Journal of Complex Networks (2022) **4**, Advance Access Publication on 3 August 2022
https://doi.org/10.1093/comnet/cnac033

An algorithm for updating betweenness centrality scores of all vertices in a graph upon deletion of a single edge

Yoshiki Satotani, Tsuyoshi Migita and Norikazu Takahashi†

Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530, Japan

†Corresponding author. Email: takahashi@okayama-u.ac.jp

Edited by: Ernesto Estrada

[Received on 6 March 2022; editorial decision on 25 June 2022; accepted on 28 June 2022]

Betweenness centrality (BC) is a measure of the importance of a vertex in a graph, which is defined using the number of the shortest paths passing through the vertex. Brandes proposed an efficient algorithm for computing the BC scores of all vertices in a graph, which accumulates pair dependencies while traversing single-source shortest paths. Although this algorithm works well on static graphs, its direct application to dynamic graphs takes a huge amount of computation time because the BC scores must be computed from scratch every time the structure of graph changes. Therefore, various algorithms for updating the BC scores of all vertices have been developed so far. In this article, we propose a novel algorithm for updating the BC scores of all vertices in a graph upon deletion of a single edge. We also show the validity and efficiency of the proposed algorithm through theoretical analysis and experiments using various graphs obtained from synthetic and real networks.

Keywords: betweenness centrality; dynamic graph; shortest path; algorithm; time complexity.

1. Introduction

In the last two decades, complex networks have attracted a great deal of attention since they efficiently describe a wide range of systems in biology, information technology, social science and so on [1]. Identifying important nodes in a network, such as super-spreaders of some disease in a population [2], influencers in a social network [3] and central cities in an air transportation network [4], is a fundamental task in network analysis. For this purpose, various centralities have been used so far [5]. Among them, the betweenness centrality (BC) [6], which defines the importance of a node based on how often the node appears on the shortest paths among all pairs of nodes, has been used for many applications including transportation networks [7] and communication networks [8].

The most well-known algorithm for computing the BC scores of all nodes is the one proposed by Brandes [9]. A key idea behind the Brandes algorithm is to compute the BC scores by accumulating pair dependencies while traversing single-source shortest paths (SSSPs). Since the seminal paper of Brandes, many variants of his algorithm have been proposed. Some of them parallelize the accumulation [10–15]. Some of the others approximate the BC scores by sampling and extrapolation [16–22] or by setting an upper bound for the distance between nodes [23] or by using the hypergraph sketch [24].

The Brandes algorithm works efficiently on a network with a fixed topology. However, many networks in the real world are dynamic in the sense that their nodes and/or links are added/deleted over time [25]. For such a dynamic network, it is very costly to compute the BC scores from scratch using the Brandes

algorithm every time the network changes its structure. Therefore, various algorithms to efficiently update and maintain the BC scores have been developed so far. One of the earliest algorithms is QUBE [26] that caches maximum union cycles (MUCs). As far as exact algorithms are concerned, many of them cache SSSPs to update the BC scores efficiently when an edge is added [27–32]. For example, Green *et al.* [28] proposed an algorithm for unweighted graphs. Kas *et al.* [29] proposed an algorithm based on the Ramalingam and Reps (RR) algorithm [33]. Nasre *et al.* [30] proposed an algorithm based on the idea of Karger *et al.* [34]. Pontecorvi and Ramachandran [32] proposed an algorithm for fully dynamic graphs based on the Demetrescu and Italiano (DI) algorithm [35]. Bergamini *et al.* [27] proposed an algorithm based on the RR algorithm. As an algorithm that can deal with an edge deletion, Nasre *et al.* [31] proposed an algorithm based on the DI algorithm. Other algorithms have also been proposed in the literature, such as an MUC-based algorithm [36], an algorithm that updates the node BC scores and the edge BC scores simultaneously [37], approximation algorithms [38–41] and a parallel algorithm [42].

In this article, we consider weighted directed/undirected graphs as mathematical models of networks and propose a novel algorithm for updating the BC scores of all vertices upon the deletion of a directed/undirected single edge. We also show the efficiency of our algorithm through time complexity analysis and experiments using various graphs obtained from synthetic and real networks. To be more specific, we show that the worst-case time complexity of our algorithm is the same as the Brandes algorithm, but the actual execution time is significantly reduced in many cases. Our algorithm makes use of the RR algorithm [33], which has an advantage in memory consumption compared to the DI algorithm [35]. The space complexity of the RR algorithm is $\mathcal{O}(|V|^2)$, where $|V|$ denotes the number of vertices, while that of the DI algorithm is $\mathcal{O}(|V|^3)$ [35]. To the best of the authors' knowledge, the RR algorithm has not been used so far in any algorithm for updating the BC scores upon deletion of an edge, though some existing algorithms [27, 29] use it to update the BC scores upon addition of an edge, as stated above. Hence, our algorithm can be considered as a missing piece in the history of algorithms for updating the BC scores on dynamic graphs. Our algorithm also makes use of the algorithm of Bergamini *et al.* [27] in order to reduce memory requirements.

The remainder of this article is organized as follows. Section 2 presents the definition of the BC for weighted strongly connected directed graphs and notations used in later discussions. Section 3 describes our algorithm in detail and analyses its time complexity. Section 4 discusses the extension of the algorithm to not strongly connected directed graphs and undirected graphs. Section 5 shows the efficiency of the algorithm through experiments. Finally, Section 6 concludes this article with some remarks and future challenges.

2. BC and Brandes algorithm

In this section, we focus our attention on simple weighted directed graphs (or digraphs) and introduce the notion of the BC and the basic idea behind the Brandes algorithm for those graphs. A digraph G is defined as an ordered pair (V, E), where V and E are the sets of vertices and directed edges, respectively. Each member of E is an ordered pair (x, y) of vertices x and y, which represents the edge from x to y. A digraph is said to be simple if it has neither self-loops nor multiple edges. A digraph is said to be weighted if each edge $(x, y) \in E$ has a weight (or length) denoted by $w_{(x,y)}$, which is assumed to be positive in this article. A digraph is said to be strongly connected if, for any pair of two vertices s and t, there is a directed path from s to t. The sets of successors and predecessors of vertex x in a digraph $G = (V, E)$ are denoted by $\mathcal{S}_G(x)$ and $\mathcal{P}_G(x)$, respectively, that is, $\mathcal{S}_G(x) = \{y \mid (x, y) \in E\}$ and $\mathcal{P}_G(x) = \{y \mid (y, x) \in E\}$.

Let $G = (V, E)$ be an weighted strongly connected digraph. The length of a path from vertex s to vertex t is defined by the sum of the weights of the edges in the path. The length of the shortest paths

from s to t, which is also called the distance from s to t, is denoted by d_{st}. Also, the number of the shortest paths from s to t is denoted by σ_{st}. For the sake of convenience, we define $d_{ss} = 0$ and $\sigma_{ss} = 1$. The number of the shortest paths from s to t that pass through x is denoted by $\sigma_{st}(x)$. The shortest paths from s to t form a subgraph of G. This subgraph is denoted by $G_{st} = (V_{st}, E_{st})$, where V_{st} and E_{st} are subsets of V and E, respectively. The shortest paths from vertex s to all other vertices of G form a subgraph of G. This subgraph is called the SSSPs from s and denoted by $G_s = (V, E_s)$, where E_s is a subset of E. The shortest paths to vertex t from all other vertices form a subgraph of G. This subgraph is called the single-target shortest paths (STSPs) to t.

With these notations, the BC of vertex x is defined as

$$B_x = \sum_{s \in V \setminus \{x\}} \left(\sum_{t \in V \setminus \{s,x\}} \frac{\sigma_{st}(x)}{\sigma_{st}} \right). \tag{2.1}$$

In other words, B_x is the summation of the ratio of the number of the shortest paths from s to t which pass through x, to the number of the shortest paths from s to t, over all pairs of s and t such that $s \neq x$ and $t \neq s, x$. The quantity $\sigma_{st}(x)/\sigma_{st}$ is sometimes called the pair-dependency of s and t on x and denoted by $\delta_{st}(x)$. Furthermore, the quantity $\sum_{t \in V \setminus \{s,x\}} \delta_{st}(x)$ is called the dependency of s on x and denoted by $\delta_{s\bullet}(x)$. Using this notation, we can rewrite (2.1) as $B_x = \sum_{s \in V \setminus \{x\}} \delta_{s\bullet}(x)$.

Brandes [9] showed that $\delta_{s\bullet}(x)$ is expressed in terms of the dependencies of s on the successors of x on G_s as

$$\delta_{s\bullet}(x) = \sum_{y \in \mathcal{S}_{G_s}(x)} \frac{\sigma_{sx}}{\sigma_{sy}} (1 + \delta_{s\bullet}(y)) \tag{2.2}$$

and that if the number σ_{sx} of the shortest paths is computed for all $x \in V$ when finding G_s then the value of $\delta_{s\bullet}(x)$ is obtained for all $x \in V \setminus \{s\}$ by computing the right-hand side of (2.2) while traversing G_s from sinks (vertices without successors) to the source s in the opposite direction of edges. Here, we should note that $\delta_{s\bullet}(x) = 0$ if x is a sink, that is, $\mathcal{S}_{G_s}(x) = \varnothing$. Making use of this idea, Brandes [9] developed an efficient algorithm for computing the BC scores of all vertices that runs in $\mathcal{O}(|V||E| + |V|^2 \log|V|)$ time, where $|V|$ is the number of vertices and $|E|$ is the number of edges in G, while a straightforward algorithm runs in $\mathcal{O}(|V|^3)$ time, due to the explicit sum of the pair-dependencies $\delta_{st}(x) = \sigma_{st}(x)/\sigma_{st}$.

3. Proposed algorithm

We propose a novel algorithm for updating the BC scores of all vertices in a simple weighted strongly connected digraph $G = (V, E)$ when a directed edge $(u, v) \in E$ is deleted. It is assumed for simplicity that the resulting digraph is also strongly connected in this section. However, this assumption will be removed in Section 4. We first give some results of theoretical analysis which play important roles in our algorithm and then explain the details of the algorithm.

