𝕋ℂ𝕊-𝕋ℝ-𝔸-**12-58**

𝕋ℂ𝕊 Technical Report

Fast Approximation Algorithm for the 1-Median Problem

by

Koji Tabata, Atsuyoshi Nakamura, Mineichi Kudo

Division of Computer Science

Report Series A

June 19, 2012

Hokkaido University
Graduate School of
Information Science and Technology

Email: {ktabata,atsu,mine}@main.ist.hokudai.ac.jp Phone: +81-011-706-6806
Fax: +81-011-706-7832

Fast Approximation Algorithm for the 1-Median Problem

June 19, 2012

Abstract

We present a fast approximation algorithm for the 1-median problem. Our algorithm can be applied to metric undirected graphs with node weight. Given a node v, our algorithm repeatedly executes a process of finding a node with higher centrality,in which an approximate centrality of each node v' is calculated for the subgraph called the best kNSPGS of (v,v'). The best kNSPGS of (v,v') is a subgraph that contains the shortest path tree of v, and approximate centralities of all the nodes v' for the subtrees can be calculated more efficiently than their exact centralities for the original graph. We empirically show that our algorithm runs much faster and has better approximation ratio than a sophisticated existing method called DTZ. We demonstrate the effectiveness of our algorithm through experiments.

We can use graphs to describe many kinds of relationships in daily life. For example, consider a graph with node weight to describe the transportation network. The node weight means the customer's demand and the edges mean transport routes with the cost of its length. In this situation, the facility location problem seeks an optimal location of facilities to minimize the total cost of the transport and the opening costs of the facilities. This problem has been studied since 1960s because it can be applied to many other situations. The k-median problem [1] is a special case of this problem. In the k-median problem, the number of facilities is fixed to k, but there is no cost to open facilities. Even now, this is one of the important subjects of research.

Approximation algorithms for the k-median problem have been presented so far ([2], [3]), since k-median problem is **NP**-hard. Jain et al. proposed an approximation algorithm for k-median problem by using LP-relaxation [2]. Indyk proposed an approximation algorithm based on the random sampling [3].

Freeman defined some centrality measures [4]. The closeness centrality is one of them. This centrality of node v is defined as the reciprocal number of the sum of the distances from v to other nodes. The solution of the 1-median problem, also known as the Fermat-Weber problem, is the highest closeness centrality node.

In the field of data mining, a high closeness centrality node has important meaning. For example, the closeness centrality is utilized to seek authorities from social networks and citation networks of papers. Recently, also in the field of biomedicine, this approach has been applied to protein-protein interaction networks to find important proteins.

In these days, we have a necessity for dealing with huge databases because of the diffusion of the Internet and mass storage. The number of nodes of a huge network reaches some millions. To the best of our knowledge, however, there is no algorithm to cope with such a huge graph even for 1-median problem. All of previous algorithms we know are not feasible for huge graphs because of its large calculation time cost or its large memory space cost.

For 1-median problem, the computation time by classical method is $\mathrm{O}(nm \log n)$[1], where n is the number of nodes, and m is the number of edges. Generally, 1-median problem is easier than k-median problem. But this calculation cost is too high to cope with huge graphs. Therefore some practical algorithms are proposed for 1-median problem. For example, Fujiwara et al. studied on 1-median problem for time-evolving graphs [6]. Rattigan et al. proposed an approximation algorithm for the centrality measure using an annotated information to the nodes [7].

In this article, we propose an effective fast approximating algorithm for 1-median problem to seek a node with high closeness centrality. Our algorithm can be used for the metric undirected graphs, or graphs with edge lengths satisfying the triangle inequality. The nodes of the graph can be weighted. Roughly speaking, our algorithm repeatedly executes a process of finding a node with higher centrality by using a shortest path tree. This process is expected to be fast because the cost of calculating closeness centralities of all nodes is $\mathrm{O}(n)$ for a tree and much faster than that for a general graph, namely, $\mathrm{O}(nm \log n)$.

To efficiently find a node with higher centrality, we make use of subgraphs of given graphs that contain the shortest path tree. Furthermore, we use sets of graphs called kNSPGS to improve the performance on accuracy. The computational complexity doesn't increase very much by using these techniques.

The remainder of this paper is organized as follows. In Section 2, we introduce necessary notions and notations. In Section 3, we explain our algorithm and analyze its complexity. In Section 4, we report the results of our experiments. Our brief conclusion and future works are described in Section 5.

1 Preliminaries

Assume that we are given an undirected connected graph $G = (V, E)$ with edge lengths satisfying the triangle inequality. Let n and m be the number of the nodes and edges respectively, that is, $n = |V|$ and $m = |E|$. Note that $m \geq n - 1$ because a given

[1]Note that when edge lengths are positive integer, we can reduce the calculation time to $\mathrm{O}(nm)$ by using the method proposed by Thorup [5]. In this paper, we treat graphs in which lengths of each edge can be assigned a positive real number.

