PAPER Special Section on Foundations of Computer Science - New Trends of Theory of Computation and Algorithm -

Efficient Computation of Betweenness Centrality by Graph Decompositions and Their Applications to Real-World Networks

Tatsuya INOHA[†*], *Nonmember*, **Kunihiko SADAKANE**[††a)], **Yushi UNO**[†b)], *Members*, *and* **Yuma YONEBAYASHI**[††**c)], *Nonmember*

SUMMARY Betweenness centrality is one of the most significant and commonly used centralities, where *centrality* is a notion of measuring the importance of nodes in networks. In 2001, Brandes proposed an algorithm for computing betweenness centrality efficiently, and it can compute those values for all nodes in O(nm) time for unweighted networks, where n and m denote the number of nodes and links in networks, respectively. However, even Brandes' algorithm is not fast enough for recent large-scale real-world networks, and therefore, much faster algorithms are expected. The objective of this research is to theoretically improve the efficiency of Brandes' algorithm by introducing graph decompositions, and to verify the practical effectiveness of our approaches by implementing them as computer programs and by applying them to various kinds of real-world networks. A series of computational experiments shows that our proposed algorithms run several times faster than the original Brandes' algorithm, which are guaranteed by theoretical analyses.

key words: *BC-tree, Brandes' algorithm, centrality of networks, graph decomposition*

1. Introduction

Network analysis has been a surprisingly active research area since the World Wide Web was invented. There are a lot of significant topics of interest in this area (e.g., [1]), and among those computing "centrality" is one of such topics. When we model real-world networks as graphs, centrality is an index of the vertices of graphs that represents the importance of each node in those networks. The notion of centrality itself has a long history in graph theory, but in the context of network analysis, some of them are rediscovered and several notions are newly proposed due to their increasing importance in applications to real-world networks. They include degree centrality, closeness centrality [2], PageRank centrality [6], and so on, and used differently depending on the purpose. Among those the *betweenness centrality* is one of the most common centralities, and has attracted much attention since its introduction by Freeman [10] in 1977. The betweenness centrality of a vertex is defined by in how many shortest paths between any two vertices the vertex is included, and considered to have a wide variety of potential applications to real-world networks such as finding significant person (like terrorist) in social networks [8], or protein structure analyses [9], and so on.

Computing centrality efficiently is an important issue since the real-world networks are so enormous and evolving. For betweenness centrality, an algorithm proposed by Brandes [4] in 2001 (the so-called Brandes' algorithm) is well-known as one of the most efficient ways to compute its exact values. The algorithm is a kind of dynamic programming approach, and it first finds shortest paths from each starting vertex, and it computes and accumulates contributions to betweenness centrality of the other vertices by traversing all the vertices in farthest-first order. The running time of the algorithm is $O(nm)$ for unweighted graphs and $O(nm + n^2 \log n)$ for weighted graphs, where n and m are the number of vertices and edges of a graph, respectively. However, this implies that even Brandes' algorithms is not fast enough for huge networks of the current era.

By this reason, many alternative approaches for computing betweenness centrality have been considered and proposed so far. One is to approximate them, instead of to compute exact values. Brandes and Pich [5] proposed an algorithm of computing approximate values of centrality by deriving shortest paths for vertices in randomly sampled vertex subset. Another approach focuses on the dynamic aspect of real-world networks, that is, nodes and links are added and/or deleted frequently. However, it is inefficient and undesirable to compute centrality from scratch every time of the change. Therefore, for those kind of networks it is useful if we can recompute them by their differences, and such algorithms have been studied extensively [3], [12], [15].

In view of this situation, the objective of this paper is to propose yet another approach of computing betweenness centrality efficiently. We tackle this problem straightforwardly, that is, we are still trying to compute their exact values of static networks based on Brandes' algorithm. To this end, we will fully exploit the decomposition structures of networks. Specifically, our algorithm first prepares biconnectivity decompositions of input networks, and then applies Brandes' algorithm on those decomposed graphs. This may reduce the time for computation since the required information for computation is aggregated in decomposition

Manuscript received March 24, 2021.
Manuscript revised July 15, 2021.
Manuscript publicized November 8, 2021.

[†]The authors are with Graduate School of Engineering, Osaka Prefecture University, Sakai-shi, 599–8531 Japan.
[††]The authors are with Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 113–8656 Japan.
[*]Presently, with Fujitsu Broad Solution & Consulting Inc., Japan.
[**]Presently, with Marubeni Corporation, Japan.
a) E-mail: sada@mist.i.u-tokyo.ac.jp
b) E-mail: uno@cs.osakafu-u.ac.jp
c) E-mail: yuma.yonebayashi@gmail.com
DOI: 10.1587/transinf.2021FCP0003

trees. We see that many of the real-world networks have such nice decomposition structures, and we show that our proposed algorithm contributes greatly to the effectiveness.

