An Adaptive Version of Brandes' Algorithm for Betweenness Centrality

Matthias Bentert
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
matthias.bentert@tu-berlin.de

Alexander Dittmann
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
alexander.dittmann@campus.tu-berlin.de

Leon Kellerhals[1]
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
leon.kellerhals@tu-berlin.de

André Nichterlein
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
andre.nichterlein@tu-berlin.de

Rolf Niedermeier
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
rolf.niedermeier@tu-berlin.de

Abstract

Betweenness centrality – measuring how many shortest paths pass through a vertex – is one of the most important network analysis concepts for assessing the relative importance of a vertex. The well-known algorithm of Brandes [2001] computes, on an n-vertex and m-edge graph, the betweenness centrality of all vertices in $O(nm)$ worst-case time. In follow-up work, significant empirical speedups were achieved by preprocessing degree-one vertices and by graph partitioning based on cut vertices. We further contribute an algorithmic treatment of degree-two vertices, which turns out to be much richer in mathematical structure than the case of degree-one vertices. Based on these three algorithmic ingredients, we provide a strengthened worst-case running time analysis for betweenness centrality algorithms. More specifically, we prove an adaptive running time bound $O(kn)$, where $k < m$ is the size of a minimum feedback edge set of the input graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases network science, social network analysis, centrality measures, shortest paths, tree-like graphs, efficient pre- and postprocessing, FPT in P

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.36

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.06701`.

[1] Supported by DFG project FPTinP, NI 369/16.

29th International Symposium on Algorithms and Computation (ISAAC 2018).
Editors: Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao; Article No. 36; pp. 36:1–36:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

One of the most important building blocks in network analysis is to determine a vertex's relative importance in the network. A key concept herein is *betweenness centrality* as introduced in 1977 by Freeman [6]; it measures centrality based on shortest paths. Intuitively, for each vertex, betweenness centrality counts the (relative) number of shortest paths that pass through the vertex. A straightforward algorithm for computing the betweenness centrality on undirected (unweighted) n-vertex graphs runs in $\Theta(n^3)$ time, and improving this to $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$ would break the so-called APSP-conjecture [1]. In 2001, Brandes [3] presented the to date theoretically fastest algorithm, improving the running time to $O(nm)$ for graphs with m edges. As many real-world networks are sparse, this is a far-reaching improvement, having a huge impact also in practice. Newman [9] presented a high-level description of an algorithm for a variant of betweenness centrality running in $O(nm)$ time.

Our work is in line with numerous research efforts concerning the development of algorithms for computing betweenness centrality. Formally, we study the following problem:

Betweenness Centrality
Input: An undirected graph G.
Task: Compute the *betweenness centrality* $C_B(v) := \sum_{s,t \in V(G)} \frac{\sigma_{st}(v)}{\sigma_{st}}$ for each vertex $v \in V(G)$.

Herein, σ_{st} is the number of shortest paths in G from vertex s to vertex t, and $\sigma_{st}(v)$ is the number of shortest paths from s to t that additionally pass through v.[2]

Extending previous, more empirically oriented work of Baglioni et al. [2], Puzis et al. [12], and Sariyüce et al. [13] (see Section 2 for a description of their approaches), our main result is the mathematically rigorous analysis of an algorithm for Betweenness Centrality that runs in $O(kn)$ time, where k denotes the feedback edge number of the input graph G. The *feedback edge number* of G is the minimum number of edges to be deleted from G in order to make it a forest.[3] Clearly, $k = 0$ holds on trees, and $k \leq m$ holds in general. Thus our algorithm is *adaptive*, i.e., it interpolates between linear time for constant k and the running time of the best unparameterized algorithm for k approaching m. Obviously, by depth-first search one can compute k in linear time; however, $k \approx m - n$, so we provide no asymptotic improvement over Brandes' algorithm for most graphs. When the input graph is very tree-like ($m = n + o(n)$), however, our new algorithm improves on Brandes' algorithm. Real-world networks showing the relation between PhD candidates and their supervisors [4, 8] or the ownership relation between companies [11] typically have a feedback edge number that is smaller than the number of vertices or edges by orders of magnitude [10]. For roughly half of their networks, $m - n$ is smaller than n by at least one order of magnitude.

Our algorithmic contribution is to complement the works of Baglioni et al. [2], Puzis et al. [12], and Sariyüce et al. [13] by, roughly speaking, additionally dealing with degree-two vertices. These vertices are much harder to cope with and to analyze since, other than degree-one vertices, they may lie on shortest paths between two vertices. Recently, Vella et al. [14] used a heuristic approach to process degree-two vertices for improving the performance of their Betweenness Centrality algorithms on several real-world networks.

---

[2] To simplify our matters, we set $\sigma_{st}(v) = 0$ if $v = s$ or $v = t$. This is equivalent to Brandes [3] but differs from Newman [9], where $\sigma_{st}(s) = 1$.

[3] Notably, Betweenness Centrality computations have also been studied when the input graph is a tree [15], hinting at the practical relevance of this special case.

Our work is purely theoretical in spirit. Our most profound contribution is to analyze the worst-case running time of the proposed betweenness centrality algorithm based on degree-one-vertex processing [2], usage of cut vertices [12, 13], and our degree-two-vertex processing. To the best of our knowledge, this provides the first proven worst-case "improvement" over Brandes' upper bound in a relevant special case.

Notation. We use mostly standard graph notation. Given a graph G, $V(G)$ and $E(G)$ denote the vertex respectively edge set of G with $n = |V(G)|$ and $m = |E(G)|$. We denote the vertices of degree one, two, and at least three by $V^{=1}(G)$, $V^{=2}(G)$, and $V^{\geq 3}(G)$, respectively. A *cut vertex* or *articulation vertex* is a vertex whose removal disconnects the graph. A connected component of a graph is *biconnected* if it does not contain any cut vertices, and hence, no vertices of degree one. A path $P = v_0 \dots v_q$ is a graph with $V(P) = \{v_0, \dots, v_q\}$ and $E(P) = \{\{v_i, v_{i+1}\} \mid 0 \leq i < q\}$. The length of the path P is $|E(P)|$. Adding the edge $\{v_q, v_0\}$ to P gives a cycle $C = v_0 \dots v_q v_0$. The distance $d_G(s,t)$ between vertices $s, t \in V(G)$ is the length of the shortest path between s and t in G. The number of shortest s-t–paths is denoted by σ_{st}. The number of shortest s-t–paths containing some vertex v is denoted by $\sigma_{st}(v)$. We set $\sigma_{st}(v) = 0$ if $s = v$ or $t = v$ (or both). Lastly, for $j \leq k$ we set $[j,k] := \{j, j+1, \dots, k\}$.