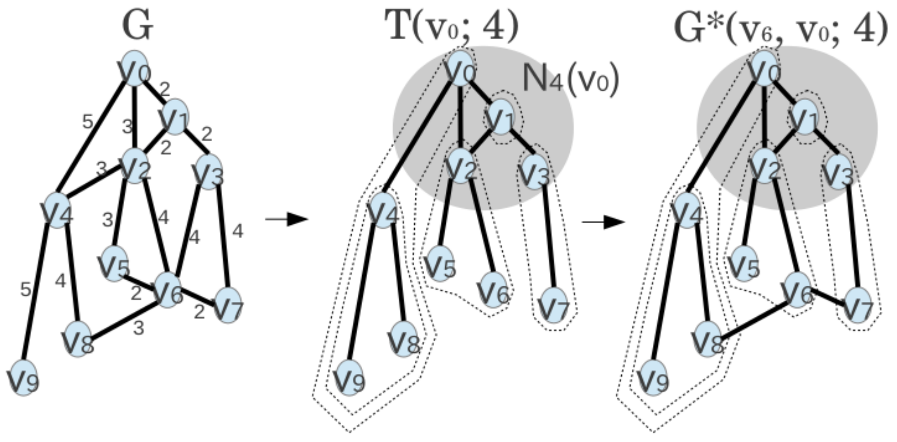

Figure 1: **Example of a kNSPG and a best kNSPGS:** The center graph is a kNSPG $T(v_0;4)$ of the left graph G. The right graph is the best kNSPGS $G^*(v_6,v_0;4)$.

graph is assumed to be connected. The nodes can be weighted. The weight of node v is denoted as $W(v)$. Assume that all the weights of nodes are positive.

The distance between two nodes v and u, which is denoted by $d(u,v)$, is defined as the minimum path length from v to u, where a path length is the sum of the length of the component edges. For a subgraph F of G, $d_F(v,u)$ denotes the minimum length of a path between v and u among the paths that are composed of edges in F only. Note that the distance for all node pairs is not given, and we know only the edge lengths, or the distances between two adjacent nodes.

We let $d(v)$ denote the weighted sum of the distances from v to all nodes u, that is, $d(v)=\sum_{u\in V}W(u)d(v,u)$. Our goal is to efficiently find a node which makes this value small. For a subgraph F of G and a node v, $d_F(v)$ is defined as $\sum_{u\in V}W(u)d_F(v,u)$.

A spanning tree of G is defined as a tree $T=(V,E')$, that satisfies $E'\subseteq E$. A spanning tree T of G is called a shortest path tree of node v if $d_T(v,u)=d_G(v,u)$ for all $u\in V$.

Definition 1 (kNSPG) *For a graph $G=(V,E)$, k-neighborhood of $v\in V$ is defined as the set of the nearest k nodes from v including v itself, and denoted as $N_k(v)$. For $v_0\in V$, the k-neighbor dense shortest path graph of v_0 (denoted as $T(v_0;k)$), the kNSPG of v_0 for short, is defined as follows:*

1. *$T(v_0;k)$ has all the edges of a shortest path tree of v_0.*

2. *For $v,u\in N_k(v_0)$, $T(v_0;k)$ has an edge $\{v,u\}$ if and only if $\{v,u\}\in E$*

Note that $T(v_0;1)$ is the shortest path tree of v_0. An example of a kNSPG is illustrated in Fig. 1. The center graph is the kNSPG $T(v_0;4)$ of the left graph G. In this graph, $N_4(v_0)=\{v_0,v_1,v_2,v_3\}$.

Definition 2 (Closest v_0-neighbor) *For $G = (V, E)$, the closest v_0-neighbor of $v \in V$ in $T(v_0; k)$, which is denoted as $C(v, v_0; k)$, is defined as the nearest node 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.*

In the graph of Fig. 1, $C(v_8, v_0; 4) = v_0$ and $C(v_1, v_0; 4) = v_1$. Note that V is partitioned into $\{\{v_0, v_4, v_8, v_9\}, \{v_1\}, \{v_2, v_5, v_6\}, \{v_3, v_7\}\}$ by the value of $C(v, v_0; 4)$.

Definition 3 (v-subtree and v-subtree weight) *For a kNSPG $T(v_0; k)$ of $G = (V, E)$ and $v \in V$, the v-subtree of $T(v_0; k)$, denoted as $ST(v, v_0; k)$, is defined as follows.*

1. *For $v \in N_k(v_0)$, $ST(v, v_0; k)$ is defined as a subtree of $T(v_0; k)$ that is composed of all the nodes of which closest v_0-neighbor is v.*
2. *For $v \notin N_k(v_0)$, $ST(v, v_0; k)$ is defined as a subtree of $T(v_0; k)$ that is composed of all the nodes u to which path from v is included in the shortest path from v_0 to u.*

The weight of the v-subtree of $T(v_0; k)$, denoted as $SW(v, v_0; k)$, is defined as the sum of all the weights of nodes in $ST(v, v_0; k)$.

In the middle graph of Fig. 1, the subgraphs surrounded by dotted lines are v-subtrees of $T(v_0; 4)$ for $v = v_0, v_1, v_2, v_3, v_4$. If all the node weights are 1, $SW(v_0, v_0; 4) = 4$ and $SW(v_4, v_0; 4) = 3$.

Definition 4 (kNSPGS) *For $G = (V, E)$ and for $v_0, v \in V$, a subgraph G' of G is said to be a kNSPG of v_0 with shortcuts from v, a kNSPGS of (v_0, v) for short, if G' is made from $T(v_0; k)$ by adding edges $\{v, u\} \subset E \setminus E'$ for at most one node u of each w-subtree for $w \in N_k(v_0) \setminus \{C(v, v_0; k)\}$, where E' is the set of edges in $T(v_0; k)$. The set of kNSPGS of (v_0, v) is denoted by $\mathbb{G}(v, v_0; k)$.*

We call $G^* \in \mathbb{G}(v, v_0; k)$ the best kNPGS of (v_0, v), which is denoted as $G^*(v, v_0; k)$, if $G^* = \min_{G' \in \mathbb{G}(v, v_0; k)} d_{G'}(v)$.

