EFFICIENT AND EFFECTIVE LOCAL ALGORITHMS FOR ANALYZING MASSIVE GRAPHS

by

YUBAO WU

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

May, 2016

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the dissertation of

Yubao Wu

candidate for the degree of **Doctor of Philosophy***.

Committee Chair

Dr. Xiang Zhang

Committee Member

Dr. Xiaofeng Zhu

Committee Member

Dr. Jing Li

Committee Member

Dr. Z. Meral Özsoyoğlu

Date of Defense

August 28, 2015

*We also certify that written approval has been obtained

for any proprietary material contained therein.

To my mother, Fengqing Yuan

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my advisor, Dr. Xiang Zhang. He had spent a lot of time and energy to teach me. When I made mistakes, he was very patient and kept correcting me. He figured out my weaknesses, and helped me to improve every aspect of my research ability. When I was depressed in the research, his supports and encouragements made me more confident in my research and see the bright future. He is insightful and devoted to the research, which stimulates me and makes me know what a good researcher should look like. The research experience under the supervision of him makes me grow a lot in every aspect of my research ability. I appreciate this precious learning experience provided by him.

I would like to thank my committee members, Dr. Xiaofeng Zhu, Dr. Jing Li, and Dr. Z. Meral Özsoyoğlu, for their constructive suggestions. I especially thank Dr. Zhu for his supports about the statistical knowledge. I thank Dr. Ruoming Jin for his supports about the graph knowledge. I also thank Dr. Mehmet Koyutürk for teaching me algorithms and systems biology.

I would like to thank Dr. Wei Cheng, Dr. Shunping Huang, and Dr. Zhaojun Zhang from UNC. They have helped me at the beginning of my research. I would like to thank my classmates. They are Jingchao Ni, Yuchen Bian, Yuan Li, Heming Wang, Ke Hu, Shi Qiao, Zhuofu Bai, Dr. Gang Shu, Jing Chen, Marzieh Ayati, Dr. Matthew Ruffalo, and Ye Fang. The insightful discussions with them substantially broaden my mind and inspire me to keep learning. Their optimistic attitudes also give me the power and keep me sane. I also would like to thank my roommates, Xu Han and Zhaojun Xue. They are magnanimous and provide me selfless help.

Finally, I would like to thank my family. They love me deeply.

Efficient and Effective Local Algorithms for Analyzing Massive Graphs

Abstract

by

YUBAO WU

Graph is a ubiquitous data structure that can model many real-world problems. There are several important problems in graph analysis, such as ranking, community detection and densest subgraph detection. The global versions of these problems have been extensively studied. Recently, the graphs are becoming larger and larger and may contain billions of nodes. The large scale of graphs makes it prohibitive to apply the existing global algorithms. In many circumstances, the entire graph is unavailable, such as the Twitter social network. On the other hand, in many applications, the user is only interested in the local patterns. Thus, the local versions of these problems and algorithms have attracted intense research interests. Ideally, the local algorithms are preferred for solving these problems efficiently without searching the entire graph.

In this dissertation, we focus on the local pattern mining problems and related local search algorithms. We study the top-k proximity query problem, the local community detection problem, and the densest connected subgraph problem in dual networks. For the top-k proximity query problem, we devise an efficient and unified local search algorithm, which can be applied for a variety of random walk based proximity measures. For the local community detection problem, we discover that the existing local community detection methods suffer from a serious free rider effect, and develop the query biased node weighting scheme to mitigate the free rider effect. The densest connected subgraph problem is motivated from real applications and is a natural extension of the densest subgraph problem from a single network to dual networks. For each problem, we perform comprehensive experiments to evaluate the effectiveness and efficiency of the proposed method on large real and synthetic networks. The results demonstrate that, for each problem, the proposed method outperforms the state-of-the-art methods in terms of effectiveness and efficiency.

Chapter 1

Introduction

Graph (or network[1]) is a ubiquitous data structure that can model many real-world problems in social science, biology, physics, and information technology. There are several primitive problems in analyzing the network data, such as ranking, community detection, and densest subgraph detection. Given a query node, the ranking problem aims at ranking the other nodes based on the closeness to the query node. Usually random walk based methods are used to measure the closeness between nodes [25]. Community detection methods aim at detecting all the communities in the entire network [33]. The densest subgraph problem finds the subgraph with the maximum density (average edge weight) in a single network [71].

In recent years, the graphs are becoming larger and larger, and the entire graph may contain billions of nodes and tens of billions of edges. Thus it is a great challenge to process the entire graph efficiently. In many circumstances, the entire graph is even unavailable. For example, the Twitter company only provides limited query functions thus we can only obtain partial social network in Twitter. On the other hand, in many applications, we are only interested in local patterns. For example, in the application of recommendation, we only need to find the top-k nodes which are closest to the query node [34, 61]. In local community detection, given a set of query nodes, we are

[1]In this dissertation, we use network and graph interchangeably.

