IEICE TRANS. INF. & SYST., VOL.E100–D, NO.5 MAY 2017

PAPER

An Efficient Approximate Algorithm for the 1-Median Problem on a Graph*

Koji TABATA[†a)], ***Nonmember***, **Atsuyoshi NAKAMURA**[†], ***and*** **Mineichi KUDO**[†], ***Members***

SUMMARY We propose a heuristic approximation algorithm for the 1-median problem. The 1-median problem is the problem of finding a vertex with the highest *closeness centrality*. Starting from a randomly selected vertex, our algorithm repeats to find a vertex with higher closeness centrality by approximately calculating closeness centrality of each vertex using simpler spanning subgraphs, which are called *k-neighbor dense shortest path graphs with shortcuts*. According to our experimental results using real networks with more than 10,000 vertices, our algorithm is more than 100 times faster than the exhaustive search and more than 20 times faster than the state-of-the-art approximation algorithm using annotated information to the vertices while the solutions output by our algorithm have higher approximation ratio.

key words: *1-median problem, closeness centrality, graph mining*

1. Introduction

Since various networks in society can be represented as weighted graphs, methods for finding important vertices in a graph have a lot of applications. What is important depends on applications, and various importance measures have been proposed so far.

One of the most popular such measures is *closeness centrality*, which measures centrality in terms of distance to all the vertices. The closeness centrality of a vertex v is calculated using the (weighted) sum of the lengths of the shortest paths from v to all the vertices, and smaller length sum means higher closeness centrality. The problem of finding the highest closeness centrality is known as the 1-*median problem*, which is a special case of the *k-median problem*: the problem of finding a k-sized set S that minimizes the (weighted) sum of the lengths of the shortest paths from $v_u \in S$ to all the vertices u, where v_u is the vertex in S that is nearest to u.

The k-median problem is known to be NP-hard for $k > 1$ [8] but the 1-median problem can be solved in $O(n^2 \log n + mn)$ time by Dijkstra's algorithm using a Fibonacci heap [3], [7], where n is the number of vertices and m is the number of edges. Recently, however, we have to deal with huge networks, for which algorithms are required to run in time close to linear in the network size.

Manuscript received September 17, 2016.
Manuscript revised December 22, 2016.
Manuscript publicized January 23, 2017.

[†]The authors are with Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.

*The preliminary version of this paper has appeared in [13].

a) E-mail: ktabata@main.ist.hokudai.ac.jp

DOI: 10.1587/transinf.2016EDP7398

In this paper, we propose a heuristic approximation algorithm for the 1-median problem. Our algorithm is iterative; starting from a randomly selected initial vertex v_0, the algorithm finds a vertex v_* with closeness centrality higher than that of v_0, and repeats the same process replacing v_0 with v_* until such vertex v_* is not found. In each iteration, the algorithm approximately calculates closeness centrality of each vertex using a simpler subgraph containing the shortest path tree from the vertex v_0. The highest-closeness-centrality vertex v_* for such simpler subgraphs has centrality that is at least the centrality of v_0 for the original graph because the centrality of v_0 is the same for both the graphs. As a subgraph, the shortest path tree from v_0 itself can be used and then we can obtain the exact solution for the subgraph in $O(n)$ time [5]. The centrality of a vertex v for the spanning tree is an upper bound of its centrality for the original graph, but the gap between them is large unless $v = v_0$ in most graphs, which results in few iterations and a bad approximate solution. To obtain a better centrality upper bound for each vertex v, we propose to use a *k-neighbor dense shortest path graph from v_0 with shortcuts*, which is composed of (1) all the edges between k-nearest vertices of v_0 and (2) at most $k-1$ edges between v's *partition* and the other $k-1$ *partitions* in addition to the shortest path tree from v_0, where each *partition* is composed of all the vertices whose nearest vertex among the k-nearest vertices of v_0 is the same. Our algorithm, which we call FAOM (Fast Approximation of One Median), runs in $O(((kn + m) \log n + k^3 \log k)\ell)$ time and $O(m + n \log n)$ space, where ℓ is the number of iterations, which is at most 7 in our experiments using a graph with 17,903 vertices and 197,031 edges.

According to the results of our experiments using real and synthetic datasets, for scale-free networks, FAOM outperforms the exhaustive search and an algorithm using DTZ (Distance To Zone) [11] to calculate approximate distances; for two real networks and one synthetic scale-free network, FAOM with $k \leq 128$ is faster than the exhaustive search and the algorithm using DTZ while the approximation ratio of FAOM is always better than that of the algorithm using DTZ. Especially for the real networks, which have more than 10,000 vertices, FAOM with $k \leq 128$ is at least 21 times faster than the algorithm using DTZ and at least 143 times faster than the exhaustive search.

2. Related Work

The graph median problem has been studied more than half

a century. Hakimi wrote a paper on the 1-median problem in 1964 [6]. The 1-median problem is related to the problem of constructing a shortest path tree from a given vertex because the 1-median problem can be solved by constructing a shortest path tree from each vertex, which we call the *exhaustive search* in this paper. For the shortest path tree construction problem, $O(n \log n + m)$-time algorithm using Fibonacci heap [3], [7] has been proposed, where n is the number of nodes and m is the number of edges.

Since the exhaustive search is too slow for a huge network, faster exact algorithms for a simpler graph or faster approximation algorithms have been developed. Burkard et al. proposed the exact algorithm for cactus graphs [2]. Rattigan et al. proposed an approximation algorithm for the centrality measure using annotated information to the vertices [11].

The 1-median problem is, in other words, the problem of finding a vertex with the highest *closeness centrality* [4]. In terms of closeness centrality, approximation algorithms for the ranking problems have been also developed [10], [15].

As for more general k-median problem, its NP-hardness was proved [8], and approximation algorithms have been proposed [1], [9], [12].

3. Problem Setting

Let $G = (V, E)$ be an undirected connected graph, where V and E are the sets of vertices and edges, respectively. Each vertex $v \in V$ has a weight $w(v) > 0$, and each edge $(u, v) \in E$ has a length $\mathrm{len}(u, v) > 0$, where (u, v) for $u, v \in V$ represents an edge between two vertices $u, v \in V$. The number of vertices and edges are denoted as n and m, respectively.

The distance between any two vertices u and v in $G = (V, E)$ is defined as the shortest path length between u and v and denoted as $d_G(u, v)$. The distance $d_G(u, v)$ is also written as $d(u, v)$ by omitting G when G is clear from context.

Let $d_G(v)$ denote the $w(u)$-weighted sum of distances $d_G(v, u)$ from v to all the vertices u, that is,

$$d_G(v) = \sum_{u \in V} d_G(v, u) w(u).$$

In this paper, we consider a problem of finding a vertex v with the minimum $d_G(v)$ among all vertices $v \in V$.

Problem 3.1 (1-median problem): For a given undirected connected graph $G = (V, E)$ with a vertex weight function $w : V \to (0, \infty)$ and an edge length function $d : E \to (0, \infty)$, find a vertex v with the minimum $d_G(v)$ among all vertices $v \in V$.