The right graph $G^*(v_6, v_0; ; 4)$ of Fig. 1 is an example of (the best) kNSPGS of (v_0, v_6). A graph which has the edge $\{v_6, v_3\}$ instead of the edge $\{v_6, v_7\}$ is also another example of kNSPGS. Any kNSPGS of (v_0, v_6) cannot have the edge $\{v_5, v_6\}$ because $C(v_5, v_0; 4) = C(v_6, v_0; 4)$ holds.

2 Algorithm

In this section, we describe our algorithm and analyze it theoretically.

2.1 Basic facts

Our algorithm is based on the following facts.

Theorem 1 *For graph $G = (V, E)$, for any subgraph $G' = (V, E')$ of G, inequality $d_G(v) \leq d_{G'}(v)$ holds for all $v \in V$.*

When E' includes all the edges of a shortest path tree of v_0, $d_G(v_0) = d_{G'}(v_0)$

Proof *For all pairs of nodes $(v, u) \in V^2$, $d_G(v, u) \leq d_{G'}(v, u)$, because all the edges in G' are included in G. Therefore, obviously $d_G(v) \leq d_{G'}(v)$ holds.*

When E' includes all the edges of a shortest path tree of v_0, $d_G(v_0) = d_{G'}(v_0)$ holds because G' has the shortest paths from v_0 to any other nodes. □

Corollary 1 *For graph $G = (V, E)$, let $G' = (V, E')$ be a subgraph of G that contains all the edges of a shortest path tree of v_0. If some $v \in V$ satisfies $d_{G'}(v) < d_{G'}(v_0)$, the following inequality holds.*

$$d_G(v) < d_G(v_0)$$

Proof

$$\begin{aligned} d_G(v) &\leq d_{G'}(v) \\ &< d_{G'}(v_0) \\ &= d_G(v_0), \end{aligned}$$

where the first inequality and the last equality hold by Theorem 1, and the second inequality holds by the assumption of this corollary. □

2.2 Outline

For given $G = (V, E)$, our algorithm repeatedly finds a node v with $d_G(v)$ that is the smaller than $d_G(v_0)$ of a node v_0, which is the node found in the last repetition. The node v_0 is initially selected randomly and the repetitions continue until no v with $d_G(v)$ that is smaller than $d_G(v_0)$ is found by our upper bound calculation of $d_G(v)$ for all $v \in V$.

To efficiently find v with $d_G(v) < d_G(v_0)$, we make use of subgraphs F that contain the shortest path tree of v_0.

For such subgraph F, it is enough to find v with $d_F(v) < d_F(v_0)$ instead of v with $d_G(v) < d_G(v_0)$ by Corollary 1. Furthermore, $d_F(v)$ for all $v \in V$ can be calculated more efficiently than $d_G(v)$ for all $v \in V$. The simplest such subgraph F is the shortest path tree $T(v_0; 1)$ of v_0 itself, for which $d_{T(v_0;1)}(v)$ for all $v \in V$ can be calculated in $O(m \log n)$ time including the construction time of $T(v_0; 1)$. We, however, found by our preliminary experiments that $d_{T(v_0;1)}(v)$ is too loose as an upper bound of $d_G(v)$.

The second simplest one is the kNSPG $T(v_0;k)$ of v_0, for which $d_{T(v_0;k)}(v)$ for all $v \in V$ can be also calculated in $\mathrm{O}(m \log n)$ time if $\mathrm{O}(k = \sqrt[3]{m})$. Accordingly to our experimental result shown in Sec. 3.1, however, $d_{T(v_0;k)}(v)$ is not enough close to $d_G(v)$ for many nodes v even if $k = \sqrt[3]{m}$.

In our algorithm, we try to find a good upper bound of $d_G(v)$ among $d_{G'}(v)$ of the kNSPGSs G' of (v_0, v).

In Fig. 2, the shortest path length from node v_6 is shown at each node in parentheses for three subgraphs of G in Fig. 1: the shortest path tree $T(v_0;1)$, the kNSPG $T(v_0;4)$ and the best kNSPGS $G^*(v_6, v_0; 4)$. Upper bounds of $d_G(v_6)$, or $d_{G'}(v_6)$ for these subgraphs G', are 98, 89 and 54, respectively, and the value of $d_G(v_6)$ is 47. Thus, the upper bound of $d_G(v_6)$ is improved well by using $G^*(v_6, v_0; 4)$.

Since the exact values of $d_{G^*(v,v_0;k)}(v)$ for all $v \in V$ require $\mathrm{O}(mn)$ time for calculation, we calculate a close upper bound of $d_{G^*(v,v_0;k)}(v)$ by stopping the calculation for effect of an added edge at $\mathrm{O}(\log n)$ depth from v. Then, such upper bounds of $d_{G^*(v,v_0;k)}(v)$ for all $v \in V$ can be also calculated in $\mathrm{O}(m \log n)$ time.