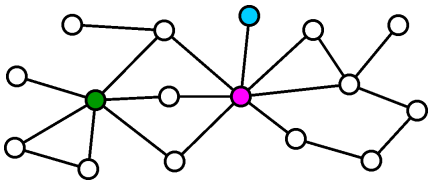


Figure 1.1: An example graph for the top-k proximity query problem

only interested in the community which is near the query nodes [24, 28]. In densest subgraph detection, we are also interested in detecting local dense subgraphs. In these applications, we only focus on detecting the local patterns in large graphs. Next, we introduce these three problems with more details.

1.1 Top-k Proximity Query Problem

Proximity measures on graphs are the primitive topic in various applications associated with graphs. Numerous proximity measures have been proposed from different disciplines. Random walk based proximity measures have been demonstrated to outperform other proximity measures, such as those based on shortest paths, common neighbors, or network flows. The random walk based proximity measure can capture multiple paths connecting two nodes. For example, in Figure 1.1, the cyan node is close to the purple query node because it is directly adjacent to the query node. However, the green node maybe closer to the query node because it has three paths connecting to the query node. Various random walk based proximity measures have been proposed, such as random walk with restart [112], effective importance [14], RoundTripRank [31], and SimRank [50].

Even though the random walk based proximity measures are effective in capturing the proximity between nodes, their computations are usually expensive. The reason is that they simulate many random walks on graphs to capture the multiple paths between two nodes. The naive method is to utilize the linear recursive equations of each proximity measure, and use the power iteration method to compute the exact proximity values. However, this simple power iteration method needs to iterate over

the entire graph, thus the complexity is prohibitive on large graphs. The Monte Carlo based algorithm is another method. It simply simulates many random walks beginning from the query node, and calculates some statistics of the samples to estimate the proximity values. The Monte Carlo based algorithm is approximate, and it needs large samples to get an accurate estimation.

In real applications, given a query node, we are usually only interested in the top ranking items. Thus, it is unnecessary to compute the exact proximity values of all the nodes in the graph. Specifically, the top-k proximity query problem is defined as

Problem 1. [Top-k Proximity Query Problem] *Suppose that we have an undirected graph $G(V, E)$, a query node q and a number k. Let $\mathbf{r}_i$ represent the proximity value of node i with regard to node q. The top-k proximity query problem aims at finding a node set $K \subseteq V \setminus \{q\}$ such that $|K| = k$ and $\mathbf{r}_i \geq \mathbf{r}_j$, for any node $i \in K$ and $j \in V \setminus K \setminus \{q\}$.*

Many research effects have been devoted to study how to efficiently retrieve the top-k nodes with regard to the query node. Some of them improve the power iteration method by computing the upper bounds and pruning the nodes during the iterations [35]. They guarantee the exactness of the top-k results. Some of them search locally, and try to find the approximate top-k results [97]. Intuitively, the top-k nodes should be in the neighborhood of the query nodes. Thus the question is how to develop an efficient local search algorithm which guarantees the exactness of the top-k results. We will show that our developed algorithm is such a local search algorithm, which searches locally and also guarantees the exactness of the top-k results.

1.2 Local Community Detection Problem

Community detection in graphs is a primitive task in graph analysis. The basic assumptions of communities are

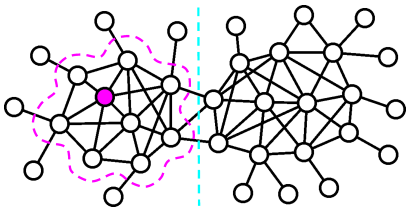


Figure 1.2: An example graph for the local community detection problem

"the nodes are densely connected within the community, and sparsely connected to the nodes outside the community".

The conventional community detection problem aims at finding all the communities in the entire graph. It is also known as graph partitioning, which partitions the entire graph into disjoint communities. For example, in Figure 1.2, the spectral clustering algorithm will partition the graph into two parts indicated by the cyan dashed line. Community detection can help us summarize the graphs, give us more insights about the structure of real graphs, and let us analyze the graph more effectively. The detected communities can be applied in many important commercial activities, such as recommendation and link prediction.

In many applications, we are interested in some query nodes, and we want to discover the community near the query nodes. All the communities in the entire graph are hard to compute and provide too much abundant information. This motivates the researchers to study the local community detection problem, which aims at only finding the local community near the query nodes. Specifically, the local community detection problem is defined as

Problem 2. [Local Community Detection Problem] *Given an undirected graph, a set of query nodes, and a goodness metric evaluating how good a subgraph is a community, find the subgraph such that*

1) it contains all the query nodes;

2) its goodness metric is maximized.

