# An Efficient Parallel Algorithm for Computing the Closeness Centrality in Social Networks

Phuong Hanh DU
Department of Information Systems, VNU University of Engineering and Technology
Hanoi, Vietnam
hanhdp@vnu.edu.vn

Hai Chau NGUYEN
Department of Information Systems, VNU University of Engineering and Technology
Hanoi, Vietnam
chaunh@vnu.edu.vn

Kim Khoa NGUYEN
Synchromedia Laboratory, Ecole Superieure de Technologie
Montreal, QC, Canada
Kim-Khoa.Nguyen@etsmtl.ca

Ngoc Hoa NGUYEN
Department of Information Systems, VNU University of Engineering and Technology
Hanoi, Vietnam
Ngoc-Hoa.Nguyen@vnu.edu.vn

## ABSTRACT

Closeness centrality is an substantial metric used in large-scale network analysis, in particular social networks. Determining closeness centrality from a vertex to all other vertices in the graph is a high complexity problem. Prior work has a strong focuses on the algorithmic aspect of the problem, and little attention has been paid to the definition of the data structure supporting the implementation of the algorithm. Thus, we present in this paper an efficient algorithm to compute the closeness centrality of all nodes in a social network. Our algorithm is based on (i) an appropriate data structure for increasing the cache hit rate, and then reducing amount of time accessing the main memory for the graph data, and (ii) an efficient and parallel complete BFS search to reduce the execution time. We tested performance of our algorithm, namely BigGraph, with five different real-world social networks and compare the performance to that of current approaches including TeexGraph and NetworKit. Experiment results show that BigGraph is faster than TeexGraph and NetworKit 1.27-2.12 and 14.78-68.21 times, respectively.

## CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; **Massively parallel algorithms**;

## KEYWORDS

Closeness Centrality, Breadth-First Search, Social Network Analysis, Multi-threaded Parallel Computing

**ACM Reference Format:**
Phuong Hanh DU, Hai Chau NGUYEN, Kim Khoa NGUYEN, and Ngoc Hoa NGUYEN. 2018. An Efficient Parallel Algorithm for Computing the Closeness Centrality in Social Networks. In *Proceedings of ACM 9th International Symposium on Information and Communication Technology (SoICT'2018).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/_4


*SoICT'2018, December 2018, Danang, Vietnam*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/_4

## 1 INTRODUCTION

Social networks are omnipresent for all country and they become a significant way in order to connect people in our networked society. Facebook, Twitter, YouTube and WhatsApp are notable ones in our modern life. As the statistic provided by The Statistics Portal in July 2018, the number of active users of Facebook is 2.196 billions; Youtube is 1.9 billion and WhatsApp surpassed 1.5 billion [19].

In the trend of developing e-government, management and exploitation of social networks is an important task that enables the promotion of citizen participation in government. In addition, social networks can be seen as the effective means of interacting between citizens and state agencies [10],[4].

Graph theory has been considered as a proper methodology for modeling social networks. A member of a social network is generally modeled by a vertex, and the direct relationship between two members is represented by an edge. In order to manage the social network, many social network analysis (SNA) methods have been proposed and exploited in practice. SNA is defined as the process of investigating social structures through the use of networks and graph theory [15]. Thus, it is now considered as a key technique in modern sociology.

One of the most important things that we have considered to perform a network analysis is determining the centrality of a node within a social network. In other words, for a SNA, we should figure out which node has the most effect on the others [14]. Thus, the centrality of a node allows us to identify the most important users within a network [5]. One of the most widely used indicators is *closeness centrality* and we focus only this indicator in our work.

Computing the closeness centrality of a node in a social network requires solving the all pairs shortest path problem. Thus, it needs to perform a complete breadth-first search (BFS) for an unweighted network or a complete run of Dijkstra's algorithm for a weighted network. The computational effort for this task is often impractical for very large real-world social networks [16].

In this paper, we propose a method to improve the performance of computing the closeness centrality indicator on unweighted social networks. To gain this purpose, we propose an appropriate data structure for modeling the network and a strategy to parallelize the execution of complete BFS search.