2. Betweenness Centrality

2.1 Definition

We assume throughout the paper, that a graph $G = (V, E)$ is simple, undirected and connected, where V and E are its vertex and edge sets, respectively, We denote $n = |V|$ and $m = |E|$.

Let s and t be two distinct vertices in V, and consider the shortest paths from s to t (s, t-*shortest paths*). Let σ_{st} be the number of distinct s, t-shortest paths, where σ_{ss} is defined to be 1. Among s, t-shortest paths, let $\sigma_{st}(v)$ denote the number of those passing through v as a midpoint which is different from s and t. Now the *betweenness centrality* $g(v)$ of v ($\in V$) is defined as follows:

$$g(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{1}$$

In addition, by introducing a new variable $\delta_{st}(v) \triangleq \frac{\sigma_{st}(v)}{\sigma_{st}}$, we can rewrite the definition formula (1) as

$$g(v) = \sum_{s,t \in V \setminus \{v\}} \delta_{st}(v). \tag{2}$$

Remark that this value of $g(v)$ is often used by being normalized to fit in the range between 0 and 1. This can be computed by dividing $g(v)$ by $(n-1)(n-2)$, since the possible maximum value of $g(v)$ is achieved when v exists on the shortest paths of all $(n-1)(n-2)$ pairs of the vertices except v itself. Figure 1 illustrates a graph and (normalized) values of betweenness centrality of each vertex.

2.2 Brandes' Algorithm

In this subsection, we explain basic ideas of Brandes' algorithm [4] that computes exact values of betweenness centrality for all vertices of an unweighted graph in $O(nm)$ time.

The betweenness centrality of a single vertex v is computed simply according to formula (1) which is a direct definition of the notion. However, when we compute the value for all the vertices, applying formula (1) may contain repeated computation of similar shortest path length many times due to the double summation on vertices, and thus, this naive approach is quite inefficient. Then, to avoid this inefficiency, Brandes [4] introduced the following value:

$$\delta_{s\bullet}(v) = \sum_{t \in V \setminus \{s,v\}} \delta_{st}(v). \tag{3}$$

The value $\delta_{s\bullet}(v)$ defined in this way has the following property, where $P_s(u)$ is a set of vertices that precede u (*predecessor* of u) in the shortest path starting at s.

Theorem 2.1 [4] For all $s, v \in V$, it holds that

$$\delta_{s\bullet}(v) = \sum_{u: v \in P_s(u)} \frac{\sigma_{sv}}{\sigma_{su}} \cdot (1 + \delta_{s\bullet}(u)). \tag{4}$$

To compute the betweenness centrality for all the vertices efficiently, Brandes' algorithm pre-computes, by using the property of Theorem 2.1, the order of vertices in their distances (farthest first) and the set of predecessors for each starting vertex s by traversing their shortest paths. We show Brandes' algorithm as Algorithm 1 in the appendix for convenience since we utilize its basic ideas for the further discussions in the subsequent sections. The time complexity of such Brandes' algorithm is stated as follows.

Theorem 2.2 [4] The betweenness centrality of all vertices can be computed in $O(nm)$ time for unweighted graphs and $O(nm + n^2 \log n)$ time for weighted graphs both in $O(n + m)$ space.

As we see in the theorem, the notion of betweenness centrality is defined not only on unweighted graphs but on edge-weighted graphs. However, since the way of computing the centrality for unweighted graphs can be generalized into weighted ones relatively in a straightforward manner, we only focus on unweighted graphs in this paper. We also remark here that any path-comparison based algorithm cannot compute the betweenness centrality for all the vertices faster than $O(nm)$ time [14].

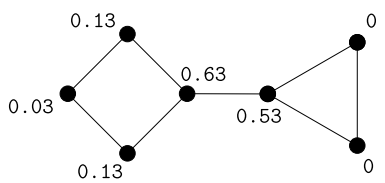

Fig. 1 A small graph and normalized betweennes centrality of its vertices.

3. Graph Decompositions

The main objective of the paper is an attempt to accelerate Brandes' algorithm by running it on small graphs after decomposing input graphs. Therefore, in this section, we introduce some notions of graph decompositions, that is, biconnectivity decompositions and BC-trees. We remark that we will use the terms vertex (and edge) for input graphs, and node (and edge) for BC-trees from now on.

