HOKKAIDO UNIVERSITY

Title	社会的ネットワークにおける効率的な重要ノードの発見の手法とその応用
Author(s)	田畑, 公次
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第12816号
Issue Date	2017-06-30
DOI	https://doi.org/10.14943/doctoral.k12816
Doc URL	https://hdl.handle.net/2115/66527
Type	doctoral thesis
File Information	Koji_Tabata.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

Doctoral Dissertation

Efficient Methods for Finding Important Nodes in Social Networks and Their Applications

Koji TABATA

Hokkaido University
Graduate School of Information Science and Technology
Division of Computer Science

June 2017

学 位 論 文 内 容 の 要 旨

博士の専攻分野の名称　　博士 (情報学)　　氏名　田畑公次

学 位 論 文 題 名

社会的ネットワークにおける効率的な重要ノードの発見の手法とその応用
(Efficient Methods for Finding Important Nodes
in Social Networks and Their Applications)

近年, インターネット上でのソーシャルネットワーキングサービスの普及により, 人と人との様々なつながり方がネットワークデータとして得られるようになってきた. ネットワーク上での重要なノードを見つける研究は従来から行われてきたが, 大規模なデータも入手可能となってきたという状況の中, 効率的に重要ノードを発見する手法の需要が高まってきている.

ネットワーク上でのノードの重要性を測る尺度として, 距離中心性 (closeness centrality), 媒介中心性 (betweenness centrality), 離心中心性 (eccentricity centrality), 次数中心性 (degree centrality), 固有ベクトル中心性 (eigenvector centrality) など様々な中心性が用いられている. さらに近年ネットワーク上での情報伝播に基づいてノードの重要性を評価する影響度 (influence) なども提案されている. これらの尺度は発見するノードの使用目的により使い分けられる.

例えば, あるノードの距離中心性は, そのノードから他のノードまでの距離の逆数で定義される. 距離中心性が最大のノードは, 言い換えると他のノードに至るまでの平均距離が最小となるノードである. したがって, 距離中心性を尺度として用いると, どのノードからも比較的アクセスしやすく重要な位置にあるノードを見つけることが可能である. ネットワーク及びユークリッド空間上で距離中心性が最大のノードを見つける問題は 1-median 問題と呼ばれる古典的な最適化問題であり, 最適施設配置問題としても扱われてきた.

また, あるノード集合の影響度は, その集合に属するノードからネットワークを介して情報伝達が行われると仮定した場合に, 情報が伝わるノード数の期待値で定義される量である. インターネット上でのソーシャルネットワークを介した方が, オフラインでのソーシャルネットワークを介するより高速かつ広範囲に情報が広がりやすいため, インターネット上での口コミを利用

してマーケティングを行う動きが高まっている. このようなマーケティングは, 情報がウイルスが拡散するように指数関数的に広範囲に広がっていく様子に例えてバイラルマーケティングと呼ばれている. 影響度の尺度は, バイラルマーケティングにおいて効果的なノード集合を見つけるのに使うことができるため, 近年注目されている.

本研究では, ネットワークにおけるノードの重要度を測る尺度として, 距離中心性と影響度の 2 つに焦点を絞り, それらの指標を用いた重要ノードの効率的な発見法を提案し, それらの有効性を示す. 距離中心性の尺度においては, 1-median 問題に対する反復法を用いた効率的な近似アルゴリズムを提案する. また, 影響度の尺度に関しては, Independent Cascade (IC) モデルと呼ばれる辺毎に伝播確率が与えられる情報伝達モデルにおいて, 二端子直並列グラフに対して影響最大化問題を正確 (exact) に解くアルゴリズムを提案する. 以下にそれぞれの概要と, 本研究における特徴について述べる.

[1-median 問題に対する反復法を用いた効率的な近似アルゴリズム]

グラフにおける 1-median 問題とは, 辺に長さのある無向グラフ $G = (V, E)$ において, ノード $v \in V$ から他のノードまでの距離の和を $d(v)$ としたとき, $d(v)$ を最小化するようなノード v を見つける問題である. 提案手法では, ノード $v_0 \in V$ に対して v_0 を根とする全域木を含む部分グラフ G' を考えると, G' においてあるノード v から他のノードまでの距離の和 $d'(v)$ が $d'(v) \geq d(v)$ を満たすことを利用して, 反復的により良い解を求める手続きを行うことで近似解を求める. 全域木を含む部分グラフ G' として単純なグラフを用いることにより, 効率的に $d'(v)$(またはその上界) を求めることができるため, 1 回の反復にかかる時間を短くでき, 更に実験によれば少ない反復回数で収束するため, 高速な計算が可能である. 提案法では 1 回の反復を $O((n+m)\log n)$ で行うことができる部分グラフを用いた.

すべてのノードに対して最短経路木を作成することにより $d(v)$ を計算すれば, $O(n^2 \log n + mn)$ 時間で問題を正確 (exact) に解くことが可能であるが, これは巨大なグラフにおいて実行が困難である. 実データや人工データを用いた実験によれば, 本手法はこの手法より 100 倍程度高速に動作する. また DTZ と呼ばれる従来の近似手法と比較してもスケールフリーネットワークに対して精度と実行速度の両方において優れていることが実験により示されている.

[二端子直並列グラフに対して影響最大化問題を正確に解くアルゴリズム]

影響最大化問題は, 与えられた小さな整数 k に対し, k 要素の情報源ノード集合 S で, その影響度 $\sigma(S)$ を最大化するものを見つける問題である. この問題は, IC モデルなどの一般的な情報伝達モデルでは NP 困難であることが知られているため, 貪欲法による効率的な近似アルゴリズムの研究が盛んに行われている.

本研究では,IC モデルにおいて, 扱うグラフを二端子直並列グラフに制限したときに影響最大化問題を正確 (exact) に解くアルゴリズムを提案する. 従来知られている一般グラフに対する貪欲法による近似解法では得られる解集合の影響度が最適解に比べ $(1 - 1/e)$ 倍以上になることが理論的に保証されているものの, ランダムに生成した二端子直並列グラフにおいては最適解の 0.8 倍程度の近似解しか得られない.

従来, 効率的に影響最大化問題の最適解を求める手法は, 木に対しては動的計画法を用いた手法が知られていたが, 二端子直並列グラフなどの木よりも複雑なクラスに対しては知られていなかった. 提案アルゴリズムでは, ある種の極大性を満たすものしか解に成り得ないという性質を利用して候補を絞ることにより効率化を図っている. 分割統治法を用いて, 部分グラフの解の候補を合成することにより全体のグラフに対する解の候補を計算するが, 解の候補の合成をする際に極大性のチェックを行うことにより解の候補を常に少なく保つことができる. 実験によれば, ランダムに生成した 800 ノードの二端子直並列グラフに対し, 提案法を用いれば $k = 7$ の影響最大化問題の最適解を 1 分程度で求めることが可能である.

Acknowledgment

I am deeply grateful to my advisor Dr. Atsuyoshi Nakamura, laboratory associate professor, for a large amount of advice and support of my Ph.D study. When my research didn't go well, he had always found a new direction and encouraged me. He also gave me an opportunity to work as a research assistant and a teaching fellow. It was a pleasure for me to work for Graduate School of Information Science and Technology in Hokkaido University, and also stimulated my research.

I would also like to express my sincere gratitude to Dr. Mineichi Kudo, laboratory professor, for a lot of advice about the way to writing a paper, making a presentation and other various necessary things for a researcher.

In addition, I am very grateful to Dr. Shin-ichi Minato, one of my examiners, for his helpful comments such as on related researches.

It was also a great help for me that Graduate School of Information Science and Technology, my affiliation, allow me to work as a research assistant. In adittion, I would like to show my thanks to all the members in Laboratory for Pattern Recognition and Machine Learning.

Lastly, I would like to greatly thank my family, my parents and my brother, for supporting financially and spiritually me for a long student life.

Contents

1 Introduction **1**

1.1 Overview 1

1.2 Contribution 2

2 Preliminaries **4**

2.1 Graph 4

2.2 Shortest Path Tree and Dijkstra's algorithm 5

2.3 Artifitial Network Models 8

3 1-median Problem **10**

3.1 Overview 10

3.2 Problem Setting 11

3.3 Iterative Algorithm Framework 12

3.4 Subgraphs for Efficient Approximation 13

3.4.1 k-Neighbor Dense Shortest Path Graph 13

3.4.2 kNSPG with Shortcuts 16

3.5 Algorithm FAOM 20

3.5.1 Algorithm 20

3.5.2 Experiments and Results 21

3.6 Algorithm FAOM2 27

3.6.1 Algorithm 27

3.6.2 Experiments 29

3.6.3 Experimental Setting 29

3.6.4 Effect of Using kNSPG with Shortcuts 30

3.6.5 Number of Main-Loop Iterations 30
3.6.6 Effect of Using Larger k 30
3.6.7 Comparison to Other Methods 32

4 Influence Maximization **35**
4.1 Overview 35
4.1.1 Related Works 37
4.2 Preliminary 38
4.2.1 Influence Maximization in Independent Cascade Model 38
4.2.2 Two-Terminal Series Parallel Graph 40
4.3 Algorithm for a TTSPG 42
4.3.1 Local Effect Function of Source Activation Probability 42
4.4 Algorithm 45
4.5 Application to a General Network 48
4.6 Experiments 51
4.6.1 Experiments for TTSPGs 51
4.6.2 Dataset 51
4.6.3 Comparative Algorithm 52
4.6.4 Experimental Settings 52
4.6.5 Results 53
4.7 Experiments for Real Networks 54

5 Conclusion **57**
5.1 Results 57
5.2 Future Directions 58

List of Figures

2.3.1 An example of graph generated by Erdös-Rényi Model. . . . 8
2.3.2 Example of generation by Barabasi-Albert Model. $n = 10$, $m_0 = 4$ and $m = 3$. 9

3.4.1 Example of a kNSPG and a kNSPG with shortcuts 15
3.5.1 Distribution of approximation ratio $d_F(v)/d_G(v)$ 25
3.6.1 Distribution of approximation ratio $d_{G_v}(v)/d_G(v)$ 31
3.6.2 Relation between parameter k and approximation ratio 33

4.2.1 An example of an influence propagation network. 39
4.2.2 An example of an SP decomposition tree of a TTSPG. 41
4.5.1 tree and pseudo-TTSPG 50
4.6.1 Running times and their 95% confidence intervals 53
4.6.2 Running times and their 95% confidence intervals for the graphs with 800 nodes. 54
4.6.3 The approximation ratio $\sigma(\hat{S})/\sigma(S^*)$ of Greedy 55

List of Tables

2.1 The calculation cost of binary heap and Fibonacci heap 7

3.1 Datasets used in our experiments 24
3.2 The number of main-loop iterations 25
3.3 Comparison of FAOM, DTZ and Exact 26
3.4 The number of main-loop iterations 31
3.5 Approximation ratio $d_G(\hat{v})/d_G(v_*)$ and running time 34

4.1 Authors extracted as the solution of influence maximization problem 56

Chapter 1

Introduction

1.1 Overview

By the recent diffusion of social networking services on the Internet, various relationships among people have become obtainable in the form of network data. Now, analyses of such social network data have become very popular and one of them is to find important nodes in a given network. In these days, the size of the network we have to analyze is increasing, so faster algorithms are required for the problem. In this thesis, we study efficient algorithms for the problem of finding important nodes in a large network.

To quantify how important a node is in a network, various centrality measures have been proposed so far: closeness centrality[1], betweenness centrality[1], eccentricity centrality[2], eigenvector centrality[3], influence[4] and so on. In this thesis, we focus on two of them, closeness centrality and influence.

The *closeness centrality* of a node is defined as the reciprocal number of the sum of distances from the node to all the other nodes. The node with the highest closeness centrality can be rephrased as the node that minimizes the the sum of distances to all the other nodes. The problem to find the node satisfying such a property is called 1-median problem, which is a kind of classical problem to find the optimal point not only in a network but also

in Euclidean space such as the facility location problem. We study heuristic approximate algorithms for 1-median problem in a network.

Influence of a node set is defined as the expected number of active nodes in the end of the information propagation process when the nodes in the set are activated initially. As the models of information propagation process, Independent Cascade (IC) and Linear Threshold (LT) models are popular, and in the both models, neighbor nodes of active nodes are activated if a certain condition is satisfied. The information tends to propagate rapidly and widely through an on-line social network compared with the propagation through an off-line social network. Therefore drawing attention to make use of the word of mouth o the Internet is considered effective as a tool for advertising. The internet marketing using the word of mouth is called as viral marketing from the analogy of virus propagation. Influence can be considered as a quantity that measures the effect of viral marketing, so it catches a great deal of attention recent years. We study the Influence Maximization Problem in IC model, which is the problem to find the k-element node set with the maximum influence assuming that nodes activated at time t can activate their neighbor nodes at time $t + 1$ through edges with probability of given edge weights. This problem is known to be NP-Hard for a general graph, so efficient approximation algorithms for a general graph and an efficient exact algorithm for a tree have been proposed so far. We propose a heuristic efficient exact algorithm for a two-terminal series parallel graph, which can represent any tree using 0-weighted edges.

1.2 Contribution

Our contributions for 1-median problem are the followings[5, 6]:

1. We proposed a novel heuristic approach to approximately solve the problem, in which a node with higher closeness centrality is repeatedly found by approximately calculating closeness centrality of each node using simple spanning subgraphs.