The rest of this paper is organized as follows. Section 2 presents preliminaries and related work. Section 3 details our efficient method for improving the performance of both updating and computing operations. In Section 4, we summarize our experiments to verify and benchmark our approach. Finally, the last section provides some conclusions and future works.

## 2 PRELIMINARIES AND RELATED WORK

### 2.1 Notations

In this article, we focus only on undirected and unweighted social networks. A undirected and unweighted network can be represented as a graph $G(V, E)$ where $V$ is the set of all members (vertices) and $E = \{(v_i, v_j)|v_i, v_j \in V\}$ represents the set of all relationships (edges) ($v_i$ and $v_j$ are connected with a single unweighted link). Note that in such graph, $(v_i, v_j) \equiv (v_j, v_i)$. The total number of edges to (incoming) and from (outgoing) a vertex $v_i$ is called the degree of $v_i$ 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 $G$ 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 $G$. If $u$ and $v$ are identical then $dst(u, v) = 0$. Moreover, if $u$ and $v$ are disconnected then $dst(u, v) = \infty$.

In social network analysis, the centrality of a node allows identifying the most important users within a network. Centrality concepts are also applied in other problems such as key infrastructure nodes in the Internet and super-spreaders of disease. There are four indicators of centrality defined as follows:

*Definition 2.1.* **Degree Centrality** is defined as the number of links incident upon a node. It is measured by the following formula:

$$CD(v) = deg(v) : v \in V \tag{1}$$

*Definition 2.2.* **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 other nodes. Closeness Centrality is computed by the following formula:

$$CC(v) = \frac{1}{\sum_u dst(u, v)} : n \in V, \tag{2}$$

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

In this paper, in order to avoid the value $\infty$ when compute the shortest distance of a disconnected graph $G$, we will compute the $CC$ of a node $v$ for the largest-component $\Gamma_G$ of $G$. Moreover, if a node $u$ cannot reach any other node in $G$, then $CC(u) = 0$.

*Definition 2.3.* **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 [6]. In his conception, vertices that have a high probability to occur on a randomly chosen shortest path between two randomly chosen vertices have a high betweenness. Betweenness Centrality is computed by the following formula:

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

where $\sigma_{st}$ is total number of shortest distances from node $s$ to node $t$ and $\sigma_{st}(v)$ is the number of those paths that pass through $v$.

*Definition 2.4.* **Eigenvector Centrality** is an indicator to measure the influence of a respective node in a social network. This indicator allows to assign a relative score to all influence nodes based on the concept of connection to high scoring participating nodes whose contribution is more to the score of the node in question than equality [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 $G$: $a_{u,v} = 1$ if node $u$ is linked to node $v$ and $a_{u,v} = 0$ otherwise. The Eigenvector Centrality $x$ of node $v$ can be defined as:

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

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

### 2.2 Related Work

Closeness is a *traditional* definition of centrality, and consequently it was not *designed with scalability in mind* [9]. Moreover, computing the closeness centrality in large-scale networks is incapable due to the computational complexity [2]. One of the simplest solutions considered was to define different measures that might be related to closeness centrality [9].

Parallelization of Algorithm 1 is one of the most effective ways to improve the performance of CC computation in a real-world social networks. This approach exploits the multicore/multichip computers and presented in a lot of works [1],[8], [20],[21]. However, these works were not considered the memory hierarchic organization in computer: if we have a good data structure, we can reduce the cache misss rate and increase the cache hit rate. Thus, due of the CPU cache organization, when a process needs to handle a big data, the consecutive item list is the best way to allow having the highest cache hit rate [3].

There are tools and libraries that can feasibly be used to perform the manipulation on the social networks. NetworkX, for instance, is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks and graph [8]. SNAP C++ library [13] is very popular for a general purpose, high-performance system for analysis and manipulation of large networks. These tools also support methods to compute the closeness centrality indicator. However, they are not optimized in order to exploit the multicore/multichip offers in the current computer architecture.

For processing large-scale graphs in distributed and parallel computation, GraphLab [22] and PowerGraph [7] are remarkable systems. They are efficient for general purposes in case of having a dominant computing platform such as clusters and supercomputers [22]. Nevertheless, they are not adequate for the closeness centrality

computation for the real-world networks in the context of medium computing platforms, similar to NetworkX and SNAP C++.