3.1 Biconnectivity Decomposition and BC-Tree

For a connected graph $G = (V, E)$, a vertex $v \in V$ is a *cut-vertex* if its removal makes G disconnected, and a *block* is a (inclusion-)maximal subgraph of G without cut-vertices. Let $\mathcal{B}$ and C be families of blocks and cut-vertices of G, respectively. Then the *block graph* of G is a graph whose node set is $\mathcal{B} \cup C$ and there is an edge (b, c) ($b \in \mathcal{B}, c \in C$) if a cut-vertex c belongs to a block b. A *BC-tree* is a block

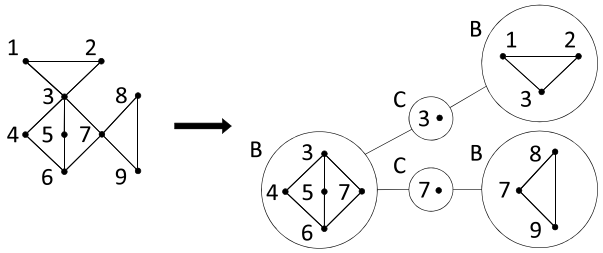

Fig. 2 A graph and its BC-tree. B and C stand for B-node and C-node, respectively.

graph rooted at an arbitrary node $b \in \mathcal{B}$. For more details of BC-trees and related notions, see [11], for example.

In a BC-tree, a node $b \in \mathcal{B}$, which represents a block, is called a B-node, and a node $c \in C$, which represents a cut-vertex, is called a C-node. By definition, no two B-nodes can be adjacent and no two C-nodes either. For a graph, its BC-tree is uniquely determined, and it shows how the graph is decomposed into biconnected components. Figure 2 illustrates an example of a graph and its BC-tree.

4. Computing Betweenness Centrality by Biconnectivity Decompositions

Decomposed structures of graphs may help computing various network invariants since such computation could be done separately within each decomposed component and merged appropriately. In this section, we propose a new approach to compute betweenness centrality efficiently by decomposing a graph into biconnected components. The idea is firstly to construct the BC-tree after biconnectivity decomposition, and then to apply Brandes' algorithm to each B-node.

To this end we introduce some values that are used during the computation. For any node $x \in \mathcal{B} \cup C$ in a BC-tree and a node y adjacent to x, we denote a block or a cut-vertex corresponding to x by S_x, a cut-vertex or a block corresponding to y, respectively, by S_y, and a unique vertex that belongs to both S_x and S_y by c_{xy}. Let $w(x, y)$ be the number of vertices except c_{xy} in block y and its descendant blocks when regarding x as a root node of a BC-tree.

Now we can describe our algorithm of computing betweenness centralities in detail.

1. For a given graph, construct its BC-tree by executing a single depth-first search.
2. Select an arbitrary B-node as a root of the BC-tree, and apply the following procedure to all nodes except the root by the post-order of a depth-first search on the BC-tree.
   Let q be a node where the depth-first search is now visiting, p be the parent of q, q_c be the children of q, and n_q be the number of vertices except c_{pq} in S_q. Since all the child nodes of q have been processed due to the post-order visit, a breadth-first search can compute n_q and thus we can compute $w(p, q)$ by the following formula:

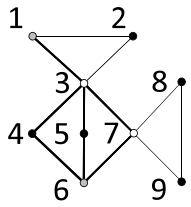

Fig. 3 Shortest paths between two end vertices 1 and 6 (drawn in grey) that belong to different biconnected components (1, 6-shortest paths are shown by bold edges). Any 1, 6-path goes through a cut-vertex that belongs to both biconnected components, one containing 1 and the other containing 6.

$$w(p, q) = \sum_{r \in q_c} w(q, r) + n_q. \tag{5}$$

   Although graphs are assumed to be undirected, we cannot compute $w(q, p)$ within the above procedure of computing $w(p, q)$. For this reason, we need to compute w for all pairs of vertices. This will be done in the next step (Step 3).
3. By a depth-first search starting at q, compute value w in pre-order, and apply Brandes' algorithm to each B-node in post-order. (This will be explained in detail below.)

Step 3 is the main part for making Brandes' algorithm more efficient by biconnectivity decomposition. This is achieved by a simple observation that any (shortest) path between two vertices in different biconnected components must go through the cut-vertices between them. To exploit this property in Brandes' algorithm and to make it work correctly, we have to give some modifications to formula (4) of the original algorithm, depending on if s or v is a cut-vertex. These are to compute the contribution (value) of end vertices, which are not contained in the currently visiting B-node, of shortest paths to the centrality value at the same time of applying Brandes' algorithm to the current B-node. Figure 3 shows an example of such shortest paths.