The goodness metric can be defined based on the basic assumptions of communities. For example, the density (average edge weight) of the subgraph has been used

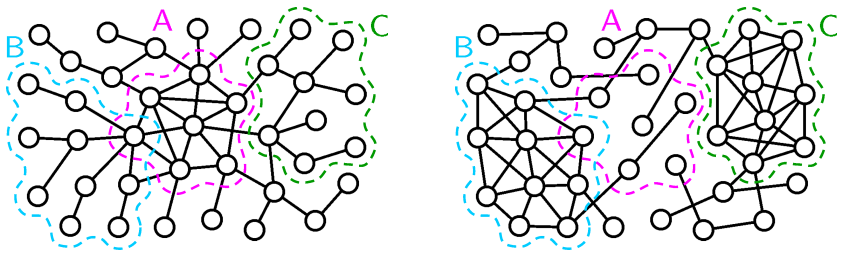
as a goodness metric for local community detection [96]. Given the example graph in Figure 1.2 and suppose that the purple node is the query node, the local community detection problem aims at only detecting the community in the purple dashed curve. This is different from the global community detection problem, which aims at partitioning the entire graph.

Most existing algorithms perform a local search heuristically to find a local optimal subgraph in terms of one specific goodness metric. We discover that when we merge the global optimal subgraph into the local optimal subgraph, some representative goodness metrics are always increasing. We call this phenomenon the free rider effect. It hinders us from discovering the desired local community accurately. We propose to use the query biased node weighting scheme to mitigate the free rider effect.

1.3 Densest Connected Subgraph Problem

Densest subgraph problem is a classic problem in graph analysis. This problem can be traced back to 1984, when the researcher discovered that it could be solved in polynomial time [39]. Before that, researchers thought that this problem was NP-hard. To further improve the efficiency, numerous algorithms have been proposed [20, 37]. This problem has many useful applications. In the previous subsection, the density goodness metric used in local community detection is an application of the densest subgraph problem.

In many applications, there exist dual networks, where one is the physical network and the other is the conceptual network. These two networks contain the same set of nodes, but different and independent sets of edges. Figure 1.3 shows one example of dual networks. The node layouts in the physical and conceptual networks are identical. The edges in the physical and conceptual networks are independent. We can see that, in this example, the overlapping between the two sets of edges in these two networks is very small.



(a) Physical network (b) Conceptual network

Figure 1.3: An example of dual networks for the densest connected subgraph problem

The dual networks model is observed and abstracted from real applications. In the biology application, each node in the dual networks represents a protein, each edge in the physical network represents that two proteins have physical bonding interaction, and each edge in the conceptual network represents that two proteins have genetic interaction. In the bibliographic information analysis application, each node represents an author, each edge in the physical network represents that two authors have co-authored a paper, and each edge in the conceptual network represents that two authors have similar research interests. In the social recommender system application, each node represents a user, each edge in the physical network represents that two users are friends, and each edge in the conceptual network represents two users have similar interests in the commercial products. Please see Section 4.10 for a detailed description of various real-life dual networks.

One edge in the physical network may not exist in the conceptual network. The global densest subgraph in the physical network represents the dense physical interaction, however, it may not be dense at all in the conceptual network. For example, in Figure 1.3, subgraph A is the global densest subgraph in the physical network, but it contains only one edge in the conceptual network. Motivated by real applications, we are interested in the local dense subgraph in the conceptual network, which is also connected in the physical network. In the dual networks in Figure 1.3, we are interested in subgraphs B and C, which are densely connected in the conceptual network and sparsely connected in the physical network. We call this pattern the densest connected subgraph in dual networks. The problem is defined as follows.

Table 1.1: Main contributions

tasks	limitations of existing works	our contributions
top-k proximity query	global: expensive; local: approximate	simple, unified and exact local search
local community detection	free rider effect	query biased densest subgraph
densest subgraph detection	single network; co-dense	dual networks; densest connected subgraph

Problem 3. [Densest Connected Subgraph Problem] *Given dual networks, find the subgraph such that*

1) it is connected in the physical network;

2) its density in the conceptual network is maximized.

We study the complexity of the densest connected subgraph problem, and prove that it is NP-hard. This is interesting because the densest subgraph problem in a single network is polynomial time solvable. We propose two heuristic algorithms to discover the densest connected subgraph from dual networks. The experimental results demonstrate the new method can discover many interesting patterns, and the proposed algorithms run efficiently.

1.4 Overview of the Proposed Methods

In this dissertation, we present three efficient and effective algorithms for finding local patterns in large graphs. Table 1.1 summarizes the contributions of these works.