2. According to our experimental results using real networks with more than 10,000 vertices, our algorithms FAOM and FAOM2 of our iterative approach are more than 100 times faster than the exhaustive search and more than 20 times faster than the state-of-the-art approximation algorithm DTZ while our algorithms have higher approximation ratio.

3. FAOM2 was empirically shown to improve the approximation ratio of FAOM with a slight increase of computational time.

Our contribution for influence maximization problem are the followings[7]:

1. We proposed an algorithm that exactly solves the influence maximization problem for TTSPGs (Two-Terminal Series Parallel Graphs), which is the first feasible exact algorithm that solves the problem for the graphs more complex than trees.

2. Empirically fast solving for the influence maximization problem for TTSPGs can be achieved by making use of a certain maximality that must be satisfied by the solution node set. Our algorithm solves the problem in the divide-and-conquer manner keeping the number of candidate node sets small in each recursion by eliminating non-maximal node sets.

3. According to our experimental results using synthetic graphs, our algorithm solves the influence maximization problem for TTSPGs with 800 nodes within one minutes in the case that the number of initial active nodes are at most 7.

Chapter 2

Preliminaries

In this chapter, I introduce some basic facts that we used in the later chapters.

2.1 Graph

Graph is an ordered pair (V, E), where V, E are a set of vertices and a set of edges. We also call a vertex as a node. A *directed edge* from vertex u to vertex v is denoted by (u, v). An *undirected edge* between vertices u and v is also denoted as (u, v), and in this case edge (u, v) is identified as edge (v, u). Graphs with directed and undirected edges are called *directed* and *undirected graphs*, respectively.

In the chapter 3, we mainly discuss the median problem on an undirected graph. In the chapter 4, we discuss the influence maximization problem on a directed graph.

The *degree* of a node is defined as the number of edges linked to the node for undirected graphs. *In-degree* and *out-degree* of a node are the number of edges from and to it, respectively, for directed graphs.

A path from u to v is an edge sequence $(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n)$ with $v_0 = u$ and $v_n = v$. An edge-weighted graph is a graph with weighted edges, and we call the weight of an edge as its *length*. The length of edge (u, v) is denoted by $\mathrm{len}(u, v)$ The *length of a path* is the sum of length of the edges

on the path. The *distance* from vertex u to vertex v, which is denoted as $d(u, v)$, is defined as the minimum length of paths from u to v. The shortest path from u to v is the path whose path length is equal to the distance from u to v. A path from vertex u to vertex v is called as a cycle when $u = v$. A graph is connected if there is a path between any pair of vertices. A tree is a connected graph which has no cycle.

A subgraph of $G = (V, E)$ is a graph (V', E') satisfying two conditions $V' \subseteq V$ and $E' \subseteq E$. A subgraph of a graph G with $V = V'$ is called a *spanning subgraph* of G.

2.2 Shortest Path Tree and Dijkstra's algorithm

For a graph $G = (V, E)$ and an arbitrary node $v \in V$, the shortest path tree rooted v is defined as follows.

Definition 1 (shortest path tree) *For a graph $G = (V, E)$ and a vertex $v \in V$, a shortest path tree $T = (V, E')$ rooted by v is a spanning tree of G in which the unique path from v to u is a shortest path of G from v to u for any $u \in V$.*

Dijkstra's algorithm[8] is originally a method to find the shortest path between given two nodes. But this method can be used to construct a shortest path tree for a given single source.

The pseudo-code is shown in Algorithm 1. In this algorithm, priority queue is used.

A priority queue Q is a structure that has the following four operations. Note that a priority is a non-negative value, and the smaller the value is, the higher the priority is.

1. Q.insert(v,p): Add element v with priority p to Q.

Algorithm 1 Dijkstra's Algorithm

```
1: function DIJKSTRA'S ALGORITHM(G = (V, E), s ∈ V)
2:     // single-source shortest paths from s
3:     Q : priority queue.
4:     parent(s) = None, Q.insert(s,0)
5:     for all v ∈ V \ {s} do
6:         Q.insert(v, ∞)
7:     end for
8:     while Q is not empty do
9:         (v, d(s, v)) = Q.deletemin()
10:        for all u with (v, u) ∈ E do
11:            if d(s, v) + len(v, u) < Q(u) then % Q(u) denotes the priority of u
12:                Q.decrease-key(u, d(s, v) + len(v, u))
13:                parent(u) = v
14:            end if
15:        end for
16:    end while
17:    return d
18: end function
```

2. Q.deletemin(): Delete the element v with the highest priority p and return (v, p)

3. Q.decrease-key(v,p): Change the priority of element v to p

4. Q.find-min(): Return the element v with the highest priority

A priority queue can be implemented via various data structures, and the most popular structures among them are binary heap and Fibonacci heap[9].

A binary tree is a data structure in which each node has two children at most. A binary heap is a kind of binary tree satisfying following properties.

- (minimum heap property) The priority of a parent is higher than those

of its children. Therefore the element with the highest priority is located in the root.

- (shape property) The height of tree is $\lfloor \log_2 n \rfloor$, where n is the number of elements. Children are filled from left to right on last level.

Fibonacci heap is implemented as the set of trees satisfying the minimum-heap property, that is, the priorities of the children are lower than that of parent. Therefore the element with the highest priority is one of the root of trees.

While we have to keep the shape property strictly in binary heaps, the looser structure is permitted by postponing some processes in Fibonacci heaps.

The calculation cost is shown in Table 2.1 for each operation.

Table 2.1: The calculation cost of binary heap and Fibonacci heap

operation	binary heap	Fibonacci heap
insert	$\log n$	$\log n$
delete-min	$\log n$	$\log n$
decrease-key	$\log n$	1
find-min	1	1

In Dijkstra's algorithm, we have to perform $|V|$ insert, $|V|$ delete-min, $|E|$ decrease-key in the worst case. When a binary heap is used for a priority queue, the time calculation cost is $O(m \log n)$ in the worst case. But when Fibonacci heap is used for priority queue, it is known that the calculation cost can be decreased to $O(m + n \log n)$.

Furthermore, it is known that the calculation cost can be upper-bounded in $O(n)$ for undirected graph in which all the edge lengths are positive integer[10].

2.3 Artifitial Network Models

To study how the real networks are generated, various network models have been proposed. Erdös-Rényi Model[11] is a random network model. In this model, for a fixed vertex set, any pair of vertices is connected via an edge with some probability. There are two approaches to realize this. Both approaches fix the vertex set. First one is to link an edge with some probability p for each pair of vertices. The other one is to randomly choose one graph from all the graphs with m edges for a given number m. In this thesis, we use the first one when we generate graphs by Erdös-Rényi Model.

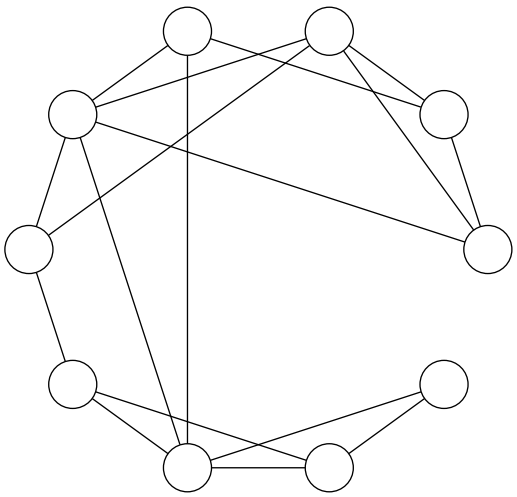

Figure 2.3.1: An example of graph generated by Erdös-Rényi Model.

Barabási-Albert Model is a network model in which generated graphs become scale-free network, that is, degree distribution follows a power law.

This model has three parameters n, m and m_0, where n is the number of nodes, m is the degree of a new added node and m_0 is the number of nodes of initial network. The construction of the network is as follows. At the first, the network starts from an connected network with m_0 nodes. Repeat adding a new node until the number of nodes reaches to n. Each time a new node is added, m edges are linked from the node to existing nodes with a

probability that is proportional to the degree.

In this model, the degree distribution will be $P(k) \sim k^{-3}$.

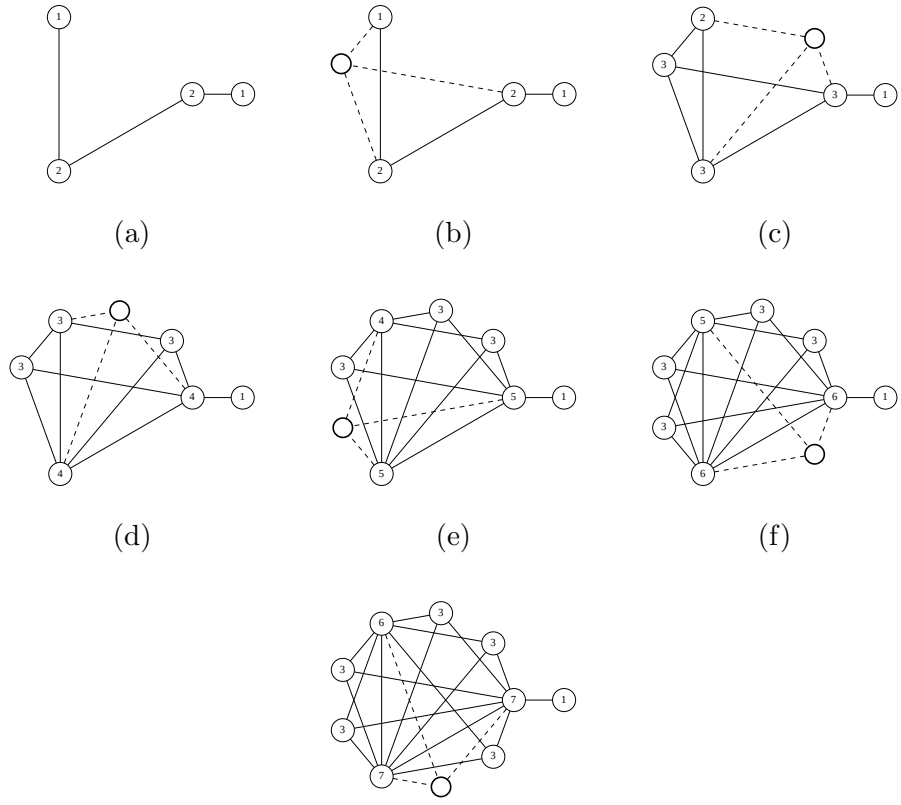

Figure 2.3.2: Example of generation by Barabasi-Albert Model. $n = 10$, $m_0 = 4$ and $m = 3$.

Chapter 3

1-median Problem

3.1 Overview

The graph median problem has been studied more than half a century. Hakimi wrote a paper on the 1-median problem in 1964 [12]. 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 [13, 9] 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 [14]. Rattigan et al. proposed an approximation algorithm for the centrality measure using annotated information to the vertices [15].

The 1-median problem is, in other words, the problem of finding a vertex with the highest *closeness centrality* [1]. In terms of closeness centrality, approximation algorithms for the ranking problems have been also developed [16, 17].

As for more general k-median problem, its NP-hardness was proved [18],

and approximation algorithms have been proposed [19, 20, 21].

3.2 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 a 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 3.2.1 (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[10] 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

Algorithm 2 Iterative algorithm framework for 1-median problem

1: $v_* \leftarrow$ a randomly selected vertex from V;
2: **repeat**
3: $v_0 \leftarrow v_*$
4: Calculate $d_G(v_0)$ constructing the shortest path tree $T(v_0)$ from v_0.
5: Calculate $d_{G_v}(v)$ for each $v \in V$,
where G_v is a subgraph of G that contains $T(v_0)$.
6: Set $v_* = \arg\min_{v \in V} d_{G_v}(v)$.
7: **until** $d_{G_{v_*}}(v_*) \geq d_G(v_0)$
8: **output** v_0;

$u \in V \setminus \{v\}$. 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.

3.3 Iterative Algorithm Framework

To obtain an approximate solution for 1-median problem, we propose an iterative algorithm framework shown in Algorithm 2. 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 $d_G(v_*) \leq d_G(v_0)$, 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 [22]. Furthermore, $d_G(v) \leq d_G(v_0)$ if $d_{G_v}(v) \leq d_{G_{v_0}}(v_0)$ because $d_G(v) \leq d_{G_v}(v)$ 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, $d_G(v_*) \leq d_G(v_0)$ 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 $d_G(v_*) \leq d_G(v_0)$ 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.

3.4 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,

3.4.1 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 3.4.1 *Let G be the leftmost graph of Fig. 3.4.1. 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 3.4.2 *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 *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.

