politician	1.13	2.28
gemsec-Facebook Artist	1.11	2.34
DBLP	1.15	2.06
Flickr	1.27	1.61
Youtube	1.23	2.19

Thus, for all six real networks, our solution *bigGraph* enhances the performance and computes the BC for all nodes in the shortest time.

V. CONCLUSION

In this paper, we have focused on improving the efficiency of the betweenness centrality computing for all vertices in large-scale graphs. Based on the Brandes's faster algorithm to compute the BC (the best algorithm today to compute BC with the computational complexity $O(|V| * (|V| + |E|))$), we proposed an efficient *bigGraph* solution based on (i) organizing the appropriate graph data that reducing amount of time accessing the main memory, and (ii) paralleling Brandes's faster algorithm using the CilkPlus library to perform in parallel the SSSP for all vertices. The proposed algorithm has a complexity of $O(\frac{|V|*(|V|+|E|)}{t}$, where t is the number of threads that can be executed in parallel.

Our experiments with six different real networks (provided by SNAP and Aminers) allow us to confirm that *bigGraph* is the most efficient in comparison with other modern graph analysis toolkits such as TeexGraph and NetworKit: bigGraph is faster than TeexGraph and NetworKit from 1.11 to 1.35 and from 2.06 to 2.44 times, respectively. The speedup of *bigGraph* is also increased proportionally with the number of threads in parallel.

VI. FUTURE SCOPE

For future work, we aim to extend our method for performing more complex graph analysis such as consumer behavior analysis in social commerce; detecting non-signature malware; finding the most influential user on the social media; controlling and forecasting the news spreading across the social networks.

ACKNOWLEDGEMENTS

This research was funded by Ministry of Science and Technology of Vietnam, grant number KC.01.19/16-20.

Conflict of Interest. The authors declare no conflict of interest.

REFERENCES

[1]. Statista (2019). Most famous social network sites worldwide as of april 2019, ranked by number of active users, https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[2]. Mikalef, P., Giannakos, M., & Pateli, A. (2013). Shopping and word-of-mouth intentions on social media. *Journal of Theoretical and Applied Electronic Commerce Research*, *8*(1), 17–34.

[3]. Zhang , K. Z., & Benyoucef, M. (2016). Consumer behavior in social commerce: A literature review. *DecisionSupport Systems, 86*, 95-108.

[4]. Hung, H. J., Shuai, H. H., Yang, D. N., Huang, L. H., Lee, W. C., Pei, J., & Chen, M. S. (2016). When social influence meets item inference, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, 915-924.

[5]. Otte, E., & Rousseau, R. (2002). Social network analysis: a powerful strategy, also for the information sciences, *Journal of Information Science*, *28*(6), 441-453.

[6]. Louni, A., & Subbalakshmi, K. P. (2018). Who spread that rumor: Finding the source of information in large online social networks with probabilistically varying internode relationship strengths, *IEEE Transactions on Computational Social Systems*, *5*(2), 335-343.