2 Algorithm overview

In this section, we review our algorithmic strategy to compute the betweenness centrality of each vertex. Before doing so, since we build on the works of Brandes [3], Baglioni et al. [2], Puzis et al. [12], and Sariyüce et al. [13], we first give the high-level ideas behind their algorithmic approaches. Then, we describe the ideas behind our extension. We remark that we assume throughout our paper that the input graph is connected. Otherwise, we can process the connected components one after another.

Existing algorithmic approaches. Brandes [3] developed an $O(nm)$-time algorithm which essentially runs modified breadth-first searches (BFS) from each vertex of the graph. In each of these modified BFS, Brandes' algorithm computes the "effect" that the starting vertex s of the modified BFS has on the betweenness centrality values of all other vertices. More formally, the modified BFS starting at vertex s computes $\sum_{t \in V(G)} \sigma_{st}(v)/\sigma_{st}$ for each vertex $v \in V(G)$.

Reducing the number of performed modified BFS in Brandes' algorithm is one way to speed up Brandes' algorithm. To this end, a popular approach is to remove in a preprocessing step all degree-one vertices from the graph [2, 12, 13]. By repeatedly removing degree-one vertices, whole "pending trees" can be deleted. Considering a degree-one vertex v, observe that in *each* shortest path P starting at v, the second vertex in P is the single neighbor u of v. Hence, after deleting v, one needs to store the information that u had a degree-one neighbor. To this end, one uses for each vertex w a counter which we call $\mathrm{Pen}[w]$ that stores the number of vertices in the subtree pending on w that where deleted before. In contrast to e. g. Baglioni et al. [2], we initialize for each vertex $w \in V$ the value $\mathrm{Pen}[w]$ with one instead of zero (so we count w as well). This simplifies most of our formulas. See Figure 1 for an example of the $\mathrm{Pen}[\cdot]$-values of the vertices at different points in time. This yields the following (weighted) problem variant.

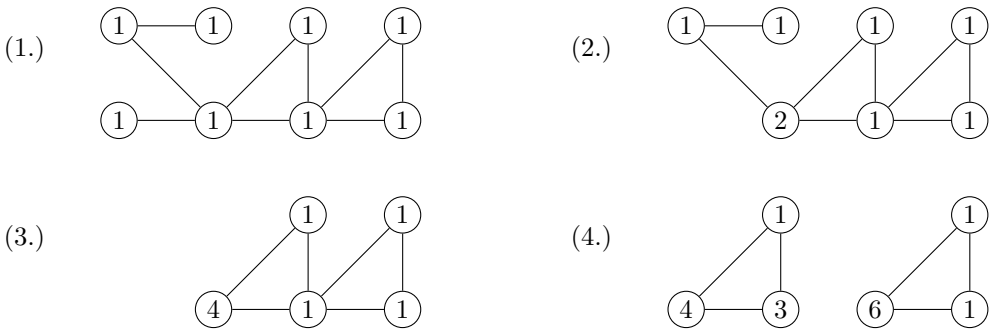

Figure 1 An initial graph where the Pen[·]-value of each vertex is 1 (top left) and the same graph after deleting one (top right) or both (bottom left) pending trees using Reduction Rule 1. The labels are the respective Pen[·]-values. Subfigure (4.) shows the graph of (3.) after applying Lemma 2 to the only remaining cut vertex of the graph.

Weighted Betweenness Centrality
Input: An undirected graph G and vertex weights $\mathrm{Pen}\colon V(G) \to \mathbb{N}$.
Task: Compute for each vertex $v \in V(G)$ the weighted betweenness centrality

$$C_B(v) := \sum_{s,t \in V(G)} \gamma(s,t,v), \tag{1}$$

where $\gamma(s,t,v) := \mathrm{Pen}[s] \cdot \mathrm{Pen}[t] \cdot \sigma_{st}(v)/\sigma_{st}$.

The effect of a degree-one vertex to the betweenness centrality value of its neighbor is captured in the next data reduction rule.

▶ **Reduction Rule 1** ([2, 12, 13]). *Let G be a graph, let $s \in V(G)$ be a degree-one vertex, and let $v \in V(G)$ be the neighbor of s. Then increase* $\mathrm{Pen}[v]$ *by* $\mathrm{Pen}[s]$*, increase the betweenness centrality of v by* $\mathrm{Pen}[s] \cdot \sum_{t \in V(G) \setminus \{s,v\}} \mathrm{Pen}[t]$*, and remove s from the graph.*

Hence, the influence of a degree-one vertex to the betweenness centrality of its neighbor can be computed in constant time as $\sum_{w \in V(G)} \mathrm{Pen}[w]$ can be precomputed once in linear time.

A second approach to speed up Brandes' algorithm is to split the input graph G into smaller components and process them separately [12, 13]. This approach is a generalization of the ideas behind removing degree-one vertices and works with cut vertices. The basic observation for this approach is as follows: Consider a cut vertex v such that removing v breaks the graph into exactly two connected components C_1 and C_2 (the idea generalizes to more components). Obviously, every shortest path P in G that starts in C_1 and ends in C_2 has to pass through v. For the betweenness centrality values of the vertices inside C_1 (inside C_2) it is not important where exactly P ends (starts). Hence, for computing the betweenness centrality values of the vertices in C_1, it is sufficient to know which vertices in C_1 are adjacent to v and how many vertices are contained in C_2. Thus, in a preprocessing step one can just add to C_1 a copy of the cut vertex v with $\mathrm{Pen}[v]$ being increased by the sum of Pen[·]-values of the vertices in C_2 (see Figure 1 (bottom)). The same is done for C_2. Formally, this is done as follows.