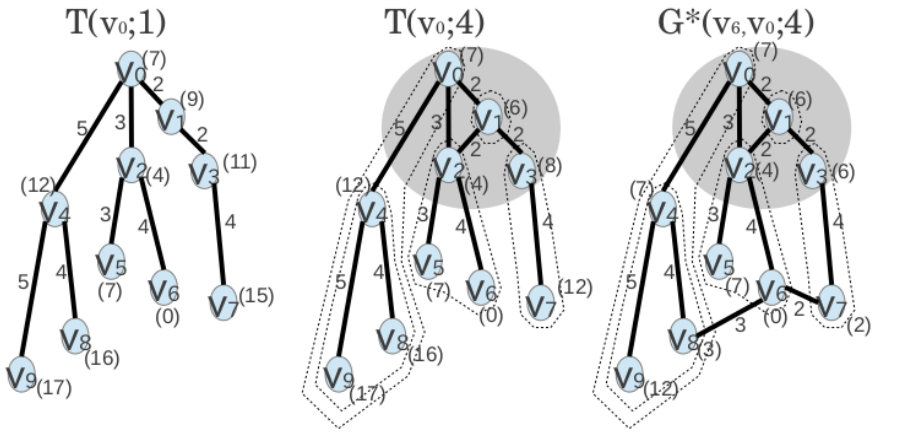

Figure 2: **Shortest path length from v_6:** The parenthesized value at each node is the shortest path length from v_6.

A pseudocode of our algorithm FAOM (Fast Approximation of One Median) is shown in Fig. 3. FAOM repeatedly executes algorithm EstimatedHigherCentralityNode which tries to find a node v with $d_G(v) < d_G(v_0)$ by calculating an upper bound of $d_{G^*(v',v_0;k)}(v')$ for each $v' \in V$ and selecting the node v with the smallest upper bound.

The calculation of an upper bound of $d_{G^*(v,v_0;k)}(v)$ is done by calculating $\delta(v)$, which is the improvement by the upper bound s of $d_{G^*(v,v_0;k)}(v)$ from $d_{T(v_0;k)}(v)$, that is, $\delta(v) = s - d_{T(v_0;k)}(v)$. The values of $\delta(v)$ for all $v \in V$ are calculated by algorithm ImprovementByShortCut, which is shown in Fig. 5. We can prove the following theorem by using lemmas that are proved in Sec. 2.3.

Theorem 2 *For $k = O(\sqrt[3]{m})$, FAOM runs in $O(ml \log n)$ time and $O(m)$ space, where l is the number of executions of EstimatedHigherCentralityNode in the algorithm.*

```
Algorithm FAOM
Require: $G = (V, E)$, $k$: parameter
Ensure: high-centrality-node
1: $v_0$ = randomly selected node in $V$
2: while TRUE do
3:     $v = EstimatedHigherCentralityNode(G, v_0, k)$;
4:     if $d_G(v) < d_G(v_0)$ then
5:         $v_0 = v$;
6:     else
7:         return $v_0$;
8:     end if
9: end while
```

```
Algorithm EstimatedHigherCentralityNode
Require: $G = (V, E)$, $v_0 \in V$: initial node, $k$: parameter
Ensure: higher-centrality-node
1: make $T(v_0; k)$;
2: calculate $d_{T(v_0;k)}(v)$ for all $v \in V$;
3: calculate $\delta(v)$ for all $v \in V$;
4: return $\arg\min_{v \in V \setminus \{v_0\}} d_{T(v_0;k)}(v) - \delta(v)$
```

Figure 3: Pseudocode of algorithm FAOM

2.3 Complexity Analysis

In algorithm EstimatedHigherCentralityNode, we first construct $T(v_0; k)$ for the node v_0 that is obtained in the last execution and for a fixed parameter k. It seems that the larger k becomes, the tighter the obtained upper bound of $d_G(v)$ for each $v \in V$ becomes, but larger k will make the algorithm slower and require more memory space.

We analyze the time and space complexities of algorithm FAOM by considering the effect of k.

Lemma 1 *For a given graph $G = (V, E)$, $d_{T(v_0;k)}(u, v)$ for $u, v \in N_k(v_0)$ and $d_{T(v_0;k)}(v, C(v, v_0; k))$ for all the nodes $v \in V$ can be calculated in $O(m \log n + k^3 \log k)$ time and $O(m + k^2)$ space.*

Proof *In this proof, we use d and $C(v)$ instead of $d_{T(v_0;k)}$ and $C(v, v_0; k)$, respectively, for simplicity.*

At first, consider the process of making a kNSPG $T(v_0; k)$. This process can be done in $O(m \log n)$ time and $O(m)$ space by making the shortest path tree using Dijkstra's algorithm and adding edges inside $N_k(v_0)$.

Dijkstra's algorithm constructs the shortest path tree of v_0 by repeatedly growing the shortest path tree of v_0 for $N_i(v_0)$ to that for $N_{i+1}(v_0)$ starting from that for

$N_1(v_0) = \{v_0\}$. Thus, we can know $N_k(v_0)$ when the number of nodes in the growing shortest path tree reaches to k. At that time, we can construct the subgraph I of G that is induced by $N_k(v_0)$ and can calculate $d_{T(v_0;k)}(u,v)$ for $u, v \in N_k(v_0)$ by executing Dijkstra's algorithm for I and for each node in $N_k(v_0)$. The total cost of this process is $O(k^3 \log k)$ time and $O(k^2)$ space.