NetworKit [20], TeexGraph [21] and GraphLab [22] are notable for processing the large social networks in parallel computation. We will use these tools to evaluate and analyze our solution and compare to them.

# 3 A FAST ALGORITHM OF CLOSENESS CENTRALITY COMPUTATION

## 3.1 Overview

Since the major of real-world social networks have the mutual unweighted relationship between two members, we focus only on the Closeness Centrality indicator $CC$ in a unweighted and undirected real-world social network $G$.

The pseudo-code is described in Algorithm 1. The later uses the breadth-first search (BFS) from each node $v$ of $V$ and accumulates to computed $CC[v]$.

**Algorithm 1:** Basic Closeness Centrality Computation

```
Data: G = (V, E)
Result: CC[.] for all v ∈ V
CC[v] ← 0, ∀v ∈ V ;
Sum[v] ← 0, ∀v ∈ V ;
foreach s ∈ V do
    FC[s] ← 0;
    Q ← empty queue;
    Q.push(s);
    dst[s] ← 0;
    CC[s] ← 0;
    dst[v] ← −1, ∀v ∈ V s;
    while Q is not empty do
        v ← Q.pop();
        forall w ∈ Γ_G(v) do
            if dis[w] = ∞ then
                Q.push(w);
                dst[w] ← dst[v] + 1;
                Sum[v] ← dst[w];
            end
        end
    end
    CC[s] ← 1/Sum[s] ;
end
returnCC[.];
```

The complexity of Algorithm 1 is $O(|V| * (|V| + |E|))$. For the large networks such a Facebook, Youtube, the execution time to compute the closeness centrality for all nodes is also very high: for a small dataset collected from ground-truth communities of Youtube, computing the closeness centrality for all 1,134,890 nodes, 2,987,624 edges consumes 147924.4 seconds (see Table 1).

Our solution to compute the closeness centrality is based on both (i) an appropriate data structure for increasing the cache hit rate and reducing amount of time accessing the main memory for the graph data, and (ii) parallelization of the closeness centrality computation in order to exploit all capability of CPU.

## 3.2 Appropriate Data Structure

We encode the vertices from 0 to $|V| - 1$. For the graph edges, there are three main structure types: (i) edge lists, (ii) adjacency matrices and (iii) adjacency lists. In large scale graphs, the adjacency matrices representation cannot be used because of the limit of main memory size. The edge list structure is simple, but the operations on the graph, such as insertion and deletion, are difficult. The appropriate way to represent the large-scale edges of the graph is the adjacency list structure [3].

For managing big data, the consecutive item list is the best way to achieve the highest cache hit rate [17]. Moreover, in this research, we mainly examine large social networks which have no more than four billion members. Therefore, each member is identified by a 32-bit integer.

From the above ideas, the graph data is represented by the adjacency lists described as follows: (i) each node/vertice is represented by a 4-byte integer; (ii) all outgoing nodes of a node $u$ are stored in a sorted vector. Thus, a graph can be represented by a vector arrays $Edges[u] \forall u \in V$.

## 3.3 Efficient Parallel Algorithm

To perform the BFS search from a node $u$, we use a bitmap array, namely $Maps$, for remarking traveled nodes. We also use a specific queue in order to store also the distance from current node to the node in queue.

To exploit profit of multicore/multichip CPUs, the computation of closeness centrality will be executed in parallel. We will use global queues and maps pre-allocated for all threads in the computing system. Cilk Plus is used for performing queries in parallel[1]. We implemented our solution based on multi-threaded programming paradigms including OpenMP[2], Pthread[3] and note that the Cilk Plus is the most efficient one and achieve outstanding performance.

Our new proposed algorithm is presented in Algorithm 2. The complexity of this algorithm is also $O(|V| * (|V| + |E|))$ as the basic closeness centrality computation.

# 4 EXPERIMENT AND EVALUATION

In this section, we perform different tests of our algorithm on several real social networks. All the networks data in our tests is collected from the SNAP (https://snap.stanford.edu/data/index.html) and the AMiner (https://aminer.org/data-sna).