▶ **Lemma 2** ([12, 13]). *Let G be a connected graph, let v be a cut vertex such that removing v yields $\ell \geq 2$ connected components $C_1, \ldots, C_\ell$, and let $\xi := \mathrm{Pen}[v]$. Then remove v, add a new vertex v_i to each component C_i, make them adjacent to all vertices in the respective*

component that were adjacent to v, and set

$$\mathrm{Pen}[v_i] = \xi + \sum\nolimits_{j \in [1,\ell] \setminus \{i\}} \sum\nolimits_{w \in V(C_j) \setminus \{v_j\}} \mathrm{Pen}[w].$$

Computing the betweenness centrality of each connected component independently, increasing the betweenness centrality of v by $\sum_{i=1}^{\ell} \left(C_B^{C_i}(v_i) + (\mathrm{Pen}[v_i] - \xi) \cdot \sum_{s \in V(C_i) \setminus \{v_i\}} \mathrm{Pen}[s] \right)$, and ignoring all new vertices v_i is the same as computing the betweenness centrality in G, that is,

$$C_B^G(u) = \begin{cases} C_B^{C_i}(u), & \text{if } u \in V(C_i) \setminus \{v_i\}; \\ \sum_{i=1}^{\ell} \left(C_B^{C_i}(v_i) + (\mathrm{Pen}[v_i] - \xi) \cdot \sum_{s \in V(C_i) \setminus \{v_i\}} \mathrm{Pen}[s] \right), & \text{if } u = v. \end{cases}$$

Applying the above procedure as preprocessing on all cut vertices and degree-one vertices takes linear time [13] leaves us with biconnected components that we can solve independently. Hence, we assume in the rest of the paper that we are given a vertex-weighted biconnected component.

Our algorithmic approach. Starting with a vertex-weighted biconnected graph, our algorithm focuses on degree-two vertices. In contrast to degree-one vertices, degree-two vertices can lie on shortest paths between two other vertices. This difference makes degree-two vertices harder to handle: Removing a degree-two vertex v in a similar way as done with degree-one vertices (see Reduction Rule 1) affects many other shortest paths that neither start nor end in v. Hence, we deal with degree-two vertices in a different manner. Instead of removing vertices one-by-one, we process multiple degree-two vertices at once. To this end, we use the following definition and exploit that adjacent degree-two vertices behave similarly.

▶ **Definition 3.** Let G be a graph. A path $P = v_0 \dots v_\ell$ is a *maximal induced path* in G if $\ell \geq 2$ and the inner vertices $v_1, \dots, v_{\ell-1}$ all have degree two in G, but the endpoints v_0 and v_ℓ do not, that is, $\deg_G(v_1) = \dots = \deg_G(v_{\ell-1}) = 2$, $\deg_G(v_0) \neq 2$, and $\deg_G(v_\ell) \neq 2$. Moreover, $\mathcal{P}^{\max}$ is the set of all maximal induced paths in G.

Note that if our biconnected graph is a cycle, then it does not contain any maximal induced path. Our algorithm (see Algorithm 1 for the pseudocode) deals with this corner case separately by using a linear-time dynamic programming algorithm for vertex-weighted cycles. Note that the vertices in the cycle can have different betweenness centrality values as they may have different Pen[·]-values.

▶ **Proposition 4** (⋆[4]). *Let $C = x_0 \dots x_q x_0$ be a cycle. Then, the weighted betweenness centrality of the vertices in C can be computed in $O(q)$ time.*

The remaining part of the algorithm deals with maximal induced paths. Note that if the (biconnected) graph is not a cycle, then all degree-two vertices are contained in maximal induced paths: If the graph is not a cycle and does not contain degree-one vertices, then the endpoints of each chain of degree-two vertices are vertices of degree at least three. If some degree-two vertex v was not contained in a maximal induced path, then v would be contained on a cycle with exactly one vertex of degree at least three. This vertex would be a cut vertex and the graph would not be biconnected; a contradiction.

Using standard arguments, we can show that the number of maximal induced paths is upper-bounded by the minimum of the feedback edge number k of the input graph and the number n of vertices. Moreover, one can easily compute all maximal induced paths in linear-time (see Line 6 in Algorithm 1).

[4] Proofs of results marked with (⋆) are deferred to the full version.

Algorithm 1: Computation of betweenness centrality in a biconnected graph.

Input: An undirected biconnected graph G with vertex weights $\mathrm{Pen}\colon V(G) \to \mathbb{N}$.
Output: The betweenness centrality values of all vertices.

```
1  foreach v ∈ V(G) do BC[v] ← 0        // BC will contain the betweenness centrality values
2  F ← feedback edge set of G                    // computable in O(n + m) time using BFS
3  if |F| = 1 then
4      update BC for the case that G is a cycle // computable in O(n) time, see Proposition 4
5  else
6      P^max ← all maximal induced paths of G          // takes O(n + m) time, see Lemma 6
7      foreach s ∈ V^{≥3}(G) do  // some precomputations taking O(|F|n) time, see Lemma 10
8          compute d_G(s, t) and σ_st for each t ∈ V(G) \ {s}
9          Inc[s, t] ← 2 · Pen[s] · Pen[t]/σ_st for each t ∈ V^{=2}(G)
10         Inc[s, t] ← Pen[s] · Pen[t]/σ_st for each t ∈ V^{≥3}(G) \ {s}
11     foreach x_0 x_1 ... x_q = P^max ∈ P^max do    // initialize W^left and W^right, in O(n) time
12         W^left[x_0] ← Pen[x_0]; W^right[x_q] ← Pen[x_q]
13         for i = 1 to q do  W^left[x_i] ← W^left[x_{i−1}] + Pen[x_i]
14         for i = q − 1 to 0 do  W^right[x_i] ← W^right[x_{i+1}] + Pen[x_i]
15     foreach x_0 x_1 ... x_q = P_1^max ∈ P^max do          // case s ∈ V^{=2}(P_1^max), see Section 3
           /* deal with the case t ∈ V^{=2}(P_2^max), see Section 3.1                          */
16         foreach y_0 y_1 ... y_r = P_2^max ∈ P^max \ {P_1^max} do
               /* update BC for the case v ∈ V(P_1^max) ∪ V(P_2^max)                           */
17             foreach v ∈ V(P_1^max) ∪ V(P_2^max) do  BC[v] ← BC[v] + γ(s, t, v)
               /* now deal with the case v ∉ V(P_1^max) ∪ V(P_2^max)                           */
18             update Inc[x_0, y_0], Inc[x_q, y_0], Inc[x_0, y_r], and Inc[x_q, y_r]
           /* deal with the case that t ∈ V^{=2}(P_1^max), see Section 3.1                     */
19         foreach v ∈ V(P_1^max) do  BC[v] ← BC[v] + γ(s, t, v)
20         update Inc[x_0, x_q]                    // this deals with the case v ∉ V(P_1^max)
21     foreach s ∈ V^{≥3}(G) do              // perform modified BFS from s, see Section 3.2
22         foreach t, v ∈ V(G) do  BC[v] ← BC[v] + Inc[s, t] · σ_st(v)
23 return BC.
```