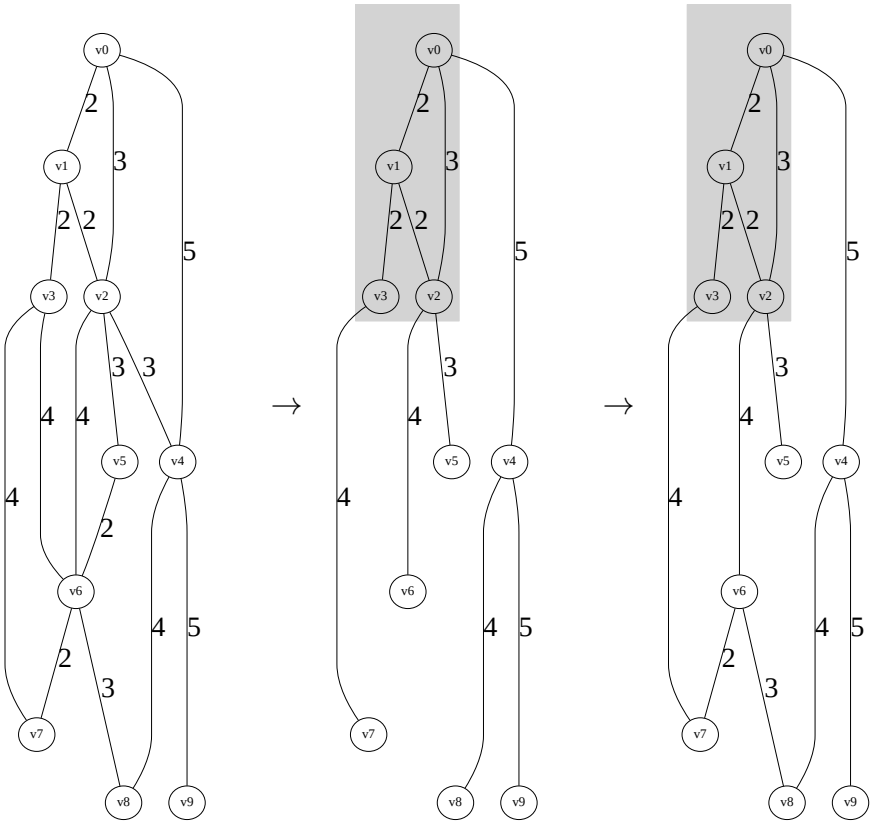

Figure 3.4.1: **Example of a kNSPG and a kNSPG with shortcuts**: The center graph is a kNSPG $T(v_0; 4)$ of the leftmost graph G from v_0. The rightmost graph is a kNSPG $T^{S_{v_6}}(v_0; 4)$ of G from v_0 with shortcuts for v_6.

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.

3.4.2 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 *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 3.4.3 *In the center graph 4NSPG $T(v_0; 4)$ of Fig. 3.4.1,*

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

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 3.4.4 *The rightmost graph in Fig. 3.4.1 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

$$\begin{aligned}
&\alpha_{T(v_0;k),e}(v,u)\\
=&d_{T(v_0:k)}(v,u) - d_{T^{\{e\}}(v_0;k)}(v,u)\\
=&d_{T(v_0:k)}(v,C(v)) + d_{T(v_0;k)}(C(v),C(t)) + d_{T(v_0;k)}(C(t),u)\\
&- \left(d_{T(v_0;k)}(v,s) + \mathrm{len}(s,t) + d_{T(v_0;k)}(u,t)\right). \qquad (3.1)
\end{aligned}$$

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

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

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,

$$\begin{aligned}
\overline{\delta}_{T(v_0;k),(s,t)}(v) =& \alpha_{T(v_0;k),(s,t)}(v, v_{\mathrm{op}}) W_{T(v_0)}(v_{\mathrm{op}})\\
&+ \sum_{i=1}^{i_{\mathrm{op}}} d_{T(v_0)}(p^i(t), p^{i-1}(t)) W_{T(v_0)}(p^{i-1}(t)) \qquad (3.2)
\end{aligned}$$

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

Theorem 3.4.5 *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 *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 $j = 0, \ldots, \log(depth_{ST(C(v))}(v))$ 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 $j = 0, \ldots, \log(depth_{ST(C(v))}(v))$,

where $depth_{ST(C(v))}(v)$ 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 $\sum_{i=1}^{depth_{T(v_0)}(v)} d_{T(v_0)}(p^i(v), p^{i-1}(v))W_{T(v_0)}(p^{i-1}(v))$ 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}, $\sum_{i=1}^{i_{op}} d_{T(v_0)}(p^i(t), p^{i-1}(t))W_{T(v_0)}(p^{i-1}(t))$ 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.

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 $v_{\rm op} \neq C(t)$ 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

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

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

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

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 3.4.6 *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 *By Theorem 3.4.5 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 $E_v^u \cup \{e_{v',u} \mid p(v') = v\}$ are calculated. So, the total number of edges is*

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

Since $\sum_{v \in V \setminus N_k(v_0)} \sum_{u \in N_k(v_0) \setminus \{C(v)\}} |E_v^u|$ 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 $(k-1)(n-k) + 2m = O(kn+m)$.

3.5 Algorithm FAOM

3.5.1 Algorithm

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 $d_G(v) < d_G(v_0)$. 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 $O(m \log n)$ time if $O(k = \sqrt[3]{m})$. Accordingly to our experimental result, however, $d_T(v_0; k)(v)$ is not enough close to $d_G(v)$ for many nodes v even if $k = \sqrt[3]{m}$.

A pseudocode of our algorithm FAOM (Fast Approximation of One Median) is shown in Algorithm.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 $d_{G(v_0;v_0;k)}(v_0)$ for each $v_0 \in V$ 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, $\delta(v) = d_{T(v0;k)}(v) - s$. The values of $\delta(v)$ for all $v \in V$ are calculated by algorithm ImprovementByShortCut. The pseudocode is shown in Algorithm. 4

3.5.2 Experiments and Results

In this section, we show the result of experiments conducted to show the effectiveness of our method.

As real datasets, we used two datasets of Stanford Large Network Dataset Collection[1]. 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.

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

Algorithm 3 FAOM

```
1: function FAOM(G = (V, E), k)
2:     v_0 = randomly selected node in V
3:     while TRUE do
4:         v = EstimatedHigherCentralityNode(G, v_0, k)
5:         if d_G(v) < d_G(v_0) then
6:             v_0 = v
7:         else
8:             return v_0
9:         end if
10:    end while
11: end function
12:
13: function EstimatedHigherCentralityNode(G = (V, E), k, v_0)
14:    Construct kNSPG T(v_0; k) of G from v_0
                   by adding edges between the vertices in N_k(v_0) to T(v_0)
15:    Calculate d_{T(v_0;k)}(v) for all v ∈ V.
16:    Calculate δ(v) for all v ∈ V.
17:    return arg min_{v∈V\{v_0}} d_{T(v_0;k)}(v) − δ(v)
18: end function
```

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 vertex degree[2]. In a graph of Barabási-Albert model, the distribution of vertex

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

Algorithm 4 ImprovementByShortCut

1: **function** ImprovementByShortCut($G = (V, E)$, $T(v_0; k) = (V, E')$, v_0)
2: **for all** $v \in V$ **do**
3: **for all** $u \in$ such that $\{u, v\} \in E$ **do**
4: **if** $C(u, v_0; k) \neq C(v, v_0; k)$ **then**
5: $\delta_u(v) =$ ImprovementByOneShortCut(v, u)
6: **end if**
7: **end for**
8: $\delta(v) = \sum_{c \in N_k(v_0)} max_{u \in ST(c, v_0; k)} \delta_u(v)$
9: **end for**
10: **return** δ
11: **end function**
12:
13: **function** ImprovementByOneShortCut($G = (V, E)$, $T(v_0; k) = (V, E')$, $u, v \in V$)
14: result $= 0$, $sw = 0$, repeat $= 0$
15: $D = d(v, C(v)) + d(C(v, v_0; k), C(u, v_0; k)) + d(C(u, v_0; k)) - d(v, u)$
16: **while** $D > 0$ **do**
17: result $=$ result $+ (SW(u, v_0; k) - sw)D$
18: **if** $u = C(u)$ or repeat $> \log n$ **then**
19: **return** result
20: **end if**
21: repeat $=$ repeat $+ 1$
22: $sw = SW(u, v_0; k)$
23: $D = D - 2d(u,$ parent of $u)$
24: u parent of u
25: **end while**
26: **return** result
27: **end function**

Table 3.1: 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

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 3.1, 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 [3]. In ca-AstroPh and Oregon1(May26), all the vertex weights and the edge lengths were set to 1.0.

First of all, we conducted experiment to confirm the effect of the subgraph used to obtain the approximation of $d(v)$ compared with tree. In our algorithm, for a given node v, we adding the edges from v to kNSPG $T(v_0; k)$ in calculating the $d(v)$. We will call them "shortcut from v" for convenient.

In this experiment, we compared the effect of adding the edges in $N_k(v_0)$ to a shortest path tree rooted by v_0 and that of adding shortcuts to kNSPG of v_0. In comparing, we measured the ratio $\frac{d_F(v)}{d_G(v)}$, where F means a subgraph we focus. We used a shortest path tree of v_0, kNSPG of v_0 and the best kKNSPGS for F.

We calculated the value $d_T(v_0; 1)/d_G(v)$, $d_T(v_0; k)/d_G(v)$ and $(d_T(v0; k)(v) - \delta(v))/d_G(v)$ for randomly selected v_0 as initial node and each $v \in V$. We show the histogram of the result for "ca-AstroPh" in fig. 3.5.1 . As we can see, the approximation ratio become better by using the shortcuts.

Next, we know the calculation cost of FAOM is $O(m\ell \log n)$ time. How-

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

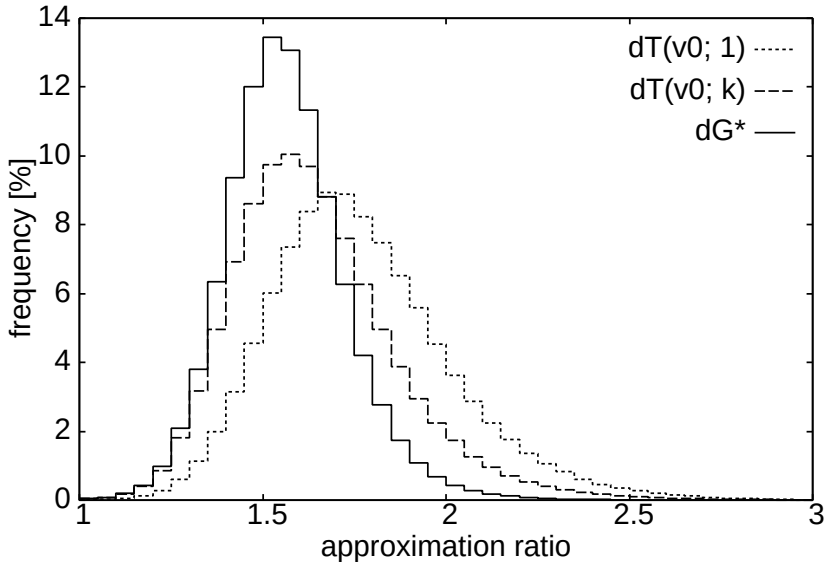

Figure 3.5.1: Distribution of approximation ratio $d_T(v_0;1)/d_G(v)$, $d_T(v_0;k)/d_G(v)$ and $(d_T(v0;k)(v) - \delta(v))/d_G(v)$.

Table 3.2: The number of main-loop iterations

	1	2	3	4	5	6	7
ca-AstroPh	2	15	30	34	15	3	1
Oregon1(May26)		54	46				
BA		48	30	15	4	3	
ER	12	34	31	14	5	2	2

ever, the repetition of main-loop ℓ, or the number of execution the procedure EstimatedHigherCentralityNode, is about proportional to the calculation time. We conducted the experiment for the number of repetition of main-loop, and show the result in Fig. 3.2.

Although the number of nodes are 4,986 and 17,903 respectively, the number of repetition of main-loop are 7 at most, and their modes are from 2 to 4. In this experiments, the numbers of repetition are extremely lower than the number of nodes in any dataset. Therefore our method is expected to run fast.

We compared with other methods. The results are shown in Table 3.3.

Table 3.3: The results shown in this table are the average over 100 times runs. The units of time are milli second.

	BA		ER	
method	time	approx.	time	approx.
Exact	97,919	1.000	100,652	1.000
DTZ(k=2,d=1)	14,261	1.361	14,424	1.141
DTZ(k=5,d=1)	15,284	1.106	15,066	1.085
DTZ(k=2,d=5)	43,212	1.069	45,096	1.071
DTZ(k=5,d=5)	56,369	1.102	54,647	1.084
FAOM(k=2)	305	1.034	263	1.081
FAOM(k=8)	332	1.033	358	1.081
FAOM(k=64)	410	1.031	409	1.080
FAOM(k=512)	3,029	1.002	2,627	1.080

	Oregon1(May26)		ca-AstroPh	
method	time	approx.	time	approx.
Exact	361,943	1.000	2,115,636	1.000
DTZ(k=2,d=1)	75,896	1.193	203,754	1.414
DTZ(k=5,d=1)	75,917	1.159	205,134	1.224
DTZ(k=2,d=5)	308,623	1.215	770,141	1.256
DTZ(k=5,d=5)	316,501	1.222	793,480	1.120
FAOM(k=2)	392	1.000	1,687	1.042
FAOM(k=8)	379	1.000	2,213	1.034
FAOM(k=64)	424	1.000	3,010	1.040
FAOM(k=512)	2,151	1.000	6,800	1.025

3.6 Algorithm FAOM2

3.6.1 Algorithm

We propose algorithm *FAOM2 (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 FAOM2 is shown in Algorithm 5. FAOM2 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 3.4.5 (Line 14), upper bounds $\overline{d}_v$ 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 Theorem3.4.2 and 3.4.6, we obtain the following theorem.