For the ith nearest node v^i is added to the growing shortest path tree as its ith node. We show that additional $O(1)$ time is enough for calculating $d(v^i, C(v^i))$ for all $i = 1, 2, ..., n$ by mathematical induction. First, $d(v^i, C(v^i)) = d(v^i, v^i) = 0$ for all $i = 1, 2, ..., k$. Assume that $d(v^j, C(v^j))$ is already calculated for $j \leq i$. When v^{i+1} is added to the growing shortest path tree, edge $\{v^j, v^{i+1}\}$ for some $j \leq i$ is also added to the tree and $d(v^{i+1}, C(v^{i+1}))$ can be calculated in $O(1)$ time using the equation $d(v^{i+1}, C(v^{i+1})) = d(v^{i+1}, v^j) + d(v^j, C(v^j))$. Thus, the calculation of $d(v, C(v))$ for all $v \in V$ costs additional $O(n)$ time and $O(n)$ space.

By the above arguments, we obtain $T(v_0; k)$ and $d(u, v)$ for $u, v \in N_k(v_0)$ and $d(v, C(v))$ for all the nodes $v \in V$ in $O(m \log n + k^3 \log k)$ time and $O(m + k^2)$ space. □

Lemma 2 *For a graph $G = (V, E)$, if $T(v_0; k)$ and $d_{T(v_0;k)}(u, v)$ for $u, v \in N_k(v_0)$ and $d_{T(v_0;k)}(v, C(v, v_0; k))$ for all the nodes $v \in V$ are given, we can calculate $d_{T(v_0;k)}(v)$ for all nodes $v \in V$ in $O(n + k^2)$ time and $O(n + k^2)$ space.*

Proof *In this proof, we also use d instead of $d_{T(v_0;k)}$ for simplicity.*

First of all, calculate $SW(v, v_0; k)$ for all $v \in V$. This can be done in $O(n)$ time and $O(n)$ space by using depth fast search for each v-subtree for $v \in N_k(v_0)$.

For $v \in N_k(v_0)$, we can show that $d(v)$ can be obtained in $O(k^2)$ time as follows. Since

$$\begin{aligned}
d(v) - d(v_0) &= \sum_{u \in V} W(u) \{d(v,u) - d(v_0,u)\} \\
&= \sum_{c \in N_k(v_0)} \sum_{u \in ST(c,v_0;k)} W(u) \{d(v,u) - d(v_0,u)\} \\
&= \sum_{c \in N_k(v_0)} \sum_{u \in ST(c,v_0;k)} W(u) \{(d(v,u) - d(c,u)) - (d(v_0,u) - d(c,u))\} \\
&= \sum_{c \in N_k(v_0)} \sum_{u \in ST(c,v_0;k)} W(u) \{d(v,c) - d(v_0,c)\} \\
&= \sum_{c \in N_k(v_0)} SW(c, v_0; k) \{d(v,c) - d(v_0,c)\}
\end{aligned}$$

holds, $d(v)$ can be calculated by $d(v_0) + \sum_{c \in N_k(v_0)} SW(c, v_0; k) \{d(v,c) - d(v_0,c)\}$, of which calculation requires $O(k)$ time. For all $v \in N_k(v_0)$, repeating this calculation k times, now we obtain the $d(v)$ for $v \in N_k(v_0)$ in $O(k^2)$ time.

Next, calculate the sum of the weight of all nodes, which is denoted as W_{all}, in $O(k)$ time because W_{all} is $\sum_{u\in N_k(v_0)} SW(u,v_0;k)$.

For any $v \notin N_k(v_0)$ and its parent node $u \in V$,

$$\begin{aligned} d(v) - d(u) &= -SW(v,v_0;k)d(v,u) + (W_{all} - SW(v,v_0;k))d(v,u) \\ &= (W_{all} - 2SW(v,v_0;k))d(v,u) \end{aligned}$$

holds. This means that $d(v)$ can be calculated in $O(1)$ time if $d(u)$ is already known. Since $d(u')$ for $u' \in N_k(v_0)$ is already known, any $v \notin N_k(v_0)$ can be calculated in $O(1)$ time by mathematical induction. Therefore, this calculation costs $O(n)$ time in total.

By summing all the costs, $O(n+k^2)$ time, $O(n+k^2)$ space is enough for calculating $d(v)$ for all $v \in V$. □

By the lemma 1 and 2, for a given graph $G = (V,E)$, we can calculate $d_{T(v_0;k)}(v)$ for all the nodes $v \in V$ in $\mathrm{O}(m\log n + k^3 \log k)$ time and $\mathrm{O}(m+k^2)$. Considering the time complexity of the algorithm, parameter k should be set to a value that is at most $\mathrm{O}(\sqrt[3]{m})$

Lemma 3 *If $SW(v,v_0;k)$ for all $v \in V, d_{T(v_0;k)}(v, C(v,v_0;k))$ for all $v \in V$ and $d_{T(v_0;k)}(v,u)$ for all $u,v \in N_k(v_0)$ are given, $\delta(v)$ for all $v \in V$ can be calculated in $O(m\log n)$ time and $O(m+k^2)$ space.*

Proof *For a given graph $G = (V,E)$ and kNSPG $T(v_0;k) = (V,E')$, fix $v \in V$.*

Let T' be the subgraph of G that is made by adding an edge $\{v,u_0\} \in E \setminus E'$ to $T(v_0;k)$ as shown in Fig. 4. Define $\mathcal{D}$ as

$$\mathcal{D} \ = \ d_{T(v_0;k)}(v,u_0) - d_{T'}(v,u_0).$$

Then

$$\begin{aligned} \mathcal{D} &= d_{T(v_0;k)}(v, C(v,v_0;k)) + d_{T(v_0;k)}(C(v,v_0;k), C(u_0,v_0;k)) \\ &+ d_{T(v_0;k)}(C(u_0,v_0;k),u_0) - d_G(v,u_0) \end{aligned}$$

holds. Suppose that $D > 0$, that is, the distance between v and u_0 can be shortened by adding an edge $\{v,u_0\}$ to $T(v_0;k)$. Then, for all the nodes $u \in ST(u_0,v_0;k)$,

$$\begin{aligned} d_{T'}(v,u) &= d_{T'}(v,u_0) + d_{T'}(u_0,u) \\ &= d_{T(v_0;k)}(v,u_0) - \mathcal{D} + d_{T(v_0;k)}(u_0,u) \\ &= d_{T(v_0;k)}(v,u) - \mathcal{D} \end{aligned}$$