This problem can be solved exactly by constructing a *shortest path tree* from each vertex, which takes $O(n^2 \log n + nm)$ time using Dijkstra's algorithm, and $O(nm)$ time using the Thorup's algorithm [14] in the case with positive integer length function.

The followings are notions and notations related to shortest path trees that are used in this paper. The *shortest path tree $T(v)$ of G from v* is the spanning tree of G that contains a shortest path from v to u for all $u \in V \setminus \{v\}$. The shortest path tree $T(v)$ can be regarded as a rooted tree with root v. We let $D_{T(v)}(u)$ denote the set of the *descendants of u* in the rooted tree $T(v)$. Note that u itself is a descendant of u. We let $p_{T(v)}(u)$ denote the *parent of u* in $T(v)$, and let $p^0_{T(v)}(u) = u$ and let $p^i_{T(v)}(u)$ denote the parent of $p^{i-1}_{T(v)}(u)$. Define $W_{T(v_0)}(v)$ and $d^D_{T(v_0)}(v)$ as

$$W_{T(v_0)}(v) = \sum_{u \in D_{T(v_0)}(v)} w(u) \text{ and}$$

$$d^D_{T(v_0)}(v) = \sum_{u \in D_{T(v_0)}(v) \setminus \{v\}} d_{T(v_0)}(v, u) w(u),$$

respectively. We sometimes omit the subscript $T(v)$ when it is clear from context.

Algorithm 1 Iterative algorithm framework for 1-median problem

```
1: v* ← a randomly selected vertex from V;
2: repeat
3:     v0 ← v*
4:     Calculate d_G(v0) constructing the shortest path tree T(v0) from v0.
5:     Calculate d_{G_v}(v) for each v ∈ V,
                   where G_v is a subgraph of G that contains T(v0).
6:     Set v* = arg min_{v∈V} d_{G_v}(v).
7: until d_{G_{v*}}(v*) ≥ d_G(v0)
8: output v0;
```

4. Iterative Algorithm Framework

To obtain an approximate solution for 1-median problem, we propose an iterative algorithm framework shown in Algorithm 1. Starting from a randomly selected vertex, an algorithm in our framework finds a better vertex (a vertex v with smaller $d_G(v)$) repeatedly until failing to find such a vertex. The point is how we can find a vertex v with $d_G(v)$ smaller than $d_G(v_0)$ of a given vertex v_0 efficiently. In our framework, an algorithm calculates $d_G(v_0)$ by constructing the shortest path tree $T(v_0)$ from v_0. Then, in order to efficiently find vertex v_* with $d_G(v_*) \le d_G(v_0)$, the algorithm calculates $d_{G_v}(v)$ for each $v \in V$, where G_v is a subgraph of G that contains $T(v_0)$. The algorithm in this framework can be implemented efficiently if G_v has a simple structure; in the case with $G_v = T(v_0)$, $d_{G_v}(v)$ for all $v \in V$ can be calculated in $O(n)$ time [5]. Furthermore, $d_G(v) \le d_G(v_0)$ if $d_{G_v}(v) \le d_{G_{v_0}}(v_0)$ because $d_G(v) \le d_{G_v}(v)$ and $d_G(v_0) = d_{G_{v_0}}(v_0)$, where the last equality holds because G_{v_0} contains the shortest path tree $T(v_0)$ of G from v_0. Thus, $d_G(v_*) \le d_G(v_0)$ is guaranteed for $v_* = \arg\min_{v \in V} d_{G_v}(v)$, which means that no worse vertex is obtained by any iteration. Note that $d_G(v_*) \le d_G(v_0)$ is guaranteed even using an upper bound $\bar{d}_{G_v}(v)$ of $d_{G_v}(v)$ if $\bar{d}_{G_{v_0}}(v_0) = d_G(v_0)$ holds.

5. Subgraphs for Efficient Approximation

In order to efficiently obtain a tighter upper bound $d_{G_v}(v)$ of $d_G(v)$, what subgraph G_v of G should be used? Under the constraint that G_v must contain the shortest path tree $T(v_0)$ from v_0, it is ranged from $T(v_0)$ to G itself. A simple subgraph G_v enables fast calculation of $d_{G_v}(v)$ but brings a loose upper bound of $d_G(v)$. Conversely, a tight upper bound of $d_G(v)$ can be obtained if G_v is close to G though slow calculation of $d_{G_v}(v)$ is inevitable. So, we should choose a subgraph balancing this trade-off,

5.1 k-Neighbor Dense Shortest Path Graph

If the shortest path tree $T(v_0)$ is used as G_v, then the difference between $d_{G_v}(v)$ and $d_G(v)$ is 0 at $v = v_0$ and expected to increase as the distance between v and v_0 increases. The difference is expected to become close to 0 for neighbors of v_0 if the edges between the neighbors are added.

From this consideration, as a simple subgraph of G that is an extension of the shortest path tree $T(v_0)$, we propose a *k-neighbor dense shortest path graph of G from v_0*, kNSPG for short. Let $N_k(v_0)$ denote the set of k nearest vertices v of v_0 in V, that is, the set of k vertices with the smallest distance $d_G(v, v_0)$ from v_0. Note that $v_0 \in N_k(v_0)$. The kNSPG of G from v_0, which is denoted as $T(v_0; k)$, is defined as a subgraph of G that is constructed from the shortest path tree $T(v_0)$ of G from v_0 by adding all the edges in E between the vertices in $N_k(v_0)$. The kNSPG $T(v_0; k)$ of G from v_0 is an extension of the shortest path tree $T(v_0)$ of G from v_0 because $T(v_0)$ is just the 1NSPG $T(v_0; 1)$.

Example 5.1: Let G be the leftmost graph of Fig. 1. Then, the center graph in the figure is a 4NSPG $T(v_0; 4)$ of G from v_0. The four vertices in the gray box are members of $N_4(v_0)$. The edge between v_1 and v_2 is not contained in the shortest path tree $T(v_0)$ but it is contained in $T(v_0; 4)$.

The construction of $T(v_0; k)$ and the calculation of $d_{T(v_0;k)}(v)$ for all the vertices $v \in V$ can be done efficiently for small k.

Theorem 5.2: For a given graph $G = (V, E)$, $d_{T(v_0;k)}(v)$ for all the vertices $v \in V$ can be calculated in $O(n \log n + m + k^3)$ time and $O(m)$ space.

(*proof*) See Appendix A. □

5.2 kNSPG with Shortcuts

By using the kNSPG $T(v_0; k)$ as G_v, the upper bound $d_{G_v}(v)$ of $d_G(v)$ is expected to become tight for neighbors v of v_0, but it might be still loose for the vertices v that are far from v_0 if k is small. In order to improve the upper bound for such vertices, we consider a further extension of kNSPG $T(v_0; k)$ for each v by adding at most $k - 1$ edges depending on v.