Top-k Proximity Query Problem: Various random walk based measures have been proposed to measure the proximity between nodes. For different proximity measures, specialized algorithms are usually needed for efficient computation. Based on their search strategy, the existing algorithms can be categorized into *global* and *local* methods. Global methods usually need to iterate over the entire graph or require an expensive preprocessing step [112, 34]. Local methods try to approximate proximities by searching a small part of the graph [97, 98]. Although more efficient than

global methods, local methods usually cannot guarantee the exactness of the results. Moreover, with a variety of measures available, designing specialized algorithms for different measures is a tedious task. Thus, we develop a unified and exact local search method, FLoS (Fast Local Search), for efficient top-k proximity query in large graphs [121, 122]. Unlike the existing local search algorithms, FLoS can guarantee the exactness of the solution. Moreover, it can be applied to a variety of commonly used proximity measures. FLoS is based on the *no local optimum* property of the proximity measures. We show that many measures have no local optimum. Utilizing this property, we introduce several operations on transition probabilities, which allow developing tight lower and upper bounds on the proximity values. The bounds monotonically converge to the exact proximity when more nodes are visited. We further extend FLoS to measures having local optimum by utilizing relationship among different measures.

Local Community Detection Problem: We discover that most existing methods [24, 96, 103] tend to include irrelevant subgraphs in the detected local community. We refer to such irrelevant subgraphs as free riders. We systematically study the existing goodness metrics and provide theoretical explanations on why they may cause the free rider effect. We further develop a query biased node weighting scheme to reduce the free rider effect [120]. In particular, each node is weighted by its proximity to the query node. We define a query biased density metric to integrate the edge and node weights. The query biased densest subgraph, which has the largest query biased density, will shift to the neighborhood of the query nodes after node weighting. We then formulate the query biased densest connected subgraph problem, study its complexity, and provide efficient algorithms to solve it.

Densest Connected Subgraph Problem: We observe that in many emerging applications, there exist *dual* networks. For example, in genetics, it is important to use protein interactions to interpret genetic interactions [106]. In this application, one network represents *physical* interactions among nodes, e.g., protein-protein in-

teractions, and another network represents *conceptual* interactions, e.g., genetic interactions. Edges in the conceptual network are usually derived based on certain correlation measure or statistical test measuring the strength of the interaction. Two nodes with strong conceptual interaction may not have direct physical interaction. Thus, we propose the novel dual network model and investigate the problem of finding the densest connected subgraph (DCS) which has the largest density in the conceptual network and is also connected in the physical network [123, 124]. Density in the conceptual network represents the average strength of the measured interacting signals among the set of nodes. Connectivity in the physical network shows how they interact physically. Such pattern cannot be identified using the existing algorithms for a single network. We show that even though finding the densest subgraph in a single network is polynomial time solvable, the DCS problem is NP-hard. We develop a two-step approach to solve the DCS problem. In the first step, we effectively prune the dual networks while guarantee that the optimal solution is contained in the remaining networks. For the second step, we develop two efficient greedy methods based on different search strategies to find the DCS. Different variations of the DCS problem are also studied.

Next, in Chapters 2, 3, and 4, we describe the proposed algorithms for the top-k proximity query problem, the local community detection problem, and the densest connected subgraph problem respectively.

Chapter 2

Unified and exact local search for random walk based top-k proximity query in large graphs

2.1 Introduction

Given a large graph and a query node, finding its k-nearest-neighbor (kNN) is a primitive operation that has recently attracted intensive research interests [98, 19, 70, 35]. In general, there are two challenges in top-k proximity query. One is to design proximity measures that can effectively capture the similarity between nodes. Another challenge is to develop efficient algorithms to compute the top-k nodes for a given measure.

Designing effective proximity (similarity) measures is a nontrivial task. Random walk based measures have been shown to be effective in many applications. Some examples include discounted/truncated hitting time [98, 97], penalized hitting probability [40, 128], random walk with restart [112, 14], RoundTripRank [31], and absorption probability [119].

Although various proximity measures have been developed, how to efficiently com-

pute them remains a challenging problem. For most random walk based measures, a naive method requires matrix inversion, which is prohibitive for large graphs. Two *global* approaches have been developed. One applies the power iteration method over the entire graph [95, 35, 61]. Another approach precomputes and stores the inversion of a matrix [112, 34, 36]. The precomputing step is usually expensive and needs to be repeated whenever the graph changes.

To improve the efficiency, local methods have been developed [98, 97, 128, 14]. The idea is to visit the nodes near the query node and dynamically expand the search range. Node proximities are estimated based on local information only. Without using the global information, however, most existing local search methods cannot guarantee to find the exact solution. Moreover, they are usually designed for specific measures and cannot be generalized to other measures.

In this chapter, we propose FLoS (Fast Local Search), a simple and unified local search method for efficient and exact top-k proximity query in large graphs. FLoS has the following properties.

- *Exact*: It guarantees to find the exact top-k nodes.
- *Unified*: It is a general method that can be applied to a variety of random walk based proximity measures. Most existing methods are designed for specific measures.
- *Efficient*: It uses a simple local search strategy that needs neither preprocessing nor iterating over the entire graph. Experimental results show that it is orders of magnitude faster than alternatives.