In what follows, the prime symbol ($'$) is used when considering the digraph after the deletion of the edge (u, v). For example, $G' = (V, E \setminus \{(u, v)\})$ and d'_{st} represents the distance from s to t in G'. A vertex s is called an affected source of a target $t \in V$ if G_{st} is changed by the deletion of the edge. The set of affected sources of t is denoted by $S(t)$, and its complement is denoted by $\overline{S(t)} = V \setminus S(t)$. Similarly, a vertex t is called an affected target of a source $s \in V$ if G_{st} is changed by the deletion of the edge. The set of affected targets of s is denoted by $T(s)$, and its complement is denoted by $\overline{T(s)} = V \setminus T(s)$. How to

find $S(t)$ and $T(s)$ is described in Section 3.2. The contribution of all $t \in T(s) \setminus \{x\}$ to $\delta_{s\bullet}(x)$ and $\delta'_{s\bullet}(x)$ are denoted by $\Delta_{s\bullet}(x) = \sum_{t \in T(s)\setminus\{x\}} \delta_{st}(x)$ and $\Delta'_{s\bullet}(x) = \sum_{t \in T(s)\setminus\{x\}} \delta'_{st}(x)$, respectively. Bergamini *et al.* [27] proposed an algorithm to compute $\Delta_{s\bullet}(x)$ and $\Delta'_{s\bullet}(x)$ for all pairs of distinct vertices s and x. The details of their algorithm are described also in Section 3.2.

3.1 *Analysis*

The next lemma characterizes the set of affected targets of a source $s \in V$ and the set of affected sources of a target $t \in V$ when an edge is deleted.

LEMMA 3.1 Let $G = (V, E)$ be a simple weighted strongly connected digraph and suppose that an edge $(u, v) \in E$ is deleted from G. Then, the set of affected targets of a source $s \in V$ and the set of affected sources of a target $t \in V$ are given by

$$T(s) = \{t \in V \mid d_{st} = d_{su} + w_{(u,v)} + d_{vt}\}, \tag{3.1}$$

$$S(t) = \{s \in V \mid d_{st} = d_{su} + w_{(u,v)} + d_{vt}\}, \tag{3.2}$$

respectively.

Proof. We consider only (3.1) because (3.2) can be proved in the same way. It is clear from the definition of $T(s)$ that $t \in T(s)$ if and only if the subgraph $G_{st} = (V_{st}, E_{st})$ of G, which consists of all the shortest paths from s to t, contains the edge (u, v). It is also clear that G_{st} contains (u, v) if and only if

$$d_{st} = d_{su} + w_{(u,v)} + d_{vt}. \tag{3.3}$$

Therefore, $t \in T(s)$ if and only if (3.3) holds, which means that $S(t)$ is given by (3.1). □

The next lemma gives a sufficient condition for $S(t)$ to be empty and one for $T(s)$ to be empty.

LEMMA 3.2 Let $G = (V, E)$ be a simple weighted strongly connected digraph and suppose that an edge $(u, v) \in E$ is deleted from G. If $s \in V$ is not an affected source of v, that is, $s \notin S(v)$, then $T(s)$ is empty. Similarly, if $t \in V$ is not an affected target of u, that is, $t \notin T(u)$, then $S(t)$ is empty.

Proof. Let us first suppose that $s \in V$ is not an affected source of v. Then it follows from Lemma 3.1 that $d_{sv} < d_{su} + w_{(u,v)}$. Furthermore, for any vertex $t \in V \setminus \{s\}$, we have $d_{st} \leq d_{sv} + d_{vt} < d_{su} + w_{(u,v)} + d_{vt}$ which implies that t is not an affected target of s. Therefore, $T(s)$ is empty. Let us next suppose that $t \in V$ is not an affected target of u. Then, it follows from Lemma 3.1 that $d_{ut} < w_{(u,v)} + d_{vt}$. Furthermore, for any vertex $s \in V \setminus \{t\}$, we have $d_{st} \leq d_{su} + d_{ut} < d_{su} + w_{(u,v)} + d_{vt}$ which implies that s is not an affected source of t. Therefore, $S(t)$ is empty. □

From Lemma 3.2, we obtain the following result which plays an important role when SSSPs are updated in the algorithm we propose in this article.

PROPOSITION 3.1 Let $G = (V, E)$ be a simple weighted strongly connected digraph and suppose that an edge $(u, v) \in E$ is deleted from G. If $G'_{st} \neq G_{st}$ for a pair of vertices s and t then $s \in S(v)$ and $t \in T(u)$. In other words, if $s \notin S(v)$ or $t \notin T(u)$ then $G'_{st} = G_{st}$.

Proof. Suppose that $G'_{st} \neq G_{st}$ for a pair of vertices s and t. Then, $T(s)$ and $S(t)$ are nonempty because $t \in T(s)$ and $s \in S(t)$. Therefore, it follows from Lemma 3.2 that $s \in S(v)$ and $t \in T(u)$. □

From Lemma 3.2, we also obtain the following result, which plays a central role when the BC scores are updated in the algorithm we propose in this article.

PROPOSITION 3.2 Let $G = (V, E)$ be a simple weighted strongly connected digraph and suppose that an edge $(u, v) \in E$ is deleted from G. Then,

$$B'_x - B_x = \sum_{s \in S(v) \setminus \{x\}} (\Delta'_{s\bullet}(x) - \Delta_{s\bullet}(x)) \tag{3.4}$$

for all $x \in V$.

Proof. For each $x \in V$, the amount of change in the BC score is expressed as follows:

$$\begin{aligned}
B'_x - B_x &= \sum_{s \in V \setminus \{x\}} \left(\delta'_{s\bullet}(x) - \delta_{s\bullet}(x)\right) \\
&= \sum_{s \in S(v) \setminus \{x\}} \left(\delta'_{s\bullet}(x) - \delta_{s\bullet}(x)\right) + \sum_{s \in \overline{S(v)} \setminus \{x\}} \left(\delta'_{s\bullet}(x) - \delta_{s\bullet}(x)\right) \\
&= \sum_{s \in S(v) \setminus \{x\}} \left(\sum_{t \in T(s) \setminus \{x\}} \left(\delta'_{st}(x) - \delta_{st}(x)\right) + \sum_{t \in \overline{T(s)} \setminus \{s,x\}} \left(\delta'_{st}(x) - \delta_{st}(x)\right) \right) \\
&\quad + \sum_{s \in \overline{S(v)} \setminus \{x\}} \left(\sum_{t \in T(s) \setminus \{x\}} \left(\delta'_{st}(x) - \delta_{st}(x)\right) + \sum_{t \in \overline{T(s)} \setminus \{s,x\}} \left(\delta'_{st}(x) - \delta_{st}(x)\right) \right).
\end{aligned}$$

Here, for each $s \in V \setminus \{x\}$, it is clear from the definition of $T(s)$ that $\delta'_{st}(x) = \delta_{st}(x)$ for all $t \in \overline{T(s)} \setminus \{s, x\}$. Also, as shown in Lemma 3.2, if $s \notin S(v)$ then $T(s)$ is empty. Therefore, we have

$$\begin{aligned}
B'_x - B_x &= \sum_{s \in S(v) \setminus \{x\}} \left(\sum_{t \in T(s) \setminus \{x\}} \left(\delta'_{st}(x) - \delta_{st}(x)\right) \right) \\
&= \sum_{s \in S(v) \setminus \{x\}} \left(\sum_{t \in T(s) \setminus \{x\}} \delta'_{st}(x) - \sum_{t \in T(s) \setminus \{x\}} \delta_{st}(x) \right) \\
&= \sum_{s \in S(v) \setminus \{x\}} \left(\Delta'_{s\bullet}(x) - \Delta_{s\bullet}(x)\right)
\end{aligned}$$

which completes the proof. □

Algorithm 1 Update BC scores in a digraph upon deletion of an edge

Input: $G = (V, E)$, $w_{(x,y)}$ for all $(x, y) \in E$, d_{xy} and σ_{xy} for all $x, y \in V$, B_x for all $x \in V$, $(u, v) \in E$
Output: d_{xy} and σ_{xy} for all $x, y \in V$, B_x for all $x \in V$

1. **if** $d_{uv} = w_{(u,v)}$ **then**
2. find $S(v)$ using Algorithm 2 with $t = v$
3. **for** $s \in S(v)$ **do**
4. find $T(s)$ using Algorithm 3
5. **end for**
6. **for** $s \in S(v)$ **do**
7. set $B_x \leftarrow B_x - \Delta_{s\bullet}(x)$ for all $x \in V \setminus \{s\}$ using Algorithm 4 with $M = -1$
8. **end for**
9. **for** $s \in S(v)$ **do**
10. update d_{sx} and σ_{sx} for all $x \in T(s)$ using Algorithm 5
11. **end for**
12. set $G \leftarrow (V, E \setminus \{(u, v)\})$
13. **for** $s \in S(v)$ **do**
14. set $B_x \leftarrow B_x + \Delta_{s\bullet}(x)$ for all $x \in V \setminus \{s\}$ using Algorithm 4 with $M = 1$
15. **end for**
16. **end if**
17. return d_{xy} and σ_{xy} for all $x, y \in V$ and B_x for all $x \in V$

3.2 *Algorithm*

The algorithm we propose in this article is formally described in Algorithm 1. It receives a simple weighted strongly connected digraph $G = (V, E)$, the weight $w_{(x,y)}$ of every edge $(x, y) \in E$, the length d_{xy} and the number σ_{xy} of the shortest paths from x to y for every pair of vertices x and y, the BC score B_x of every vertex $x \in V$ and an edge (u, v) to be deleted from G as inputs, and returns the length d'_{xy} and the number σ'_{xy} of the shortest paths from x to y in $G' = (V, E \setminus \{(u, v)\})$ for every pair of vertices x and y, and the BC score B'_x of every $x \in V$ in G' as outputs.