▶ **Lemma 5** (⋆). *Let G be a graph with feedback edge number k containing no degree-one vertices. Then the cardinalities $|V^{\geq 3}(G)|$ and $|\mathcal{P}^{\max}|$ are upper-bounded by $O(\min\{n, k\})$.*

▶ **Lemma 6** (⋆). *The set $\mathcal{P}^{\max}$ of all maximal induced paths of a graph with n vertices and m edges can be computed in $O(n + m)$ time.*

Our algorithm processes the maximal induced paths one by one (see Lines 7 to 22). This part of the algorithm requires its own pre- and postprocessing (see Lines 7 to 14 and Lines 21 to 22 respectively). In the preprocessing, we initialize tables used frequently in the main part (of Section 3). The postprocessing computes the final betweenness centrality values of each vertex as this computation is too time-consuming to be executed for each maximal induced path. When explaining our basic ideas, we will first present the postprocessing as this explains why certain values will be computed during the algorithm.

Recall that we want to compute $\sum_{s,t \in V(G)} \gamma(s, t, v)$ for each $v \in V(G)$ (see Equation (1)). Using the following observations, we split Equation (1) into different parts:

▶ **Observation 7.** *For $s, t, v \in V(G)$ it holds that $\gamma(s, t, v) = \gamma(t, s, v)$.*

▶ **Observation 8** (⋆). *Let G be a biconnected graph with at least one vertex of degree three. Let $v \in V(G)$. Then,*

$$\sum_{s,t \in V(G)} \gamma(s,t,v) = \sum_{s \in V^{\geq 3}(G),\, t \in V(G)} \gamma(s,t,v) + \sum_{s \in V^{=2}(G),\, t \in V^{\geq 3}(G)} \gamma(t,s,v) + \sum_{\substack{s \in V^{=2}(P_1^{\max}),\, t \in V^{=2}(P_2^{\max}) \\ P_1^{\max} \neq P_2^{\max} \in \mathcal{P}^{\max}}} \gamma(s,t,v) + \sum_{\substack{s,t \in V^{=2}(P^{\max}) \\ P^{\max} \in \mathcal{P}^{\max}}} \gamma(s,t,v).$$

In the remaining graph, by Lemma 5, there are $O(\min\{k,n\})$ vertices of degree at least three and $O(k)$ maximal induced paths. This implies that we can afford to run the modified BFS (similar to Brandes' algorithm) from each vertex $s \in V^{\geq 3}(G)$ in $O(\min\{k,n\} \cdot (n+k)) = O(kn)$ time. This computes the first summand and, by Observation 7, also the second summand in Observation 8. However, we cannot afford to run such a BFS from every vertex of degree two. Thus we need to compute the third and fourth summand differently.

To this end, note that $\sigma_{st}(v)$ is the only term in $\gamma(s,t,v)$ that depends on v. Our goal is then to precompute $\gamma(s,t,v)/\sigma_{st}(v) = \mathrm{Pen}[s] \cdot \mathrm{Pen}[t]/\sigma_{st}$ for as many vertices as possible. Hence, we store precomputed values in a table $\mathrm{Inc}[\cdot,\cdot]$ (see Lines 10, 18 and 20). Then, we plug this factor into the next lemma which provides our postprocessing.

▶ **Lemma 9** (⋆). *Let s be a vertex and let $f\colon V(G)^2 \mapsto \mathbb{N}$ be a function such that for each $u,v \in V(G)$ the value $f(u,v)$ can be computed in $O(\tau)$ time. Then, one can compute $\sum_{t \in V(G)} f(s,t) \cdot \sigma_{st}(v)$ for all $v \in V$ overall in $O(n \cdot \tau + m)$ time.*

Our strategy is to start the algorithm behind Lemma 9 only from vertices in $V^{\geq 3}(G)$ (see Line 22). Since the term τ in the above lemma will be constant, we obtain a running time of $O(kn)$ for running this postprocessing for all vertices. The most intricate part will be to precompute the factors in $\mathrm{Inc}[\cdot,\cdot]$ (see Lines 18 and 20). We defer the details to Section 3.1. In these parts, we need the tables W^{left} and W^{right}. These tables store values depending on the maximal induced path a vertex is in. More precisely, for a vertex x_i in a maximal induced path $P^{\max} = x_0 x_1 \dots x_q$, we store in $W^{\mathrm{left}}[x_k]$ the sum of the $\mathrm{Pen}[\cdot]$-values of vertices "left of" x_k in $P^{\max}$; formally, $W^{\mathrm{left}}[x_k] = \sum_{i=1}^{k} \mathrm{Pen}[x_j]$. Similarly, we have $W^{\mathrm{right}}[x_k] = \sum_{i=k}^{q-1} \mathrm{Pen}[x_k]$. The reason for having these tables is easy to see: Assume for the vertex $x_k \in P^{\max}$ that the shortest paths to $t \notin V(P^{\max})$ leave $P^{\max}$ through x_0. Then, it is equivalent to just consider the shortest path(s) starting in x_0 and simulate the vertices between x_k and x_0 in $P^{\max}$ by "temporarily increasing" $\mathrm{Pen}[x_0]$ by $W^{\mathrm{left}}[x_k]$. This is also the idea behind the argument that we only need to increase the values $\mathrm{Inc}[\cdot,\cdot]$ for the endpoints of the maximal induced paths in Line 18.