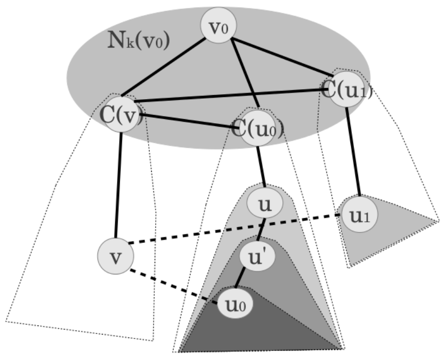

Figure 4: **Improvement by shortcuts.**

holds. This means that $d_{T'}(v)$ is smaller than $d_{T(v_0;k)}(v)$ by at least $\mathcal{D}SW(u_0, v_0; k)$.

Furthermore, let u be a node on the shortest path between u_0 and $C(u, v_0; k)$ of $T(v_0; k)$. Let $\mathcal{D}'$ denote the shortened distance between v and u by adding an edge $\{v, u_0\}$ to $T(v_0; k)$, that is,

$$\mathcal{D}' = d_{T(v_0;k)}(v, u) - d_{T'}(v, u).$$

Then

$$\begin{array}{rcl} \mathcal{D}' & = & d_{T(v_0;k)}(v, C(v, v_0; k)) + d_{T(v_0;k)}(C(v, v_0; k), C(u_0, v_0; k)) \\ & + & d_{T(v_0;k)}(C(u_0, v_0; k), u) - (d_G(v, u_0) + d_{T(v_0;k)}(u_0, u)) \end{array}$$

holds. As long as $\mathcal{D}' > 0$ and $u \notin N_k(v_0)$, for all nodes $w \in ST(u, v_0; k) \setminus ST(u', v_0; k)$, where u' is the child node of u on the path to u_0,

$$\begin{array}{rcl} d_{T'}(v, w) & = & d_{T'}(v, u) + d_{T'}(u, w) \\ & = & d_{T(v_0;k)}(v, u) - \mathcal{D}' + d_{T(v_0;k)}(u, w) \\ & = & d_{T(v_0;k)}(v, w) - \mathcal{D}' \end{array}$$

holds. This means that $d_{T'}(v)$ is smaller than $d_{T(v_0;k)}(v)$ by additional $\mathcal{D}'(SW(u, v_0; k) - SW(u', v_0; k))$.

We can keep on this calculation to the upstream as long as $\mathcal{D}' > 0$ and $u \notin N_k(v_0)$. For each step, calculation cost is $O(1)$ time. Therefore the total calculation time is proportional to the calculation depth. If we stop this calculation at $O(\log n)$ depth, the total calculation cost is upper bounded by $O(\log n)$ even in the worst case.

During calculating $\delta(v)$ for all the node $v \in V$, the above calculation is done twice for all the edges in $E \setminus E'$. Therefore, this can be done in $O(m \log n)$ time in total. As for the required space, $O(m + k^2)$ space is enough for keeping the original graph and the node distances in $T(v_0; k)$. □

According to Theorem 1, for any $v \in V$, $d_G(v) \leq d_{T(v_0;k)}(v) - \delta(v)$. Therefore if the some $v \in V$ satisfies $d_{T(v_0;k)}(v) - \delta(v) \leq d_{T(v_0;k)}(v_0) = d_G(v_0)$, we get $d_G(v) \leq d_G(v_0)$.

We will test its effectiveness in experiments.

```
Algorithm ImprovementByShortCut
Require: $G = (V, E)$, $T(v_0; k) = (V, E')$, initial node $v_0 \in V$,
Ensure: $\delta(v)$ for all $v \in V$
1: for all $v \in V$ do
2:     for all $u \in V$ such that $\{v, u\} \in E$ do
3:         if $C(u, v_0; k) \neq C(v, v_0; k)$ then
4:             $\delta_u(v) = ImprovementByOneShortCut(v, u)$;
5:         end if
6:     end for
7:     $\delta(v) = \sum_{c \in N_k(v_0)} \max_{u \in ST(c, v_0; k)} \delta_u(v)$;
8: end for
9: return $\delta$
```

```
Algorithm ImprovementByOneShortCut
Require: $G = (V, E)$, $T(v_0; k) = (V, E')$, $v, u \in V$
Ensure: $\delta_{u'}(v)$ for $u' = C(v, v_0; k)$
1: $result = 0$;
2: $D = d(v, C(v)) + d(C(v, v_0; k), C(u, v_0; k)) + d(C(u, v_0; k), u) - d(v, u)$;
3: $sw = 0$;
4: $repeat = 0$;
5: while $D > 0$ do
6:     $result = result + (SW(u, v_0; k) - sw)D$;
7:     if $u = C(u)$ or $repeat > \log n$ then
8:         return $result$
9:     end if
10:    $repeat = repeat + 1$;
11:    $sw = SW(u, v_0; k)$;
12:    $D = D - 2d(u,$ parent of $u)$;
13:    $u =$ parent of $u$;
14: end while
15: return $result$
```

Figure 5: calculation of $\delta(v)$ for all $v \in V$

3 Experiments

We conducted experiments to demonstrate the effectiveness of our algorithm.