Before describing the extension, we introduce some notions and notations. For each vertex $v \in V$, define

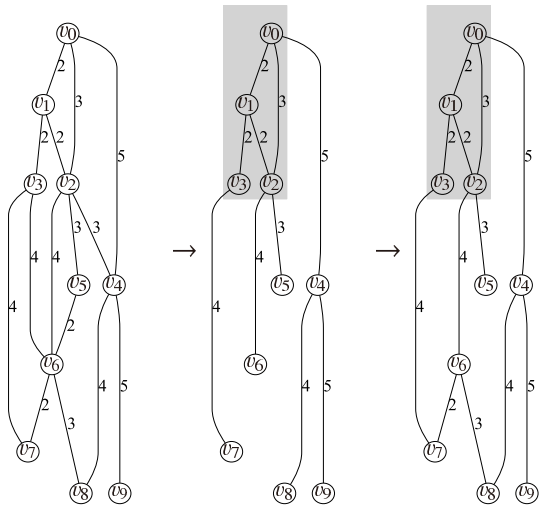

Fig. 1 **Example of a kNSPG and a kNSPG with shortcuts**: The center graph is a kNSPG $T(v_0; 4)$ of the leftmost graph G from v_0. The rightmost graph is a kNSPG $T^{S_{v_6}}(v_0; 4)$ of G from v_0 with shortcuts for v_6.

the closest v_0-neighbor $C(v, v_0; k)$ of $v \in V$ in $T(v_0; k)$ as the nearest vertex in $N_k(v_0)$ from v, that is, $C(v, v_0; k) = \arg\min_{u \in N_k(v_0)} d_{T(v_0;k)}(v, u)$. If v is in $N_k(v_0)$, $C(v, v_0; k)$ is v itself. For each $v \in N_k(v_0)$, we define *the v-subtree $ST(v, v_0; k)$ of $T(v_0; k)$* as the vertex-induced subgraph of $T(v_0; k)$ that is composed of all the vertices $u \in V$ with $C(u, v_0; k) = v$. A kNSPG $T(v_0; k)$ is partitioned into k disjoint v-subtrees $ST(v, v_0; k)$ for $v \in N_k(v_0)$. In the following, $C(v, v_0; k)$ and $ST(v, v_0; k)$ are also written as $C(v)$ and $ST(v)$ by omitting "$v_0; k$" when it is clear from context.

Example 5.3: In the center graph 4NSPG $T(v_0; 4)$ of Fig. 1,

$$C(v_0) = C(v_4) = C(v_8) = C(v_9) = v_0,$$
$$C(v_1) = v_1,$$
$$C(v_2) = C(v_5) = C(v_6) = v_2,$$
$$C(v_3) = C(v_7) = v_3.$$

Thus, $T(v_0; 4)$ are partitioned into $ST(v_0), ST(v_1), ST(v_2)$ and $ST(v_3)$, which are the vertex-induced subgraphs of vertex-sets $\{v_0, v_4, v_8, v_9\}, \{v_1\}, \{v_2, v_5, v_6\}$ and $\{v_3, v_7\}$, respectively.

Now, we describe our extension of kNSPG $T(v_0; k)$ of G from v_0. A kNSPG of G from v_0 *with shortcuts for v* is defined as a subgraph of G that is constructed from $T(v_0; k)$ by adding at most one edge (s, t) in E between $V_{ST(C(v))}$ and $V_{ST(u)}$ for each $u \in N_k(v_0) \setminus \{C(v)\}$, where $V_{ST(C(v))}$ and $V_{ST(u)}$ are the set of vertices in $ST(C(v))$ and $ST(u)$, respectively.

Example 5.4: The rightmost graph in Fig. 1 is a kNSPG of the leftmost graph G from v_0 with shortcuts for v_6.

Let $T^S(v_0; k)$ denote the kNSPG of G with shortcuts for v that is constructed from $T(v_0; k)$ by adding edges in S. Now, the problem is how to efficiently find S that is good for v. To address to the problem, we first consider effect of

adding an edge.

For $v \in V$, $s \in D_{T(v_0)}(v)$, $e = (s,t)$ and $u \in D_{T(v_0)}(C(t))$ $(C(t) \neq C(v))$, define

$$\begin{aligned}&\alpha_{T(v_0;k),e}(v,u)\\ =&d_{T(v_0;k)}(v,u) - d_{T^{\{e\}}(v_0;k)}(v,u)\\ =&d_{T(v_0;k)}(v,C(v)) + d_{T(v_0;k)}(C(v),C(t)) + d_{T(v_0;k)}(C(t),u)\\ &- \left(d_{T(v_0;k)}(v,s) + \mathrm{len}(s,t) + d_{T(v_0;k)}(u,t)\right). \end{aligned} \quad (1)$$

We call $u \in ST(C(t))$ an *effected vertex of* $e = (s,t)$ *for* v if $\alpha_{T(v_0;k),e}(v,u) > 0$. For a descendant s of v in the tree $ST(C(v))$ rooted by $C(v)$ and $u \in N_k(v_0) \setminus \{C(v)\}$, define the *nearest effected vertex* $v_{op} \in V_{ST(u)}$ *of an edge* $(s,t) \in (V_{ST(C(v))} \times V_{ST(u)}) \cap E$ *for* v as

$$v_{\mathrm{op}} = \operatorname*{arg\,min}_{u' \in V_{ST(u)}, \alpha_{T(v_0;k),e}(v,u') > 0} \alpha_{T(v_0;k),e}(v,u')$$

Note that v_{op} is the vertex that is nearest to v_0 among the vertices u' in $V_{ST(u)}$ for which the distance from v to u' is shortened by the path using edge (s,t). The *effect* $\delta_{T(v_0;k),e}(v)$ *of edge* e *for* v is defined as $d_{T(v_0;k)}(v) - d_{T^{\{e\}}(v_0;k)}(v)$ and the *restricted effect* $\overline{\delta}_{T(v_0;k),e}(v)$ of e for v is defined as the effect of $e = (s,t)$ restricted to the set of vertices $ST(C(t))$, that is,

$$\overline{\delta}_{T(v_0;k),(s,t)}(v) = \sum_{u \in V_{ST(C(t))}} \alpha_{T(v_0;k),(s,t)}(v,u) w(u).$$

Then,

$$\begin{aligned}\overline{\delta}_{T(v_0;k),(s,t)}(v) =&\alpha_{T(v_0;k),(s,t)}(v,v_{\mathrm{op}}) W_{T(v_0)}(v_{\mathrm{op}})\\ &+ \sum_{i=1}^{i_{\mathrm{op}}} d_{T(v_0)}(p^i(t), p^{i-1}(t)) W_{T(v_0)}(p^{i-1}(t)) \end{aligned} \quad (2)$$

holds, where $p^{i_{\mathrm{op}}}(t) = v_{\mathrm{op}}$.

Theorem 5.5: Given a kNSPG $T(v_0;k)$ of G from v_0 and $d_{T(v_0;k)}(v,u)$ for all $v,u \in N_k(v_0)$ $(v \neq u)$, after $O(n \log n)$-time $O(n \log n)$-space preparation, the restricted effect $\overline{\delta}_{T(v_0;k),e}(v)$ of any edge e for any vertex v can be calculated in $O(\log n)$ time.