Based on the proposed method, we built and implemented our solution, namely **BigGraph**, in C++ language. 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 maximum 36-threads in parallel without hyperthreading.

[1] https://www.cilkplus.org/cilk-documentation-full
[2] http://openmp.org/wp/
[3] https://computing.llnl.gov/tutorials/pthreads/

**Algorithm 2:** Fast Closeness Centrality Computation

```
Data: G = (V, E) represented by Edges
Result: CC[.] for all v ∈ V
CC[v] ← 0, Sum[v] ← 0, Maps[v] ← 0∀v ∈ V ;
// Perform in parallel the queries by Cilk Plus method
for s = 1 to Edges.size() do
    Q ← empty queue; Q.push(s);
    // We mark s was visited in Maps buffer
    SetBit(s, Maps);
    CC[s] ← 0; FC[s] ← 0; dst ← 0;
    while Q is not empty do
        dst ← dst + 1;
        // We scan all nodes moved in the queue Q in same
          level/distance from s
        while Q is not empty do
            v ← Q.pop();
            // We scan all nodes connected directly to s
            forall w ∈ Edges[s] do
                // if this node is not visited
                if !TestBit(w) then
                    //we save also the distance from v to w
                    Q.push(w, dst);
                    // we mark w was visited
                    SetBit(w, Maps);
                end
            end
        end
        // We move to the next level
        Q.nextLevel() ;
    end
    if Sum[s] ≠ 0 then
        CC[s] ← 1/Sum[s] ;
    end
end
returnCC[.];
```

## 4.1 Datasets

To validate our method for computing the closeness centrality on a network, five datasets from the Stanford Large Network Dataset Collection [12] and one from Aminer Datasets for Social Network Analysis [23] are selected to evaluate the results.

- *gemsec-Facebook*: These datasets contain eight networks built to represent blue verified Facebook page networks. Facebook pages those are represented by nodes and edges are mutual likes among them. Due to time constraint, we choose only two big dataset in *gemsec-Facebook* for our experiment: Potician and Artist.
- *ego-Facebook*: This dataset is built from the 'friends lists' of Facebook, collected from survey participants using this Facebook app.
- *com-DBLP*: This dataset represent the DBLP co-authorship network.
- *com-Youtube*: This dataset is collected from the ground-truth communities in Youtube social network.
- *Flickr*: This dataset represents 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 showed in Table 1:

**Table 1: Graph Collection Statistics**

| Dataset | Edges | Nodes | Diameter |
|---|---|---|---|
| gemsec-Facebook Politician | 41,729 | 5,908 | 14 |
| ego-Facebook | 88,234 | 4,039 | 8 |
| gemsec-Facebook Artist | 819,306 | 50,515 | 11 |
| DBLP | 1,049,866 | 317,080 | 23 |
| Youtube | 2,987,624 | 1,134,890 | 24 |
| Flickr | 9,114,557 | 215,495 | 10 |

## 4.2 Results and Evaluation

Based on work of P. H. Du et al. [18], we implemeted our method in C++ language using the Cilk Plus parallel library and published both source codes and test results on the GitHub at: https://github.com/hanhdp/parallel_closeness_centrality/.

To evaluate our solution, several recent network analysis tools presented at Section 2 were chosen to compare the performance with BigGraph: TeexGraph and NetworKit. We implemented these tools and BigGraph in the platform mentioned above.

To analyze the parallel speed up, we firstly evaluate our solution BigGraph, with different number of parallel threads varied from 1 to maximum number of threads 36-threads in our testing machine. For each dataset, we perform computing the closeness centrality 10-times. For the big datasets such as Youtube, DBLP and Flickr, their execution times for computing the closeness centrality are very high (as illustrated by the Table 3). Thus, we focus on the first three datasets: gemsec-Facebook Politician named DS1, ego-Facebook named DS2 and gemsec-Facebook Artist named DS3. The experiment results we obtained are synthesis by computing the average of testing execution times and illustrated by the following table:

**Table 2: Time (in second) and Speedup of BigGraph**

| NumOfThread | DS1 | DS1 Speedup | DS2 | DS2 Speedup | DS3 | DS3 Speedup |
|---|---|---|---|---|---|---|
| 1 | 1.546 | 1.00 | 1.031 | 1.00 | 195.276 | 1.00 |
| 2 | 0.819 | 1.89 | 0.552 | 1.87 | 97.527 | 2.00 |
| 4 | 0.415 | 3.72 | 0.306 | 3.37 | 49.136 | 3.97 |
| 6 | 0.285 | 5.43 | 0.223 | 4.63 | 34.475 | 5.66 |
| 8 | 0.223 | 6.95 | 0.169 | 6.11 | 26.418 | 7.39 |
| 10 | 0.181 | 8.56 | 0.130 | 7.93 | 21.406 | 9.12 |
| 12 | 0.160 | 9.66 | 0.120 | 8.57 | 18.716 | 10.43 |
| 14 | 0.155 | 9.98 | 0.100 | 10.33 | 16.731 | 11.67 |
| 16 | 0.129 | 12.03 | 0.095 | 10.82 | 14.491 | 13.48 |
| 18 | 0.111 | 13.88 | 0.086 | 11.95 | 13.263 | 14.72 |
| 20 | 0.101 | 15.33 | 0.074 | 13.92 | 11.784 | 16.57 |
| 22 | 0.091 | 16.97 | 0.067 | 15.40 | 10.856 | 17.99 |
| 24 | 0.088 | 17.52 | 0.062 | 16.54 | 9.884 | 19.76 |
| 26 | 0.084 | 18.37 | 0.061 | 16.79 | 9.116 | 21.42 |
| 28 | 0.077 | 20.03 | 0.057 | 18.16 | 8.775 | 22.25 |
| 30 | 0.075 | 20.61 | 0.054 | 18.94 | 8.063 | 24.22 |
| 32 | 0.069 | 22.48 | 0.054 | 19.07 | 7.481 | 26.10 |
| 34 | 0.066 | 23.43 | 0.053 | 19.45 | 7.041 | 27.74 |
| 36 | 0.060 | 25.96 | 0.052 | 19.89 | 6.648 | 29.37 |

The following figure shows more clearly the speedup of BigGraph as the number of parallel threads changes.

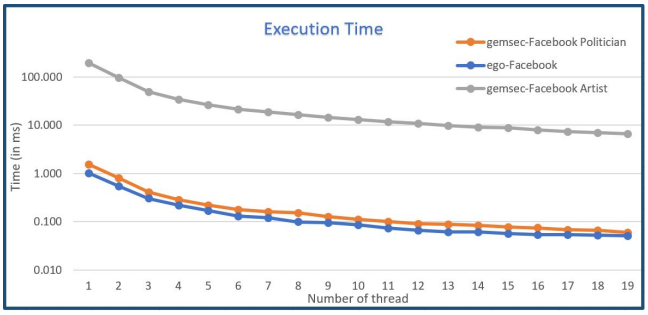


**Figure 1: BigGraph Execution Time (in second)**

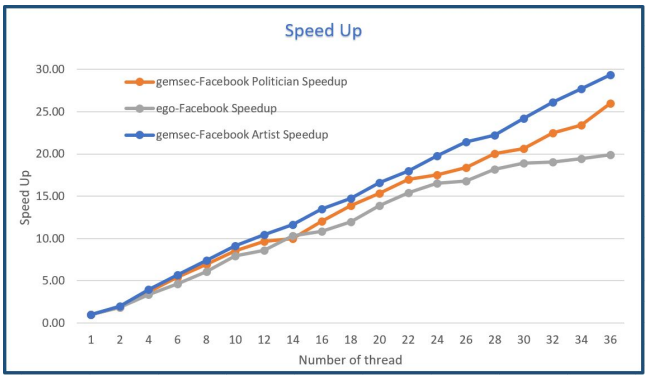


**Figure 2: BigGraph Parellel Speedup Evaluation**

As illustrated in Figure 2, the more parallel threads we use, the shorter computation time of closeness centrality is. Therefore, we decided to set to 36-threads in parallel for all three tools: NetworKit, TeexGraph and BigGraph.

The following table illustrates the execution times we obtained for all three tools. Note that they are the average runtime of 10 different tests.