Theorem 3.6.1 *Function HigherCentralityVertex runs in* $O((kn+m)\log n + k^3)$ *time and* $O(m + n\log n)$ *space.*

Trivially, time complexity of FAOM2 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 3.6.2 *Assume that the number of the main-loop iterations is* $O(1)$*, and let us compare the time and space complexities of FAOM2 to those of other algorithms that are used as comparative methods in our experiments. The exhaustive search using Dijkstra's algorithm runs in* $O(n^2 \log n + mn)$

Algorithm 5 FAOM2

function FAOM2($G = (V, E)$, k)
  $v_* \leftarrow$ a randomly selected vertex from V
  repeat
    $v_0 \leftarrow v_*$;
    $(v_*, d_*) \leftarrow$ HigherCentralityVertex (G, k, v_0)
  until $d_* \geq d_G(v_0)$
  output v_0;
end function

function HigherCentralityVertex($G = (V, E)$, k, v_0)
  Calculate $d_G(v_0)$ constructing the shortest path tree $T(v_0)$ from v_0.
  Construct kNSPG $T(v_0; k)$ of G from v_0
    by adding edges between the vertices in $N_k(v_0)$ to $T(v_0)$
  Calculate $d_{T(v_0;k)}(v)$ for all $v \in V$.
  Do preparation for fast calculation of $\overline{\delta}_{T(v_0;k),(s,t)}(v)$
    $(v \in V \setminus N_k(v_0),\ s \in D_{T(v_0)}(v), t \notin V_{ST(C(v))})$.
  $v_* \leftarrow v_0$, $d_* \leftarrow d_G(v_0)$
  for all $v \in V \setminus N_k(v_0)$ **do**
    $\overline{d}_v \leftarrow d_{T(v_0;k)}(v) - \sum_{e \in S_v} \overline{\delta}_{T(v_0;k),e}(v)$
    if $\overline{d}_v < d_*$ **then**
      $v_* \leftarrow v$, $d_* \leftarrow \overline{d}_v$
    end if
  end for
  return (v_*, d_*);
end function

time and $O(m)$ space. So, FAOM2 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[4] *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. FAOM2 asymptotically runs faster than this method when $m = o(n^2/(\log n))$ but consumes more memory when $m = o(n \log n)$.*

Remark 3.6.3 *Unfortunately, for $n > 2k$, FAOM2's approximation ratio is not good in the worst case even when $w(v) = 1$ for all $v \in V$ because we have*

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

for $v_* = \arg\max_{v \in V} d_G(v)$ and FAOM2'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 $V = \{v_1, ..., v_n\}$, $E = E_1 \cup E_2$, where $E_i = \{(v_i, v_j) | j > i\}$, and $\mathrm{len}(u, v) = a$ for $(u, v) \in E_1$ and $\mathrm{len}(u, v) = \delta$ for $(u, v) \in E_2$. Assume that $a \gg \delta$. Then, starting from v_1, FAOM2 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 $\sup_{len} \frac{d_G(\hat{v})}{d_G(v_*)} = n - 1$ holds for this G. Approximation ratio of FAOM2's outputs is close to 1 in our experiments, so what condition makes FAOM2 output a vertex with good approximation ratio is an interesting issue to pursue.*

3.6.2 Experiments

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

3.6.3 Experimental Setting

We used four datasets shown in Table 3.1.

[4]See Sec. 3.6.7 for detailed description.

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.

3.6.4 Effect of Using kNSPG with Shortcuts

In algorithm FAOM2, 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. 3.6.1. As we can see on this graph, the approximation ratio is improved by using $T(v_0;k)$ and furthermore by using $T^{S_v}(v_0;k)$.

3.6.5 Number of Main-Loop Iterations

Efficiency of algorithm FAOM2 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 3.4. 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.

3.6.6 Effect of Using Larger k

FAOM2 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

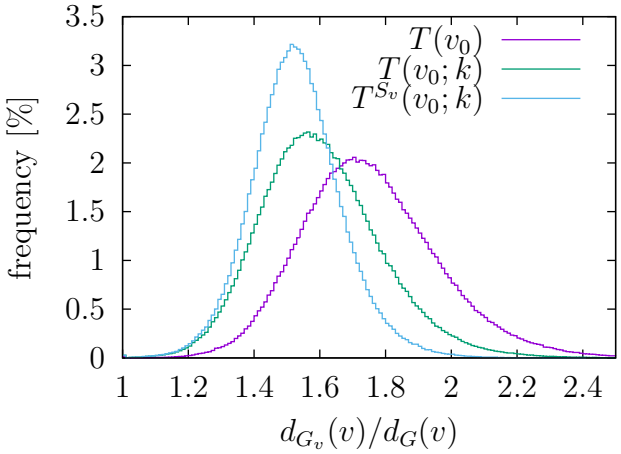

Figure 3.6.1: Distribution of approximation ratio $d_{G_v}(v)/d_G(v)$ of $v \in V$ in ca-AstroPh for $G_v = T(v_0)$, $T(v_0;k)$, $T^{S_v}(v_0;k)$. For $T^{S_v}(v_0;k)$, the distribution is not for $d_{T^{S_v}(v_0;k)}(v)$ but for their upper bounds $d_{T(v_0;k)}(v) - \sum_{e \in S_v} \overline{\delta}_{T(v_0;k),e}(v)$. 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 3.4: 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

FAOM2's output $\hat{v}$ for the optimal vertex $v_* = \arg\min_{v \in V} d_G(v)$ while it increases FAOM2'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 FAOM2 1000 times by giving randomly selected initial vertices and calculated average approximation ratio and wall clock time. The result is shown in Fig.3.6.2. (See also Table3.5 for detailed data.) For Oregon1(May26), given any initial ver-

tex, FAOM2 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.

3.6.7 Comparison to Other Methods

We compared FAOM2'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 approximation 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 [15]. 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[5]. 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 3.5 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

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

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

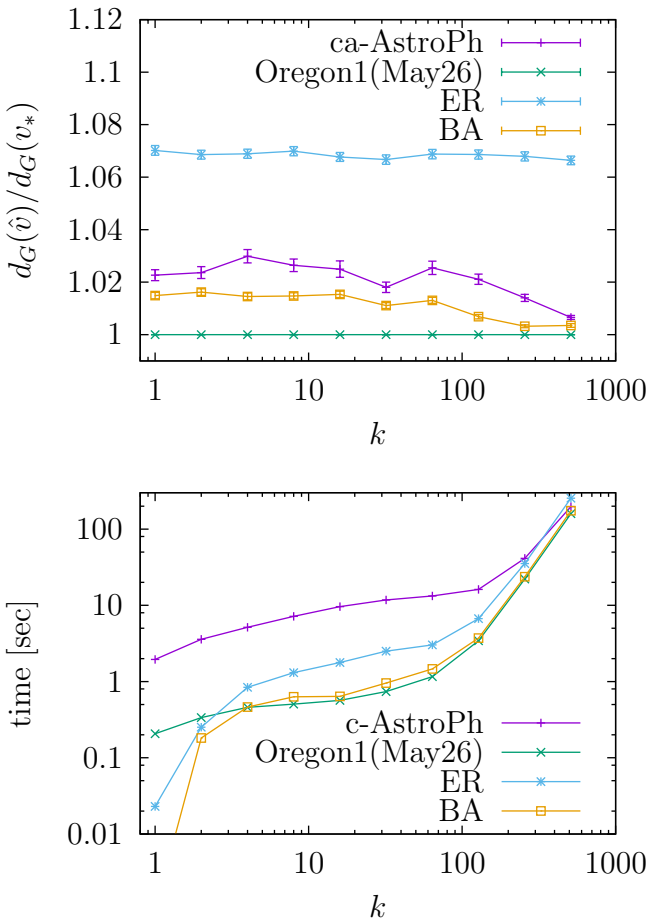

Figure 3.6.2: **Upper graph:** Relation between parameter k and approximation ratio $d_G(\hat{v})/d_G(v_*)$, where v_* is the optimal vertex and $\hat{v}$ is the output of FAOM. The error bars show 95% confidence interval. **Lower graph:** Relation between parameter k and running time of FAOM2.

ratio of FAOM2 for any k is better than that of DTZ of any parameter settings used in our experiment. FAOM2's approximation ratio is not good for the random graph (ER) compared to DTZ. As for running time, FAOM2 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, FAOM2 with $k \leq 128$ is at least 21 times faster than DTZ and at least 143 times faster than Exact.

Table 3.5: Approximation ratio $d_G(\hat{v})/d_G(v_*)$ and running time [sec] of three methods, FAOM2, 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 FAOM2, 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)	
		app. ratio	time	app. ratio	time
FAOM2	$k = 1$	1.023(±0.002)	1.951	1.000(±0.000)	0.207
	$k = 2$	1.024(±0.002)	3.588	1.000(±0.000)	0.337
	$k = 4$	1.030(±0.003)	5.143	1.000(±0.000)	0.459
	$k = 8$	1.026(±0.002)	7.154	1.000(±0.000)	0.506
	$k = 16$	1.025(±0.003)	9.602	1.000(±0.000)	0.567
	$k = 32$	1.018(±0.002)	11.764	1.000(±0.000)	0.738
	$k = 64$	1.025(±0.003)	13.290	1.000(±0.000)	1.160
	$k = 128$	1.021(±0.002)	16.172	1.000(±0.000)	3.434
	$k = 256$	1.014(±0.001)	41.303	1.000(±0.000)	22.111
	$k = 512$	1.007(±0.001)	197.273	1.000(±0.000)	159.656
DTZ	$k = 2, d = 10$	1.167(±0.001)	1677.122	1.232(±0.001)	713.922
	$k = 5, d = 4$	1.111(±0.001)	883.119	1.181(±0.001)	349.941
	$k = 10, d = 2$	1.084(±0.000)	525.265	1.145(±0.000)	198.245
	$k = 20, d = 1$	1.064(±0.001)	344.416	1.111(±0.003)	123.802
Exact	-	1.000(±0.000)	3037.151	1.000(±0.000)	493.967

method	parameters	ER		BA	
		app. ratio	time	app. ratio	time
FAOM2	$k = 1$	1.070(±0.002)	0.023	1.015(±0.001)	0.001
	$k = 2$	1.069(±0.002)	0.252	1.016(±0.001)	0.182
	$k = 4$	1.069(±0.002)	0.841	1.015(±0.001)	0.463
	$k = 8$	1.070(±0.002)	1.309	1.015(±0.001)	0.631
	$k = 16$	1.068(±0.002)	1.776	1.015(±0.001)	0.640
	$k = 32$	1.067(±0.002)	2.508	1.011(±0.001)	0.957
	$k = 64$	1.069(±0.002)	3.028	1.013(±0.001)	1.467
	$k = 128$	1.069(±0.002)	6.688	1.007(±0.001)	3.716
	$k = 256$	1.068(±0.002)	35.363	1.003(±0.001)	23.716
	$k = 512$	1.066(±0.002)	254.720	1.004(±0.001)	174.233
DTZ	$k = 2, d = 10$	1.083(±0.000)	110.153	1.134(±0.001)	121.654
	$k = 5, d = 4$	1.064(±0.001)	56.540	1.137(±0.001)	61.958
	$k = 10, d = 2$	1.055(±0.000)	34.714	1.085(±0.002)	37.046
	$k = 20, d = 1$	1.045(±0.000)	21.091	1.050(±0.000)	22.039
Exact	-	1.000(±0.000)	103.010	1.000(±0.000)	100.203

Chapter 4

Influence Maximization

4.1 Overview

The information or influence propagation on social networks has gathered attention in this decade. For example, let us consider a diffusion of reputations of a new product. Assume a few people try it, and tell to their friends how useful it is. Some of their friends may have interest in it and try it and recommend it to other people. In this way, the reputations will spread gradually. We can know the influence diffusion dynamics underlying the social network from the phenomenon like this. Therefore, the dynamics has been studied to understand the influence diffusion process in the social networks.

The study on the influence diffusion dynamics is useful for a marketing using a word-of-mouth communication [23, 24], a spreading of epidemics [25], a diffusion of innovations [26] and so on. In this study area, how to make the influence spread widely is an important theme on the viral marketing [24].

To analyze the influence diffusion dynamics, some diffusion models are proposed, and Independent Cascade (IC) model and Linear Threshold (LT) model are two of the most popular models. In both the models, each node takes either active state or inactive state. A node with active state corresponds to an affected individual. The diffusion process progresses step by step, that is, the nodes activated at time round t try to activate inactive

neighbors on a network at time round $t+1$. Once the state of a node becomes active, it stays active later also. So the spread of the influence monotonically increases. The basic idea of these models is the same as that of a Susceptible-Infected-Recovered (SIR) epidemic model [27], in which the *susceptible* (the people not yet infected with the disease) don't spread the disease, but the *infected* can spread it and the *recovered* don't spread it any more.

In this paper, we study the influence maximization problem on IC model formulated by Kempe et al. [28], whose objective is to find the most influential k nodes from a given network for a given number $k \in \mathbb{N}$. This problem is one of the most important problems in the area of network analysis because of its rich applications. A problem of finding the best testers for new products in marketing using word-of-mouth communication is an instance of this problem.