Algorithm 1 first checks whether the equality $d_{uv} = w_{(u,v)}$ holds or not (see Line 1). If $d_{uv} < w_{(u,v)}$, then it returns d_{xy} and σ_{xy} for all $x, y \in V$ and B_{xy} for all $x \in V$ (see Line 17) because the deletion of the edge (u, v) does not affect any shortest path between vertices. Otherwise, it finds $S(v)$ using Algorithm 2 with $t = v$ (see Line 2). Algorithm 2 is a depth-first search for the set $S(t)$ of affected sources of a given target $t \in V$. Algorithm 1 next finds $T(s)$ for each $s \in S(v)$ using Algorithm 3 (see Lines 3–5). Algorithm 3 is a depth-first search for the set $T(s)$ of affected targets of a given source $s \in V$. Algorithm 1 then decreases the BC score B_x by $\sum_{s \in S(v) \setminus \{x\}} \Delta_{s\bullet}(x)$ for each $x \in V$ using the algorithm of Bergamini *et al.* [27] (see Lines 6–8). It then updates the values of d_{sx} and σ_{sx} for each pair of vertices s and x such that $s \in S(v)$ and $x \in T(s)$ using the RR algorithm (see Lines 9–11). The new values of d_{sx} and σ_{sx} are equal to the length d'_{sx} and the number σ'_{sx} of the shortest paths from s to x in $G' = (V, E \setminus \{(u, v)\})$. Note that we do not need to consider updating the values of d_{sx} and σ_{sx} with $s \notin S(v)$ because $s \notin S(v)$ implies $T(s) = \varnothing$ as shown in Lemma 3.2. Algorithm 1 then deletes the edge (u, v) (see Line 12) and increases the BC score B_x by $\sum_{s \in S(v) \setminus \{x\}} \Delta_{s\bullet}(x)$ for each $x \in V$ using the algorithm of Bergamini *et al.* [27] (see Lines 13–15), where we should note that $\Delta_{s\bullet}(x)$ is computed for $G' = (V, E \setminus \{(u, v)\})$ and thus equal to $\Delta'_{s\bullet}(x)$. It finally

Algorithm 2 Find affected sources of a target in a digraph

Input: $G = (V, E)$, $w_{(x,y)}$ for all $(x, y) \in E$, d_{xy} for all $x, y \in V$, $(u, v) \in E$, $t \in V$
Output: $S(t)$

1. $S(t) \leftarrow \varnothing$
2. **if** $d_{ut} = w_{(u,v)} + d_{vt}$ **then**
3. $S(t) \leftarrow \{u\}$
4. create an empty stack A and then push u to A
5. $R \leftarrow \varnothing$
6. **while** A is nonempty **do**
7. pop x from A
8. **for** $y \in \mathcal{P}_G(x) \setminus R$ **do**
9. **if** $d_{yt} = d_{yu} + w_{(u,v)} + d_{vt}$ **then**
10. $S(t) \leftarrow S(t) \cup \{y\}$
11. push y to A
12. **end if**
13. **end for**
14. $R \leftarrow R \cup \{x\}$
15. **end while**
16. **end if**
17. return $S(t)$

returns the values of $d_{xy}(= d'_{xy})$ and $\sigma_{xy}(= \sigma'_{xy})$ for all $x \in V$ and $y \in V$, and the value of $B_x(= B'_x)$ for all $x \in V$ (see Line 17).

In what follows, we explain Algorithms 2–5 in detail. Algorithm 2 receives a simple weighted strongly connected digraph $G = (V, E)$, the weight $w_{(x,y)}$ of every edge $(x, y) \in E$, the length d_{xy} of the shortest paths from x to y for every pair of vertices x and y, an edge (u, v) to be deleted from G, and a target $t \in V$ as inputs and returns the set of affected source $S(t)$ as an output. It first sets $S(t) \leftarrow \varnothing$ (see Line 1). It next checks whether the equality $d_{ut} = w_{(u,v)} + d_{vt}$ holds or not, in other words, whether $t \in T(u)$ or not (see Line 2). If $d_{ut} < w_{(u,v)} + d_{vt}$ then it returns $S(t) = \varnothing$ (see Line 17) because $t \notin T(u)$ implies $S(v) = \varnothing$ as shown in Lemma 3.2. Otherwise, it sets $S(t) \leftarrow \{u\}$ (see Line 3) and performs a depth-first search traversing vertices from u in the opposite direction of edges (see Lines 4–15). It finally returns $S(t)$ (see Line 17).

Algorithm 3 receives a simple weighted strongly connected digraph $G = (V, E)$, the weight $w_{(x,y)}$ of every edge $(x, y) \in E$, the length d_{xy} of the shortest paths for every pair of $x \in V$ and $y \in V$, an edge (u, v) to be deleted from G, and a source $s \in V$ as inputs, and returns the set of affected targets $T(s)$ as an output. The basic idea of this algorithm is the same as Algorithm 2.

Algorithm 4 receives a simple weighted strongly connected digraph $G = (V, E)$, the weight $w_{(x,y)}$ of every edge $(x, y) \in E$, the length d_{xy} and the number σ_{xy} of the shortest paths for every pair of $x \in V$ and $y \in V$, the BC score B_x for all $x \in V$, a vertex $s \in V$, the set $T(s)$ of affected targets of s, and $M \in \{-1, 1, -2, 2\}$ as inputs, and returns the updated BC score B_x for all $x \in V$. The value of M determines the mode of this algorithm. When $M = -1$ (1, resp.), it decrements (increments, resp.) B_x by $\Delta_{s\bullet}(x)$ for all $x \in V \setminus \{s\}$. The values $M = -2$ and $M = 2$ are used only when Algorithm 1 is extended so that it can handle undirected graphs (see Section 4.2 for details). Algorithm 4 is based on a theorem

Algorithm 3 Find affected targets of a source in a digraph

Input: $G = (V, E)$, $w_{(x,y)}$ for all $(x, y) \in E$, d_{xy} for all $x, y \in V$, $(u, v) \in E$, $s \in V$
Output: $T(s)$

```
1.  T(s) ← ∅
2.  if d_sv = d_su + w_(u,v) then
3.      T(s) ← {v}
4.      create an empty stack A and then push v to A
5.      R ← ∅
6.      while A is nonempty do
7.          pop x from A
8.          for y ∈ S_G(x) \ R do
9.              if d_sy = d_su + w_(u,v) + d_vy then
10.                 T(s) ← T(s) ∪ {y}
11.                 push y to A
12.             end if
13.         end for
14.         R ← R ∪ {x}
15.     end while
16. end if
17. return T(s)
```

given by Bergamini *et al.* [27] which says that $\Delta_{s\bullet}(x)$ is expressed in terms of $\Delta_{s\bullet}(y)$ with $y \in \mathcal{S}_{G_s}(x)$ as

$$\Delta_{s\bullet}(x) = \sum_{y \in \mathcal{S}_{G_s}(x) \cap T(s)} \frac{\sigma_{sx}}{\sigma_{sy}} (1 + \Delta_{s\bullet}(y)) + \sum_{y \in \mathcal{S}_{G_s}(x) \setminus T(s)} \frac{\sigma_{sx}}{\sigma_{sy}} \Delta_{s\bullet}(y). \tag{3.5}$$

Algorithm 4 first initializes $\Delta_{s\bullet}(x)$ to zero for all $x \in V$ (see Lines 1–3). It next creates an empty priority queue Q (see Line 4), and then enqueues each $t \in T(s)$ to Q with priority d_{st} (see Lines 5–7). It then computes the right-hand side of (3.5) while traversing G_s from vertices in $T(s)$ to the source s in the opposite direction of edges, and decrements or increments, depending on the value of M, the BC score of each vertex in Q when dequeued (see Lines 8–23). It finally returns the new BC scores of all vertices.

Algorithm 5 receives a simple weighted strongly connected digraph $G = (V, E)$, the weight $w_{(x,y)}$ of every edge $(x, y) \in E$, the length d_{xy} and the number σ_{xy} of the shortest paths for every pair of $x \in V$ and $y \in V$, an edge (u, v) to be deleted, a vertex $s \in V$ and the set $T(s)$ of affected targets of s as inputs, and returns the new values of d_{xy} and σ_{xy} for all $x \in V$ and $y \in V$. It first checks whether $T(s)$ is empty or not (see Line 1). If $T(s)$ is empty then it just returns the values of d_{sx} and σ_{sx} for all $x \in V$ (see Line 25) because $G_s = G'_s$ follows from Proposition 3.1. Otherwise, it initializes the values of d_{sx} and σ_{sx} to H and 0, respectively, where H is a sufficiently large number (see Lines 2–4). This process is necessary only when $G' = (V, E \setminus \{(u, v)\})$ is not strongly connected (see Section 4 for details) and thus can be skipped under the assumption that G' is strongly connected. It then updates the values of d_{sx} and σ_{sx} in increasing order of distance from vertex s to vertex x in $G' = (V, E') = (V, E \setminus \{(u, v)\})$ using the idea of RR algorithm [33] (see Lines 5–24). It finally returns the updated values of d_{xy} and σ_{xy} for all $x \in V$ and $y \in V$ (see Line 25).

Algorithm 4 Update BC scores with respect to a source in a digraph

Input: $G = (V, E)$, $w_{(x,y)}$ for all $(x, y) \in E$, d_{xy} and σ_{xy} for all $x, y \in V$, B_x for all $x \in V$, $s \in V$, $T(s)$, $M \in \{-1, 1, -2, 2\}$

Output: B_x for all $x \in V$

1. **for** $x \in V$ **do**
2. $\quad \Delta_{s\bullet}(x) \leftarrow 0$
3. **end for**
4. create an empty priority queue Q
5. **for** $t \in T(s)$ **do**
6. $\quad$ enqueue t to Q with priority d_{st}
7. **end for**
8. **while** Q is nonempty **do**
9. $\quad$ dequeue x with the highest priority from Q
10. $\quad B_x \leftarrow B_x + M \cdot \Delta_{s\bullet}(x)$
11. $\quad$ **for** $y \in \mathcal{P}_G(x)$ **do**
12. $\quad\quad$ **if** $y \neq s$ and $d_{sx} = d_{sy} + w_{(y,x)}$ **then**
13. $\quad\quad\quad$ **if** $x \in T(s)$ **then**
14. $\quad\quad\quad\quad \Delta_{s\bullet}(y) \leftarrow \Delta_{s\bullet}(y) + (\sigma_{sy}/\sigma_{sx}) \cdot (1 + \Delta_{s\bullet}(x))$
15. $\quad\quad\quad$ **else**
16. $\quad\quad\quad\quad \Delta_{s\bullet}(y) \leftarrow \Delta_{s\bullet}(y) + (\sigma_{sy}/\sigma_{sx}) \cdot \Delta_{s\bullet}(x)$
17. $\quad\quad\quad$ **end if**
18. $\quad\quad\quad$ **if** y is not in Q **then**
19. $\quad\quad\quad\quad$ enqueue y to Q with priority d_{sy}.
20. $\quad\quad\quad$ **end if**
21. $\quad\quad$ **end if**
22. $\quad$ **end for**
23. **end while**
24. return B_x for all $x \in V$

3.3 *Complexity analysis*

Here, we examine the efficiency of the proposed algorithm in terms of time complexity. For the convenience of further discussion, we begin with introducing some notations. The set of vertices visited in Line 7 of Algorithm 1 is denoted by $\tau(s) = T(s) \cup \{x \mid \Delta_{s\bullet}(x) > 0\}$. Analogously, the set of vertices visited in Line 14 of Algorithm 1 is denoted by $\tau'(s) = T(s) \cup \{x \mid \Delta'_{s\bullet}(x) > 0\}$. For the digraph $G = (V, E)$ and a subset W of V, the sum of the number of vertices in W and their indegrees and outdegrees is denoted by $\|W\|_G$, that is, $\|W\|_G = |W| + \sum_{v \in W} (|\mathcal{P}_G(v)| + |\mathcal{S}_G(v)|)$. When considering only indegrees (outdegrees, resp.), we use the notation $\|W\|_G^{\text{in}}$ ($\|W\|_G^{\text{out}}$, resp.), that is, $\|W\|_G^{\text{in}} = |W| + \sum_{v \in W} |\mathcal{P}_G(v)|$ and $\|W\|_G^{\text{out}} = |W| + \sum_{v \in W} |\mathcal{S}_G(v)|$. For the digraph $G' = (V, E \setminus \{(u, v)\})$ and a subset W of V, we use the notations $\|W\|_{G'}$, $\|W\|_{G'}^{\text{in}}$ and $\|W\|_{G'}^{\text{out}}$ in the same way as above.