[7]. Farooq, A., Joyia, G. J., Uzair, M., & Akram, U. (2018). Detection of influential nodes using social networks analysis based on network metrics, *in2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 1-6.
[8]. Puzis, R., Elovici, Y., Zilberman, P., Dolev, S., & Brandes, U. (2015). Topology manipulations for speeding betweenness centrality computation, *Journal of Complex Networks*, *3*(1), 84-112.
[9]. Freeman, L. C. (1977). A set of measures of centrality based on betweenness, *Sociometry*, *40*(1), 35-41.
[10]. Sariyüce, A. E., Kaya, K., Saule, E., & Çatalyürek, U. V. (2017). Graph manipulations for fast centrality computation, *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *11*(3), 26:1–26:25.
[11]. Kim, J., & Lee, J. G. (2015). Community detection in multi-layer graphs: A survey, *ACM SIGMOD Record*, *44*(3), 37-48.
[12]. Yang, J., & Chen, Y. (2011). Fast computing betweenness centrality with virtual nodes on large sparse networks, *PLOS ONE*, *6*(7), 1-5.
[13]. Bullmore, E., & Sporns, O. (2009). Complex brain networks: graph theoretical analysis of structural and functional systems,*Nature Reviews Neuroscience*, *10*, 186-198.
[14]. Chikhaoui, B., Chiazzaro, M., Wang, S., & Sotir, M. (2017). Detecting communities of authority and analyzing their influence in dynamic social networks, *ACM Transactions on Intelligent Systems and Technology*, *8*(6), 82:1–82:28.
[15]. Kaya, M., Kawash, J., Khoury, S., & Day, M. Y. (2018). Social Network Based Big Data Analysis and Applications, 1st edition, *Lecture Notes in Social Networks, Springer International Publishing*. [16]. Vella, F., Bernaschi, M., & Carbone, G. (2018). Dynamic merging of frontiers for accelerating the evaluation of betweenness centrality,*Journal of Experimental Algorithmics*, *23*, 1.4:1–1.4:19.
[17]. Brandes, U. (2001). A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, *25*(2), 163-177.
[18]. Eppstein, D., & Wang, J. (2004). Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, *8*(1), 39-45.
[19]. Chehreghani, M. H. (2014). An efficient algorithm for approximate betweenness centrality computation, *The Computer Journal*, *57*(9), 1371-1382.
[20]. Riondato, M., & Kornaropoulos, E. M. (2016). Fast approximation of betweenness centrality through sampling, *Data Mining and Knowledge Discovery, 30*, 438-475.
[21]. Mahmoody, A., Tsourakakis, C. E., & Upfal, E. (2016). Scalable betweenness centrality maximization via sampling, *in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* New York, NY, USA: ACM, 1765-1773.
[22]. Baglioni, M., Geraci, F., Pellegrini, M., & Lastres, E. (2012). Fast exact computation of betweenness centrality in social networks," *in 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM),* 450-456.
[23]. Bentert, M., Dittmann, A., Kellerhals, L., Nichterlein, A., & Niedermeier, R. (2018). An adaptive version of brandes' algorithm for betweenness centrality, *Computing Research Repository-CoRR*, abs/1802.06701.
[24]. Staudt, C. L., SAZOnOvS, A. & Meyerhenke, H. (2016). Networkit: A tool suite for large-scale complex network analysis. *Network Science*, *4*(4), 508-530.
[25]. Takes, F. W., & Heemskerk, E. M. (2016). Centrality in the global network of corporate control. *Social Network Analysis and Mining*, *6*(1), 1-18.
[26]. Jamour, F., Skiadopoulos, S., & Kalnis, P. (2018). Parallel algorithm for incremental betweenness centrality on large graphs. *IEEE Transactions on Parallel and Distributed Systems*, 29(3), 659-672.
[27]. Bernaschi, M., Carbone, G., & Vella, F. (2016). Scalable betweenness centrality on multi-gpu systems, *in Proceedings of the ACM International Conference on Computing Frontiers*, USA: ACM, 29-36.
[28]. Fan, R., Xu, K., & Zhao, J. (2017). A GPU-based solution for fast calculation of the betweenness centrality in large weighted networks. *Peer J Computer Science*, *3*, e140.
[29]. McLaughlin. A., & Bader, D. A. (2018). Accelerating GPU betweenness centrality. *Communications of the ACM*, *61*(8), 85-92.
[30]. Wei, J., Chen, K., Zhou, Y., Zhou, Q., & He, J. (2016). Benchmarking of distributed computing engines spark and GraphLab for big data analytics. *In 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, 10-13.
[31]. Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., & Muthukrishnan, S. (2015). One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment, 8*(12), 1804-1815.
[32]. Avery Ching, Nitay Joffe, Maja Kabiljo, Greg Malewicz, Ravi Murthy, and Alessandro Presta, (2013). Scaling apache giraph to a trillion edges, https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920, (accessed October, 15 2019)
[33]. Intel, (2020). Cilk plus programming, *https://www.cilkplus.org/cilk-documentation-full*, (accessed January, 20 2020).
[34]. Du, P. H., Nguyen, H. C., Nguyen, K. K., & Nguyen, N. H. (2018). An efficient parallel algorithm for computing the closeness centrality in social networks. *In the Ninth International Symposium on Information and Communication Technology, ACM, New York, NY, USA*, 1-6.
[35]. Du, P. H., Pham, H. D., & Nguyen, N. H. (2018). An efficient parallel method for optimizing concurrent operations on social networks," *Transactions on Computational Collective Intelligence*, LNCS (Scimago Q2), *10840* (29), 182-199.
[36]. Leist, A., & Gilman, A. (2014). A comparative analysis of parallel programming models for C++, *in The Ninth International Multi-Conference on Computing in the Global Information Technology,* ICCGI 2014, 121-127.
[37]. Leskovec, J., & Krevl, A. (2019). SNAP Datasets: Stanford large network dataset collection, *http://snap.stanford.edu/data*, (accessed November, 10, 2019).
[38]. Zhang, Y., Tang, J., Yang, Z., Pei, J., & Yu, P. S. (2015). Cosnet: Connecting heterogeneous social networks with local and global consistency. *In*

Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD '15. New York, NY, USA: ACM, 1485-1494.
[39]. Nasiruzzaman, A. B. M., & Pota, H. R. (2013). Complex Network Framework Based Comparative Study of Power Grid Centrality Measures, *International Journal of Electrical and Computer Engineering (IJECE)*, *3*(4), 543-552.
[40]. Daniel, C., Furno, A., & Zimeo, E. (2019). Cluster-based Computation of Exact Betweenness Centrality in Large Undirected Graphs, *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 603-608.
[41]. Du, P. H. Source code of bigGraph, https://github.com/hanhdp/parallel_betweenness_centrality/
[42]. OpenMP, http://openmp.org/wp/, (accessed September, 10 2019).
[43]. Pthread Programming, https://computing.llnl.gov/tutorials/pthreads/, (accessed September, 10 2019).

How to cite this article: Du, P. H., Duong, N. S., Nguyen, N. C. and Nguyen, N. H. (2020). A Fast Computation of Betweenness Centrality in Large-Scale Unweighted Graphs. *International Journal on Emerging Technologies*, *11*(2): 370–377.