As datasets, we used the maximum connected component of "Collaboration network of Arxiv Astro Physics category" (17903 nodes and 197031 edges, 'CA-AstroPh' for short) and the maximum connected component of "AS peering information inferred from Oregon route-views"(11174 nodes and 23409 edges, 'oregon1_010526' for short) of Stanford Large Network Dataset Collection[8], and also used automatically generated graphs based on Erdos-Renyi model [9] (4986 nodes and 15118 edges, 'ER' for short) and Barabasi-Albert model [10] (5000 nodes and 14968 edges, 'BA' for short).

ER is a random graph model, in which all the pairs of nodes are connected randomly with a given probability, which was set to in our experiments. BA is scale-free graph model, in which generated graphs have a similarity to real-world datasets like a characteristic of power-law distribution of node degrees. Since a graph of ER is not guaranteed to be connected, we used a maximum connected component of a generated graph. In ER and BA, we added the node weights randomly between 0 and 1, the edge lengths randomly between 1 and 2. Note that, these edge lengths satisfy triangle inequality. In CA-AstroPh and oregon1_010526, we assigned 1.0 to all the node weights and 1.0 to all the edge lengths.

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.

3.1 Effect of edges inside $N_k(v_0)$ and shortcuts

To see the effect of adding the edges inside $N_k(v_0)$ to the shortest path tree of v_0, and the effect of adding shortcuts to a kNSPG of v_0, we conducted experiments on the approximation ratio for estimation of $d_G(v)$ for all the node v.

For each of the randomly selected 100 initial nodes v_0, we calculated $d_{T(v_0;1)}/d_G(v)$, $d_{T(v_0;k)}/d_G(v)$ and $(d_{T(v_0;k)}(v) - \delta(v))/d_G(v)$ for all the nodes v. Then, we made a histogram of the values for each of the three. The result for the CA-AstroPh is shown in Fig. 6.

As you can see on this graph, the approximation ratio is improved by using edge inside $N_k(v_0)$ and shortcuts.

3.2 Number of repetitions of main loop

We examined the number of repetitions of the main loop, that is, the number of execution of EstimatedHigherCentralityNode. We randomly selected 100 nodes from a given graph and counted the number of repetitions of main loop by giving each selected node to our algorithm as an initial node. The result is shown in Table 1. As compared with the number of nodes, the number of repetitions is very small on any graph in our experiments. As a result, our algorithm runs fast for the datasets.

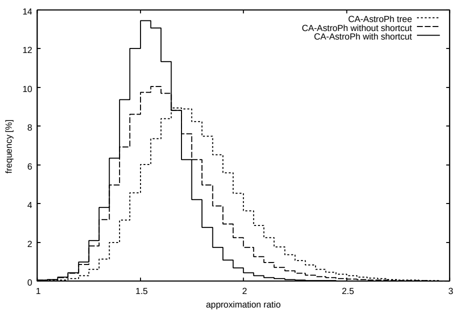

Figure 6: **Effect of edges inside $N_k(v_0)$ and shortcuts:** The frequency of the value of $d_{T(v_0;1)}(v)/d_G(v)$, $d_{T(v_0;k)}(v)/d_G(v)$ and $(d_{T(v_0;k)}(v) - \delta(v))/d_G(v)$ are shown as a dotted line, a dashed line and a solid line, respectively. Note that parameter k was set to $\lfloor \sqrt[3]{m} \rfloor = 58$.

Table 1: **Repetition times of main loop**

			#repetition						
	#node	#edge	1	2	3	4	5	6	7
CA-AstroPh	17903	197031	2	15	30	34	15	3	1
oregon1_010526	11174	23409	0	54	46	0	0	0	0
BA	5000	14968	0	48	30	15	4	3	0
ER	4986	15118	12	34	31	14	5	2	2

3.3 Effect of k

We also conducted an experiment on the effect of k.

We evaluated our algorithm's accuracy by the number of times where the exact 1-median is obtained, and also evaluated its efficiency by its average wall clock time. For each $k = 1, 2, 4, \ldots, 512$, we executed our algorithm 100 times by giving randomly selected initial nodes.

The result is shown in Fig. 7.

As you can see on this graph, the larger k becomes, the larger the number of exact outputs becomes, though the calculation cost increases.

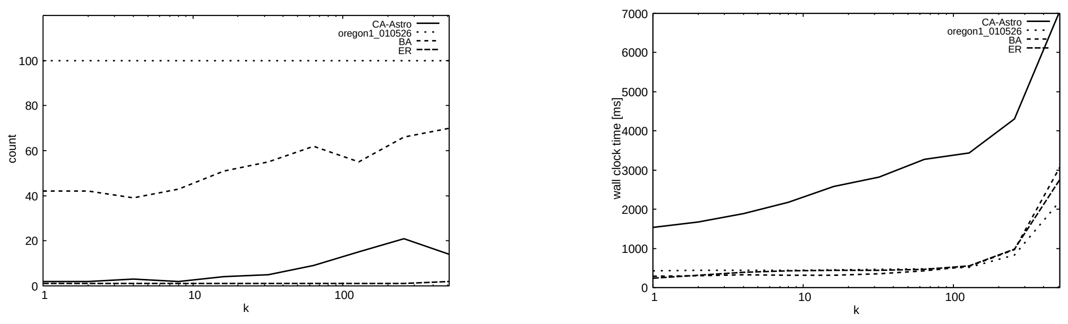

Figure 7: **Effect of k:** The horizontal axis is a logarithm scale. In this experiment, k was set to $1, 2, 4, 8, ..., 512$. The result was averaged over 100 runs for each k.