This leaves us with the remaining part of the preprocessing: the computation of the distances $d_G(s,t)$, the number of shortest paths σ_{st}, and $\mathrm{Inc}[s,t]$ for $s \in V^{\geq 3}(G), t \in V(G)$ (see Lines 7 to 10 in Algorithm 1). This can be done in $O(kn)$ time as well:

▶ **Lemma 10** (⋆). *The initialization in the for-loop in Lines 7 to 10 of Algorithm 1 can be done in $O(kn)$ time.*

Putting all parts together, we arrive at our main theorem (see Section 3.2 for the proof).

▶ **Theorem 11.** Betweenness Centrality *can be solved in $O(kn)$ time, where k is the feedback edge number of the input graph.*

3 Dealing with maximal induced paths

In this section, we focus on degree-two vertices contained in maximal induced paths. Recall that the goal is to compute the betweenness centrality $C_B(v)$ (see Equation (1)) for all $v \in V(G)$ in $O(kn)$ time. In the end of this section, we finally prove Theorem 11.

Based on Observation 8 and Equation (1), we compute $C_B(v)$ in three steps. By starting a modified BFS from vertices in $V^{\geq 3}(G)$ similarly to Baglioni et al. [2] and Brandes [3], we can compute the following term in $O(kn)$ time:

$$\sum_{s \in V^{\geq 3}(G),\, t \in V(G)} \gamma(t, s, v) + \sum_{s \in V^{=2}(G),\, t \in V^{\geq 3}(G)} \gamma(s, t, v).$$

3.1 Paths with endpoints in maximal induced paths

In this subsection, we show how to compute the remaining two summands given in Observation 8. In the next subsection, we prove Theorem 11.

Paths with endpoints in different maximal induced paths. We now focus on shortest paths between pairs of maximal induced paths $P_1^{\max}$ and $P_2^{\max}$, and how to efficiently determine how these paths affect the betweenness centrality of each vertex.

▶ **Proposition 12 (⋆).** *In $O(kn)$ time the following values can be computed for all $v \in V(G)$:*

$$\sum_{\substack{s \in V^{=2}(P_1^{\max}),\, t \in V^{=2}(P_2^{\max}) \\ P_1^{\max} \neq P_2^{\max} \in \mathcal{P}^{\max}}} \gamma(s, t, v).$$

Recall that in the course of the algorithm, we first gather values in $\mathrm{Inc}[\cdot, \cdot]$ and in the final step we compute for each $s, t \in V^{\geq 3}(G)$ the values $\mathrm{Inc}[s, t] \cdot \sigma_{st}(v)$ in $O(m)$ time (Lemma 9). This postprocessing (see Lines 21 and 22 in Algorithm 1) takes $O(kn)$ time.

In the proof of Proposition 12 (deferred to the full version), we consider two cases for every pair $P_1^{\max} \neq P_2^{\max} \in \mathcal{P}^{\max}$ of maximal induced paths: First, we look at how the shortest paths between vertices in $P_1^{\max}$ and $P_2^{\max}$ affect the betweenness centrality of those vertices that are not contained in the two maximal induced paths, and second, how they affect the betweenness centrality of those vertices that are contained in the two maximal induced paths.

Paths with endpoints in the same maximal induced paths. Subsequently, we look at shortest paths starting and ending in a maximal induced path $P^{\max} = x_0 \dots x_q$ and show how to efficiently compute how these paths affect the betweenness centrality. Our goal is to prove the following:

▶ **Proposition 13.** *In $O(kn)$ time the following values can be computed for all $v \in V(G)$:*

$$\sum_{\substack{s, t \in V^{=2}(P^{\max}) \\ P^{\max} \in \mathcal{P}^{\max}}} \gamma(s, t, v).$$

As in Section 3.1, we first gather all increments to $\mathrm{Inc}[\cdot, \cdot]$ and in the final step, we compute for each $s, t \in V^{\geq 3}(G)$ the values $\mathrm{Inc}[s, t] \cdot \sigma_{st}(v)$. We start with the following simple observation.

▶ **Observation 14.** *Let $P^{\max} = x_0 \dots x_q$, where $x_0, x_q \in V^{\geq 3}(G)$ and $x_i \in V^{=2}(G)$ for $1 \leq i \leq q-1$. Then*

$$\sum_{s,t \in V^{=2}(P^{\max})} \gamma(s,t,v) = \sum_{i,j \in [1,q-1]} \gamma(x_i, x_j, v) = 2 \cdot \sum_{i=1}^{q-1} \sum_{j=i+1}^{q-1} \gamma(x_i, x_j, v).$$

For the sake of readability we set $[x_p, x_r] := \{x_p, x_{p+1}, \dots, x_r\}$, $p < r$. We will distinguish between two different cases that we then treat separately: Either $v \in [x_i, x_j]$ or $v \in V(G) \setminus [x_i, x_j]$. We will show that both cases can be solved in overall $O(q)$ time for $P^{\max}$. Doing this for all maximal induced paths results in a running time of $O(\sum_{P^{\max} \in \mathcal{P}^{\max}} V^{=2}(P^{\max})) \subseteq O(n)$. We will distinguish between the two main cases in the calculations – all shortest $x_i x_j$-paths are fully contained in $P^{\max}$, or all shortest $x_i x_j$-paths leave $P^{\max}$ – and the corner case that there are some shortest paths inside $P^{\max}$ and some that partially leave it. Observe that for any fixed pair $i < j$ the distance between x_i and x_j is given by $d_{\text{in}} = j - i$ if a shortest path is contained in $P^{\max}$ and by $d_{\text{out}} = i + d_G(x_0, x_q) + q - j$ if a shortest $x_i x_j$-path leaves $P^{\max}$. The corner case that there are shortest paths both inside and outside of $P^{\max}$ occurs when $d_{\text{in}} = d_{\text{out}}$. In this case it holds that $j - i = i + d_G(x_0, x_q) + q - j$ which is equivalent to

$$j = i + \frac{d_G(x_0, x_q) + q}{2}, \tag{2}$$

where j is an integer smaller than q. For convenience, we will use a notion of "mid-elements" for a fixed starting vertex x_i. We distinguish between the two cases that this mid-element has a higher index in $P^{\max}$ or a lower one. Formally, we say that $i^+_{\text{mid}} = i + (d_G(x_0, x_q) + q)/2$ and $j^-_{\text{mid}} = j - (d_G(x_0, x_q) + q)/2$. We next analyze the factor $\sigma_{x_i x_j}(v)/\sigma_{x_i x_j}$. We also distinguish between the cases $v \in V(P^{\max})$ and $v \notin V(P^{\max})$. Observe that