We have the following four cases depending on whether s or v is a cut-vertex or not.

(i) Neither s nor v is a cut-vertex.

We do not need any change in this case.

(ii) Only s is a cut-vertex.

We need to consider the contributions of v to the betweenness centrality by the shortest paths between the vertices that are reachable to s and are outside of the current B-node, and the vertices in the currently visiting B-node. Those shortest paths always go through s, and thus traverse the same route within the current B-node. This implies that $\delta_{s\bullet}(v)$ increases, for each u, by the ratio of shortest paths passing through v, out of all the shortest paths starting at outside vertices that are reachable to s (Fig. 4). Therefore, in this case, we can change formula (4) as follows:

$$\delta_{s\bullet}(v) = \sum_{u: v \in P_s(u)} \left(\frac{\sigma_{sv}}{\sigma_{su}} \cdot (1 + \delta_{s\bullet}(u)) + \frac{\sigma_{sv}}{\sigma_{su}} \cdot w(q, p) \right).$$

(iii) Only v is a cut-vertex.

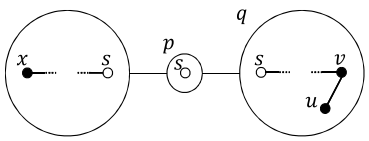

Fig. 4 Each white vertex denotes a cut-vertex, and q is the current B-node, s is a cut-vertex, and p is a C-node that shares the cut-vertex s with q. The contributions of v to the centrality by x, u-shortest paths starting at x, which is reachable to s and is in a descendant B-node of p rooted at q, is also computed.

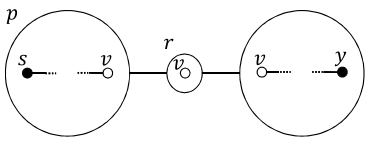

Fig. 5 Each white vertex denotes a cut-vertex, and p is the current B-node, v is a cut-vertex, and r is a C-node that shares the cut-vertex v with p. The contributions of v to the centrality by s, y-shortest paths ending at y, which is reachable from v and is in a descendant B-node of r when rooted at p, is also computed.

We need to consider the contributions of v to the betweenness centrality by the shortest paths between s and the vertices that are reachable from v and are outside of the current B-node. Those shortest paths always go through v, which forms a C-node r, and they traverse the same route within the current B-node in case that v is the end vertex (Fig. 5). Therefore, since $\delta_{s\bullet}(v)$ increases by the number of vertices reachable from v and outside of the current B-node, we need to change formula (4) as follows:

$$\delta_{s\bullet}(v) = w(q,r) + \sum_{u:v\in P_s(u)} \frac{\sigma_{sv}}{\sigma_{su}} \cdot (1 + \delta_{s\bullet}(u)).$$

(iv) Both s and v are cut-vertices.

In addition to cases (ii) and (iii), we need to consider the contributions of v to the betweenness centrality by the shortest paths between the vertices that are reachable to s and are outside of the current B-node, and the vertices that are reachable from v and outside of the current B-node (Fig. 6). Therefore, $\delta_{s\bullet}(v)$ increases not only by cases (ii) and (iii) but the number of vertices reachable to s (and outside of the current B-node) for each vertex that is reachable from v (and outside of the current B-node). Thus, we need to change formula (4) as follows:

$$\delta_{s\bullet}(v) = w(q,r) + w(q,r)\cdot w(q,p) + \sum_{u:v\in P_s(u)} \left(\frac{\sigma_{sv}}{\sigma_{su}} \cdot (1+\delta_{s\bullet}(u)) + \frac{\sigma_{sv}}{\sigma_{su}} \cdot w(q,p) \right).$$

As we analyzed so far, we can compute exact betweenness centrality values correctly only by using vertices within a B-node, once we have computed w values beforehand. The following Procedure 1 illustrates a function WCALC that computes w values in a post-order of a depth-first search. The overall algorithm of our proposed approach is presented as Algorithm 2 in the Appendix.

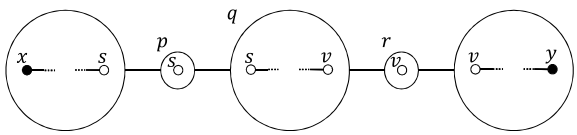

Fig. 6 Each white vertex denotes a cut-vertex, and q is the current B-node, s and v are cut-vertices, p is a C-node that shares the cut-vertex s with q, and r is a C-node that shares the cut-vertex v with q. In addition to the shortest paths of Figs. 4 and 5, the contributions of v to the centrality by x, y-shortest paths, starting at x which is reachable to s and is in a descendant B-node of p and ending at y which is reachable from v and is in a descendant B-node of r, is also computed.