Unfortunately, it seems to be hard to obtain the exact solution of this problem feasibly because the problem is known to be NP-hard on both IC model and LT model.

Kempe et al. [28] has proposed a greedy approximation algorithm for this problem. Their algorithm finds a set of k initially activated nodes by repeatedly adding the node that increases the effect most. The approximation ratio of the solution is guaranteed by the analysis using a sub-modular function. Their algorithm, however, must calculate the effect for a given initial active nodes many times, and they estimated it by a large number of simulations in their experiments because its fast exact calculation has not been known yet. On the other hand, Lappas et al. proposed an exact algorithm for a tree that runs in polynomial time [29], and suggested a method for a general directed graph by applying their algorithm to its spanning tree with edges important for propagation.

We propose an exact method for a *two-terminal series parallel graph* (TTSPG). Since any weighted directed tree can be converted to a weighted TTSPG with the same propagation structure by adding a new node v and 0-weighted edges from all the leaves to v, the class of TTSPGs can be con-

sidered to be larger than the class of trees, for which exact algorithms has been already developed so far. In our algorithm, the effect on a TTSPG G for a set of initial active nodes is represented as a linear function of the activation probability of its source node. This enables the algorithm to calculate the effect on each subTTSPG without knowing the activation probability of its source node that depends on what upstream nodes are selected as initial active nodes. Thus, effect calculation in the divide-and-conquer manner becomes possible. The number of candidate sets of initial active nodes are exponential to k but we have to check only the sets whose local effect functions are maximal, and the number of such *maximal* sets is not large according to the result of our experiments using randomly generated TTSPGs. Our algorithm calculates candidate node sets in the divide-and-conquer manner efficiently by eliminating non-maximal ones in merge of subproblems' solutions. We also propose a way of obtaining a TTSPG with edges important for propagation from a general directed graph to apply our algorithm to a real network.

According to the result using synthetically generated random TTSPGs, our exact algorithm is significantly faster than the greedy approximation algorithm developed by Leskovec et al. [30] though its average approximation ratio does not reach 0.9. In the experiments using networks constructed from DBLP records, more influential authors are extracted by our method compared with the method using a spanning tree developed by Lappas et al. [29] in terms of the h-index sum.

4.1.1 Related Works

The influence maximization problem is proposed by Domingos and Richardson as an algorithmic problem [31, 23]. They modeled the influence propagation between nodes in a social network as a Markov random field. Later, Kempe et al. formalized it as a more operational model based on mathematical sociology [28]. They introduced IC and LT models to the influence maximization problem and proposed a greedy algorithm with a guaranteed

approximation ratio based on mathematical theory of submodular functions. More efficient implementation of their method by making use of the submodularity of the objective function has been also developed [30, 32].

IC model belongs to the type of SIR model in the sense that the affected are never affected again. As an extension of SIR model, there is another type of model called the Susceptible-Infective-Susceptible (SIS) model, in which individuals can be affected many times. Saito et al. proposed several methods to solve the influence maximization problem in the SIS model [33, 34].

On IC model, there are recent studies that attempt to develop an exact algorithm for simple structures. Lappas et al. formulated an extended influence maximization problem as the k-effector problem and proposed an exact algorithm for a tree using a dynamic programming [29]. Bharathi et al. proposed an approximation algorithm with arbitrary precision for bidirected trees using a dynamic programming and rounding probabilities [35].

Our study is the first attempt to develop an exact algorithm on influence maximization problem for more complex structure than the tree structure.

4.2 Preliminary

In this section, we introduce some definitions and notations used through the paper.

4.2.1 Influence Maximization in Independent Cascade Model

Independent Cascade (IC) Model is one of the most popular information propagation models.

An edge-weighted graph $G = (V, E, p)$ is used in this model, where V and E are the set of nodes and directed edges, respectively, and p is a weight function of edges that satisfies $p(e) \in [0, 1]$ for all edges e. Nodes correspond to individuals, who propagate information, and the edges correspond to re-

lationship between individuals. Information propagates from a node to its neighbor nodes via edges only. The weight $p(e)$ of an edge e is the success probability of propagation via the edge e. We assume that the success of the propagation via one edge is independent from that via any other edge. Active state propagates via edges step by step, that is, if a node v turns into active state at time round t, a neighbor inactive node u of v turns into active state at time round $t+1$ via an edge $v \to u$ by probability $p(v \to u)$. If the node v fails to activate u at time round $t+1$, v doesn't try to activate u anymore. Once a node becomes active state, it never turns into inactive in future. Therefore, the propagation process progresses monotonically. The propagation process starts from given initial active nodes at time round 0 and ends by time round n, where n is the number of nodes.

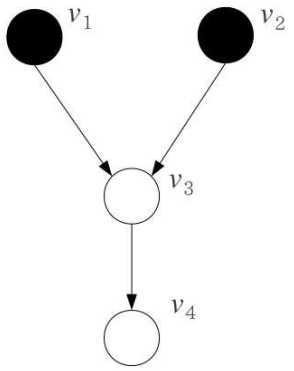

Figure 4.2.1: An example of an influence propagation network.

For an initial active set of nodes $S \subset V$, its *effect*, which is denoted by $\sigma(S)$, is defined as the expected number of active nodes at the end of the state propagation process.

An example of an influence propagation network is illustrated in Figure 4.2.1. Assume that the states of black and white nodes in the figure are active and inactive, respectively, at time round t. At time round $t+1$, v_3 can be activated by v_1 or v_2 and becomes active by probability $1-(1-p(v_1 \to v_3)) \times (1-p(v_2 \to v_3))$ because the success of each propaga-

tion is independent. At time round $t+2$, node v_4 is activated by probability $p(v_3 \to v_4) \times (1 - (1 - p(v_1 \to v_3)) \times (1 - p(v_2 \to v_3)))$.

An influence maximization problem is formalized as follows.

Problem 4.2.1 (Influence Maximization Problem [28]) *Given a weighted directed graph $G = (V, E, p)$ and $k \in \mathbb{N}$, find a k-element subset S of V that maximizes $\sigma(S)$.*

This problem is known to be NP-Hard [28]. So it seems difficult to solve the problem in feasible time.

4.2.2 Two-Terminal Series Parallel Graph

Lappas et al. [29] developed a polynomial-time algorithm which can find an optimal solution of a given influence maximization problem in the case that the graph G is a tree. In this paper, we develop an algorithm that solves the problem for a graph called a *two-terminal series parallel graph*, TTSPG for short. Note that a TTSPG has a more complex structure than a tree and a simpler structure than a directed acyclic graph.

A TTSPG is defined as follows.

Definition 4.2.2 (TTSPG [36]) *A two-terminal series parallel graph (TTSPG) is defined recursively as follows.*

1. *A directed graph G consisting of two vertices v_1 and v_2 joined by an edge $v_1 \to v_2$ is a TTSPG. The node v_1 is called the source of G, and the node v_2 is called the sink of G.*

2. *If G_1 and G_2 are TTSPG S, so the directed graph obtained by either of the following operations.*

   - *(Parallel Composition): identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.*
   - *(Series Composition): identify the sink of G_1 with the source of G_2.*

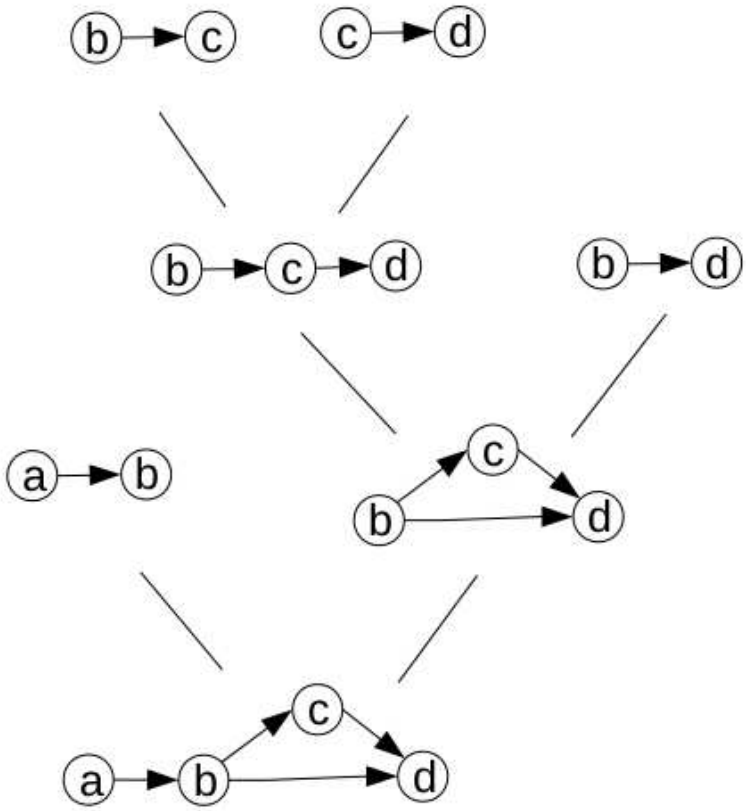

Figure 4.2.2: An example of an SP decomposition tree of a TTSPG.

By Definition 4.2.2, any TTSPG with more than two nodes can be decomposed in parallel or series, and we call such decomposition as *SP decomposition*. By applying SP decomposition repeatedly, TTSPG can be decomposed into its component non-decomposable subgraphs with one edge in polynomial time [36]. The decomposition process can be represented as a tree, which is called as an *SP decomposition tree*. (See Figure 4.2.2.)

4.3 Algorithm for a TTSPG

Let $G = (V, E, p)$ be a TTSPG. As the number k of initially activated nodes, we assume $k < |V|$ because $S = V$ is the solution of the information maximization problem otherwise, where $|V|$ denotes the number of elements in V. Since any node subset S which contains the sink node cannot be a unique solution when $k < |V|$, we only have to search the solution S among k-element subsets of V that exclude the sink node.

4.3.1 Local Effect Function of Source Activation Probability

By SP decomposition, a TTSPG $G = (V, E, p)$ can be decomposed into $G_1 = (V_1, E_1, p)$ and $G_2 = (V_2, E_2, p)$ in parallel or series. Let us try to solve the influence maximization problem for G and k in the divide-and-conquer manner using this fact. Consider the case that the decomposition is series. In this case, the activation probability of the source s_1 of G_1 is 0 or 1 depending on whether s_1 is chosen as an initial active node or not. The activation probability of the source s_2 of G_2, however, can take a variety of values depending of what subset of V_1 is selected as the set of initial active nodes in G_1 when s_2 is not chosen as an initial active node: the number of values taken by it can be $\Omega(((|V_1| - 1)/k)^k)$. This prevents the problem from being efficiently solved by a divide-and conquer approach. Note that the number of possible values for the activation probability of any node is at most its depth plus two in the case with a tree, which enables an efficient algorithm of the influence maximization problem for a tree.

To overcome the above problem, we propose a method in which the effect of the set $S_2 \subseteq V_2$ of initial active nodes in G_2 is calculated as a function of the activation probability p_0 of s_2 without knowing the specific values of p_0. We introduce two functions σ_G^S and q_G^S of p_0 defined for each node subset S of G.

Definition 4.3.1 *Let $G = (V, E, p)$ be a TTSPG with a sink node t. For*

a node subset $S \subseteq V \setminus \{t\}$, $\sigma_G^S(p_0)$ and $q_G^S(p_0)$ are defined as the expected number of active non-sink nodes and the probability to have an active sink node, respectively, at the end of the state propagation process in which all the nodes in S are initially activated and the source node of G is also activated (initially) with probability of p_0.

The effect $\sigma(S)$ of S on G can be expressed as $\sigma(S) = \sigma_G^S(0) + q_G^S(0)$ using notations introduced above. It is trivial that σ_G^S and q_G^S are monotonically increasing functions. Note that $\sigma_G^S(p_0)$ and $q_G^S(p_0)$ are constantly equal to $\sigma_G^S(1)$ and $q_G^S(1)$, respectively, when the source node of G is included in S. By the following theorem, we know that σ_G^S and q_G^S are representable by the form of $ap_0 + b$ using some constant reals a and b.

Theorem 1 *For a TTSPG $G = (V, E, p)$ and $k < |V|$, let S be a k-element subset of V that excludes the sink node of G. Then, functions σ_G^S and q_G^S are linear.*

Proof *We prove the theorem by mathematical induction on the number m of edges in E. We only have to prove the case that the source node is not contained in S because the functions are constant otherwise.*

For a non-decomposable TTSPG $G = (\{v_1, v_2\}, \{v_1 \to v_2\}, p)$ and $p_0 \in [0, 1]$,

$$\sigma_G^{\emptyset}(p_0) = p_0 \text{ and } q_G^{\emptyset}(p_0) = p(v_1 \to v_2)p_0$$

hold. Thus, the theorem holds for a non-decomposable TTSPG.