$$\frac{\sigma_{x_i x_j}(v)}{\sigma_{x_i x_j}} = \begin{cases} 0, & \text{if } d_{\text{out}} < d_{\text{in}} \wedge v \in [x_i, x_j] \text{ or } d_{\text{in}} < d_{\text{out}} \wedge v \notin [x_i, x_j]; \\ 1, & \text{if } d_{\text{in}} < d_{\text{out}} \wedge v \in [x_i, x_j]; \\ 1, & \text{if } d_{\text{out}} < d_{\text{in}} \wedge v \notin [x_i, x_j] \wedge v \in V(P^{\max}); \\ \frac{\sigma_{x_0 x_q}(v)}{\sigma_{x_0 x_q}}, & \text{if } d_{\text{out}} < d_{\text{in}} \wedge v \notin V(P^{\max}); \\ \frac{1}{\sigma_{x_0 x_q}+1}, & \text{if } d_{\text{in}} = d_{\text{out}} \wedge v \in [x_i, x_j]; \\ \frac{\sigma_{x_0 x_q}}{\sigma_{x_0 x_q}+1}, & \text{if } d_{\text{in}} = d_{\text{out}} \wedge v \notin [x_i, x_j] \wedge v \in V(P^{\max}); \\ \frac{\sigma_{x_0 x_q}(v)}{\sigma_{x_0 x_q}+1}, & \text{if } d_{\text{in}} = d_{\text{out}} \wedge v \notin V(P^{\max}). \end{cases} \tag{3}$$

The denominator $\sigma_{x_0 x_q} + 1$ is correct since there are $\sigma_{x_0 x_q}$ shortest paths from x_0 to x_q (and therefore $\sigma_{x_0 x_q}$ shortest paths from x_i to x_j that leave $P^{\max}$) and one shortest path from x_i to x_j within $P^{\max}$. Note that if there are shortest paths that are not contained in $P^{\max}$, then $d_G(x_0, x_q) < q$ as we are in the case that $0 < i < j < q$. Thus $P^{\max}$ is not a shortest path from x_0 to x_q.

We will now compute the value for all paths that only consist of vertices in $P^{\max}$, that is, we will compute for each x_k with $i < k < j$ the term $2 \cdot \sum_{i=1}^{q-1} \sum_{j=i+1}^{q-1} \gamma(x_i, x_j, x_k)$ with a dynamic program in $O(q)$ time. Since $i < k < j$ this can be simplified to

$$2 \cdot \sum_{\substack{i \in [1,q-1] \\ i<k}} \sum_{\substack{j \in [i+1,q-1] \\ k<j}} \gamma(x_i, x_j, x_k) = 2 \cdot \sum_{i \in [1,k-1]} \sum_{j \in [k+1,q-1]} \gamma(x_i, x_j, x_k).$$

▶ **Lemma 15.** *For a fixed maximal induced path $P^{\max} = x_0 x_1 \dots x_q$, for all x_k with $0 \leq k \leq q$ we can compute $2 \cdot \sum_{i \in [1,k-1]} \sum_{j \in [k+1,q-1]} \gamma(x_i, x_j, x_k)$ in $O(q)$ time.*

Proof. For the sake of readability we define

$$\alpha_{x_k} = 2 \cdot \sum_{i\in[1,k-1]} \sum_{j\in[k+1,q-1]} \gamma(x_i, x_j, x_k).$$

Note that $i \geq 1$ and $k > i$ and thus for x_0 we have $\alpha_{x_0} = 2\sum_{i\in\emptyset}\sum_{j\in[1,q-1]}\gamma(x_i, x_j, x_0) = 0$. This will be the base case of the dynamic program.

For every vertex x_k with $1 \leq k < q$ it holds that

$$\alpha_{x_k} = 2\cdot \sum_{\substack{i\in[1,k-1]\\ j\in[k+1,q-1]}} \gamma(x_i, x_j, x_k) = 2\cdot \sum_{\substack{i\in[1,k-2]\\ j\in[k+1,q-1]}} \gamma(x_i, x_j, x_k) + 2\cdot \sum_{j\in[k+1,q-1]} \gamma(x_{k-1}, x_j, x_k).$$

Similarly, for x_k with $1 < k \leq q$ it holds that

$$\alpha_{x_{k-1}} = 2\cdot \sum_{\substack{i\in[1,k-2]\\ j\in[k,q-1]}} \gamma(x_i, x_j, x_{k-1}) = 2\cdot \sum_{\substack{i\in[1,k-2]\\ j\in[k+1,q-1]}} \gamma(x_i, x_j, x_{k-1}) + 2\cdot \sum_{i\in[1,k-2]} \gamma(x_i, x_k, x_{k-1}).$$

Next, observe that any path from x_i to x_j with $i \leq k-2$ and $j \geq k+1$ that contains x_k also contains x_{k-1} and vice versa. Substituting this into the equations above yields

$$\alpha_{x_k} = \alpha_{x_{k-1}} + 2 \cdot \sum_{j\in[k+1,q-1]} \gamma(x_{k-1}, x_j, x_k) - 2 \cdot \sum_{i\in[1,k-2]} \gamma(x_i, x_k, x_{k-1}).$$