Procedure 1 W-Calculation function

```
1: N[vBC]: the number of vertices in a node vBC of the
   BC-tree of an input graph
2:
3: function WCALC(v', s')
4:     visit[v'] = 1;
5:     for neighbor w' of v' do
6:         if visit[w'] = 0 then
7:             p[w'] = v';
8:             WCALC(w', s');
9:         end if
10:    end for
11:    if v' ≠ s' then
12:        for neighbor w' of v' do
13:            if w' ≠ p[v'] then
14:                W[p[v']][v'] ← W[p[v']][v'] + W[v'][w'];
15:            end if
16:        end for
17:        W[p[v']][v'] ← W[p[v']][v'] + N[v'] − 1;
18:    end if
19: end function
```

We finally discuss the computational complexity of our proposed algorithm. This is estimated as follows.

Theorem 4.1 Let n_b and m_b be the number of vertices and edges contained in each biconnected component $b \in \mathcal{B}$, respectively. Then we can compute the betweenness centrality $g(v)$ for all the vertices $v \in V$ of an unweighted graph $G = (V, E)$ in time $\sum_{b\in\mathcal{B}} O(n_b m_b)$.

Proof. In the algorithm, we spend $O(n+m)$ time for a depth-first search to decompose a graph into biconnected components and to construct a BC-tree as a preprocess. Then it requires $O(|\mathcal{B}|+|\mathcal{C}|)$ time for computing w and w' values and $O(|\mathcal{B}|+|\mathcal{C}|)+\sum_{b\in\mathcal{B}} O(n_b m_b)$ time for computing betweenness centralities. Altogether, since $\sum_{b\in\mathcal{B}} O(n_b m_b)$ is dominant the result follows. □

Observe in the theorem that if we let $N_{\mathrm{B}} = \max_{b\in\mathcal{B}} n_b$, then the total computational time could be rephrased by $\sum_{b\in\mathcal{B}} O(n_b m_b) = O(N_{\mathrm{B}} m)$. This observation implies that if an input graph has a biconnectivity decomposition whose size of its maximum component is relatively small, then the computational time can be much smaller compared to $O(nm)$ of the original Brandes' algorithm. This suggests that we

Table 1 Fundamental information about selected networks, and their biconnectivity decompositions: n and m are the number of vertices and edges of networks, respectively; B, N_B and M_B are the number of biconnected components, the number of vertices and edges of the maximum biconnected components, respectively.

Network (abbreviation)	category	n	m	B	N_B	M_B
U.Rovira i Virgili (U.R)	communication	1,133	5,451	157	976	5,293
Facebook(NIPS) (Fa)	online social	2,888	2,981	2,798	69	140
US power grid (US)	infrastructure	4,941	6,594	1,688	3,040	4,555
Pretty Good Privacy (PGP)	online contact	10,680	24,316	5,992	3,670	15,910
CAIDA (CA)	computer	26,475	53,381	10,195	16,264	43,155
Douban (Do)	online social	154,908	327,162	103,265	51,634	223,878

can expect the algorithm to work faster in practice when the real networks are decomposed well.

5. Application to Real-World Networks

In this section, we verify the effectiveness of our proposed approaches for computing betweenness centralities based on graph decompositions. We are also interested in the properties about how the real-world networks are decomposed. To this end, we implement those ideas as computer programs and performed a series of computational experiments by applying them to real-world network benchmark data. The environment of computational experiments is as follows: OS: openSUSE 12.3 64bit, Memory: 32GB, CPU: Intel Core i7-4771 CPU @ 3.50Hz x 8, compiler: g++ 4.7.2.

Benchmark network data are taken from KONECT – The Koblenz Network Collection (http://konect.uni-koblenz.de/), which maintains a large collection of network datasets from many different application areas. Among those, we selected undirected, connected and unweighted networks.