Assume that the theorem holds for any TTSPG $G' = (V', E', p')$ with $|E'| \leq m$. Let $G = (V, E, p)$ be an arbitrary decomposable TTSPG with $|E| = m+1$, and let S be an arbitrary subset S of $V \setminus \{s, t\}$, where s and t are respectively the source and sink nodes of G. Then, G can be decomposed into two TTSP subgraphs $G_1 = (V_1, E_1, p)$ and $G_2 = (V_2, E_2, p)$ with $|E_1|, |E_2| \leq m$ by SP decomposition of G. Let $S_1 = S \cap V_1 \setminus \{t_1\}$ and $S_2 = S \cap V_2 \setminus \{t_2\}$, where t_1 and t_2 are the sink nodes of G_1 and G_2, respectively. Note that the sink node t_1 of G_1 and the source s_2 of G_2 can be included in $S \cap V_1$

and $S \cap V_2$, respectively, in the case that the decomposition type is series and $t_1 = s_2$. By the above assumption, all the four functions $\sigma_{G_1}^{S_1}$, $q_{G_1}^{S_1}$, $\sigma_{G_2}^{S_2}$ and $q_{G_2}^{S_2}$ are linear.

In the case that the decomposition type is parallel, both $\sigma_{G_1}^{S_1}(p_0)$ and $\sigma_{G_2}^{S_2}(p_0)$ contain the effect for the common source node, which is p_0, so

$$\sigma_G^S(p_0) = \sigma_{G_1}^{S_1}(p_0) + \sigma_{G_2}^{S_2}(p_0) - p_0 \tag{4.1}$$

holds. Thus, σ_G^S is linear from the linearity of $\sigma_{G_1}^{S_1}$ and $\sigma_{G_2}^{S_2}$. Since

$$\begin{aligned} q_G^S(p_0) = \{1 - (1 - q_{G_1}^{S_1}(1))(1 - q_{G_2}^{S_2}(1))\}p_0 + \\ \{1 - (1 - q_{G_1}^{S_1}(0))(1 - q_{G_2}^{S_2}(0))\}(1 - p_0) \end{aligned} \tag{4.2}$$

holds, q_G^S is also a linear function.

Consider the case that the decomposition type is series. Without loss of generality, we can assume that G_1 precedes G_2, that is, the sink node of G_1 is identical with the source node of G_2. In this case,

$$\sigma_G^S(p_0) = \sigma_{G_1}^{S_1}(p_0) + \sigma_{G_2}^{S_2}(q_{G_1}^{S_1}(p_0)) \tag{4.3}$$

holds, so σ_G^S is linear from the linearities of $\sigma_{G_1}^{S_1}$ and $\sigma_{G_2}^{S_2}$. Since

$$q_G^S(p_0) = q_{G_2}^{S_2}(q_{G_1}^{S_1}(p_0)) \tag{4.4}$$

holds, q_G^S is also a linear function.

Therefore, the theorem also holds for $G = (V, E, p)$ with $|E| = m + 1$. □

Corollary 1 *Assume that a TTSPG $G = (V, E, p)$ can be decomposed into two TTSPGs $G_1 = (V_1, E_1, p)$ and $G_2 = (V_2, E_2, p)$ with by an SP decomposition of G. Let $S_1 = S \cap V_1$ and $S_2 = S \cap V_2$ for $S \subseteq V$. If all the parameters of linear functions of $\sigma_{G_1}^{S_1}(x) = a_1x + b_1$, $q_{G_1}^{S_1}(x) = c_1x + d_1$, $\sigma_{G_2}^{S_2}(x) = a_2x + b_2$ and $q_{G_2}^{S_2}(x) = c_2x + d_2$ are known, then parameters a, b, c and d of linear functions of $\sigma_G^S(x) = ax + b$ and $q_G^S(x) = cx + d$ can be calculated in $O(1)$ time.*

Proof *When S does not contain the source node of G, the parameters a, b, c and d can be calculated in $O(1)$ time from the parameters a_i, b_i, c_i and d_i for $i = 1, 2$ using Eq. (4.1), (4.2), (4.3) and (4.4) in Theorem 1. When the case that S contains the source node of G, they can be also calculated in $O(1)$ time similarly using the same equations by setting p_0 to 1 additionally.* □

For a TTSPG $G = (V, E, p)$ and two subsets $S_1, S_2 \subseteq V$, let us write

$$(S_1, \sigma_G^{S_1}, q_G^{S_1}) \leq (S_2, \sigma_G^{S_2}, q_G^{S_2})$$

if $\sigma_G^{S_1}(p_0) \leq \sigma_G^{S_2}(p_0)$ and $q_G^{S_1}(p_0) \leq q_G^{S_2}(p_0)$ hold for all $p_0 \in [0, 1]$. We say that a triplet (S, σ_G^S, q_G^S) is *maximal* in $\mathcal{S}$ if $(S, \sigma_G^S.q_G^S) \in \mathcal{S}$ and $(S, \sigma_G^S, q_G^S) \not\leq (S', \sigma_G^{S'}, q_G^{S'})$ for any $S' \in \mathcal{S} \setminus \{S\}$.

4.4 Algorithm

For a given TTSPG $G = (V, E, p)$ and a natural number $k < |V|$, let

$$\mathcal{S}_{\mathrm{all}}(G, k) = \{(S, \sigma_G^S, q_G^S) | S \subseteq V \setminus \{t\}, |S| = k\},$$

where t is the sink node of G. Then, the set S^* of the most influential k nodes is expressed as

$$S^* = \underset{S:(S,\sigma_G^S,q_G^S)\in\mathcal{S}_{\mathrm{all}}(G,k)}{\arg\max} (\sigma_G^S(0) + q_G^S(0)).$$

Unfortunately, $\mathcal{S}_{\mathrm{all}}(G, k)$ cannot be enumerated efficiently because $\mathcal{S}_{\mathrm{all}}(G, k)$ contains $\Omega(((n-1)/k)^k)$ subsets. Considering the fact that S for non-maximal triplets (S, σ_G^S, q_G^S) cannot be S^*, the algorithm only have to check the effect of S for (S, σ_G^S, q_G^S) in

$$\mathcal{S}(G, k) = \{(S, \sigma_G^S, q_G^S) \in \mathcal{S}_{\mathrm{all}}(G, k) \mid (S, \sigma_G^S, q_G^S) \text{ is maximal in } \mathcal{S}_{\mathrm{all}}(G, k)\}.$$

There has been yet no theoretical analysis on how much reduced the set $\mathcal{S}(G, k)$ is from $\mathcal{S}_{\mathrm{all}}(G, k)$, but the effect seems significantly large according

to the following our empirical result. Using the procedure GenTTSPG in Sec. 4.6.2, we randomly generated 100 TTSPGs (V, E, p) with $|V| = 800$ and $p \equiv 0.1$ and calculate $|\mathcal{S}(G, k)|$. Their largest size is 104 and their average size is 43.2, which is overwhelmingly smaller than $|\mathcal{S}_{\mathrm{all}}(G, 5)| = (799 \cdot 798 \cdot 797 \cdot 796 \cdot 795)/(5 \cdot 4 \cdot 3 \cdot 2 \cdot 1) \approx 2.7$ trillion.

So, in the rest of this subsection, we explain how efficiently our algorithm calculates $\mathcal{S}(G, k)$.

Let $\mathcal{S}(G, k, 1)$ denote the set of all the elements in $\mathcal{S}(G, k)$ that contains the source of G and let $\mathcal{S}(G, k, 0)$ denote the set of all the other elements in $\mathcal{S}(G, k)$. Consider the problem of calculating the the list $((\mathcal{S}(G, 0, 0), \mathcal{S}(G, 0, 1)), \cdots, (\mathcal{S}(G, k, 0), \mathcal{S}(G, k, 1)))$ instead of $\mathcal{S}(G, k)$ only. Since $\mathcal{S}(G, k) = \mathcal{S}(G, k, 0) \cup \mathcal{S}(G, k, 1)$, this is a finer problem. This finer problem can be solved recursively by making use of SP decomposition of G.

Let $G_1 = (V_1, E_1, p)$ and $G_2 = (V_2, E_2, p)$ be TTSPGs that is made by some SP decomposition of G. Define k_1 and k_2 as $\min\{|V_1| - 1, k\}$ and $\min\{|V_2| - 1, k\}$, respectively. Assume that $((\mathcal{S}(G_1, 0, 0), \mathcal{S}(G_1, 0, 1)), \cdots, (\mathcal{S}(G_1, k_1, 0), \mathcal{S}(G_1, k_1, 1)))$ and $((\mathcal{S}(G_2, 0, 0), \mathcal{S}(G_2, 0, 1)), \cdots, (\mathcal{S}(G_2, k_2, 0), \mathcal{S}(G_2, k_2, 1)))$ have been already obtained. We show that $((\mathcal{S}(G, 0, 0), \mathcal{S}(G, 0, 1)), \cdots, (\mathcal{S}(G, k, 0), \mathcal{S}(G, k, 1)))$ can be obtained from those.

Let us consider the case with series decomposition. We can assume that G_1 is positioned upstream of G_2 without loss of generality in this case. Then, for each $l \in \{0, 1, \ldots, k\}$ and $x \in \{0, 1\}$, $\mathcal{S}(G, l, x)$ can be calculated from $\mathcal{S}(G_1, k', x)$ and $\mathcal{S}(G_2, l - k', 0) \cup \mathcal{S}(G_2, l - k', 1)$ for $\max\{0, l - k_2\} \leq k' \leq \min\{l, k_1\}$ as

$$
\begin{aligned}
\mathcal{S}(G, l, x) = \{&(S, \sigma_G^S, q_G^S) \in \mathcal{S}'(G, l, x) \mid \\
&(S, \sigma_G^S, q_G^S) \text{ is maximal in } \mathcal{S}'(G, l, x)\}, \text{ where} \\
\mathcal{S}'(G, l, x) = &\bigcup_{k' = \max\{0, l - k_2\}}^{\min\{l, k_1\}} \{(S_1 \cup S_2, \sigma_G^{S_1 \cup S_2}, q_G^{S_1 \cup S_2}) \mid \\
&(S_1, \sigma_{G_1}^{S_1}, q_{G_1}^{S_1}) \in \mathcal{S}(G_1, k', x), \\
&(S_2, \sigma_{G_2}^{S_2}, q_{G_2}^{S_2}) \in \mathcal{S}(G_2, l - k', 0) \cup \mathcal{S}(G_2, l - k', 1)\}, \quad (4.5)
\end{aligned}
$$

where $\sigma_G^{S_1 \cup S_2}$ is calculated from $\sigma_{G_1}^{S_1}$, $\sigma_{G_2}^{S_2}$ and $q_{G_1}^{S_1}$ using Eq. (4.3), and $q_G^{S_1 \cup S_2}$ is also calculated from $q_{G_1}^{S_1}$ and $q_{G_2}^{S_2}$ using Eq. (4.4).

Consider the case that G can be made from G_1 and G_2 by parallel composition. In this case, for each $l \in \{0, 1, \ldots, k\}$, $\mathcal{S}(G, l, 1)$ can be calculated from $\mathcal{S}(G_1, k', 1)$ and $\mathcal{S}(G_2, l - k' + 1, 1)$ for $\max\{1, l - k_2 + 1\} \leq k' \leq \min\{l, k_1\}$ as

$$\begin{aligned}
\mathcal{S}(G, l, 1) = \{(S, \sigma_G^S, q_G^S) \in \mathcal{S}'(G, l, 1) \mid \\
(S, \sigma_G^S, q_G^S) \text{ is maximal in } \mathcal{S}'(G, l, 1)\}, \text{ where} \\
\mathcal{S}'(G, l, 1) = \bigcup_{k' = \max\{1, l-k_2+1\}}^{\min\{l, k_1\}} \{(S_1 \cup S_2, \sigma_G^{S_1 \cup S_2}, q_G^{S_1 \cup S_2}) \mid \\
(S_1, \sigma_{G_1}^{S_1}, q_{G_1}^{S_1}) \in \mathcal{S}(G_1, k', 1), \\
(S_2, \sigma_{G_2}^{S_2}, q_{G_2}^{S_2}) \in \mathcal{S}(G_2, l - k' + 1, 1)\}, \quad (4.6)
\end{aligned}$$

and $\mathcal{S}(G, l, 0)$ can be calculated from $\mathcal{S}(G_1, k', 0)$ and $\mathcal{S}(G_1, l - k', 0)$ for $\max\{0, l - k_2\} \leq k' \leq \min\{l, k_1\}$ as

$$\begin{aligned}
\mathcal{S}(G, l, 0) = \{(S, \sigma_G^S, q_G^S) \in \mathcal{S}'(G, l, 0) \mid \\
(S, \sigma_G^S, q_G^S) \text{ is maximal in } \mathcal{S}'(G, l, 0)\}, \text{ where} \\
\mathcal{S}'(G, l, 0) = \bigcup_{k' = \max\{0, l-k_2\}}^{\min\{l, k_1\}} \{(S_1 \cup S_2, \sigma_G^{S_1 \cup S_2}, q_G^{S_1 \cup S_2}) \mid \\
(S_1, \sigma_{G_1}^{S_1}, q_{G_1}^{S_1}) \in \mathcal{S}(G_1, k', 0), \\
(S_2, \sigma_{G_2}^{S_2}, q_{G_2}^{S_2}) \in \mathcal{S}(G_2, l - k', 0)\}, \quad (4.7)
\end{aligned}$$

where $\sigma_G^{S_1 \cup S_2}$ is calculated from $\sigma_{G_1}^{S_1}$ and $\sigma_{G_2}^{S_2}$ using Eq. (4.1), and $q_G^{S_1 \cup S_2}$ is also calculated from $q_{G_1}^{S_1}$ and $q_{G_2}^{S_2}$ using Eq. (4.2). Note that p_0 must be set to 1 in the calculation of $\sigma_G^{S_1 \cup S_2}$ and $q_G^{S_1 \cup S_2}$ of $\mathcal{S}'(G, l, 1)$.