**Table 3: Execution Time (in second)**

| Dataset | Networkit | Teexgraph | BigGraph |
|---|---|---|---|
| gemsec-Facebook Politician | 1.192 | 0.071 | **0.056** |
| ego-Facebook | 0.468 | 0.052 | **0.032** |
| gemsec-Facebook Artist | 182.890 | 9.808 | **6.405** |
| DBLP | 3363.286 | 326.659 | **153.753** |
| Youtube | 147924.400 | 4418.191 | **2168.677** |
| Flickr | | 540.944 | **309.058** |

In this table, due of Flickr is disconnected, NetworKit cannot compute the closeness centrality. Other tools can exactly perform computing the closeness centrality for all datasets.

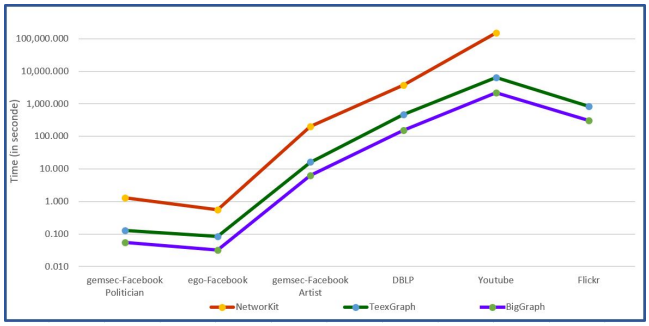


**Figure 3: Experiment Runtime**

The results obtained from experiment allow to validate our solution of computing the closeness centrality in a social network. Its performance is outstanding in comparison with the others. Table 4 illustrates the speedup factor between BigGraph and the others tools for all 5 datasets: BigGraph is faster than TeexGraph and NetworKit from 1.27-2.12 and 14.78-68.21 times.

**Table 4: BigGraph Speedup**

| Dataset | Teexgraph/BigGraph | Networkit/BigGraph |
|---|---|---|
| gemsec-Facebook Politician | 1.27 | 21.23 |
| ego-Facebook | 1.66 | 14.78 |
| gemsec-Facebook Artist | 1.53 | 28.56 |
| DBLP | 2.12 | 21.87 |
| Youtube | 2.04 | 68.21 |
| Flickr | 1.75 | |

For all datasets, the BigGraph solution performs the closeness centrality in shortest time. Moreover, based on the appropriate data structure (for reducing amount of time accessing the main memory for the graph data by increasing the cache hit rate), the method for parallelizing BFS algorithm, the performance of BigGraph is clearly improved compared to both TeexGraph and NetworKit.

## 5 CONCLUSION

Computing the closeness centrality for all node in a real-world social network have been a huge challenge today. We proposed in this paper an efficient algorithm with (i) the appropriate data structure for reducing amount of time accessing the main memory for the graph data by increasing the cache hit rate, (ii) optimization and parallelization of complete BFS search to reduce the execution time. The experiment results confirmed that BigGraph is the most efficient tool in comparison with other social network analysis libraries such as TeexGraph and NetworKit. It obtained the good performance in comparison with the two libraries for computing the closeness centrality of five different network datasets:BigGraph is faster than TeexGraph and NetworKit from 1.27-2.12 and 14.78-68.21 times. The execution time is also reduced proportionally with the number of real parallel threads.

For future works, we aim to extend our method for performing more complex operations on social networks such as computing the others indicators of centrality like node Betweenness, Eigenvector Centrality.

## ACKNOWLEDGMENTS

This work is partially supported by the national research project No. KC.01/16-20, granted by the Ministry of Science and Technology of Vietnam (MOST).