PROPOSITION 3.3 The time complexity of Algorithm 1 is given by

$$\mathcal{O}\left(\|S(v)\|_G^{\text{in}} + \sum_{s \in S(v)} \left(\|\tau(s)\|_G + |\tau(s)| \log |\tau(s)| + \left\|\tau'(s)\right\|_{G'}^{\text{in}} + \left|\tau'(s)\right| \log \left|\tau'(s)\right|\right)\right). \tag{3.6}$$

Algorithm 5 Update SSSPs with respect to a source in a digraph

Input: $G = (V, E)$, $w_{(x,y)}$ for all $(x, y) \in E$, d_{xy} and σ_{xy} for all $x, y \in V$, $(u, v) \in E$, $s \in V$, $T(s)$
Output: d_{sx} and σ_{sy} for all $x \in V$

1. **if** $T(s) \neq \varnothing$ **then**
2. **for** $x \in T(s)$ **do**
3. set $d_{sx} \leftarrow H$, where H is a sufficiently large number, and $\sigma_{sx} \leftarrow 0$
4. **end for**
5. $G' \leftarrow (V, E \setminus \{(u, v)\})$
6. create an empty priority queue Q
7. **for** $x \in T(s)$ **do**
8. **if** $\mathcal{P}_{G'}(x) \setminus T(s) \neq \varnothing$ **then**
9. enqueue x to Q with priority $\min_{y \in \mathcal{P}_{G'}(x) \setminus T(s)} \{d_{sy} + w_{(y,x)}\}$
10. **end if**
11. **end for**
12. **while** Q is nonempty **do**
13. dequeue x with the lowest priority d from Q
14. $d_{sx} \leftarrow d$
15. $\sigma_{sx} \leftarrow \sum_{y \in \{z \in \mathcal{P}_{G'}(x) \mid d_{sx} = d_{sz} + w_{(z,x)}\}} \sigma_{sy}$
16. **for** $y \in \mathcal{S}_{G'}(x) \cap T(s)$ **do**
17. **if** y does not exist in Q **then**
18. enqueue y to Q with priority $d_{sx} + w_{(x,y)}$
19. **else if** $d_{sx} + w_{(x,y)}$ is less than the priority of y in Q **then**
20. update the priority of y in Q to $d_{sx} + w_{(x,y)}$
21. **end if**
22. **end for**
23. **end while**
24. **end if**
25. return d_{sx} and σ_{sx} for all $x \in V$

Proof. In Algorithm 1, the process of finding $S(v)$ in Line 2 runs in $\mathcal{O}\left(\|S(v)\|_G^{\text{in}}\right)$ time because all vertices in $S(v)$ and their predecessors are scanned (see Algorithm 2). The process of finding $T(s)$ for all $s \in S(v)$ in Lines 3–5 runs in $\mathcal{O}\left(\sum_{s \in S(v)} \|T(s)\|_G^{\text{out}}\right)$ time, because for each $s \in S(v)$ all vertices in $T(s)$ and their successors are scanned (see Algorithm 3). Since $\|T(s)\|_G^{\text{out}} = |T(s)| + \sum_{v \in T(s)} |\mathcal{S}_G(v)| \leq |T(s)| + \sum_{v \in T(s)} (|\mathcal{P}_G(v)| + |\mathcal{S}_G(v)|) = \|T(s)\|_G \leq \|\tau(s)\|_G$, where the last inequality follows from the fact that $T(s) \subseteq \tau(s)$, we can replace $\mathcal{O}\left(\sum_{s \in S(v)} \|T(s)\|_G^{\text{out}}\right)$ in the previous sentence with $\mathcal{O}\left(\sum_{s \in S(v)} \|\tau(s)\|_G\right)$. In the following analysis, we assume that a binary max heap is used for the priority queue in Algorithm 4 and a binary min heap is used in Algorithm 5. Then, the process of decreasing the BC scores of all $s \in S(v)$ in Lines 6–8 runs in

$$\mathcal{O}\left(\sum_{s \in S(v)} \left(\|\tau(s)\|_G^{\text{in}} + |\tau(s)| \log |\tau(s)|\right)\right) \tag{3.7}$$

and the process of increasing the BC scores of all $s \in S(v)$ in Lines 13–15 runs in

$$\mathcal{O}\left(\sum_{s\in S(v)}\left(\left\|\tau'(s)\right\|_{G'}^{\text{in}} + \left|\tau'(s)\right|\log\left|\tau'(s)\right|\right)\right).$$

These results are based on the work of Bergamini *et al.* [27], but more precise because $\|\tau(s)\|_G$ and $\|\tau'(s)\|_{G'}$ in their work are replaced with $\|\tau(s)\|_G^{\text{in}}$ and $\|\tau'(s)\|_{G'}^{\text{in}}$, respectively. Since $\|\tau(s)\|_G^{\text{in}} = |\tau(s)|_G + \sum_{v\in\tau(s)}|\mathcal{P}_G(v)| \le |\tau(s)|_G + \sum_{v\in\tau(s)}(|\mathcal{P}|_G(v) + |\mathcal{S}_G(v)|) = \|\tau(s)\|_G$, we can replace (3.7) with

$$\mathcal{O}\left(\sum_{s\in S(v)}(\|\tau(s)\|_G + |\tau(s)|\log|\tau(s)|)\right). \tag{3.8}$$

The time complexity for updating the values of d_{sx} and σ_{sx} for all pairs $(s,x) \in S(v)\times T(s)$ in Lines 9–11 is given by

$$\mathcal{O}\left(\sum_{s\in S(v)}(\|T(s)\|_{G'} + |T(s)|\log|T(s)|)\right). \tag{3.9}$$

This result is based on the work of RR [33]. Their assumption on how to implement the priority queue is different from ours, but it is easy to see that the same conclusion is reached. Since $\|T(s)\|_{G'} \le \|T(s)\|_G \le \|\tau(s)\|_G$ and $|T(s)| \le |\tau(s)|$ follow from the fact that $E' \subset E$ and $T(s) \subseteq \tau(s)$, we can replace (3.9) with (3.8).

Summarizing the observations above, we can conclude that the computational complexity of Algorithm 1 is given by (3.6). □

Some remarks should be made concerning Proposition 3.3. The first one is about the worst-case complexity. Substituting $\|S(v)\|_G^{\text{in}} = |V| + |E|$, $|S(v)| = |V|$, $\|\tau(s)\|_G = |V| + |E|$, $|\tau(s)| = |V|$, $\|\tau'(s)\|_{G'}^{\text{in}} = |V| + |E|$ and $|\tau'(s)| = |V|$ into (3.6), we have $\mathcal{O}(|V||E| + |V|^2\log|V|)$. Hence, the worst-case complexity of Algorithm 1 is equal to that of Brandes algorithm. However, the execution time of the proposed algorithm is expected to be much shorter than the worst case because $S(\cdot)$ and $T(\cdot)$ contain a very small number of vertices in general.

The second one is about the time complexity in the case where G is unweighted, that is, all edges of G have weight 1. In this case, we may be able to further reduce the execution time of the proposed algorithm by using another heap to implement the priority queues in Algorithms 4 and 5. In fact, if a relaxed heap [43] or a radix heap [44] is used for example, enqueueing of a new item and updating the priority of an existing item can be done in constant amortized time. However, the time complexity does not change because these heaps take $\mathcal{O}(\log p)$ time to dequeue the item with the highest priority costs, where p is the number of items in the heap [33].