(*proof*) See Appendix B. □

The restricted effect $\overline{\delta}_{T(v_0;k),e}(v)$ is a lower bound of effect $\delta_{T(v_0;k),e}(v)$, and the both coincides if $v_{\mathrm{op}} \neq C(t)$ for $e = (s,t)$. The merits of calculating $\overline{\delta}_{T(v_0;k),e}(v)$ are not only computational efficiency but also disjointness from the effect of other edge between $ST(C(v))$ and u-subtree for other $u \in N_k(v_0)$. Let S be the set of at most $k-1$ edges in E that satisfies

$$\begin{aligned}(s_1,t_1),(s_2,t_2) \in S \Rightarrow &s_1, s_2 \in D_{T(v_0)}(v),\\ &C(t_1), C(t_2) \neq C(v), C(t_1) \neq C(t_2).\end{aligned}$$

Then,

$$d_{T^S(v_0;k)}(v) \leq d_{T(v_0;k)}(v) - \sum_{e \in S} \overline{\delta}_{T(v_0;k),e}(v)$$

is derived from disjointness property of $\overline{\delta}_{T(v_0;k),e}(v)$. Since our purpose is to obtain a tighter upper bound of $d_G(v)$ efficiently and $d_{T^S(v_0;k)}(v)$ is an upper bound of $d_G(v)$, it is no problem to use $d_{T(v_0;k)}(v) - \sum_{e \in S} \overline{\delta}_{T(v_0;k),e}(v)$ as an upper bound of $d_G(v)$; it is tighter than $d_{T(v_0;k)}(v)$ by $\sum_{e \in S} \overline{\delta}_{T(v_0;k),e}(v)$.

For each $u \in N_k(v_0) \setminus \{C(v)\}$, we want to know the maximum $\overline{\delta}_{T(v_0;k),e}(v)$ among $e \in (D_{T(v_0)}(v) \times V_{ST(u)}) \cap E$, but it is computationally too heavy. Giving up to find the optimal value, we try to find a good edge e with large $\overline{\delta}_{T(v_0;k),e}(v)$. Let

$$E_v^u = \{(v,t) \in E \mid t \in V_{ST(u)}\}.$$

We define an edge $e_{v,u} \in (D_{T(v_0)}(v) \times V_{ST(u)}) \cap E$ for each $v \in V$ and $u \in N_k(v_0) \setminus \{C(v)\}$ as

$$e_{v,u} = \begin{cases} \operatorname*{arg\,max}_{e \in E_v^u} \overline{\delta}_{T(v_0;k),e}(v) & (D_{T(v_0)}(v) = \{v\}) \\ \operatorname*{arg\,max}_{e \in E_v^u \cup \{e_{v',u} \mid p(v') = v\}} \overline{\delta}_{T(v_0;k),e}(v) & (D_{T(v_0)}(v) \neq \{v\}). \end{cases}$$

Note that $e_{v,u}$ can be calculated in a bottom-up manner from the leaf nodes v of $T(v_0)$. For leaf nodes v of $T(v_0)$, edges $e_{v,u}$ are optimal but the optimality is not guaranteed for other vertices. Since $\overline{\delta}_{T(v_0;k),e}(p(v))$ is expected to be close to $\overline{\delta}_{T(v_0;k),e}(v)$, $\overline{\delta}_{T(v_0;k),e_{v,u}}(p(v))$ is expected to be large if $\overline{\delta}_{T(v_0;k),e_{v,u}}(v)$ is large. Thus, even for non-leaf nodes v, edges $e_{v,u}$ are expected to have large $\overline{\delta}_{T(v_0;k),e_{v,u}}(v)$. Define S_v as $S_v = \{e_{v,u} \mid u \in N_k(v_0) \setminus \{C(v)\}\}$. Then, we use $T^{S_v}(v_0;k)$ for a subgraph of G to calculate an upper bound of $d_G(v)$.

Theorem 5.6: Given a kNSPG $T(v_0;k)$ of G from v_0 and $d_{T(V_0;k)}(v,u)$ for all $v,u \in N_k(v_0)$ $(v \neq u)$, after $O(n \log n)$-time $O(n \log n)$-space preparation, upper bounds

$$d_{T(v_0;k)}(v) - \sum_{e \in S_v} \overline{\delta}_{T(v_0;k),e}(v)$$

of $d_{T^{S_v}(v_0;k)}(v)$ for all $v \in V \setminus N_k(v_0)$ can be calculated in $O((kn+m)\log n)$ time.

(*proof*) See Appendix C. □

6. Algorithm FAOM

We propose algorithm *FAOM (Fast Approximation of One Median)* of the iterative algorithm framework using kNSPGs $T^{S_v}(v_0;k)$ from v_0 with the set S_v of shortcuts for each vertex v as a subgraph G_v of a given graph G to calculate an upper bound of $d_G(v)$.

A pseudocode of FAOM is shown in Algorithm 2. FAOM repeats the execution (Line 5) of procedure HigherCentralityVertex which returns v_* and an upper bound d_* of $d_G(v_*)$ with $d_* \leq d_G(v_0)$ given an input vertex v_0. In HigherCentralityVertex, the kNSPG $T(v_0 : k)$ of G from v_0 is constructed (Line 12) and $d_{T(v_0;k)}(v)$ is calculated for all $v \in V$ (Line 13). After the preparation for fast calculation of $\overline{\delta}_{T(v_0;k),(s,t)}(v)$ $(v \in V \setminus N_k(v_0),\ s \in D_{T(v_0)}(v),\ t \notin V_{ST(C(v))})$ explained in the proof of Theorem 5.5 (Line 14), upper bounds

Algorithm 2 FAOM

```
1: function FAOM(G = (V, E), k)
2:     v* ← a randomly selected vertex from V
3:     repeat
4:         v0 ← v*;
5:         (v*, d*) ← HigherCentralityVertex (G, k, v0)
6:     until d* ≥ dG(v0)
7:     output v0;
8: end function
9:
10: function HigherCentralityVertex(G = (V, E), k, v0)
11:     Calculate dG(v0) constructing the shortest path tree T(v0) from v0.
12:     Construct kNSPG T(v0; k) of G from v0
                by adding edges between the vertices in Nk(v0) to T(v0)
13:     Calculate d_{T(v0;k)}(v) for all v ∈ V.
14:     Do preparation for fast calculation of δ̄_{T(v0;k),(s,t)}(v)
            (v ∈ V \ Nk(v0), s ∈ D_{T(v0)}(v), t ∉ V_{ST(C(v))}). (See Appendix B.)
15:     v* ← v0,  d* ← dG(v0)
16:     for all v ∈ V \ Nk(v0) do
17:         d̄v ← d_{T(v0;k)}(v) − Σ_{e∈Sv} δ̄_{T(v0;k),e}(v)
18:         if d̄v < d* then
19:             v* ← v,  d* ← d̄v
20:         end if
21:     end for
22:     return (v*, d*);
23: end function
```

$\overline{d}_v$ of $d_{T^{S_v}(v_0;k)}(v)$ are calculated for all $v \in V \setminus N_k(v_0)$, v_* is set to the vertex v with the minimum $\overline{d}_v$, and d_* is set to $\overline{d}_{v_*}$ (Line 15-21). Then, (v_*, d_*) is returned to the main function.