## REFERENCES

[1] V. T. Chakaravarthy, F. Checconi, F. Petrini, and Y. Sabharwal. 2014. Scalable Single Source Shortest Path Algorithms for Massively Parallel Systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 889–901. https://doi.org/10.1109/IPDPS.2014.96

[2] Duanbing Chen, Linyuan LÃij, Ming-Sheng Shang, Yi-Cheng Zhang, and Tao Zhou. 2012. Identifying influential nodes in complex networks. *Physica A: Statistical Mechanics and its Applications* 391, 4 (2012), 1777 – 1787. https://doi.org/10.1016/j.physa.2011.09.017

[3] Phuong-Hanh DU, Hai-Dang PHAM, and Ngoc-Hoa NGUYEN. 2016. Optimizing the Shortest Path Query on Large-scale Dynamic Directed Graph. In *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT '16)*. ACM, New York, NY, USA, 210–216. https://doi.org/10.1145/3006299.3006321

[4] Yogesh K. Dwivedi, Nripendra P. Rana, Mina Tajvidi, Banita Lal, G. P. Sahu, and Ashish Gupta. 2017. Exploring the Role of Social Media in e-Government: An Analysis of Emerging Literature. In *Proceedings of the 10th International Conference on Theory and Practice of Electronic Governance (ICEGOV '17)*. ACM, New York, NY, USA, 97–106. https://doi.org/10.1145/3047273.3047374

[5] A. Farooq, G. J. Joyia, M. Uzair, and U. Akram. 2018. Detection of influential nodes using social networks analysis based on network metrics. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 1–6. https://doi.org/10.1109/ICOMET.2018.8346372

[6] Linton C. Freeman. 1977. A Set of Measures of Centrality Based on Betweenness. *Sociometry* 40, 1 (1977), 35–41. http://www.jstor.org/stable/3033543

[7] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. USENIX Association, Berkeley, CA, USA, 17–30. http://dl.acm.org/citation.cfm?id=2387880.2387883

[8] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[9] U. Kang, Spiros Papadimitriou, Jimeng Sun, and Hanghang Tong. 2011. Centralities in Large Networks: Algorithms and Observations. In *SDM*.

[10] R. T. Khasawneh and M. M. Tarawneh. 2016. Citizens' attitudes towards e-government presence on social networks (e-government 2.0): An empirical study. In *2016 7th International Conference on Information and Communication Systems (ICICS)*. 45–49. https://doi.org/10.1109/IACS.2016.7476084

[11] Jungeun Kim and Jae-Gil Lee. 2015. Community Detection in Multi-Layer Graphs: A Survey. *SIGMOD Rec.* 44, 3 (Dec. 2015), 37–48. https://doi.org/10.1145/2854006.2854013

[12] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[13] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.

[14] A. Louni and K. P. Subbalakshmi. 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 (June 2018), 335–343. https://doi.org/10.1109/TCSS.2018.2801310

[15] Evelien Otte and Ronald Rousseau. 2002. Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science* 28, 6 (2002), 441–453. https://doi.org/10.1177/016555150202800601

[16] M. Park, S. Lee, O. Kwon, and A. Seuret. 2018. Closeness-Centrality-Based Synchronization Criteria for Complex Dynamical Networks With Interval Time-Varying Coupling Delays. *IEEE Transactions on Cybernetics* 48, 7 (July 2018), 2192–2202. https://doi.org/10.1109/TCYB.2017.2729164

[17] Du PH., Pham HD., and Nguyen NH. 2018. An Efficient Parallel Method for Optimizing Concurrent Operations on Social Networks. *Transactions on Computational Collective Intelligence* 10840, XXIX (April 2018), 182–199. https://doi.org/10.1007/978-3-319-90287-6_10

[18] NH. Nguyen PH. Du, HD. Pham. 2017. Source code of bigGraph. https://github.com/nnhoa/bigGraph.

[19] The Statistics Portal. 2018. Most famous social network sites worldwide as of July 2018. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[20] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2014. NetworKit: An Interactive Tool Suite for High-Performance Network Analysis. *CoRR* abs/1403.3005 (2014). http://arxiv.org/abs/1403.3005

[21] Frank W. Takes and Eelke M. Heemskerk. 2016. Centrality in the Global Network of Corporate Control. *CoRR* abs/1605.08197 (2016).

[22] J. Wei, K. Chen, Y. Zhou, Q. Zhou, and J. He. 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. https://doi.org/10.1109/BigDataService.2016.11

[23] Yutao Zhang, Jie Tang, Zhilin Yang, Jian Pei, and Philip S. Yu. 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 (KDD '15)*. ACM, New York, NY, USA, 1485–1494. https://doi.org/10.1145/2783258.2783268