The third one is about redundant operations. For each $s \in S(v)$, Algorithm 1 visits all vertices in $\tau(s)$ in Line 7 and those in $\tau'(s)$ in Line 14 to update their BC scores. These steps may contain a lot of redundant operations. This can happen when $\Delta_{s\bullet}(x) = \Delta'_{s\bullet}(x) > 0$ holds for many $x \in \tau(s)\cup\tau'(s)$ because many pairs of updates of the BC score B_x in Lines 7 and 14 cancel out each other. To see this, let us consider the digraph shown in Fig. 1 and suppose that the directed edge (u,v) is deleted. The set of all edges directed from left to right shows the SSSPs from s in G, and the set of those edges

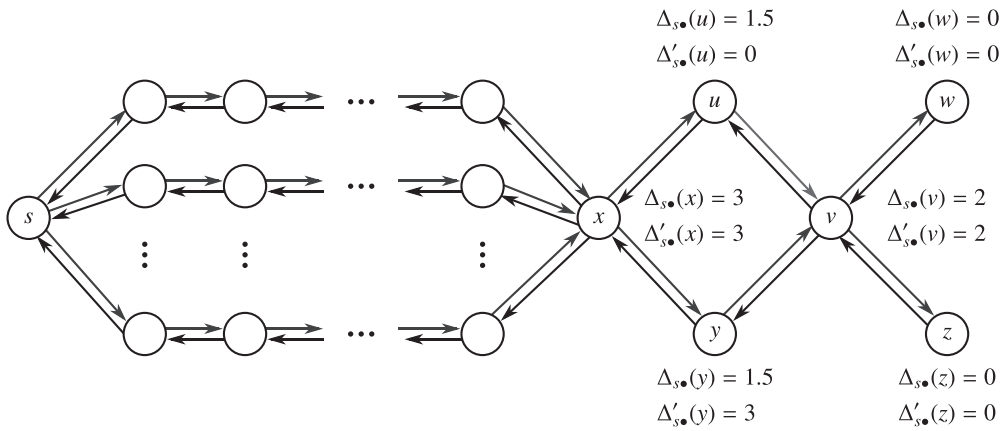

FIG. 1. An example of a digraph which causes many redundant operations in Algorithm 1.

except (u, v) shows the SSSPs from s in G'. It is easily seen from the figure that the set of affected targets of s is $T(s) = \{v, w, z\}$. Also, it is seen from the figure that $\Delta_{s\bullet}(v) = \sum_{t\in\{w,z\}} \sigma_{st}(v)/\sigma_{st} = 2$, $\Delta_{s\bullet}(u) = \sum_{t\in\{v,w,z\}} \sigma_{st}(u)/\sigma_{st} = 1/2 + 1/2 + 1/2 = 3/2$, $\Delta_{s\bullet}(y) = \sum_{t\in\{v,w,z\}} \sigma_{st}(y)/\sigma_{st} = 1/2 + 1/2 + 1/2 = 3/2$, $\Delta_{s\bullet}(x) = \sum_{t\in\{v,w,z\}} \sigma_{st}(x)/\sigma_{st} = 3$ and $\Delta'_{s\bullet}(v) = \sum_{t\in\{w,z\}} \sigma'_{st}(v)/\sigma'_{st} = 2 = \Delta_{s\bullet}(v)$, $\Delta'_{s\bullet}(u) = \sum_{t\in\{v,w,z\}} \sigma'_{st}(u)/\sigma'_{st} = 0$, $\Delta'_{s\bullet}(y) = \sum_{t\in\{v,w,z\}} \sigma'_{st}(y)/\sigma'_{st} = 3$ and $\Delta'_{s\bullet}(x) = \sum_{t\in\{v,w,z\}} \sigma'_{st}(x)/\sigma'_{st} = 3 = \Delta_{s\bullet}(x)$. Furthermore, for any vertex $a \in V \setminus \{s, u, v, w, x, y, z\}$

$$\Delta_{s\bullet}(a) = \sum_{t\in T(s)\setminus\{a\}} \delta_{st}(a) = \sum_{t\in\{v,w,z\}} \frac{\sigma_{st}(a)}{\sigma_{st}} = \sum_{t\in\{v,w,z\}} \frac{\sigma_{sx}(a)\sigma_{xt}}{\sigma_{sx}\sigma_{xt}} = 3\frac{\sigma_{sx}(a)}{\sigma_{sx}} > 0,$$

$$\Delta'_{s\bullet}(a) = \sum_{t\in T(s)\setminus\{a\}} \delta'_{st}(a) = \sum_{t\in\{v,w,z\}} \frac{\sigma'_{st}(a)}{\sigma'_{st}} = \sum_{t\in\{v,w,z\}} \frac{\sigma_{sx}(a)\sigma'_{xt}}{\sigma_{sx}\sigma'_{xt}} = 3\frac{\sigma_{sx}(a)}{\sigma_{sx}} = \Delta_{s\bullet}(a).$$

Algorithm 1 thus visits all vertices in $\tau(s) = V \setminus \{s\}$ ($\tau'(s) = V \setminus \{s, u\}$, resp.) and updates their BC scores in Line 7 (Line 14, resp.) when vertex s is considered as a source. However, for any vertex in $V \setminus \{s, u, y\}$, these two updates are unnecessary because they cancel out with each other.

4. Extension of the proposed algorithm

In this section, we show that the algorithm proposed in the previous section is easily extended so that it can be applied to not strongly connected digraphs and undirected graphs.

4.1 *Extension to not strongly connected digraphs*

We first consider an extension of the proposed algorithm to not strongly connected digraphs. The definition of the BC of vertex x shown in (2.1) is directly applied to not strongly connected digraphs by assuming that $\sigma_{st}(x)/\sigma_{st} = 0$ when there is no directed path from s to t, or by using the convention that zero divided by zero is zero [6, 45]. The algorithm proposed in the previous section is also directly applied to not strongly connected digraphs by setting the value of d_{xy} to a sufficiently large number H for all pairs of vertices x and y such that $\sigma_{xy} = 0$ or there is no directed path from x to y. In what follows, we shall show

that Algorithms 2–5 work correctly when $H = |V| \cdot \max_{(x,y)\in E}\{w_{(x,y)}\}$ for example. It is clear that in this case H is greater than d_{xy} for any pair of x and y such that $\sigma_{xy} \neq 0$.

We first show that the output $S(t)$ of Algorithm 2 does not contain x if $\sigma_{xt} = 0$, and that the output $T(s)$ of Algorithm 3 does not contain y if $\sigma_{sy} = 0$. We consider only the former statement because the latter one can be analysed in the same way. If $\sigma_{xt} = 0$ then there exist two possible cases: (i) $\sigma_{vt} = 0$ and (ii) $\sigma_{vt} \neq 0$ and $\sigma_{xu} = 0$. In the first case, the condition $d_{ut} = w_{(u,v)} + d_{vt}$ in Line 2 is not satisfied independent of the value of d_{ut} because $w_{(u,v)} + d_{vt} = w_{(u,v)} + H > H \geq d_{ut}$. Note that the equal sign in the last inequality is necessary because $d_{ut} = H$ when $\sigma_{ut} = 0$. In the second case, it is clear from Lines 3–15 that only those vertices having a directed path to t can enter $S(t)$. Therefore, in both cases, vertex x such that $\sigma_{xt} = 0$ is never contained in $S(t)$. We next show that Algorithm 4 does not update B_y if $\sigma_{sy} = 0$. It is clear from Lines 12–21 that B_y is updated only when the condition $d_{sx} = d_{sy} + w_{(y,x)}$ in Line 12 holds. However, this is impossible because $d_{sy} + w_{(y,x)} = H + w_{(y,x)} > H \geq d_{sx}$. We then show that Algorithm 5 does not update d_{sx} and σ_{sx} if $\sigma_{sx} = 0$. Note that if $\sigma_{sx} = 0$ then x is not contained in $T(s)$. It is clear from Lines 2–4 and Lines 13–15 that, in order for d_{sx} and σ_{sx} to be updated, x must be a member of $T(s)$ or must be enqueued to the priority queue Q. In addition, it is clear from Lines 7–11 and Lines 16–22 that only vertices in $T(s)$ are enqueued to Q. Therefore, d_{sx} and σ_{sx} are not updated if $\sigma_{sx} = 0$. We finally show that Algorithm 5 updates d_{sx} and σ_{sx} to H and 0, respectively, for any $x \in T(s)$ such that the number of the shortest paths from s to x decreases to zero due to deletion of the edge (u, v). It is clear from Lines 2–4 that d_{sx} and σ_{sx} are set to H and 0, respectively. Since $\mathcal{P}_{G'}(x) \setminus T(s) = \varnothing$ (otherwise there is at least one directed path from s to x in G' which is a contradiction), x is not enqueued to Q in Line 9. In addition, it is easily seen from Lines 8–10 and Lines 16–22 that only those vertices having a directed path from s in G' are enqueued to Q. Therefore, d_{sx} and σ_{sx} are not updated after Line 4.

4.2 *Extension to undirected graphs*

We next consider an extension of the proposed algorithm to undirected graphs. An undirected graph G^{U} is defined as an ordered pair (V, E^{U}) where V and E^{U} are the sets of vertices and edges, respectively. Each member of E^{U} is an unordered pair $\{x, y\}$ of vertices x and y, which represents the undirected edge connecting x and y. G^{U} is said to be simple if it has neither self-loops nor multiple edges. G^{U} is said to be weighted if each edge $\{x, y\} \in E^{\mathrm{U}}$ has a weight (or length) denoted by $w_{\{x,y\}}$, which is assumed to be positive in this article. G^{U} is said to be connected if, for any pair of two vertices x and y, there is a path connecting x and y.

Let $G^{\mathrm{U}} = (V, E^{\mathrm{U}})$ be a simple weighted undirected graph, and suppose that an edge $\{u, v\} \in E^{\mathrm{U}}$ is deleted from G^{U}. Since G^{U} can be viewed as a digraph $G = (V, E)$ by replacing each undirected edge $\{x, y\} \in E^{\mathrm{U}}$ with two directed edges (x, y) and (y, x), and setting the weights of these edges as $w_{(x,y)} = w_{(y,x)} = w_{\{x,y\}}$, the definition of the BC for digraphs is directly applied to G^{U} using this correspondence. Also, it is easy to see that we can update the BC scores of all vertices in G^{U} by running Algorithm 1 on the corresponding digraph G twice, one for deletion of the edge (u, v) and the other for deletion of the edge (v, u). However, as shown later, we can obtain the same result by running a slightly modified version of Algorithm 1 only once.

In what follows, we first give some results of our analysis on the amount of change in the BC scores of each vertex in G^{U} upon deletion of the edge $\{u, v\}$, and then present the modified algorithm. Let $G = (V, E)$ be the digraph obtained from $G^{\mathrm{U}} = (V, E^{\mathrm{U}})$ in the way stated above. Let $G' = (V, E')$ be the digraph obtained from G by deleting the edge $(u, v) \in E$. We use the prime symbol ($'$) to denote quantities in G' like B'_x and d'_{st}. The set of affected targets of a source $s \in V$ and the one of affected sources of a target $t \in V$ due to deletion of the edge (u, v) from G are denoted by $T_{u\to v}(s)$ and $S_{u\to v}(t)$, respectively.

Let $G'' = (V, E'')$ be the digraph obtained from G by deleting the two edges (u, v) and (v, u), or from G' by deleting the edge (v, u). We use the double prime symbol ($''$) to denote quantities in G'' like B''_x and d''_{st}. The set of affected targets of a source $s \in V$ and the one of affected sources of a target $t \in V$ due to deletion of the two edges (u, v) and (v, u) from G are denoted by $T_{u\leftrightarrow v}(s)$ and $S_{u\leftrightarrow v}(t)$, respectively.

The next lemma gives relationships among the four sets $T_{u\leftrightarrow v}(s)$, $S_{u\leftrightarrow v}(t)$, $T_{u\to v}(s)$ and $S_{u\to v}(t)$.