Lastly, we prove that $\sum_{j\in[k+1,q-1]}\gamma(x_{k-1}, x_j, x_k)$ and $2\cdot\sum_{i\in[1,k-2]}\gamma(x_i, x_k, x_{k-1})$ can be computed in constant time once W^{left} and W^{right} are precomputed (see Lines 11 to 14 in Algorithm 1). These tables can be computed in $O(q)$ time as well. For convenience we say that $\gamma(x_i, x_j, x_k) = 0$ if i or j are not integral or are not in $[1, q-1]$ and define $W[x_i, x_j] = \sum_{\ell=i}^{j}\text{Pen}[x_\ell] = W^{\text{left}}[x_j] - W^{\text{left}}[x_{i-1}]$. Then we can use Equations (2) and (3) to show that

$$\begin{aligned}
&\sum_{j\in[k+1,q-1]} \gamma(x_{k-1}, x_j, x_k) = \sum_{j\in[k+1,q-1]} \text{Pen}[x_{k-1}]\cdot\text{Pen}[x_j]\cdot\frac{\sigma_{x_{k-1}x_j}(x_k)}{\sigma_{x_{k-1}x_j}}\\
&= \gamma(x_{k-1}, x_{(k-1)^+_{\text{mid}}}, x_k) + \sum_{j\in[k+1,\min\{\lceil(k-1)^+_{\text{mid}}\rceil-1,q-1\}]} \text{Pen}[x_{k-1}]\cdot\text{Pen}[x_j]\\
&= \begin{cases}
\text{Pen}[x_{k-1}]\cdot W[x_{k+1}, x_{q-1}], & \text{if } (k-1)^+_{\text{mid}} \geq q;\\
\text{Pen}[x_{k-1}]\cdot W[x_{k+1}, x_{\lceil(k-1)^+_{\text{mid}}\rceil-1}], & \text{if } (k-1)^+_{\text{mid}} < q \wedge (k-1)^+_{\text{mid}} \notin \mathbb{Z};\\
\text{Pen}[x_{k-1}]\cdot(\text{Pen}[x_{(k-1)^+_{\text{mid}}}]\cdot\frac{1}{\sigma_{x_0,x_q}+1} + W[x_{k+1}, x_{(k-1)^+_{\text{mid}}-1}]), & \text{otherwise.}
\end{cases}
\end{aligned}$$

Herein we use $(k-1)^+_{\text{mid}} \notin \mathbb{Z}$ to say that $(k-1)^+_{\text{mid}}$ is not integral. Analogously,

$$\begin{aligned}
&\sum_{i\in[1,k-2]} \gamma(x_i, x_k, x_{k-1}) = \sum_{i\in[1,k-2]} \text{Pen}[x_i]\cdot\text{Pen}[x_k]\cdot\frac{\sigma_{x_ix_k}(x_{k-1})}{\sigma_{x_ix_k}}\\
&= \gamma(x_{k-1}, x_{k^-_{\text{mid}}}, x_{k-1}) + \sum_{i\in[\max\{1,\lfloor(k-1)^-_{\text{mid}}\rfloor+1\},k-2]} \text{Pen}[x_i]\cdot\text{Pen}[x_k]\\
&= \begin{cases}
\text{Pen}[x_k]\cdot W[x_1, x_{k-2}], & \text{if } k^-_{\text{mid}} < 1;\\
\text{Pen}[x_k]\cdot W[x_{\lfloor k^-_{\text{mid}}\rfloor+1}, x_{k-2}], & \text{if } k^-_{\text{mid}} \geq 1 \wedge k^-_{\text{mid}} \notin \mathbb{Z};\\
\text{Pen}[x_k]\cdot(\text{Pen}[x_{k^-_{\text{mid}}}]\cdot\frac{1}{\sigma_{x_0,x_a}+1} + W[x_1, x_{k^-_{\text{mid}}+1}]), & \text{otherwise.}
\end{cases}
\end{aligned}$$

This completes the proof since $(k-1)^+_{\text{mid}}$, k^-_{mid}, every entry in $W[\cdot]$, and all other variables in the equation above can be computed in constant time once $W^{\text{left}}[\cdot]$ is computed. Thus, computing α_{x_i} for each vertex x_i in $P^{\max}$ takes constant time. As there are q vertices in $P^{\max}$, the computations for the whole maximal induced path $P^{\max}$ take $O(q)$ time. ◀

We still need to compute the value for all paths that partially leave $P^{\max}$. Note that $\mathrm{Inc}[s,t] \cdot \sigma_{st}(v)$ will be computed in the postprocessing step (see Lines 21 and 22 in Algorithm 1).

▶ **Lemma 16** (⋆). *Let $P^{\max} = x_0 x_1 \dots x_q \in \mathcal{P}^{\max}$. Then, assuming that $\mathrm{Inc}[s,t] \cdot \sigma_{st}(v)$ can be computed in constant time for some $s,t \in V^{\geq 3}(G)$, for $v \in V(G) \setminus [x_i, x_j]$ one can compute $\sum_{i\in[1,q-1]} \sum_{j\in[i+1,q-1]} \gamma(x_i, x_j, v)$ in $O(q)$ time.*

3.2 Postprocessing and algorithm summary

We are now ready to combine all parts and prove our main theorem.

▶ **Theorem 11** (Restated). Betweenness Centrality *can be solved in $O(kn)$ time, where k is the feedback edge number of the input graph.*

Proof. We show that in the Lines 7 to 22 Algorithm 1 computes the value

$$C_B(v) = \sum_{s,t\in V(G)} \mathrm{Pen}[s] \cdot \mathrm{Pen}[t] \cdot \frac{\sigma_{st}(v)}{\sigma_{st}} = \sum_{s,t\in V(G)} \gamma(s,t,v)$$

for all $v \in V(G)$ in $O(kn)$ time. We use Observation 8 to split the sum as follows:

$$\begin{aligned}\sum_{s,t\in V(G)} \gamma(s,t,v) = &\sum_{s\in V^{\geq 3}(G),\, t\in V(G)} \gamma(s,t,v) + \sum_{s\in V^{=2}(G),\, t\in V^{\geq 3}(G)} \gamma(t,s,v) \\ &+ \sum_{\substack{s\in V^{=2}(P_1^{\max}),\, t\in V^{=2}(P_2^{\max}) \\ P_1^{\max}\neq P_2^{\max}\in\mathcal{P}^{\max}}} \gamma(s,t,v) + \sum_{\substack{s,t\in V^{=2}(P^{\max}) \\ P^{\max}\in\mathcal{P}^{\max}}} \gamma(s,t,v).\end{aligned}$$