For a non-decomposable TTSPG $G = (\{s, t\}, \{s \to t\}, p)$,

$$\begin{aligned}
&\mathcal{S}(G, 0, 0) = \{(\emptyset, p_0, p(s \to t)p_0)\}, \mathcal{S}(G, 0, 1) = \emptyset \\
&\mathcal{S}(G, 1, 0) = \emptyset \text{ and } \mathcal{S}(G, 1, 1) = \{(\{s\}, 1, p(s \to t))\}
\end{aligned}$$

hold. The correctness of our algorithm can be proved by mathematical induction on the number of edges of G using this fact and the correctness of recursive formulas (4.5),(4.6) and (4.7).

Theorem 2 *Algorithm described above calculates the k-element subset that maximizes its effect for a given TTSPG G.*

The pseudo code of the proposed algorithm is shown in Algorithm 1 of our supplementary material.

4.5 Application to a General Network

Lappas et al. [29] developed an algorithm to solve an influence maximization problem for a tree and used it to approximately solve that for a general weighted directed graph G by applying their algorithm to a spanning tree T of G. In order to obtain a good approximate solution for the original graph $G = (V, E, p)$, the set E' of edges in the spanning tree $T = (V, E', p)$ are selected so as to maximize its *likelihood* $\prod_{e \in E'} p(e)$ in their method. Such a spanning tree can be found by solving the problem of finding the minimum spanning tree for the graph $G' = (V, E, -\ln p)$, which is known to be solved in $O(|E| + |V| \ln |V|)$ time [37].

Different from trees, there may be no spanning TTSPGs even for the graphs in which there is a node that is reachable to any node. Such graphs, however, have a spanning pseudo-TTSPG, which is defined as follows.

Definition 2 *Let $G = (V, E)$ be a directed graph and let $V_t (\subseteq V)$ be the set of nodes with no outgoing edge. A directed graph (V, E) is said to be a pseudo-TTSPG if the directed graph $(V \cup \{w\}, E \cup \{v \to w \mid v \in V_t\})$ is a TTSPG for $w \notin V$.*

If a spanning tree exists for G, a spanning pseudo-TTSPG also exists because a directed tree is a pseudo-TTSPG. For a given weighted pseudo-TTSPG (V, E', p), a corresponding weighted TTSPG $(V \cup \{w\}, E' \cup \{v \to$

$w \mid v \in V_t\}, p')$, which can be seen as a graph with the same propagation structure, is obtained easily by adding 0-weight edges $v \to w$ to E' for all the nodes v in V_t, where V_t and w are notations used in Definition 2, and weight function p' is defined as $p'(e) = p(e)$ for $e \in E'$ and $p'(e) = 0$ otherwise.

Therefore, in order to use our algorithm to approximately solve an influence maximization problem for a general weighted directed graph G, we only have to find a spanning pseudo-TTSPG with many edges and high likelihood. Since no efficient algorithm to find a minimum spanning pseudo-TTSPG with a maximal set of edges has not been known yet, we took the following greedy approach for a given weighted directed graph (V, E, p).

1. Calculate the spanning tree (V, E', p) with the maximal likelihood using the algorithm of Gabow et al. [37].

2. For each $e \in E \setminus E'$, do the following in decreasing order of $p(e)$: if $(V, E' \cup \{e\}, p)$ is still a pseudo-TTSPG, add e to E'.

To visually illustrate improvement by using a spanning pseudo-TTSPG instead of a spanning tree, we applied the above greedy approach to a small synthetic collaborative network constructed as follows. Consider a set V of individuals and a set W of works. Assign a randomly selected pair $\{u, v\}$ of individuals in V to each work $w \in W$. Let $m(w)$ denote the set of individuals assigned to w. Construct a weighted directed graph (V, E, p) from V and W by defining E and p as follows:

$$E = \{u \to v \mid \{u, v\} \subseteq m(w) \text{ for some } w \in W\} \tag{4.8}$$

and

$$p(u \to v) = \frac{|\{w \in W \mid \{u, v\} \subseteq m(w)\}|}{|\{w \in W \mid v \in m(w)\}|}, \tag{4.9}$$

where $|S|$ denotes the number of elements in a set S.

The pseudo-TTSPG shown in Figure 4.5.1 was constructed by applying our greedy approach to the graph generated using the above procedure with $|V| = 30$ and $|W| = 200$. Note that the solid edges represent the edges

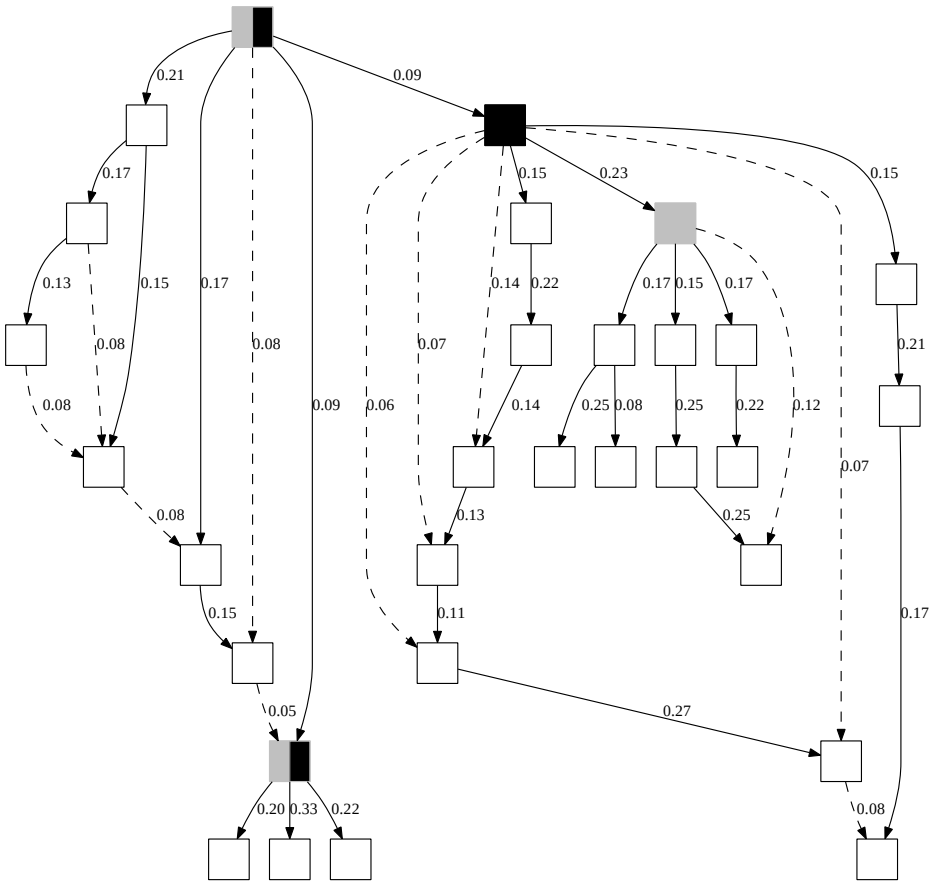

Figure 4.5.1: The pseudo-TTSPG constructed by applying our greedy approach to the graph generated using our randomized procedure. Solid edges belong to both the spanning tree and pseudo-TTSPG but dashed edges belong to the spanning pseudo-TTSPG only. The set of three nodes colored by gray and that colored by black are solutions of the influence maximization problems with $k = 3$ for the tree and the pseudo-TTSPG, respectively.

that also belong to the spanning tree calculated in the first step, and the dashed edges represent the edges that were added in the second step. The number of edges increased from 29 to 40 (38% increased). We can see that the dashed edges shrink the distances between the nodes. The solutions of the influence maximization problem for the spanning tree and pseudo-TTSPG

are also shown in Figure 4.5.1 in the case with three initial active nodes ($k = 3$): gray nodes for the tree and black nodes for the pseudo-TTSPG. The two nodes are common in both the solutions but the other node in each solution is different from each other. The node unique to the solution for the pseudo-TTSPG has several neighbor nodes connected by the dashed edges and can affect more nodes than the node unique to the solution for the tree. In fact, in the original graph, the effect of the solution for the pseudo-TTSPG averaged over 10000 simulations was 13.22 while that for the tree was 12.29. Thus, a solution with higher approximation ratio can be expected using a pseudo-TTSPG instead of a tree.

4.6 Experiments

4.6.1 Experiments for TTSPGs

In this subsection, we describe empirical evaluation on calculation time and accuracy of our method comparing to those of the greedy approximation method developed by Leskovec et al. [30].

4.6.2 Dataset

We generated n-node TTSPGs by recursive procedure GenTTSPG(n) that is described as the following steps.

Step 1 If $n = 2$, return a directed graph consisting of two vertices joined by one edge.

Step 2 Choose "series" or "parallel" composition by equal probability.

Step 3 If "series" composition is chosen, select an integer n' from $\{2, 3, \ldots, n-1\}$ randomly, generate an n'-node TTSPG G_1 and an $(n-n'+1)$-node TTSPG G_2 by calling GenTTSPG(n') and GenTTSPG($n - n' + 1$), respectively, and return a TTSPG G that is made from G_1 and G_2 by series composition.

Step 4 Otherwise (if "parallel" composition is chosen), select an integer n' from $\{2, 3, \ldots, n\}$ randomly, generate an n'-node TTSPG G_1 and an $(n-n'+2)$-node TTSPG G_2 by calling GenTTSPG(n') and GenTTSPG($n-n'+2$), respectively, and return a TTSPG G that is made from G_1 and G_2 by parallel composition.

For each number of nodes $n \in \{100, 200, 400, 800\}$, we generated 5 TTSPGs by GenTTSPG. The weights of the edges are set to a constant value $p \equiv 0.1$.

4.6.3 Comparative Algorithm

Since no algorithm to exactly solve an influence maximization problem for a TTSPG has been developed so far, we compare the performance of our algorithm to that of a hill-climbing approximate algorithm for a general network developed by Kempe et al. [28] and improved by Leskovec et al. [30], which we call Greedy for short. For a give weighted directed graph (V, E, p), Greedy selects a k-node subset S of V as an approximate solution of the influence maximization problem for (V, E, p) by starting from an empty subset S and repeatedly adding a node $v \in V \setminus S$ to S in such a way as to maximally increase the effect of S. Greedy is guaranteed to output $(1 - 1/e - \epsilon)$-approximate solution for any given $\epsilon > 0$ by theoretical analysis based on submodularity of the effect function. Greedy estimates the effect of a given set of initial active nodes by simulating the state propagation many times.

4.6.4 Experimental Settings

All the experiments were conducted on a Linux machine (Ubuntu 12.04.2) with an Intel Xeon E5-2430 2.20GHz CPU and 64GB memory. The proposed algorithm and Greedy were implemented in g++ 4.6.3 using Boost Graph Library. To estimate the effect of a given set of initial active nodes, we let Greedy do 10,000 simulations, which are the same number of simulations as Kempe et al. [28] had let it do in their experiments. All the results are averaged over 5 TTSPGs.

4.6.5 Results

The running time dependency on the number of nodes in a given graph is shown in Figure 4.6.1. In this experiment, the number of the initial active nodes are fixed to 8. Our algorithm is significantly faster than Greedy: at least about 50 times faster for all the numbers of nodes in our setting ($n = 100, 200, 400, 800$).

In Figure 4.6.2, we show the running time dependency on the number of initial active nodes. The number of nodes in a given graph is fixed to 800 in the experiment. Our algorithm is faster than Greedy for all the numbers of initial active nodes in our setting ($k = 3, 4, 5, 6, 7, 8, 9$). The growth rate of the running time of our algorithm, however, is larger than that of Greedy.

The approximation ratio $\sigma(\hat{S})/\sigma(S^*)$ of Greedy is shown in Figure 4.6.3, where $\hat{S}$ is the Greedy's approximate solution and S^* is the exact solution. Note that our algorithm always outputs the exact solution. The approximation ratio of Greedy is less than 0.9 in all the settings.

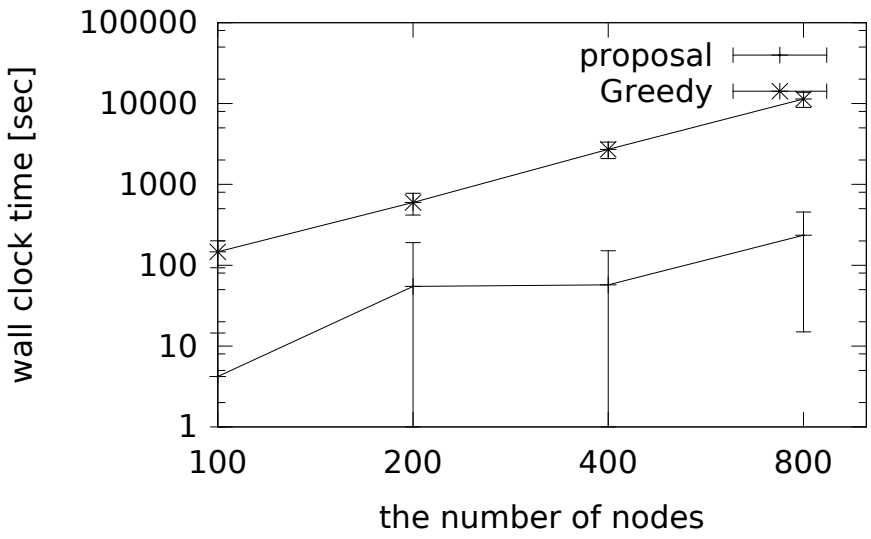

Figure 4.6.1: Running times and their 95% confidence intervals of our algorithm and Greedy for the influence maximization problem with $k = 8$.