LEMMA 4.1 For each $s \in V$, $T_{u\leftrightarrow v}(s)$ is given by

$$T_{u\leftrightarrow v}(s) = T_{u\to v}(s) \cup S_{u\to v}(s). \tag{4.1}$$

Also, for each $t \in V$, $S_{u\leftrightarrow v}(t)$ are given by

$$S_{u\leftrightarrow v}(t) = S_{u\to v}(t) \cup T_{u\to v}(t). \tag{4.2}$$

Proof. We consider only (4.1) because (4.2) can be proved in the same way. It is clear that $t \in T_{u\leftrightarrow v}(s)$ if and only if the set of all the shortest paths $G_{st} = (V_{st}, E_{st})$ contains at least one of the two edges (u, v) and (v, u), which implies that

$$T_{u\leftrightarrow v}(s) = \{t \in V \mid d_{st} = d_{su} + w_{(u,v)} + d_{vt} \text{ or } d_{st} = d_{sv} + w_{(v,u)} + d_{ut}\}.$$

The first condition $d_{st} = d_{su} + w_{(u,v)} + d_{vt}$ means that t is a member of $T_{u\to v}(s)$. The second condition $d_{st} = d_{sv} + w_{(v,u)} + d_{ut}$ is equivalent to $d_{ts} = d_{tu} + w_{(u,v)} + d_{vs}$ due to symmetry of G, and the latter equation means that t is a member of $S_{u\to v}(s)$. Therefore $T_{u\leftrightarrow v}(s)$ is given by (4.1). □

From Lemma 4.1, we obtain the following results which play important roles in the algorithm we propose later.

PROPOSITION 4.1 If $G''_{st} \neq G_{st}$, then s belongs to one of the two disjoint sets $S_{u\to v}(v)$ and $T_{u\to v}(u)$.

Proof. Suppose that $G''_{st} \neq G_{st}$. Then, we see from Lemma 4.1 that $s \in S_{u\to v}(t) \cup T_{u\to v}(t)$. If $s \in S_{u\to v}(t)$, $G'_{st} \neq G_{st}$ and thus $s \in S_{u\to v}(v)$ follows from Proposition 3.1. Similarly, if $s \in T_{u\to v}(t)$, $G'_{ts} \neq G_{ts}$ and thus $s \in T_{u\to v}(u)$ follows from Proposition 3.1. Therefore, $s \in S_{u\to v}(v) \cup T_{u\to v}(u)$. Hence, we only have to show that $S_{u\to v}(v)$ and $T_{u\to v}(u)$ are disjoint. We prove this by contradiction. Let us assume that there exists an x such that $x \in S_{u\to v}(v) \cap T_{u\to v}(u)$. Then $d_{xv} = d_{xu} + w_{(u,v)}$ and $d_{ux} = w_{(u,v)} + d_{vx}$ hold simultaneously. From these two equations and symmetry of G, we have $d_{xu} + w_{(u,v)} = d_{xu} - w_{(u,v)}$, which implies that $w_{(u,v)} = 0$. This is a contradiction. □

PROPOSITION 4.2 For each $x \in V$, the difference between B''_x and B_x is expressed in terms of B'_x and B_x as follows:

$$B''_x - B_x = 2(B'_x - B_x). \tag{4.3}$$

Proof. It follows from Proposition 4.1 that the difference between B''_x and B_x is expressed as follows:

$$B''_x - B_x$$

$$= \sum_{s \in (S_{u\to v}(v) \cup T_{u\to v}(u)) \setminus \{x\}} \left(\sum_{t \in T_{u\leftrightarrow v}(s) \setminus \{x\}} (\delta''_{st}(x) - \delta_{st}(x)) \right)$$

$$= \sum_{s \in S_{u\to v}(v) \setminus \{x\}} \left(\sum_{t \in T_{u\to v}(s) \setminus \{x\}} (\delta''_{st}(x) - \delta_{st}(x)) \right) + \sum_{s \in T_{u\to v}(u) \setminus \{x\}} \left(\sum_{t \in S_{u\to v}(s) \setminus \{x\}} (\delta''_{st}(x) - \delta_{st}(x)) \right).$$

Note that $s \in T_{u\to v}(u) \setminus \{x\}$ and $t \in S_{u\to v}(s) \setminus \{x\}$ if and only if $t \in S_{u\to v}(v) \setminus \{x\}$ and $s \in T_{u\to v}(t) \setminus \{x\}$. Using this fact and the symmetry of G and G'', the above equation can be rewritten as

$$\begin{aligned} B''_x - B_x &= 2 \sum_{s \in S_{u\to v}(v) \setminus \{x\}} \left(\sum_{t \in T_{u\to v}(s) \setminus \{x\}} (\delta''_{st}(x) - \delta_{st}(x)) \right) \\ &= 2 \sum_{s \in S_{u\to v}(v) \setminus \{x\}} \left(\sum_{t \in T_{u\to v}(s) \setminus \{x\}} (\delta''_{st}(x) - \delta'_{st}(x) + \delta'_{st} - \delta_{st}(x)) \right) \\ &= 2 \sum_{s \in S_{u\to v}(v) \setminus \{x\}} \left(\sum_{t \in T_{u\to v}(s) \setminus \{x\}} (\delta''_{st}(x) - \delta'_{st}(x)) \right) + 2 \sum_{s \in S_{u\to v}(v) \setminus \{x\}} \left(\sum_{t \in T_{u\to v}(s) \setminus \{x\}} (\delta'_{st}(x) - \delta_{st}(x)) \right). \end{aligned}$$

We now prove that $G''_{st} = G'_{st}$ for all pairs of s and t such that $s \in S_{u\to v}(v) \setminus \{x\}$ and $t \in T_{u\to v}(s) \setminus \{x\}$. If this is true, the first term of the above equation is zero and thus

$$B''_x - B_x = 2 \sum_{s \in S_{u\to v}(v) \setminus \{x\}} \left(\sum_{t \in T_{u\to v}(s) \setminus \{x\}} (\delta'_{st}(x) - \delta_{st}(x)) \right) = 2(B'_x - B_x)$$

holds. The proof is done by contradiction. Namely, we assume that $G''_{st} \neq G'_{st}$ for some $s \in S_{u\to v}(v) \setminus \{x\}$ and $t \in T_{u\to v}(s) \setminus \{x\}$, and then draw a contradiction. Since G'' is obtained from G' by deleting the edge (v, u), the condition $G''_{st} \neq G'_{st}$ implies that $d'_{su} = d'_{sv} + w_{(v,u)}$. Also, the condition $s \in S_{u\to v}(v) \setminus \{x\}$ implies that $d_{sv} = d_{su} + w_{(u,v)}$. In addition, since G_{su} does not contain the edge (u, v), we have $G_{su} = G'_{su}$ and thus $d_{su} = d'_{su}$. Furthermore, since G_{sv} contains the edge (u, v), we have $G'_{sv} \neq G_{sv}$ and thus $d'_{sv} \geq d_{sv}$. Using these relationships, we have $d_{sv} = d_{su} + w_{(u,v)} = d'_{su} + w_{(u,v)} = d'_{sv} + 2w_{(u,v)} \leq d'_{sv}$. From the last inequality, we have $w_{(u,v)} \leq 0$ which is a contradiction. □

Proposition 4.2 says that if we slightly modify Algorithm 1 then we can update the BC scores by running it only once. The modifications needed are as follows. First, we need to replace $\Delta_{s\bullet}(x)$ with $2\Delta_{s\bullet}(x)$ in the process of updating the BC scores in Lines 7 and 14. This is clear from Proposition 4.2. Second, after the process of updating the length d_{sx} and number σ_{sx} of all the shortest paths from s to x in Line 10, we need to copy d_{sx} and σ_{sx} to d_{xs} and σ_{xs}, respectively, because the set of the shortest paths from x to s, which is affected by deletion of the edge (v, u), is not considered. By applying these modifications to Algorithm 1, we obtain an algorithm for updating the BC scores of all vertices in G^{U} on deletion of an undirected edge $\{u, v\}$, which is shown in Algorithm 6.

Algorithm 6 Update BC scores upon deletion of an edge in an undirected graph

Input: $G^{\mathrm{U}} = (V, E^{\mathrm{U}})$, $w_{\{x,y\}}$ for all $\{x, y\} \in E^{\mathrm{U}}$, d_{xy} and σ_{xy} for all $x, y \in V$, B_x for all $x \in V$, $\{u, v\} \in E^{\mathrm{U}}$
Output: d_{xy} and σ_{xy} for all $x, y \in V$, B_x for all $x \in V$

1. Convert G^{U} into a digraph $G = (V, E)$ by replacing each edge $\{x, y\} \in E^{\mathrm{U}}$ with two directed edges (x, y) and (y, x) and setting $w_{(x,y)} = w_{(y,x)} = w_{\{x,y\}}$
2. **if** $d_{uv} = w_{(u,v)}$ **then**
3. find $S(v)$ using Algorithm 2 with $t = v$
4. **for** $s \in S(v)$ **do**
5. find $T(s)$ using Algorithm 3
6. **end for**
7. **for** $s \in S(v)$ **do**
8. set $B_x \leftarrow B_x - 2\Delta_{s\bullet}(x)$ for all $x \in V \setminus \{s\}$ using Algorithm 4 with $M = -2$
9. **end for**
10. **for** $s \in S(v)$ **do**
11. update d_{sx} and σ_{sx} for all $x \in T(s)$ using Algorithm 5
12. set $d_{xs} \leftarrow d_{sx}$ and $\sigma_{xs} \leftarrow \sigma_{sx}$ for all $x \in T(s)$
13. **end for**
14. set $G \leftarrow (V, E \setminus \{(u, v)\})$
15. **for** $s \in S(v)$ **do**
16. set $B_x \leftarrow B_x + 2\Delta_{s\bullet}(x)$ for all $x \in V \setminus \{s\}$ using Algorithm 4 with $M = 2$
17. **end for**
18. **end if**
19. return d_{xy} and σ_{xy} for all $x, y \in V$ and B_x for all $x \in V$

5. Experiments

In this section, we examine the efficiency of the proposed algorithm through experiments using synthetic and real networks. The proposed algorithm was written in C language[1], compiled with gcc 9.3.0 with -Ofast flag and with igraph 0.9.6, and ran in a single thread. The priority queue was implemented with a radix heap. All experiments were performed on a computer with an Intel® Core™ i7-10700 and 16GB RAM running Ubuntu 20.04.3 LTS.