By Theorem 5.2 and 5.6, we obtain the following theorem.

Theorem 6.1: Function HigherCentralityVertex runs in $O((kn + m) \log n + k^3)$ time and $O(m + n \log n)$ space.

Trivially, time complexity of FAOM is $O(((kn + m) \log n + k^3)\ell)$ time and its space complexity is the same as HigherCentralityVertex, where ℓ is the number of the main-loop iterations. Though we have not obtained any non-trivial upper bound of ℓ yet, ℓ was at most 7 in our experiment even for the network with $n > 10,000$ and $m > 100,000$.

Remark 6.2: Assume that the number of the main-loop iterations is $O(1)$, and let us compare the time and space complexities of FAOM to those of other algorithms that are used as comparative methods in our experiments. The exhaustive search using Dijkstra's algorithm runs in $O(n^2 \log n + mn)$ time and $O(m)$ space. So, FAOM asymptotically runs faster than the exhaustive search but consumes more memory when $m = o(n \log n)$. A method using approximation distance calculated by DTZ[†] runs in $O(mkd + n^2 d)$ time and $O(nkd + m)$ space, where k and d are parameters that controls distance accuracy and computational time. FAOM asymptotically runs faster than this method when $m = o(n^2/(\log n))$ but consumes more memory when $m = o(n \log n)$.

Remark 6.3: Unfortunately, for $n > 2k$, FAOM's approximation ratio is not good in the worst case even when $w(v) = 1$ for all $v \in V$ because we have

$$\sup_{G,\text{len}} \frac{d_G(\hat{v})}{d_G(v_*)} = n - 1$$

for $v_* = \arg\max_{v \in V} d_G(v)$ and FAOM's output $\hat{v}$. Inequality $\frac{d_G(\hat{v})}{d_G(v_*)} \leq n - 1$ for any G and len holds because $d_G(\hat{v}) \leq d_{T(\hat{v})}(\hat{v}) = \sum_{v \in V} d_{T(\hat{v})}(v, \hat{v})$ and $d_{T(\hat{v})}(v, \hat{v}) \leq d_{T(v_*)}(v, \hat{v}) \leq d_G(v_*)$. Let $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $E = E_1 \cup E_2$, where $E_i = \{(v_i, v_j) | j > i\}$, and $\text{len}(u, v) = a$ for $(u, v) \in E_1$ and $\text{len}(u, v) = \delta$ for $(u, v) \in E_2$. Assume that $a \gg \delta$. Then, starting from v_1, FAOM stops by outputting v_1, and in this case $\frac{d_G(v_1)}{d_G(v_2)} = \frac{(n-1)a}{(n-2)\delta + a}$ and $\lim_{\delta \to +0} \frac{(n-1)a}{(n-2)\delta + a} = n - 1$ hold. Thus, Equality $\sup_{\text{len}} \frac{d_G(\hat{v})}{d_G(v_*)} = n - 1$ holds for this G. Approximation ratio of FAOM's outputs is close to 1 in our experiments, so what condition makes FAOM output a vertex with good approximation ratio is an interesting issue to pursue.

7. Experiments

We conducted experiments to check the effectiveness of our method using synthetic and real datasets.

7.1 Experimental Setting

We used four datasets shown in Table 1.

As real datasets, we used two datasets of Stanford Large Network Dataset Collection[††]. Dataset ca-AstroPh is a collaboration network of Arxiv Astro Physics category, in which vertices represent authors of scientific papers and edges represent co-author relationship. Dataset Oregon-1(May26) is autonomous system peering information inferred from Oregon route-views, in which vertices represent autonomous systems and edges represent existences of communication between them.

As synthetic datasets, we generate a random and a scale-free graphs. Dataset ER is a random graph generated using Erdős-Rényi model, in which all the pairs of vertices are connected randomly with a given probability p. We set $p = 0.0012$ in our experiments. Dataset BA is a scale-free graph generated using Barabási-Albert model, which was generated starting from the complete graph of three vertices by repeatedly adding a vertex v and edges between v and (at most) three other existing vertices that were selected according to the probability distribution proportional to the current

Table 1 Datasets used in our experiments

dataset	network type	#vertex	#edge
ca-AstroPh	collaboration network	17,903	197,031
Oregon-1(May26)	autonomous system	11,174	23,409
ER	synthetic (random)	4,986	15,118
BA	synthetic (scale-free)	5,000	14,968

[†]See Sect. 7.5 for detailed description.

[††]https://snap.stanford.edu/data/index.html

vertex degree[†††]. In a graph of Barabási-Albert model, the distribution of vertex degrees is known to obey power-law distribution.

All the datasets but BA are disconnected, so the maximum connected components, whose numbers of vertices and edges are shown in Table 1, were used in those datasets.

In ER and BA, we generated the vertex weights and edge lengths according to uniform distribution over $[0, 1]$ and $[1, 2]$, respectively[†]. In ca-AstroPh and Oregon1(May26), all the vertex weights and the edge lengths were set to 1.0.

All the experiments were conducted using a machine with Intel(R) Core(TM) i7-2600 3.40GHz processor, 8G of RAM, and Ubuntu 12.04. We implemented our algorithms in Python 2.7.

7.2 Effect of Using kNSPG with Shortcuts

In algorithm FAOM, we use an upper bound of $d_{T^{S_v}(v_0;k)}(v)$ to obtain an upper bound of $d_G(v)$ for $v \in V$. As a subgraph of G containing $T(v_0)$, we checked effectiveness of using $T^{S_v}(v_0;k)$ comparing with simpler subgraphs $T(v_0)$ and $T(v_0;k)$.

For each of the randomly selected 100 initial vertices v_0, we calculated $d_{T(v_0)}(v)/d_G(v)$, $d_{T(v_0;k)}(v)/d_G(v)$ and $(d_{T(v_0;k)}(v) - \sum_{e\in S_v} \overline{\delta}_{T(v_0;k),e}(v))/d_G(v)$, which is an upper bound of $d_{T^{S_v}(v_0;k)}(v)/d_G(v)$, for all the vertices v. Then, we made histograms of the values with range width 0.01 for each of the three. The result for the ca-AstroPh is shown in Fig. 2. As we can see on this graph, the approximation ratio is improved by using $T(v_0;k)$ and furthermore by using $T^{S_v}(v_0;k)$.

7.3 Number of Main-Loop Iterations

Efficiency of algorithm FAOM depends on the number of main-loop iterations, that is, the number of executions of HigherCentralityVertex. So, we checked the distribution of the number of the iterations using randomly selected 100 initial vertices v_0 for each graph. The result is shown in Table 2. As compared with the number of vertices, the number of iterations is very small (at most 7) on any graph in our experiments. As a result, our algorithm runs fast for the datasets.