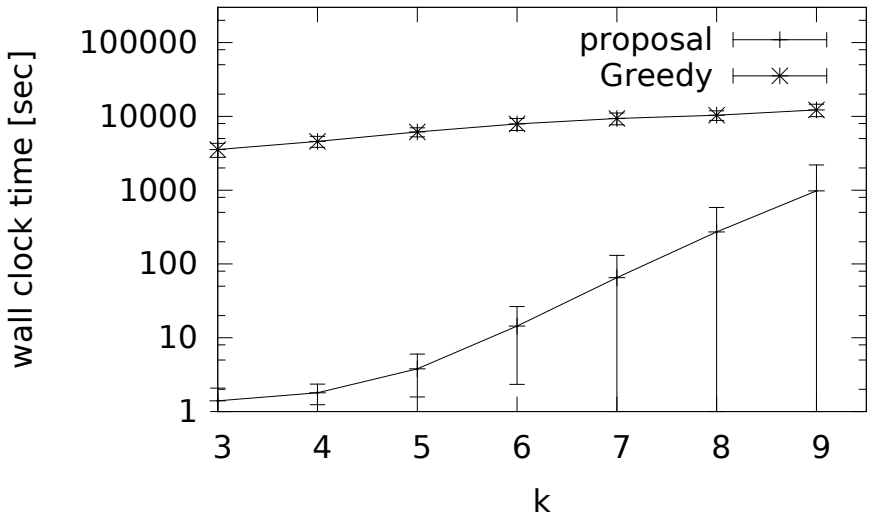

Figure 4.6.2: Running times and their 95% confidence intervals of our algorithm and Greedy for the influence maximization problem in the graphs with 800 nodes.

4.7 Experiments for Real Networks

In order to demonstrate the effectiveness of our approach to application for the general networks, we conducted experiments using real networks. Real networks used in our experiments are the collaborative networks made from the snapshot of the DBLP database on April. 4th, 2014. We extracted the sets of influential researchers from the collaborative networks using our approach and compared them with those extracted by the conventional approach using a spanning tree developed by Lappas et al. [29].

We made a weighted directed graphs (V, E, p) of conference A from the set D of papers recorded in the DBLP database as follows. Let $W(\subseteq D)$ denote the set of papers presented in conference A in some past year and let V denote the set of authors of the papers in W. Define $m(w)$ as the set of authors of a paper w for each paper $w \in W$. Then, an edge set E and a weight function p can be obtained using Eq. (4.8) and Eq. (4.9).

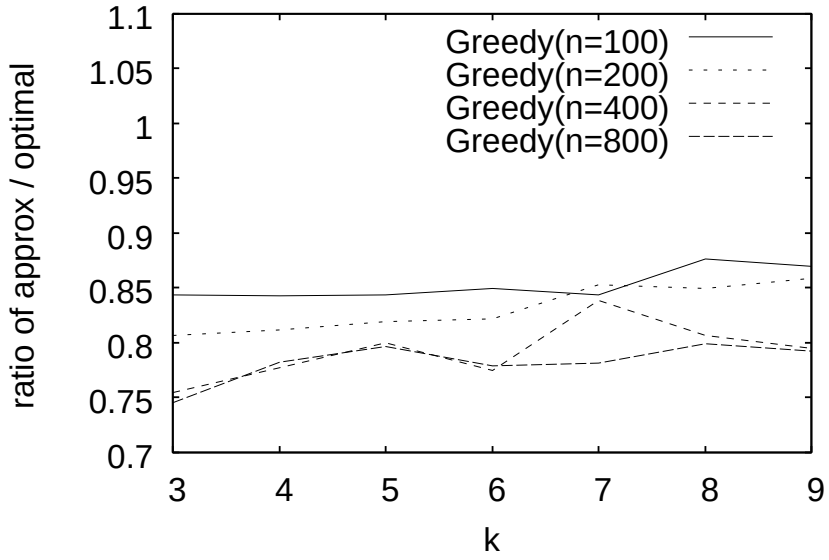

Figure 4.6.3: The approximation ratio $\sigma(\hat{S})/\sigma(S^*)$ of Greedy averaged over 5 TTSPGs.

We made graphs for several major conferences on machine learning and data mining and conducted our experiments using them, but we report the results for ACML and SDM only due to space limitation. The numbers of nodes and edges are 341 and 1,146 in ACML graph, and 1,804 and 12,836 in SDM graph.

The results of our experiments are shown in Table 4.1. The number k of initial active nodes was set to 10 for ACML and 20 for SDM considering the sizes of the graphs. In both ACML and SDM graphs, there are three authors unique to the set of the influential authors extracted by each the approach, and more influential authors seem to be extracted by using spanning pseudo-TTSPGs in terms of the h-index sum, where h-index of an author x is the maximum integer h satisfying the condition that there are at least h x's papers which have been cited by at least h papers.

Table 4.1: **Authors extracted as the solution of influence maximization problem**: The solutions for a spanning pseudo-TTSPG and a spanning tree are shown. We used the h-indeces provided by ArnetMiner (http://arnetminer.org/) on October. 8, 2014.

Result for ACML graph			
Extracted authors	h-idx	Tree	TTSPG
Qiang Yang	66		✓
Zhi-Hua Zhou	58	✓	✓
Thomas G. Dietterich	49		✓
Kristian Kersting	29	✓	✓
Masashi Sugiyama	27	✓	✓
Alan Fern	25	✓	
Steven C. H. Hoi	25	✓	
Aapo Hyvärinen	24	✓	✓
Yang Zhang	24	✓	✓
Jan Ramon	22	✓	✓
Mohammad Ghavamzadeh	15		✓
Kouzou Ohara	11	✓	✓
Qian Zhou	10	✓	
H-index sum of extracted authors		255	325
Result for SDM graph			
Extracted authors	h-idx	Tree	TTSPG
Philip S. Yu	127	✓	✓
Jiawei Han	117	✓	✓
Christos Faloutsos	94	✓	✓
Vipin Kumar	70	✓	✓
Hans-Peter Kriegel	68	✓	✓
Heikki Mannila	58	✓	✓
Eamonn J. Keogh	53	✓	✓
Luc De Raedt	50	✓	✓
Alok N. Choudhary	50	✓	✓
Joydeep Ghosh	43		✓
Ee-Peng Lim	39	✓	
Carla E. Brodley	36	✓	✓
I. V. Ramakrishnan	35		✓
Aidong Zhang	31	✓	✓
Wei Liu	29		✓
Haesun Park	28	✓	✓
Pang-Ning Tan	24	✓	✓
Carlotta Domeniconi	20	✓	
Alexander G. Gray	14	✓	✓
Einoshin Suzuki	14	✓	✓
Michael C. Burl	11	✓	✓
Sameep Mehta	11	✓	✓
Ruchita Bhargava	3	✓	
H-index sum of extracted authors		918	963

Chapter 5

Conclusion

5.1 Results

In this thesis, we proposed approximation algorithms 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 bound of closeness centrality of each vertex. FAOM and FAOM2 use k-neighbor dense shortest path graphs with shortcuts, which results in empirical efficiency and approximation ratio close to 1.

We also proposed a method to find the most influential k nodes in a TTSPG. Our method can exactly solve the influence maximization problem on a TTSPG, and outperformed the greedy approximation method developed by Leskovec et al.[30] in our experiments using synthetically generated random TTSPGs. We also proposed a way to convert arbitrary weighted directed graphs into weighted TTSPGs with edges important for propagation. The effectiveness of our method for the real dataset was demonstrated compared with the Lappas et al's method[29].

5.2 Future Directions

For FAOM and FAOM2, 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.

For our method to solve the Influence Maximization Problem on TTSPs, it is our future work to theoretically guarantee the efficiency of our algorithm.

Bibliography

[1] L. C. Freeman, Centrality in social networks conceptual clarification. *Social networks*, **1**, 3(1978), 215–239.

[2] P. Hage and F. Harary, Eccentricity and centrality in networks. *Social networks*, **17**, 1(1995), 57–63.

[3] P. Bonacich, Power and centrality: A family of measures. *American journal of sociology*, **92**, 5(1987), 1170–1182.

[4] D. Kempe, J. Kleinberg and É. Tardos, Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, 137–146.

[5] K. Tabata, A. Nakamura and M. Kudo, Fast Approximation Algorithm for the 1-Median Problem. *Proceedings of the 15th International conference on Discovery Science*, 2012, 169–183.

[6] K. Tabata, A. Nakamura and M. Kudo, An Efficient Approximate Algorithm for the 1-Median Problem on a Graph. *IEICE TRANSACTIONS on Information and Systems*, **E100-D**(2017), In press.

[7] K. Tabata, A. Nakamura and M. Kudo, An Algorithm for Influence Maximization in a Two-Terminal Series Parallel Graph and its Application to a Real Network. *Proceedings of the 18th International conference on Discovery Science*, 2012, 275–283.

[8] E. W. Dijkstra, A note on two problems in connexion with graphs. *Numerische mathematik*, **1**, 1(1959), 269–271.

[9] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, **34**, 3(1987), 596–615.

[10] M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, **46**, 3(1999), 362–394.

[11] P. Erdös and A. Rényi, On random graphs I. *Publicationes Mathematicae Debrecen*, **6**(1959), 290–297.

[12] S. L. Hakimi, Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research*, **12**, 3(1964), 450–459.

[13] D. B. Johnson, Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, **24**, 1(1977), 1–13.

[14] R. E. Burkard and J. Krarup, A linear algorithm for the pos/neg-weighted 1-median problem on a cactus. *Computing*, **60**, 3(1998), 193–215.

[15] M. J. Rattigan, M. Maier and D. Jensen, Using Structure Indices for Efficient Approximation of Network Properties. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, 357–366.

[16] D. E. J. Wang, Fast approximation of centrality. *Graph Algorithms and Applications 5*, **5**(2006), 39.

[17] K. Okamoto, W. Chen and X.-Y. Li, Ranking of closeness centrality for large-scale social networks. *International Workshop on Frontiers in Algorithmics*, 2008, 186–195.

[18] O. Kariv and S. L. Hakimi, An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, **37**, 3(1979), 539–560.

[19] O. Alp, E. Erkut and Z. Drezner, An efficient genetic algorithm for the p-median problem. *Annals of Operations research*, **122**, 1-4(2003), 21–42.

[20] M. G. Resende and R. F. Werneck, On the implementation of a swap-based local search procedure for the p-median problem. *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03)*, 2003, 119–127.

[21] Y. Kochetov, T. Levanova, E. Alekseeva and M. Loresh, Large neighborhood local search for the p-median problem. *Yugoslav Journal of Operations Research*, **15**, 1(2005), 53–63.

[22] A. Goldman, Optimal center location in simple networks. *Transportation science*, **5**, 2(1971), 212–221.

[23] P. Domingos and M. Richardson, Mining the network value of customers. *Proc. of KDD*, 2001, 57–66.

[24] J. Goldenberg, B. Libai and E. Muller, Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, **12**, 3(2001), 211–223.

[25] H. W. Hethcote, The mathematics of infectious diseases. *SIAM review*, **42**, 4(2000), 599–653.

[26] E. M. Rogers, *Diffusion of innovations*. Simon and Schuster, 2010.

[27] W. O. Kermack and A. G. McKendrick, A Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **115**(1927), 700.

[28] D. Kempe, J. Kleinberg and É. Tardos, Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, 137–146.

[29] T. Lappas, E. Terzi, D. Gunopulos and H. Mannila, Finding effectors in social networks. *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, 1059–1068.

[30] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen and N. Glance, Cost-effective outbreak detection in networks. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, 420–429.

[31] M. Richardson and P. Domingos, Mining knowledge-sharing sites for viral marketing. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, 61–70.

[32] A. Goyal, W. Lu and L. V. Lakshmanan, Celf++: optimizing the greedy algorithm for influence maximization in social networks. *Proceedings of the 20th international conference companion on World wide web*, 2011, 47–48.

[33] K. Saito, M. Kimura and H. Motoda, Discovering influential nodes for SIS models in social networks. *Discovery Science*, 2009, 302–316.

[34] K. Saito, M. Kimura, K. Ohara and H. Motoda, Efficient discovery of influential nodes for SIS models in social networks. *Knowledge and information systems*, **30**, 3(2012), 613–635.

[35] S. Bharathi, D. Kempe and M. Salek. Competitive influence maximization in social networks, 2007.

[36] J. Valdes, R. E. Tarjan and E. L. Lawler, The recognition of series parallel digraphs. *SIAM Journal on Computing*, **11**, 2(1982), 298–313.

[37] H. N. Gabow, Z. Galil, T. Spencer and R. E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, **6**(1986), 109–122.