3.4 Comparison to a previous method

We used approximation algorithm DTZ(Distance To Zone) for comparison. DTZ is a method to estimate the distance between nodes using annotating approach. We selected DTZ because DTZ shows the best performance among the methods using annotating approach according to the report in [7]. DTZ has a two parameters k and d. These are the number of divided regions and the number of repetitions, respectively.

We compared Exact, DTZ and our method (FAOM) on a computation time and an approximation ratio, where Exact is a method of calculating the correct 1-median using Dijkstra's algorithm for each node. The approximation ratio is defined as $d_G(\hat{v})/d_G(v_*)$, where v_* is the correct 1-median and $\hat{v}$ is its estimation by an algorithm. The result is shown in Table 2.

As you can see on this table, our method FAOM is overwhelmingly fast. Our algorithm outperformed DTZ also on accuracy except for ER dataset.

4 Conclusion and Future Work

In this paper, we have proposed a fast approximation algorithm for the 1-median problem.

Our method, FAOM, is based on the fact that closeness centralities of all nodes of a tree can be calculated much faster than those of a general graph.

FAOM shows good performance on accuracy and calculation time as compared with an existing method called DTZ through experiments. Especially on the calculation speed, FAOM runs more than 100 times faster than DTZ for real datasets when parameter k is not so large.

Furthermore, it is shown that our algorithm don't require much space as compared to the space to memorize a given graph in order evaluation.

How to choose parameter k is remained as the future work.

Table 2: **Comparison Exact, DTZ and FAOM**: The results are the average over 100 runs for each method. The unit of calculation time is milli second.

	BA		ER		oregon1_010526		CA-AstroPh	
Method	time	approx.	time	approx.	time	approx.	time	approx.
Exact	97,919	1.000000	100,652	1.000000	361,943	1.000000	2,115,636	1.000000
DTZ(k=2,d=1)	14,261	1.361994	14,424	1.141948	75,896	1.193440	203,754	1.414486
DTZ(k=5,d=1)	15,284	1.106395	15,066	1.085089	75,917	1.159146	205,134	1.224624
DTZ(k=2,d=5)	43,212	1.069461	45,096	1.071289	308,623	1.215059	770,141	1.256972
DTZ(k=5,d=5)	56,369	1.102151	54,647	1.084036	316,501	1.222761	793,480	1.120214
FAOM(k=1)	275	1.034788	200	1.081792	381	1.000000	1,526	1.040308
FAOM(k=2)	305	1.034788	263	1.081792	392	1.000000	1,687	1.042018
FAOM(k=4)	320	1.031950	325	1.081792	396	1.000000	1,912	1.039726
FAOM(k=8)	332	1.033483	358	1.081792	379	1.000000	2,213	1.034022
FAOM(k=16)	318	1.036259	377	1.081792	372	1.000000	2,356	1.035492
FAOM(k=32)	326	1.024823	385	1.081792	398	1.000000	2,749	1.023037
FAOM(k=64)	410	1.031825	409	1.080757	424	1.000000	3,010	1.040712
FAOM(k=128)	510	1.033758	490	1.079958	472	1.000000	3,299	1.025930
FAOM(k=256)	993	1.010458	885	1.080207	788	1.000000	4,053	1.024322
FAOM(k=512)	3,029	1.002329	2,627	1.080207	2,151	1.000000	6,800	1.025059

References

[1] Hakimi, S.L.: Optimal locations of switching centers and the absolute centers and the medians of a graph. Operations Research 12, 450–459. (1964)

[2] Jain, Kamal and Vazirani, Vijay V.: Approximation algorithms for metric facility location and k-Median problems using the primal-dual schema and Lagrangian relaxation. J. ACM 48, 274–296. (2001)

[3] Indyk, Piotr: Sublinear time algorithms for metric space problems. Proceedings of the thirty-first annual ACM symposium on Theory of computing, 428–434. (1999)

[4] Freeman, Linton C and Roeder, Douglas and Mulhollan, Robert R .: Centrality in social networks conceptualclarification Social Networks, Volume 2, Issue 2, 119–141 (1979)

[5] Thorup, Mikkel: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM 46, 362–394. (1999)

[6] Fujiwara, Yasuhiro and Onizuka, Makoto and Kitsuregawa, Masaru: Efficient Centrality Monitoring for Time-Evolving Graphs. ADVANCES IN KNOWLEDGE DISCOVERY AND DATA MINING 6635/2011, 38-50. (2011)

[7] Rattigan, Matthew J. and Maier, Marc and Jensen, David: Using structure indices for efficient approximation of network properties. KDD ’06, 357–366. (2006)

[8] Stanford Large Network Dataset Collection, http://snap.stanford.edu/data/index.html (accessed: May 04, 2012)

[9] Erdos, P. and Renyi, A. : On the evolution of random graphs, Citeseer. (1960)

[10] Barabasi, A.L. and Albert, R. and Jeong, H. : Mean-field theory for scale-free random networks, Statistical Mechanics and its Applications, 272, 173–187. (1999)