7.4 Effect of Using Larger k

FAOM has parameter k which controls the complexity of subgraphs $T^{S_v}(v_0;k)$ of G. Larger k is expected to improve approximation ratio $d_G(\hat{v})/d_G(v_*)$ of FAOM's output $\hat{v}$ for the optimal vertex $v_* = \arg\min_{v\in V} d_G(v)$ while it increases

[†††]Selection were done three times independently according to the same distribution, and distinct ones of the three selected vertices were chosen.

[†]The triangle inequality is satisfied for the edge lengths that are generated according to this distribution, so $\mathrm{len}(u, v) = d_G(u, v)$ holds for any vertices u,v of the generated graphs G.

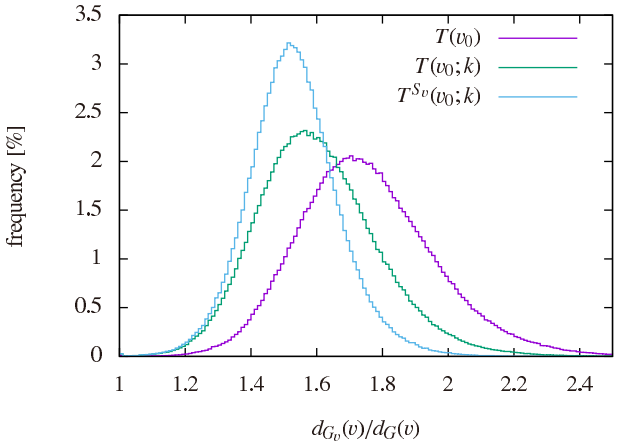

Fig. 2 Distribution of approximation ratio $d_{G_v}(v)/d_G(v)$ of $v \in V$ in ca-AstroPh for $G_v = T(v_0)$, $T(v_0;k)$, $T^{S_v}(v_0;k)$. For $T^{S_v}(v_0;k)$, the distribution is not for $d_{T^{S_v}(v_0;k)}(v)$ but for their upper bounds $d_{T(v_0;k)}(v) - \sum_{e\in S_v} \overline{\delta}_{T(v_0;k),e}(v)$. The frequency for range $[a, a + 0.01]$ ($a = 1.00, 1.01, \ldots, 2.50$) is the ratio of the number of vertices v that took the approximation ratio in that range for randomly chosen 100 initial vertices v_0.

Table 2 The number of main-loop iterations

	1	2	3	4	5	6	7	8
ca-AstroPh			14	38	27	15	6	
Oregon1(May26)			22	41	28	9		
BA			10	48	36	5	1	
ER		2	24	40	21	6	5	2

FAOM's running time. So, we conducted an experiment for checking both the relation between k and approximation ratio and the relation between k and running time. For each $k = 1, 2, 4, \ldots, 512$, we executed FAOM 1000 times by giving randomly selected initial vertices and calculated average approximation ratio and wall clock time. The result is shown in Fig.3. (See also Table3 for detailed data.) For Oregon1(May26), given any initial vertex, FAOM always found the optimal vertex for all k. So the spanning tree $T(v_0)$ is enough as G_v for this network. For other scale-free networks, ca-AstroPh and BA, approximation ratio surely improved in the range $k > 100$ though running time significantly increased in that range. Approximation ratio for ER looks almost the same for all values of k. In a random graph $G = (V, E)$, the function d_G over V has a lot of local minimums with high probability, so the probability of being caught in one of them is considered not to be improved by a smaller upper bound of d_G brought by larger k.

7.5 Comparison to Other Methods

We compared FAOM's performance to those of exact method and one state-of-the-art approximation method. Exact method (Exact) is the exhaustive search for the optimal vertex by constructing shortest path trees from all the vertices using Dijkstra's algorithm. As an approxima-

[†]DTZ and Exact are too slow to execute 1000 times.

Table 3 Approximation ratio $d_G(\hat{v})/d_G(v_*)$ and running time [sec] of three methods, FAOM, DTZ and Exact, where $\hat{v}$ is the vertex found by a method and v_* is the optimal vertex. The results are averaged over 1000 runs for FAOM, 100 runs for DTZ and 1 run for Exact[†]. The width of 95% confidence interval is shown in parentheses for approximation ratio and omitted for running time because it is ignorably small.

method	parameters	ca-AstroPh		Oregon1(May26)		ER		BA	
		app. ratio	time	app. ratio	time	app. ratio	time	app. ratio	time
FAOM	$k = 1$	1.023(±0.002)	1.951	1.000(±0.000)	0.207	1.070(±0.002)	0.023	1.015(±0.001)	0.001
	$k = 2$	1.024(±0.002)	3.588	1.000(±0.000)	0.337	1.069(±0.002)	0.252	1.016(±0.001)	0.182
	$k = 4$	1.030(±0.003)	5.143	1.000(±0.000)	0.459	1.069(±0.002)	0.841	1.015(±0.001)	0.463
	$k = 8$	1.026(±0.002)	7.154	1.000(±0.000)	0.506	1.070(±0.002)	1.309	1.015(±0.001)	0.631
	$k = 16$	1.025(±0.003)	9.602	1.000(±0.000)	0.567	1.068(±0.002)	1.776	1.015(±0.001)	0.640
	$k = 32$	1.018(±0.002)	11.764	1.000(±0.000)	0.738	1.067(±0.002)	2.508	1.011(±0.001)	0.957
	$k = 64$	1.025(±0.003)	13.290	1.000(±0.000)	1.160	1.069(±0.002)	3.028	1.013(±0.001)	1.467
	$k = 128$	1.021(±0.002)	16.172	1.000(±0.000)	3.434	1.069(±0.002)	6.688	1.007(±0.001)	3.716
	$k = 256$	1.014(±0.001)	41.303	1.000(±0.000)	22.111	1.068(±0.002)	35.363	1.003(±0.001)	23.716
	$k = 512$	1.007(±0.001)	197.273	1.000(±0.000)	159.656	1.066(±0.002)	254.720	1.004(±0.001)	174.233
DTZ	$k = 2, d = 10$	1.167(±0.001)	1677.122	1.232(±0.001)	713.922	1.083(±0.000)	110.153	1.134(±0.001)	121.654
	$k = 5, d = 4$	1.111(±0.001)	883.119	1.181(±0.001)	349.941	1.064(±0.001)	56.540	1.137(±0.001)	61.958
	$k = 10, d = 2$	1.084(±0.000)	525.265	1.145(±0.000)	198.245	1.055(±0.000)	34.714	1.085(±0.002)	37.046
	$k = 20, d = 1$	1.064(±0.001)	344.416	1.111(±0.003)	123.802	1.045(±0.000)	21.091	1.050(±0.000)	22.039
Exact	-	1.000(±0.000)	3037.151	1.000(±0.000)	493.967	1.000(±0.000)	103.010	1.000(±0.000)	100.203