5.1 OGDF

In implementing our proposed ideas as computer programs, we used OGDF: The Open Graph Drawing Framework (http://www.ogdf.net/) libraries. This is a C++ library to provide algorithms and data structures for graph drawing purposes and has been developed by Chimani et al. [7]. In addition to the functions specific to graph drawing, various kinds of classes that are related to processing graphs are available, from fundamental to advanced ones: data structures such as arrays, stacks; algorithms such as Dijkstra's, planarizations, decompositions, and so on. It greatly facilitated our programming and made programs succinct. Below we list some of its functions that we used in our implementation:

- input/output functions for graph (network) data,
- fundamental classes that manipulate graphs, vertices, edges, etc.,
- data structures necessary for implementing Brandes' algorithm such as arrays, stacks, queues, lists, etc.,
- hashes to store w values in biconnectivity decompositions, and
- BC-tree classes.

Remark that BC-tree classes are implemented to construct BC-trees optimally in linear time, based on the theoretical results (such as [13], and so on).

5.2 Decomposition Structures of Real-World Networks

We first show the results of biconnectivity decompositions of some networks in Table 1.

From this table, we can see that the biconnectivity decomposition structures greatly depend on each network. For example, Facebook(NIPS) is decomposed into extremely many components and its maximum size is small; it is only about 2.39% of the entire network. On the other hand, U.Rovira i Virgili is decomposed into relatively small number of components but the maximum size is large; it occupies about 86.1% of the total number of vertices. Similar observation can be seen in US power grid, that is, it is decomposed into relatively small number of components and the maximum component size is about 61.5% of the total number of vertices. We could see that these phenomena are due to the category of networks; virtual networks (such as online social) are known to have many low-degree vertices (e.g., pendant vertices) on its fringe, and thus easily to be decomposed into a single edge component. In contrast, physical networks (such as infrastructure) usually avoid to have cut-vertices, and thus easy to form a big component.

It is hard to find any common properties of decompositions shared by all networks, but at least we could say that most of the networks are decomposed into "small number of large biconnected components" and "large number of small biconnected components". We also remark that, to the best of our knowledge, there are quite few observations about decomposition structures of real-world networks, so these results are valuable on their own.

5.3 Computational Results for Biconnectivity Decompositions

We now show in Table 2 the experimental results of computing betweenness centralities by implementing (a) naive Brandes' algorithm (for comparison purpose), and (b) our proposed algorithm by biconnectivity decompositions. For the proposed algorithm, we measured the computational time for constructing BC-trees and applying Brandes' algorithm after decompositions, separately.

We have the following observations from Table 2.

Table 2 Computational time (in seconds) by (a) Brandes' algorithm and (b) the proposed algorithm by biconnectivity decompositions. We did multiple (five) trials for each network. For (a), we take their average. For (b), we take the median in terms of the total time, and BC-tree and Brandes are the times of that trial. We did so so that the sum of BC-tree and Brandes coincides with the total time.

Network	Brandes	Biconnectivity decomposition		
		BC-tree	Brandes	total
U.R	0.199294	0.001451	0.199611	0.201062
Fa	0.332757	0.001698	0.015221	0.016919
US	1.95956	0.003268	0.994061	0.997329
PGP	13.1339	0.012537	2.82199	2.83453
CA	115.075	0.030944	92.2047	92.2356
Do	10,083.6	0.253606	2,574.98	2,575.23

1. It takes almost no time (within a second) for biconnectivity decompositions and obtaining BC-trees, for any types of networks.
2. For all networks except one (U.R), the proposed algorithm runs faster than a naive application of Brandes' algorithm.
3. The effectiveness of the proposed algorithm depends on the property of decompositions of networks.

We verify the third point in detail. For example, U.Rovira i Virgili (U.R), which is not well decomposed into biconnected components, takes almost as much time as for Brandes' algorithm, and thus the proposed algorithm has disadvantage in spending time for decompositions. On the other hand, Facebook(NIPS) (Fa), which is decomposed into so many small components, can receive the advantage of the decomposition, and the proposed algorithm outperforms a naive application of Brandes' algorithm without decompositions. It runs about 20 times faster than a naive implementation.

We can conclude that using biconnectivity decompositions shows great advantage for computing betweenness centralities as expected.

6. Discussions and Conclusion

In this paper, we studied the problem of computing betweenness centralities of all vertices of a given network. Especially, we focused on the well-known Brandes' algorithm and proposed an approach of exploiting decomposition structures of networks to make Brandes' algorithm work more efficiently. Furthermore, we implemented these ideas as computer programs and verified the effectiveness of the proposed algorithm by performing computational experiments on real-world networks.

As a result, we confirmed that using biconnectivity decompositions (applying Brandes' algorithm after biconnectivity decompositions) has great advantage in the computational time. We also observed that real-world networks have various types of biconnectivity decomposition structures and the effectiveness of proposed approach depends on those structures. However, the proposed algorithm performs much better than a naive algorithm (i.e., without using decompositions) for large-scale networks, so we claim that we could always prefer our proposed algorithm for computing exact betweenness centrality values.