The key idea behind FLoS is that we can develop upper and lower bounds on the proximity of the nodes near the query node. These bounds can be dynamically updated when a larger portion of the graph is explored and will finally converge to the exact proximity value. The top-k nodes can be identified once the differences between their upper and lower bounds are small enough to distinguish them from the remaining nodes.

The theoretical basis of FLoS relies on the no local optimum property of proximity measures. That is, given a query node q, for any node i ($i \neq q$) in the graph, i always has a neighbor that is closer to q than i is. We show that many measures have no local optimum. This property ensures that the proximity of unvisited nodes is bounded by the maximum proximity (or minimum proximity for some measures) in the boundary of the visited nodes. It can be utilized to find the top-k nodes without exploring the entire graph under the assumption that the exact proximity can be computed based on local information. However, for most measures, the exact proximity cannot be computed without searching the entire graph. To tackle this challenge, we introduce several simple operations to modify transition probabilities, which enable developing upper and lower bounds on the proximity of visited nodes. The developed upper (lower) bounds monotonically decrease (increase) when more nodes are visited. We further study the relationship among different measures and show that FLoS can also be applied to measures having local optimum. Extensive experimental results show that, for a variety of measures, FLoS can dramatically improve the efficiency compared to the state-of-the-art methods.

2.2 Related Work

Various random walk based proximity measures have been proposed recently [25]. Examples include truncated or discounted hitting time [97, 99, 98], penalized hitting probability [40, 128], random walk with restart [112, 98], effective importance (degree normalized random walk with restart) [14], RoundTripRank [31], and absorption probability [119]. The Katz score [58] captures multiple paths between two nodes and is closely related to the random walk based proximity measures [61].

The basic approach for proximity query is to use the power iteration method [95]. An improved iteration method designed for random walk with restart decomposes the proximity into random walk probabilities of different length [35]. It tries to reduce the

number of iterations by estimating the proximity based on the information collected so far. The iteration method can also be improved by the prioritized execution of the iterative computation, where the node with the largest residual proximity value is updated first [61]. Another approach precomputes the information needed for proximity estimation during the query process [112, 34, 36]. However, this step is time consuming and becomes infeasible when the graph is large or constantly changing. Graph embedding method embeds nodes into geometric space so that node proximities can be preserved as much as possible [129]. The embedding step is also time-consuming. Moreover, the proximities in the new space are not exactly the same as the ones in the original graph.

Based on the intuition that nodes near the query node tend to have high proximity, local search methods try to visit a small number of nodes to approximate the proximities. Best-first [128] and depth-first [87] search strategies simply extract a fixed number of nodes near the query node. An approximate local search algorithm is proposed for truncated hitting time [97]. The key idea is to develop upper and lower bounds that can be used to approximate the proximities of local nodes. A similar local search algorithm is developed for personalized PageRank and degree normalized personalized PageRank [98]. The push style method is first developed in [11] for random walk with restart, and later improved by [42, 19] for the top-k proximity query problem. Starting from the query node, the push style method propagates the proximity value to the nodes in the neighborhood of the query node, and obtains approximate proximity values for them. This basic idea has been adapted to compute the top-k nodes for effective importance [14], RoundTripRank [31], and Katz score [30]. Most of the existing local search methods cannot guarantee the exactness. Moreover, all of them are designed for specific proximity measures. It is unclear whether the local search methods can be generalized to other measures.

Table 2.1: Main symbols

Symbols	Definitions
$G(V, E)$	undirected graph G with node set V and edge set E
N_i	neighbors of node i
$w_{i,j}$	weight of edge (i, j)
w_i	degree of node i, $w_i = \Sigma_{j \in N_i} w_{i,j}$
q	query node
$\mathbf{e}$	$n \times 1$ vector with $\mathbf{e}_q = 1$ and $\mathbf{e}_i = 0$ if $i \neq q$
k	number of returned nodes
S	a set of nodes
$\bar{S}$	complement of S : $\bar{S} = V \setminus S$
δS	boundary of S : $\{i \in S \mid \exists j \in N_i \cap \bar{S}\}$
$\delta \bar{S}$	boundary of $\bar{S}$: $\{i \in \bar{S} \mid \exists j \in N_i \cap S\}$
$\mathbf{r}$	$n \times 1$ vector, $\mathbf{r}_i$ is the proximity of node i w.r.t. query
$\overline{\mathbf{r}}$	upper bound of $\mathbf{r}$: $\overline{\mathbf{r}}_i \geq \mathbf{r}_i$, $\forall i \in S$
$\underline{\mathbf{r}}$	lower bound of $\mathbf{r}$: $\underline{\mathbf{r}}_i \leq \mathbf{r}_i$, $\forall i \in S$
$p_{i,j}$	transition probability from node i to j
$\mathbf{P}$	transition prob. matrix : $\mathbf{P}_{q,j} = 0; \mathbf{P}_{i,j} = p_{i,j}$ if $i \neq q$
d, $\mathbf{r}_\mathrm{d}$	a dummy node d with constant proximity value $\mathbf{r}_\mathrm{d}$
c	decay factor in PHP, DHT, RWR, EI, or RT
κ	decay factor in KZ
$\mathbf{I}$	identity matrix with $\mathbf{I}_{i,i} = 1$ and $\mathbf{I}_{i,j} = 0, \forall i \neq j$
$\mathbf{W}$	adjacency matrix with $\mathbf{W}_{i,j} = w_{i,j}$
$\mathbf{R}$	proximity matrix, $\mathbf{R}_{i,j}$: proximity between i and j
$\mathbf{D}$	diagonal matrix with $\mathbf{D}_{i,i} = w_i$
$\mathbf{\Lambda}$	diagonal matrix with $\mathbf{\Lambda}_{i,i} = \lambda_i$ (λ_i is a constant)
$u \rightsquigarrow v$	node u can reach node v in the transition graph
$u \not\rightsquigarrow v$	node u cannot reach node v in the transition graph
$u \rightsquigarrow S$	node u can reach at least one node in S
$u \not\rightsquigarrow S$	node u cannot reach any node in S