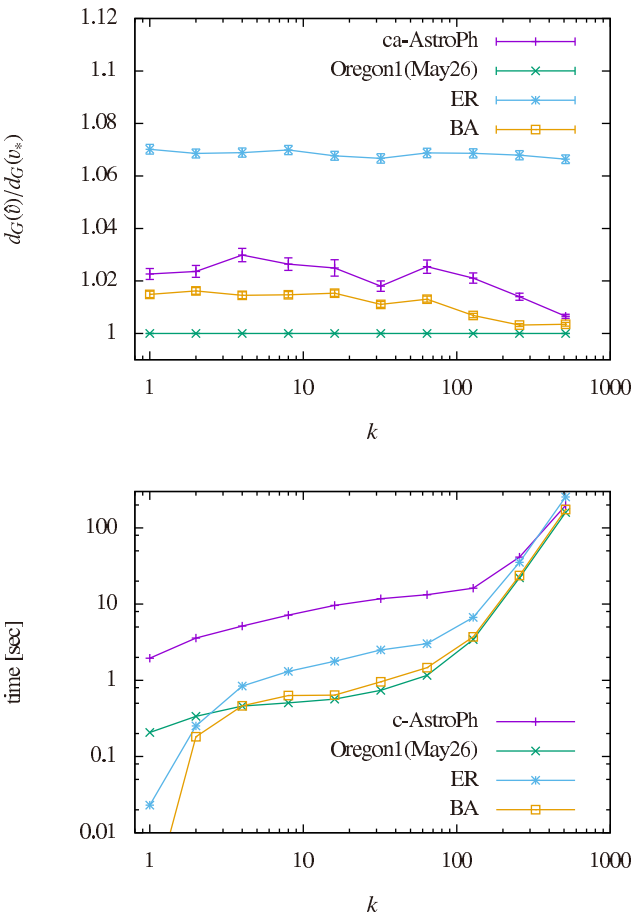

Fig. 3 **Upper graph:** Relation between parameter k and approximation ratio $d_G(\hat{v})/d_G(v_*)$, where v_* is the optimal vertex and $\hat{v}$ is the output of FAOM. The error bars show 95% confidence interval. **Lower graph:** Relation between parameter k and running time of FAOM.

tion method to compare, we selected an algorithm using DTZ (Distance To Zone) to calculate approximate distances. DTZ is a general method to efficiently estimate the distance between any two vertices of a graph using annotations that are prepared in preprocessing. DTZ was reported to perform best among the methods of annotating approach [11]. DTZ's parameters are the number of divided regions k and the number of repetitions d, and larger values of them improve approximation ratio but increase running time and memory usage. In this experiment, d is set to $20/k$ for each $k = 2, 5, 10, 20$ keeping memory usage the same[††]. Here, we also use the word "DTZ" as the algorithm using DTZ abusing the usage of the word.

Approximation ratio and running time are shown in Table 3 for the three methods, for all the four networks and for various parameters of FOAM and DTZ. Here, approximation ratio is defined as $d_G(\hat{v})/d_G(v_*)$, where v_* is the optimal vertex, that is, $\arg\min_{v \in V} d_G(v)$, and $\hat{v}$ is its estimation by a method. For real networks and the synthetic scale-free network (BA), approximation ratio of FAOM for any k is better than that of DTZ of any parameter settings used in our experiment. FAOM's approximation ratio is not good for the random graph (ER) compared to DTZ. As for running time, FAOM with $k \leq 128$ is at least three times faster than DTZ of the fastest setting ($k = 20, d = 1$), and at least 15 times faster than Exact. Restricted to real networks, which have more than 10,000 vertices, FAOM with $k \leq 128$ is at least 21 times faster than DTZ and at least 143 times faster than Exact.

8. Conclusion and Future Work

In this paper, we proposed an approximation algorithm for the 1-median problem that repeats to find a vertex with higher closeness centrality starting a randomly selected initial vertex. The key of the success of our iterative approach is what subgraph is used to efficiently obtain a tight upper

[††]DTZ stores kd values per vertex in memory for fast calculation.

bound of closeness centrality of each vertex. FAOM uses k-neighbor dense shortest path graphs with shortcuts, which results in empirical efficiency and approximation ratio close to 1. It is an important remaining issue to theoretically clarify input graph conditions under which those subgraphs are effective. Furthermore, there may be better subgraphs for this approach. It is an interesting research direction to study what subgraph is appropriate for the approach. Another interesting research direction is extension of our algorithm to the k-median problem.

References

[1] O. Alp, E. Erkut, and Z. Drezner, "An efficient genetic algorithm for the p-median problem," Annals of Operations research, vol.122(1-4), pp.21–42, 2003.
[2] R.E. Burkard and J. Krarup, "A linear algorithm for the pos/neg-weighted 1-median problem on a cactus," Computing, vol.60, no.3, pp.193–215, 1998.
[3] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," J. ACM (JACM), vol.34, no.3, pp.596–615, 1987.
[4] L.C. Freeman, "Centrality in social networks conceptual clarification," Social networks, vol.1, no.3, pp.215–239, 1978.
[5] A.J. Goldman, "Optimal center location in simple networks," Transportation science, vol.5, no.2, pp.212–221, 1971.
[6] S.L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," Operations research, vol.12, no.3, pp.450–459, 1964.
[7] D.B. Johnson, "Efficient algorithms for shortest paths in sparse networks," J. ACM (JACM), vol.24, no.1, pp.1–13, 1977.
[8] O. Kariv and S.L. Hakimi, "An algorithmic approach to network location problems. II: The p-medians," SIAM J. Applied Mathematics, vol.37, no.3, pp.539–560, 1979.
[9] Y. Kochetov, T. Levanova, E. Alekseeva, and M. Loresh, "Large neighborhood local search for the p-median problem," Yugoslav J. Operations Research, vol.15, no.1, pp.53–66, 2005.
[10] K. Okamoto, W. Chen, and X.-Y. Li, "Ranking of closeness centrality for large-scale social networks," International Workshop on Frontiers in Algorithmics, pp.186–195. Springer, 2008.
[11] M.J. Rattigan, M. Maier, and D. Jensen, "Using structure indices for efficient approximation of network properties," Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 357–366, 2006.
[12] M.G.C. Resende and R.F. Werneck, "On the implementation of a swap-based local search procedure for the p-median problem," Proc. Fifth Workshop on Algorithm Engineering and Experiments (ALENEX '03), pp.119–127, 2003.
[13] K. Tabata, A. Nakamura, and M. Kudo, "Fast approximation algorithm for the 1-median problem," Proc. 15th International Conference on Discovery Science, pp.169–183, 2012.
[14] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," J. ACM (JACM), vol.46, no.3, pp.362–394, 1999.
[15] D. Eppstein and J. Wang, "Fast approximation of centrality," J. Graph Algorithms and Applications, vol.8, no.1, pp.39–45, 2004.

Appendix A: Proof of Theorem 5.2

(*proof*) Construction of the kNSPG $T(v_0;k)$ can be done in $O(n\log n+m)$ time and $O(m)$ space by making the shortest path tree $T(v_0)$ from v_0 using Dijkstra's algorithm and adding all the edges between the nodes in $N_k(v_0)$. The calculation of $d_{T(v_0)}(v_0)(=d_G(v_0))$ can be done during the construction in $O(n)$ additional time and $O(1)$ additional space.