By Propositions 12 and 13, we can compute the third and fourth summand in $O(kn)$ time provided that $\mathrm{Inc}[s,t] \cdot \sigma_{st}(v)$ is computed for every $s,t \in V^{\geq 3}(G)$ and $v \in V(G)$ in a postprocessing step (see Lines 15 to 20). We incorporate this postprocessing into the computation of the first two summands in the equation, that is, we next show that for all $v \in V(G)$ the following value can be computed in $O(kn)$ time:

$$\sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V(G)}} \gamma(s,t,v) + \sum_{\substack{s\in V^{=2}(G) \\ t\in V^{\geq 3}(G)}} \gamma(s,t,v) + \sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V^{\geq 3}(G)}} \mathrm{Inc}[s,t] \cdot \sigma_{st}(v).$$

To this end, observe that

$$\begin{aligned}&\sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V(G)}} \gamma(s,t,v) + \sum_{\substack{s\in V^{=2}(G) \\ t\in V^{\geq 3}(G)}} \gamma(s,t,v) + \sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V^{\geq 3}(G)}} \mathrm{Inc}[s,t] \cdot \sigma_{st}(v) \\ &= \sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V(G)}} \mathrm{Pen}[s] \cdot \mathrm{Pen}[t] \cdot \frac{\sigma_{st}(v)}{\sigma_{st}} + \sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V^{=2}(G)}} \mathrm{Pen}[s] \cdot \mathrm{Pen}[t] \cdot \frac{\sigma_{st}(v)}{\sigma_{st}} + \sum_{\substack{s\in V^{\geq 3}(G) \\ t\in V^{\geq 3}(G)}} \mathrm{Inc}[s,t] \cdot \sigma_{st}(v) \\ &= \sum_{s\in V^{\geq 3}(G)} \Big((2 \cdot \sum_{t\in V^{=2}(G)} \mathrm{Pen}[s]\,\mathrm{Pen}[t] \frac{\sigma_{st}(v)}{\sigma_{st}}) + \sum_{t\in V^{\geq 3}} \sigma_{st}(v) (\frac{\mathrm{Pen}[s]\,\mathrm{Pen}[t]}{\sigma_{st}} + \mathrm{Inc}[s,t]) \Big).\end{aligned}$$

Note that we initialize $\mathrm{Inc}[s,t]$ in Lines 10 and 9 in Algorithm 1 with $2 \cdot \mathrm{Pen}[s]\,\mathrm{Pen}[t]/\sigma_{st}$ and $\mathrm{Pen}[s]\,\mathrm{Pen}[t]/\sigma_{st}$ respectively. Thus we can use the algorithm described in Lemma 9 for each vertex $s \in V^{\geq 3}(G)$ with $f(s,t) = \mathrm{Inc}[s,t]$.

Since the values Pen[s], Pen[t], σ_{st} and Inc[s, t] can all be looked up in constant time, the algorithm takes $O(n + m)$ time to run a modified BFS from some vertex s (see Lines 21 and 22). By Lemma 5 there are $O(\min\{k, n\})$ vertices of degree at least three. The algorithm therefore take $O(\min\{n, k\} \cdot m) = O(\min\{n, k\} \cdot (n + k)) = O(kn)$ time to run the modified BFS from all vertices of degree at least three. ◀

4 Conclusion

Lifting the processing of degree-one vertices due to Baglioni et al. [2, 13] to a technically much more demanding processing of degree-two vertices, we derived a new algorithm for Betweenness Centrality running in $O(kn)$ worst-case time (k is the feedback edge number of the input graph). Our work focuses on algorithm theory and contributes to the field of adaptive algorithm design [5] as well as to the recent "FPT in P" program [7]. It would be of high interest to identify structural parameterizations "beyond" the feedback edge number that might help to get more results in the spirit of our work. In particular, extending our algorithmic approach with the treatment of twin vertices [12, 13] might help to get a running time bound involving the vertex cover number of the graph. From a practical viewpoint it remains to be investigated for which classes of real-world networks our (more complicated) algorithmic approach yields faster algorithms in empirical studies.

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic Equivalences Between Graph Centrality Problems, APSP and Diameter. In *Proc. of 26th SODA*, pages 1681–1697. SIAM, 2015.

2 Miriam Baglioni, Filippo Geraci, Marco Pellegrini, and Ernesto Lastres. Fast exact computation of betweenness centrality in social networks. In *Proc. of 4th ASONAM*, pages 450–456. IEEE Computer Society, 2012.

3 Ulrik Brandes. A faster algorithm for betweenness centrality. *J Math Sociol*, 25(2):163–177, 2001.

4 Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, 2011.

5 Vladimir Estivill-Castro and Derick Wood. A Survey of Adaptive Sorting Algorithms. *ACM Comput Surv*, 24(4):441–476, 1992.

6 Linton Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

7 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theor Comput Sci*, 689:67–95, 2017.

8 David S. Johnson. The genealogy of theoretical computer science: A preliminary report. *ACM SIGACT News*, 16(2):36–49, 1984.

9 Mark E. J. Newman. Who Is the Best Connected Scientist? A Study of Scientific Coauthorship Networks. In *Proc. of 23rd CNLS*, pages 337–370. Springer, 2004.

10 André Nichterlein, Rolf Niedermeier, Johannes Uhlmann, and Mathias Weller. On tractable cases of Target Set Selection. *SNAM*, 3(2):233–256, 2013.

11 Kim Norlen, Gabriel Lucas, Michael Gebbie, and John Chuang. EVA: Extraction, visualization and analysis of the telecommunications and media ownership network. In *Proc. of 14th ITS*, 2002.

12 Rami Puzis, Yuval Elovici, Polina Zilberman, Shlomi Dolev, and Ulrik Brandes. Topology manipulations for speeding betweenness centrality computation. *J Comp Net*, 3(1):84–112, 2015.

13 Ahmet Erdem Sariyüce, Kamer Kaya, Erik Saule, and Ümit V. Çatalyürek. Graph Manipulations for Fast Centrality Computation. *ACM Trans Knowl Discov Data*, 11(3):26:1–26:25, 2017.

14 Flavio Vella, Massimo Bernaschi, and Giancarlo Carbone. Dynamic Merging of Frontiers for Accelerating the Evaluation of Betweenness Centrality. *ACM JEA*, 23(1):1.4:1–1.4:19, 2018.

15 Wei Wang and Choon Yik Tang. Distributed computation of node and edge betweenness on tree graphs. In *Proc. of 52nd CDC*, pages 43–48. IEEE, 2013.