Computing betweenness centrality efficiently is an important task by its own, but it has great spreading effect. For future work, we can incorporate our proposed approach into other algorithms that use centrality computation as subroutines, such as Girvan-Newman algorithm for community detection in social networks. We can expect that they will be greatly accelerated by our proposed algorithm.

Acknowledgments

We express our hearty gratitude for anonymous refrees for their careful reading and helpful comments.

This work is partially supported by JST CREST Grant Number JPMJCR1402, Japan. This work is also partially supported by JSPS KAKENHI Grant Numbers JP17K00017, 20H05964 and 20H05967.

References

[1] A.-L. Barabási, Network Science, Cambridge University Press, 2016.
[2] A. Bavelas, "Communication patterns in task-oriented groups," J. Acoust. Soc. Am., vol.22, no.6, pp.725–730, 1950.
[3] E. Bergamini, H. Meyerhenke, and C.L. Staudt, "Approximating betweenness centrality in large evolving networks," In the 17th Meeting on Algorithm Engineering and Experiments (ALENEX), pp.133–146, 2015.
[4] U. Brandes, "A faster algorithm for betweenness centrality," J. Mathematical Sociology, vol.25, no.2, pp.163–177, 2001.
[5] U. Brandes and C. Pich, "Centrality estimation in large networks," Int. J. Bifurcation and Chaos, vol.17, no.07, pp.2303–2318, 2007.
[6] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," Computer Networks and ISDN Systems, vol.33, no.1-7, pp.107–117, April 1998.
[7] M. Chimani, C. Gutwenger, M. Jünger, G.W. Klau, K. Klein, and P. Mutzel, The Open Graph Drawing Framework (OGDF). Chapter 17 in: R. Tamassia (ed.), Handbook of Graph Drawing and Visualization, CRC Press, pp.543–569, 2014.
[8] T. Coffman, S. Greenblatt, and S. Marcus, "Graph-based technologies for intelligence analysis," Communications of the ACM, vol.47, no.3, pp.45–47, March 2004.
[9] A. Del Sol, H. Fujihashi, and P. O'Meara, "Topology of small-world networks of protein-protein complex structures," Bioinformatics, vol.vol.21, no.8, pp.1311–1315, Jan. 2005.
[10] L.C. Freeman, "A set of measures of centrality based on betweenness," Sociometry, vol.40, no.1, pp.35–41, 1977.
[11] C. Gutwenger, "Application of SPQR-trees in the planarization approach for drawing graphs," Ph. D Thesis, Dortmund University of Technology, 2010.

[12] T. Hayashi, T. Akiba, and Y, Yoshida, "Fully dynamic betweenness centrality maintenance on massive networks," In the 42nd Int. Conf. the Very Large Data Bases Endowment (VLDB), vol.9, no.2, pp.48–59, Oct. 2015.
[13] J. Hopcroft and R.E. Tarjan, "Dividing a graph into triconnected components," SIAM J. Comput., vol.2, no.3, pp.135–158, 1973.
[14] S. Kintali, "Betweenness centrality: algorithms and lower bounds," arXiv: 0809.1906v2, 2008.
[15] M.-J. Lee, J. Lee, J.Y. Park, R.H. Choi, and C.-W. Chung, "QUBE: a quick algorithm for updating betweenness centrality," In the 21st International World Wide Web Conference (WWW), pp.351–360, April 2012.

Appendix A: Brandes' algorithm

Algorithm 1 Brandes' algorithm

```
1: g[v] ← 0, v ∈ V;
2: for s ∈ V do
3:     S ← empty stack;
4:     P[w] ← empty list, w ∈ V;
5:     σ[t] ← 0, t ∈ V; σ[s] ← 1;
6:     d[t] ← −1, t ∈ V; d[s] ← 0;
7:     Q ← empty queue;
8:     enqueue s → Q;
9:     while Q not empty do
10:        dequeue v ← Q;
11:        push v → S;
12:        for neighbor w of v do
13:            if d[w] < 0 then
14:                enqueue w → Q;
15:                d[w] ← d[v] + 1;
16:            end if
17:            if d[w] = d[v] + 1 then
18:                σ[w] ← σ[w] + σ[v];
19:                append v → P[w];
20:            end if
21:        end for
22:    end while
23:    δ[v] ← 0, v ∈ V;
24:    while S not empty do
25:        pop w ← S;
26:        for v ∈ P[w] do
27:            δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
28:        end for
29:        if w ≠ s then
30:            g[w] ← g[w] + δ[w];
31:        end if
32:    end while
33: end for
```