Let $T_{N_k(v_0)}(v_0)$ be the subgraph of $T(v_0)$ induced by $N_k(v_0)$. Define $W_{N_k(v_0)}(v)=W_{T(v_0)}(v)$ and $d_{N_k(v_0)}(v)=d^D_{T(v_0)}(v)$ for all the descendants $v\in D_{T(v_0)}(u)$ of leaf nodes $u\in T_{N_k}(v_0)$, and define $W_{N_k(v_0)}(v)=w(v)$ and $d_{N_k(v_0)}(v)=0$ for internal nodes $v\in T_{N_k}(v_0)$. Note that all the $W_{N_k(v_0)}(v)$ and $d_{N_k(v_0)}(v)$ are calculated in $O(n)$ time and space by a recursive algorithm. Let $G_{N_k(v_0)}$ be the subgraph of G induced by $N_k(v_0)$. Then, for $v\in N_k(v_0)$,

$$d_{T(v_0;k)}(v)=\sum_{u\in N_k(v_0)\setminus\{v\}}\left(d_{G_{N_k(v_0)}}(v,u)W_{N_k(v_0)}(u)+d_{N_k(v_0)}(u)\right)$$

holds. Thus, for each $v\in N_k(v_0)$, $d_{T(v_0;k)}(v)$ can be obtained in $O(k\log k+k^2)=O(k^2)$ time and $O(m)$ space calculating all the distance $d_{G_{N_k(v_0)}}(v,u)$ by Dijkstra's algorithm.

For $v\in V\setminus N_k(v_0)$, $d_{T(v_0;k)}(v)$ can be calculated by

$$d_{T(v_0;k)}(p_{T(v_0)}(v))+((W-2W(v))d(p_{T(v_0)}(v),v),$$

where W is the sum of all the weights in G. So, all the $d_{T(v_0;k)}(v)$ for $v\in V\setminus N_k(v_0)$ can be calculated in $O(n)$ time and space.

Totally, all the $d_{T(v_0;k)}(v)$ for $v\in V$ can be calculated in $O(n\log n+m+k^3)$ time and $O(m)$ space. □

Appendix B: Proof of Theorem 5.5

(*proof*) As preparation, for all $v\in V$, we calculate

(a) $W_{T(v_0)}(v)$, $C(v)$ and $d_{T(v_0;k)}(v,C(v))$,
(b) $d_{T(v_0)}(v,p^{2^j})$ for $j=0,\ldots,\log(\mathrm{depth}_{ST(C(v))}(v))$ and
(c) $\sum_{i=1}^{2^j}d_{T(v_0)}(p^i(v),p^{i-1}(v))W_{T(v_0)}(p^{i-1}(v))$ for $j=0,\ldots,\log(\mathrm{depth}_{ST(C(v))}(v))$,

where $\mathrm{depth}_{ST(C(v))}(v)$ is the depth of v in the rooted tree $ST(C(v))$.

(a) can be calculated by recursive call starting from v_0 and traversing $T(v_0)$ in $O(n)$-time and $O(n)$-space. (b) and (c) can be also calculated by similar recursive call stacking $d_{T(v_0)}(v_0,v)$ and $\sum_{i=1}^{\mathrm{depth}_{T(v_0)}(v)}d_{T(v_0)}(p^i(v),p^{i-1}(v))W_{T(v_0)}(p^{i-1}(v))$ to a stack array, from which (b) and (c) for v can be calculated in $O(\log n)$-time and $O(n)$-space. Totally, (a), (b) and (c) are calculated in $O(n\log n)$-time and $O(n\log n)$-space.

We show that $\overline{\delta}_{T(v_0;k),e}(v)$ can be calculated in $O(\log n)$ time using above (a), (b) and (c). Let $e=(s,t)$. First, v_{op} can be calculated by a kind of binary search using (b) starting from the comparison between $d_{T(v_0;k)}(v,t)=d_{T(v_0;k)}(v,C(v))+d_{T(v_0;k)}(C(v),C(t))+d_{T(V_0;k)}(C(t),t)$ and $d_{T^{\{e\}}(v_0;k)}(v,t)=d_{T(v_0;k)}(v,s)+d(s,t)$. This can be done in $O(\log n)$ time. During the binary search for the calculation of v_{op}, $\sum_{i=1}^{i_{\mathrm{op}}}d_{T(v_0)}(p^i(t),p^{i-1}(t))W_{T(v_0)}(p^{i-1}(t))$ can be calculated in $O(\log n)$ time using (c). Thus, $\overline{\delta}_{T(v_0;k),e}(v)$ can be calculated in $O(\log n)$ time. □

Appendix C: Proof of Theorem 5.6

(*proof*) By Theorem 5.5 after $O(n\log n)$-time $O(n\log n)$-

space preparation, the restricted effect $\overline{\delta}_{T(v_0;k),e}(v)$ of any edge e for any vertex v can be calculated in $O(\log n)$ time. So, we show that total number of pairs (v, e) to calculate $\overline{\delta}_{T(v_0;k),e}(v)$ is $O(kn + m)$. For each $v \in V \setminus N_k(v_0)$ and each $u \in N_k(v_0) \setminus \{C(v)\}$, $\overline{\delta}_{T(v_0;k),e}(v)$ for edges e in $E^u_v \cup \{e_{v',u} \mid p(v') = v\}$ are calculated. So, the total number of edges is

$$\sum_{v \in V \setminus N_k(v_0)} \sum_{u \in N_k(v_0) \setminus \{C(v)\}} (|E^u_v| + |\{e_{v',u} \mid p(v') = v\}|).$$

Since $\sum_{v \in V \setminus N_k(v_0)} \sum_{u \in N_k(v_0) \setminus \{C(v)\}} |E^u_v|$ is upper bounded by $2m$ and $\sum_{v \in V \setminus N_k(v_0)} |\{e_{v',u} \mid p(v') = v\}|$ is upper bounded by $n - k$, the total number of edges is $(k-1)(n-k) + 2m = O(kn + m)$. □

Koji Tabata received his M.E. degree in computer science from Hokkaido University in 2013. He is currently a doctor course student in Graduate School of Information Science and Technology, Hokkaido University.

Atsuyoshi Nakamura received his M.S. and D.S. degrees in computer science from the Tokyo Institute of Technology in 1988 and 2000. From 1988 to 2002, he worked for NEC Corporation, where he was engaged in development of database management systems, research in machine learning and its application to the WWW. He has been an associate professor at Hokkaido University since 2002. His research interests have been in the area of machine learning and data mining, especially computational learning theory, information filtering, web mining and string mining.

Mineichi Kudo received his Dr. Eng. degree in Information Engineering from the Hokkaido University in 1988. Starting from an instructor, since 2001, he is a professor (2001-) in Hokkaido University. In 2001 he received with professor Jack Sklansky the twenty-seventh annual pattern recognition society award. He was elected to a fellow of the International Association for Pattern Recognition on December 10, 2008. His current research interests include design of pattern recognition systems, image processing, data mining and computational learning theory. He is a member of the Pattern Recognition Society and the IEEE.