2.3 No Local Optimum Property

In this section, we first introduce the basic concept of no local optimum property of proximity measures and discuss how it can be used to bound the proximity of the unvisited nodes. Then we study whether commonly used measures have no local optimum and discuss the relationship between them. Table 2.1 lists the main symbols and their definitions.

2.3.1 Theoretical Basis

Note that for some measures, such as penalized hitting probability, random walk with restart, effective importance, RoundTripRank, Katz score and absorption probability, the larger the proximity the closer the nodes. In this case, no local optimum means no local maximum. For other measures, such as discounted/truncated hitting time, the smaller the proximity the closer the nodes. In this case, no local optimum means no local minimum.

Given an undirected and edge weighted graph $G = (V, E)$ and a query node $q \in V$, let $\mathbf{r}$ be the proximity vector with $\mathbf{r}_i$ representing the proximity of node $i \in V$ with respect to the query node q.

Definition 1. [No Local Maximum] *A proximity measure has no local maximum if for any node $i \neq q$, there exists a neighbor node j of i (i.e., $j \in N_i$), such that $\mathbf{r}_j > \mathbf{r}_i$.*

Definition 2. [No Local Minimum] *A proximity measure has no local minimum if for any node $i \neq q$, there exists a neighbor node j of i (i.e., $j \in N_i$), such that $\mathbf{r}_j < \mathbf{r}_i$.*

We say that a proximity measure has no local optimum if it has no local maximum or minimum. In Section 2.3.2, we will examine whether the commonly used proximity measures have no local optimum. Unless otherwise mentioned, in the next, we assume that the larger the proximity the closer the nodes, and focus on the no local maximum property. All conclusions can also be applied to the proximity measures with no local minimum.

Let S be a set of nodes, and $\bar{S} = V \setminus S$ be the remaining nodes. We use $\delta S = \{i \in S \,|\, \exists j \in N_i \cap \bar{S}\}$ to denote the *boundary* of S, and $\delta \bar{S} = \{i \in \bar{S} \,|\, \exists j \in N_i \cap S\}$ to denote the boundary of $\bar{S}$.

Figure 2.1(a) shows an undirected graph with 8 nodes. Suppose that the node set $S = \{1, 2, 3, 4\}$, then we have $\bar{S} = \{5, 6, 7, 8\}$, $\delta S = \{3, 4\}$, and $\delta \bar{S} = \{5, 6, 7\}$.

Theorem 1. *Let S be a node set containing the query node, and u be the node with the*

Algorithm 1 The basic top-k local search algorithm

Input: $G(V, E)$, query node q, proximity vector $\mathbf{r}$, number k
Output: Top-k node set K

1: $S \leftarrow \{q\}$; $K \leftarrow \{\}$;
2: **while** $|K| < k + 1$ **do**
3: $\quad u \leftarrow \text{argmax}_{i \in S \setminus K} \mathbf{r}_i$;
4: $\quad K \leftarrow K \cup \{u\}$; $S \leftarrow S \cup N_u$;
5: **return** $K \setminus \{q\}$;

largest proximity in δS. If a proximity has no local maximum, we have that $\mathbf{r}_u > \mathbf{r}_j$ $(\forall j \in \bar{S})$.