5.1 *Performance evaluation on synthetic networks*

The authors first applied the proposed and Brandes algorithms to unweighted/weighted undirected graphs based on random regular graphs (RRGs) and those based on the Barabási–Albert (BA) model [46]. Since RRGs and graphs generated by the BA model are unweighted, they were converted to weighted graphs by assigning an integer weight randomly selected from 1 to 5 with equal probability to every edge. The number of vertices, denoted by n, was set to ten different values $100, 200, \ldots, 1000$, and the average degree, denoted by k, was set to two different values 4 and 8. For each value of the pair (n, k), 50 different graphs were generated. For each of the 50 graphs, an edge to be deleted was randomly selected 100

[1] The source codes are publicly available at https://github.com/y-satotani/dynamic-betweenness.

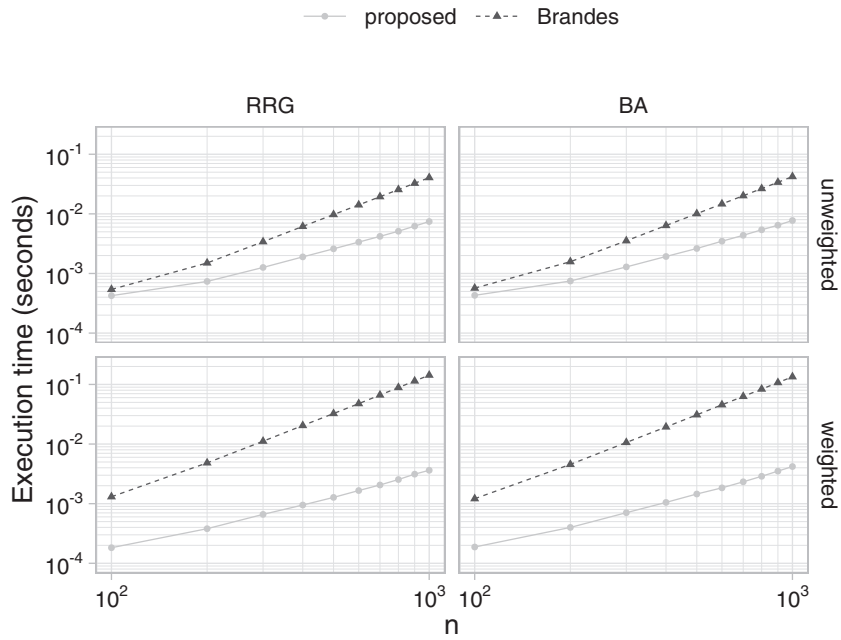

FIG. 2. Comparison of mean execution time on synthetic undirected graphs.

times. For each of the 100 selections of an edge, the two algorithms were run and the execution time was measured.

Figure 2 shows the relationship between the number of vertices and the mean execution time of the two algorithms on the four types of undirected graphs: (i) unweighted ones based on RRGs with $k = 4$ (upper left), (ii) weighted ones based on RRGs with $k = 4$ (lower left), (iii) unweighted ones based on the BA model with $k = 4$ (upper right), and (iv) weighted ones based on the BA model with $k = 4$ (lower right). We see from the figure that the proposed algorithm always outperforms the Brandes algorithm. Also, using the log–log regression analysis, we see that the mean execution time of the proposed algorithm is proportional to (i) $n^{1.28}$, (ii) $n^{1.31}$, (iii) $n^{1.30}$ and (iv) $n^{1.36}$ while the one of the Brandes algorithm is proportional to (i) $n^{1.92}$, (ii) $n^{2.06}$, (iii) $n^{1.92}$ and (iv) $n^{2.06}$. These observations suggest that the main idea behind the proposed algorithm, which is to update dependencies $\delta_{s\bullet}(x)$ only when it is necessary, is effective in practice.

Table 1 shows the mean and maximum execution time of the proposed and Brandes algorithms on synthetic unweighted graphs with $n \in \{500, 1000\}$. It is seen from the table that the proposed algorithm is about 3.8–6.8 times faster than the Brandes algorithm on average. However, the maximum execution time of the proposed algorithm is longer than that of the Brandes algorithm in some cases. One possible reason for this is that the proposed algorithm needs to allocate and free memory many times when many vertices are affected by the edge deletion. Table 2 shows the mean and maximum execution times of the two algorithms on synthetic weighted graphs with $n \in \{500, 1000\}$. It is seen from the table that the proposed algorithm is about 21.3–68.9 times faster than the Brandes algorithm on average. In addition,

TABLE 1 *Execution time (in milliseconds) on synthetic unweighted graphs*

Model	n	k	Proposed		Brandes	
			Mean	Maximum	Mean	Maximum
RRG	500	4	2.593	4.300	9.730	11.120
RRG	500	8	2.300	4.381	13.807	15.687
RRG	1000	4	7.433	11.141	40.531	44.396
RRG	1000	8	7.925	12.015	53.938	58.554
BA	500	4	2.625	15.992	10.070	11.464
BA	500	8	3.399	13.938	13.492	14.888
BA	1000	4	7.738	61.783	42.183	45.260
BA	1000	8	10.679	60.935	56.996	71.297

TABLE 2 *Execution time (in milliseconds) on synthetic weighted graphs*

Model	n	k	Proposed		Brandes	
			Mean	Maximum	Mean	Maximum
RRG	500	4	1.274	6.014	32.492	37.904
RRG	500	8	0.948	5.674	41.992	59.189
RRG	1000	4	3.629	15.197	142.977	164.676
RRG	1000	8	2.719	14.771	187.212	197.707
BA	500	4	1.452	12.939	30.916	33.208
BA	500	8	1.216	13.537	41.307	44.678
BA	1000	4	4.195	78.782	134.309	145.359
BA	1000	8	3.822	68.322	180.481	190.646

unlike the case of unweighted graphs, the maximum execution time of the proposed algorithm is shorter than that of the Brandes algorithm in all cases.

5.2 *Performance evaluation on real networks*

The authors next applied the proposed and Brandes algorithms to various real networks taken from Stanford large network dataset collection [47]. This collection has more than 50 large networks such as social networks, web graphs, road networks, Internet, citation networks, collaboration networks and communication networks. Among them, five collaboration networks [48] (beginning with 'ca' in Table 3), six networks collected by the GEMSEC project [49] (beginning with 'gemsec'), 10 networks collected by the MUSAE project [50] (beginning with 'musae'), two trust networks on Bitcoin platform [51, 52] (beginning with 'soc-sign-bitcoin') and two networks related to Wikipedia [53–55] (beginning with 'wiki') were selected. The two trust networks on the Bitcoin platform are directed and weighted, but their weights were ignored in this experiment because some of the weights are negative. Namely, these two were regarded as directed and unweighted networks in this experiment. The two networks related

TABLE 3 *Mean execution time on real networks (in seconds)*

Original network	Vertices	Edges	Proposed	Brandes
ca-AstroPh	18772	198110	1.340	31.790
ca-CondMat	23133	93497	0.991	29.860
ca-GrQc	5242	14496	0.035	0.858
ca-HepPh	12008	118521	0.405	9.620
ca-HepTh	9877	25998	0.166	4.261
gemsec-facebook-athletes	13866	86858	0.698	18.839
gemsec-facebook-company	14113	52310	0.499	15.643
gemsec-facebook-government	7057	89455	0.328	6.312
gemsec-facebook-politician	5908	41729	0.086	3.127
gemsec-facebook-public-figure	11565	67114	0.602	12.275
gemsec-facebook-tvshow	3892	17262	0.041	1.045
musae-facebook	22470	171002	1.697	52.239
musae-twitch-DE	9498	153138	1.122	12.979
musae-twitch-ENGB	7126	35324	0.364	4.086
musae-twitch-ES	4648	59382	0.318	2.651
musae-twitch-FR	6549	112666	0.765	6.398
musae-twitch-PTBR	1912	31299	0.073	0.439
musae-twitch-RU	4385	37304	0.318	1.845
musae-wikipedia-chameleon	2277	31421	0.015	0.488
musae-wikipedia-crocodile	11631	170918	3.821	16.016
musae-wikipedia-squirrel	5201	198493	0.392	5.040
soc-sign-bitcoinalpha	3783	24186	0.084	0.685
soc-sign-bitcoinotc	5881	35592	0.162	1.621
wiki-RfA	11381	189003	0.316	5.193
wiki-Vote	7115	103689	0.060	1.331

to Wikipedia are directed and unweighted. All other networks are undirected and unweighted. For each of the graphs and digraphs obtained from the selected networks, an edge to be deleted was randomly selected 100 times. For each of the 100 selections of an edge, the proposed and Brandes algorithms were run and the execution time was measured.

Table 3 shows the mean execution time of the two algorithms. It is seen from the table that the proposed algorithm is about 4.2–36.5 times faster than the Brandes algorithm on average. Therefore, we can conclude that the proposed algorithm is practically useful for updating BC scores when an edge is deleted from a large network.

6. Conclusions

In this article, we proposed an algorithm for updating the BC scores of all vertices in a graph when an edge is deleted. The proposed algorithm is based on the results of the theoretical analysis presented in this article, the algorithm of Bergamini *et al.* for updating the BC scores, and the RR algorithm for updating SSSPs. To the best of the authors' knowledge, this is the first algorithm that makes use of the

RR algorithm for this purpose. We showed through experiments using RRGs and graphs based on the BA model with two values of the average degree that the proposed algorithm and the Brandes algorithm have mean execution times proportional to about $n^{1.3}$ and $n^{2.0}$, respectively. We also showed through experiments using datasets of various real networks that the proposed algorithm runs much faster than the Brandes algorithm. These results indicate that the number of affected sources and targets is very small on average in a variety of graphs. Therefore, the proposed algorithm combined with, for example, the algorithm of Bergamini *et al.* is useful for updating the BC scores of large dynamic graphs.

Although the proposed algorithm shows very good performance on large graphs, it may contain many redundant operations as explained in Section 3.3. Hence reducing the number of redundant operations in the algorithm is a future challenge. Another challenge is to extend the algorithm so that it can cope with the simultaneous deletion of multiple edges.

Funding

Japan Society for the Promotion of Science (JSPS) KAKENHI (JP21H03510).