Appendix B: Brandes' algorithm with BC-tree

Algorithm 2 Brandes' algorithm with BCTree

```
1: O[vB]: vertices of an input graph corresponding to a B-node vB
2: C[vO]: a C-node corresponding to a cut-vertex vO of an input graph
3:
4: p[v′] ← 0, v′ ∈ V′;
5: visit[t′] ← 0, t′ ∈ V′;
6: for e′ ∈ E′ do
7:     v′, w′ ← endpoint of e′
8:     W[v′][w′] = 0;
9: end for
10: WCALC(s′, s′);
11:
12: g[v] ← 0, v ∈ V;
13: visit[t′] ← 0, t′ ∈ V′;
14: S′ ← empty stack;
15: push s′ → S′;
16: while S′ not empty do
17:     pop v′ ← S′;
18:     visit[v′] ← 1;
19:     if N[v′] > 1 then
20:         for s ∈ V[v′] do
21:             S ← empty stack;
22:             P[w] ← empty list, w ∈ V[v′];
23:             σ[t] ← 0, t ∈ V[v′]; σ[s] ← 1;
24:             d[t] ← −1, t ∈ V[v′]; d[s] ← 0;
25:             Q ← empty queue;
26:             enqueue s → Q;
27:             while Q not empty do
28:                 dequeue v ← Q;
29:                 push v → S;
30:                 for neighbor w of v do
31:                     if d[w] < 0 then
32:                         enqueue w → Q;
33:                         d[w] ← d[v] + 1;
34:                     end if
35:                     if d[w] = d[v] + 1 then
36:                         σ[w] ← σ[w] + σ[v];
37:                         append v → P[w];
38:                     end if
39:                 end for
40:             end while
41:             δ[v] ← 0, v ∈ V[v′];
42:             while S not empty do
43:                 pop w ← S;
44:                 if O[w] cut-vertex then
45:                     δ[w] ← δ[w] + W[v′][C[w]];
46:                     if O[s] cut-vertex then
47:                         δ[w] ← δ[w] + W[v′][C[w]] · W[v′][C[s]];
48:                     end if
49:                 end if
50:                 for v ∈ P[w] do
51:                     δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
52:                     if O[s] cut-vertex then
53:                         δ[v] ← δ[v] + (σ[v]/σ[w]) · W[v′][C[s]];
54:                     end if
55:                 end for
56:                 if w ≠ s then
57:                     g[w] ← g[w] + δ[w];
58:                 end if
59:             end while
60:         end for
61:     end if
62:     for neighbor w′ of v′ do
63:         if visit[w′] = 0 then
64:             for neighbor x′ of v′ do
65:                 if x′ ≠ w′ then
66:                     W[w′][v′] ← W[w′][v′] + W[v′][x′];
67:                 end if
68:             end for
```

69: $W[w'][v'] \leftarrow W[w'][v'] + N[v'] - 1$;
70: push $w' \rightarrow S'$;
71: **end if**
72: **end for**
73: **end while**

Tatsuya Inoha received the M.S. degree in Engineering from Osaka Prefecture University in 2017. Currently, he is working at Fujitsu Broad Solutions & Consulting Inc., Nagoya, Japan.

Kunihiko Sadakane received B.S., M.S., and Ph.D. degrees from Department of Information Science, University of Tokyo in 1995, 1997 and 2000, respectively. He was a research associate at Graduate School of Information Sciences, Tohoku University from 2000 to 2003, an associate professor at Faculty of Information Science and Electrical Engineering, Kyushu University from 2003 to 2009, an associate professor at National Institute of Informatics from 2009 to 2014. Since 2014, he has been a professor at Graduate School of Information Science and Technology, The University of Tokyo. His research interest includes information retrieval, data structures, and data compression.

Yushi Uno received the B.E., M.E. and Ph.D. degrees from Kyoto University, in 1987, 1989 and 1995, respectively. Currently, he is working at Graduate School of Science, Osaka Prefecture University, Sakai, 599–8531 Japan. His research interests include combinatorial optimization, algorithmic graph theory, discrete mathematics, operations research, design and analysis of algorithms, and network analysis. He is a member of Association for Computing Machinery, Operations Research Society of Japan and Information Processing Society of Japan.

Yuma Yonebayashi received B.S. and M.S. degrees from Faculty of Engineering and Graduate School of Information Science and Technology, The University of Tokyo, Japan, respectively. Currently, he is working at Marubeni Corporation, Japan.