Proof. Suppose otherwise. We have that $\exists j \in \bar{S}$, such that $\mathbf{r}_u \leq \mathbf{r}_j$. Now suppose that node v is the node with the largest proximity in $\bar{S}$. We have that $\forall i \in \bar{S} \cup \delta S$, $\mathbf{r}_v \geq \mathbf{r}_i$. The neighbors of node v must exist in $\bar{S} \cup \delta S$, i.e., $N_v \subseteq \bar{S} \cup \delta S$. Therefore, we have $\mathbf{r}_v \geq \mathbf{r}_i$ $(\forall i \in N_v)$,which means node v is a local maximum. This contradicts the assumption. □

Based on Theorem 1, assuming that we already have the exact proximity vector $\mathbf{r}$, we can design a simple local search strategy as shown in Algorithm 1 to find the top-k nodes. It starts from the query node q and uses K and S to store the top-k nodes and visited nodes respectively. In each iteration, the algorithm finds the node u that has the largest proximity in $S \setminus K$ and expands S to the neighbors of node u. Since $\delta S \subseteq S \setminus K$, the maximum proximity value in $S \setminus K$ must be no less than the maximum proximity value in δS, and greater than the maximum proximity value in the unvisited nodes $\bar{S}$ based on Theorem 1. Thus, K contains the top-k nodes. The algorithm continues until $|K| = k + 1$.

Let h be the average number of neighbors of a node. In each iteration t, on average h nodes are added to S. This takes $O(h \log ht)$ time for a sorted list. The overall complexity of Algorithm 1 is thus $O(\sum_{t=1}^{k} h \log ht) = O(hk \log hk)$.

Table 2.2: No local optimum property of some measures

Proximity measures	Abbr.	Ref.	Property
Penalized hitting probability	PHP	[40]	No local maximum
Effective importance	EI	[14]	No local maximum
Discounted hitting time	DHT	[98]	No local minimum
Truncated hitting time	THT	[97]	No local minimum (within L hops)
Random walk with restart	RWR	[112]	Local maximum
RoundTripRank	RT	[31]	Local maximum
Katz score	KZ	[58]	Local maximum
Absorption probability	AP	[119]	Local maximum

(a) original graph (b) transition graph (PHP, $q = 1$)

Figure 2.1: An example graph and its transition graph

2.3.2 Measures With and Without Local Optimum

Table 2.2 summarizes whether the commonly used proximity measures have no local optimum property. Next, we use penalized hitting probability (PHP) [40, 128] as an example to illustrate that it has no local maximum. We use w_i to denote the degree of node i, and $w_{i,j}$ to denote the edge weight between i and j. The transition probability from i to j is thus $p_{i,j} = w_{i,j}/w_i$.

Suppose the undirected graph in Figure 2.1(a) has unit edge weight. Node 3 has degree 3, thus its transition probability to node 4 is $p_{3,4} = 1/3$. Based on these transition probabilities, we can construct the corresponding transition graph as shown in Figure 2.1(b). In the transition graph, each directed edge and the number on the edge represent the transition probability from one node to the other.

PHP penalizes the random walk for each additional step. Given a query node q, let $\mathbf{r}$ denote the PHP proximity vector, with $\mathbf{r}_i$ representing the proximity value of node i. PHP can be defined recursively as

$$\mathbf{r}_i = \begin{cases} 1, & \text{if } i = q, \\ c \sum_{j \in N_i} p_{i,j} \mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where c $(0 < c < 1)$ is the decay factor in the random walk process. In [40], $c = e^{-1}$ is used as the decay factor. The query node q has constant proximity value 1, and there is no transition probability going out of the query node. For example, there is no outgoing edges from the query node 1 in the transition graph in Figure 2.1(b).

Let $\mathbf{P}$ be the transition probability matrix with

$$\mathbf{P}_{i,j} = \begin{cases} 0, & \text{if } i = q, \\ p_{i,j}, & \text{if } i \neq q. \end{cases}$$

Then the above recursive definition can be expressed as the following matrix form

$$\mathbf{r} = c\mathbf{P}\mathbf{r} + \mathbf{e},$$

where $\mathbf{e}_i = 1$ if $i = q$, and $\mathbf{e}_i = 0$ if $i \neq q$.

Lemma 1. PHP *has no local maximum.*

Proof. Suppose that node i is a local maximum. We have $\mathbf{r}_i = c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j \leq c\sum_{j\in N_i} p_{i,j}\mathbf{r}_i = c\mathbf{r}_i < \mathbf{r}_i$. We get a contradiction that $\mathbf{r}_i < \mathbf{r}_i$. □

Effective importance (EI) [14] is the degree normalized version of random walk with restart [112], which can be defined as

$$\mathbf{r}_i = \begin{cases} c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j + \frac{1-c}{w_i}, & \text{if } i = q, \\ c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where c $(0 < c < 1)$ is a constant decay factor.

Lemma 2. EI *has no local maximum.*

Proof. Suppose that node i is a local maximum. We have $\mathbf{r}_i = c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j \leq c\sum_{j\in N_i} p_{i,j}\mathbf{r}_i = c\mathbf{r}_i < \mathbf{r}_i$. Thus we get a contradiction that $\mathbf{r}_i < \mathbf{r}_i$. □

Discounted hitting time (DHT) [98] is a variant of hitting time [87], which can be defined as

$$\mathbf{r}_i = \begin{cases} 0, & \text{if } i = q, \\ 1 + c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where c $(0 < c < 1)$ is a constant decay factor.