References

1. Molontay, R. & Nagy, M. (2021) Twenty years of network science: a bibliographic and co-authorship network analysis. *Big Data and Social Media Analytics* (M. Çakırtaş & M. K. Ozdemir eds). Cham: Springer, pp. 1–24.
2. Madotto, A. & Liu, J. (2016) Super-spreader identification using meta-centrality. *Sci. Rep.*, **6**, 1–10.
3. Kiss, C. & Bichler, M. (2008) Identification of influencers-measuring influence in customer networks. *Decis. Support Syst.*, **46**, 233–253.
4. Guimera, R., Mossa, S., Turtschi, A. & Amaral, L. N. (2005) The worldwide air transportation network: anomalous centrality, community structure, and cities' global roles. *Proc. Natl. Acad. Sci. USA*, **102**, 7794–7799.
5. Oldham, S., Fulcher, B., Parkes, L., Arnatkevičiūtė, A., Suo, C. & Fornito, A. (2019) Consistency and differences between centrality measures across distinct classes of networks. *PLoS One*, **14**, e0220061.
6. Freeman, L. C. (1977) A set of measures of centrality based on betweenness. *Sociometry*, **40**, 35–41.
7. Agryzkov, T., Tortosa, L. & Vicent, J. F. (2019) A variant of the current flow betweenness centrality and its application in urban networks. *Appl. Math. Comput.*, **347**, 600–615.
8. Tizghadam, A. & Leon-Garcia, A. (2010) Betweenness centrality and resistance distance in communication networks. *IEEE Netw.*, **24**, 10–16.
9. Brandes, U. (2001) A faster algorithm for betweenness centrality. *J. Math. Sociol.*, **25**, 163–177.
10. Bader, D. A. & Madduri, K. (2006) Parallel algorithms for evaluating centrality indices in real-world networks. *Proceedings of the 2006 International Conference on Parallel Processing*. IEEE, pp. 539–550.
11. Bernaschi, M., Carbone, G. & Vella, F. (2015) Betweenness centrality on multi-GPU systems. *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. ACM, pp. 12:1–12:4.
12. Edmonds, N., Hoefler, T. & Lumsdaine, A. (2010) A space-efficient parallel algorithm for computing betweenness centrality in distributed memory. *Proceedings of the 17th International Conference on High Performance Computing*. IEEE, pp. 1–10.
13. Sariyüce, A. E., Kaya, K., Saule, E. & Çatalyürek, Ü. V. (2013) Betweenness centrality on GPUs and heterogeneous architectures. *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*. ACM, pp. 76–85.
14. Shi, Z. & Zhang, B. (2011) Fast network centrality analysis using GPUs. *BMC Bioinformatics*, **12**, 1–7.
15. Tan, G., Tu, D. & Sun, N. (2009) A parallel algorithm for computing betweenness centrality. *Proceedings of the 2009 International Conference on Parallel Processing*. IEEE, pp. 340–347.
16. Bader, D. A., Kintali, S., Madduri, K. & Mihail, M. (2007) Approximating betweenness centrality. *Algorithms and Models for the Web-Graph* (A. Bonato & F. R. K. Chung eds). Berlin, Heidelberg: Springer, pp. 124–137.
17. Borassi, M. & Natale, E. (2019) KADABRA is an adaptive algorithm for betweenness via random approximation. *J. Exp. Algorithmics*, **24**, 1–35.

18. Brandes, U. & Pich, C. (2007) Centrality estimation in large networks. *Int. J. Bifurc. Chaos*, **17**, 2303–2318.

19. Chehreghani, M. H. (2014) An efficient algorithm for approximate betweenness centrality computation. *Comput. J.*, **57**, 1371–1382.

20. Geisberger, R., Sanders, P. & Schultes, D. (2008) Better approximation of betweenness centrality. *Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, pp. 90–100.

21. Riondato, M. & Kornaropoulos, E. M. (2014) Fast approximation of betweenness centrality through sampling. *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. ACM, pp. 413–422.

22. Riondato, M. & Upfal, E. (2016) ABRA: approximating betweenness centrality in static and dynamic graphs with Rademacher averages. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1145–1154.

23. Pfeffer, J. & Carley, K. M. (2012) k-Centralities: local approximations of global measures based on shortest paths. *Proceedings of the 21st Annual Conference on World Wide Web*. ACM, pp. 1043–1050.

24. Yoshida, Y. (2014) Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1416–1425.

25. Holme, P. & Saramäki, J. (2012) Temporal networks. *Phys. Rep.*, **519**, 97–125.

26. Lee, M., Lee, J., Park, J. Y., Choi, R. H. & Chung, C.-W. (2012) QUBE: a quick algorithm for Updating betweenness centrality. *Proceedings of the 21st International Conference on World Wide Web*. ACM, pp. 351–360.

27. Bergamini, E., Meyerhenke, H., Ortmann, M. & Slobbe, A. (2017) Faster betweenness centrality updates in evolving networks. *Proceedings of the 16th International Symposium on Experimental Algorithms*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)* (C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi & R. Raman eds). Wadern, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 23:1–23:16.

28. Green, O., McColl, R. & Bader, D. A. (2012) A fast algorithm for streaming betweenness centrality. *Proceedings of the 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*. IEEE, pp. 11–20.

29. Kas, M., Wachs, M., Carley, K. M. & Carley, L. R. (2013) Incremental algorithm for updating betweenness centrality in dynamically growing networks. *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE/ACM, pp. 33–40.

30. Nasre, M., Pontecorvi, M. & Ramachandran, V. (2014) Betweenness centrality – incremental and faster. *Mathematical Foundations of Computer Science 2014* (E. Csuhaj-Varjú, M. Dietzfelbinger & Z. Ésik eds). Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 577–588.

31. Nasre, M., Pontecorvi, M. & Ramachandran, V. (2014) Decremental all-pairs all shortest paths and betweenness centrality. *Algorithms and Computation* (H.-K. Ahn & C.-S. Shin eds). Cham: Springer Cham, pp. 766–778.

32. Pontecorvi, M. & Ramachandran, V. (2015) Fully dynamic betweenness centrality. *Algorithms and Computation* (K. Elbassioni & K. Makino eds). Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 331–342.

33. Ramalingam, G. & Reps, T. (1996) On the computational complexity of dynamic graph problems. *Theor. Comput. Sci.*, **158**, 233–277.

34. Karger, D. R., Koller, D. & Phillips, S. J. (1993) Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM J. Comput.*, **22**, 1199–1217.

35. Demetrescu, C. & Italiano, G. F. (2003) A new approach to dynamic all pairs shortest paths. *Proceedings of the 35th ACM Symposium on Theory of Computing*. ACM, pp. 159–166.

36. Singh, R. R., Goel, K., Iyengar, S. R. & Gupta, S. (2015) A faster algorithm to update betweenness centrality after node alteration. *Internet Math.*, **11**, 403–420.

37. Kourtellis, N., De Francisci Morales, G. & Bonchi, F. (2015) Scalable online betweenness centrality in evolving graphs. *IEEE Trans. Knowl. Data Eng.*, **27**, 2494–2506.

38. Bergamini, E. & Meyerhenke, H. (2015) Fully-dynamic approximation of betweenness centrality. *Algorithms - ESA 2015* (N. Bansal & I. Finocchi eds). Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 155–166.

39. Bergamini, E., Meyerhenke, H. & Staudt, C. L. (2015) Approximating betweenness centrality in large evolving networks. *Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, pp. 133–146.
40. Chernoskutov, M., Ineichen, Y. & Bekas, C. (2015) Heuristic algorithm for approximation betweenness centrality using graph coarsening. *Proc. Comput. Sci.*, **66**, 83–92.
41. Hayashi, T., Akiba, T. & Yoshida, Y. (2015) Fully dynamic betweenness centrality maintenance on massive networks. *Proc. VLDB Endowm.*, **9**, 48–59.
42. Jamour, F., Skiadopoulos, S. & Kalnis, P. (2017) Parallel algorithm for incremental betweenness centrality on large graphs. *IEEE Trans. Parallel Distrib. Syst.*, **29**, 659–672.
43. Driscoll, J. R., Gabow, H. N., Shrairman, R. & Tarjan, R. E. (1988) Relaxed heaps: an alternative to Fibonacci heaps with applications to parallel computation. *Commun. ACM*, **31**, 1343–1354.
44. Ahuja, R. K., Mehlhorn, K., Orlin, J. & Tarjan, R. E. (1990) Faster algorithms for the shortest path problem. *J. ACM*, **37**, 213–223.
45. Brandes, U. (2008) On variants of shortest-path betweenness centrality and their generic computation. *Soc. Netw.*, **30**, 136–145.
46. Barabási, A.-L. & Albert, R. (1999) Emergence of scaling in random networks. *Science*, **286**, 509–512.
47. Leskovec, J. & Sosič, R. (2016) SNAP: a general purpose network analysis and graph mining library. *ACM Trans. Intell. Syst. Technol.*, **8**, 1–20.
48. Leskovec, J., Kleinberg, J. & Faloutsos, C. (2007) Graph evolution: densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, **1**, 2–es.
49. Rozemberczki, B., Davies, R., Sarkar, R. & Sutton, C. (2019) GEMSEC: graph embedding with self clustering. *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE/ACM, pp. 65–72.
50. Rozemberczki, B., Allen, C. & Sarkar, R. (2021) Multi-scale attributed node embedding. *J. Compl. Netw.*, **9**.
51. Kumar, S., Hooi, B., Makhija, D., Kumar, M., Faloutsos, C. & Subrahmanian, V. (2018) Rev2: fraudulent user prediction in rating platforms. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, pp. 333–341.
52. Kumar, S., Spezzano, F., Subrahmanian, V. & Faloutsos, C. (2016) Edge weight prediction in weighted signed networks. *Proceedings of the 2016 IEEE 16th International Conference on Data Mining*. IEEE, pp. 221–230.
53. Leskovec, J., Huttenlocher, D. & Kleinberg, J. (2010) Predicting positive and negative links in online social networks. *Proceedings of the 19th International Conference on World Wide Web*. ACM, pp. 641–650.
54. Leskovec, J., Huttenlocher, D. & Kleinberg, J. (2010) Signed networks in social media. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 1361–1370.
55. West, R., Paskov, H. S., Leskovec, J. & Potts, C. (2014) Exploiting social network structure for person-to-person sentiment analysis. *Trans. Assoc. Comput. Linguist.*, **2**, 297–310.