Lemma 3. DHT *has no local minimum.*

Proof. The maximum discounted hitting time that a node could have is $\frac{1}{1-c}$, when it cannot reach q. For connected graph, we have $\mathbf{r}_i < \frac{1}{1-c}$. Now suppose that node i is a local minimum. We have that $\mathbf{r}_i = 1 + c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j \geq 1 + c\sum_{j\in N_i} p_{i,j}\mathbf{r}_i = 1 + c\mathbf{r}_i$. Thus $\mathbf{r}_i \geq \frac{1}{1-c}$, which contradicts the fact that $\mathbf{r}_i < \frac{1}{1-c}$. □

Truncated hitting time (THT) [97] is another variant of hitting time [87]. The L-truncated hitting time only considers paths of length less than L. Let $\mathbf{r}_i^{\mathsf{L}}$ and $\mathbf{r}_i^{\mathsf{L}-1}$ be the L- and $(\mathsf{L}-1)$-truncated hitting time of node i respectively when the query is q. The L-truncated hitting time can be defined as

$$\mathbf{r}_i^{\mathsf{L}} = \begin{cases} 0, & \text{if } i = q, \\ 1 + \sum_{j\in N_i} p_{i,j}\mathbf{r}_j^{\mathsf{L}-1}, & \text{if } i \neq q. \end{cases}$$

If a node is no less than L hops away from the query node, its proximity is set to L. If node i is within L hops away from the query node, we have that $\mathbf{r}_i^{\mathsf{L}} < \mathsf{L}$.

Lemma 4. THT *has no local minimum for the nodes within* L *hops from the query node.*

Proof. Suppose that node i is within L hops from q. We can prove that $\forall j \in N_i$, $1 + \mathbf{r}_j^{\mathsf{L}-1} \geq \mathbf{r}_j^{\mathsf{L}}$, and there exists at least one node $j \in N_i$ such that $1 + \mathbf{r}_j^{\mathsf{L}-1} > \mathbf{r}_j^{\mathsf{L}}$ by mathematical induction on L. Now suppose that i is a local minimum, i.e., $\mathbf{r}_j^{\mathsf{L}} \geq \mathbf{r}_i^{\mathsf{L}}$ $(\forall j \in N_i)$. We have that $\mathbf{r}_i^{\mathsf{L}} = 1 + \sum_{j\in N_i} p_{i,j}\mathbf{r}_j^{\mathsf{L}-1} = \sum_{j\in N_i} p_{i,j}(1 + \mathbf{r}_j^{\mathsf{L}-1}) > \sum_{j\in N_i} p_{i,j}\mathbf{r}_j^{\mathsf{L}} \geq \sum_{j\in N_i} p_{i,j}\mathbf{r}_i^{\mathsf{L}} = \mathbf{r}_i^{\mathsf{L}}$. We get a contradiction that $\mathbf{r}_i^{\mathsf{L}} > \mathbf{r}_i^{\mathsf{L}}$. □

Some proximity measures have inherent relationship. The following theorem says that PHP, EI, and DHT are equivalent in terms of ranking.

Theorem 2. PHP*,* EI*, and* DHT *give the same ranking results.*

Proof. First, we show that PHP and EI are equivalent. Suppose the decay factors in PHP and EI are both set to be c. Then PHP and EI have the same recursive definition for any node $i \neq q$, and we have $\frac{\mathrm{EI}(i)}{\mathrm{PHP}(i)} = \frac{\mathrm{EI}(q)}{\mathrm{PHP}(q)}$, which is a constant when q is fixed. Thus $\mathrm{PHP}(i)$ is a linear function of $\mathrm{EI}(i)$.

Next, we show that PHP and DHT are equivalent. Suppose the decay factors in PHP and DHT are both set to be c. The relationship between PHP and DHT is $\text{PHP}(i) = 1-(1-c)\cdot\text{DHT}(i)$, which can be proved by substituting it in the recursive definition of PHP. Thus $\text{PHP}(i)$ is a linear function of $\text{DHT}(i)$. □

The proofs of other proximity measures in Table 2.2 can be found in Section 2.6, where we will first introduce the proximity measures, and then prove whether they have local optimum.

From the discussion in this section, we know that if a proximity measure has no local optimum, we can apply local search described in Algorithm 1 to find the top-k nodes under the assumption that the proximity values of all the nodes are given. However, without exploring the entire graph, the exact values of all the nodes are unknown. To tackle this challenge, we can develop lower and upper bounds on the proximity values of the visited nodes. When more nodes are visited, the lower and upper bounds become tighter and eventually converge to the exact proximity value.

Next, we will use PHP, which has no local optimum, as a concrete example to explain how to derive the lower and upper bounds. The strategy can be applied to other proximity measures with no local optimum. How to apply the local search strategy to the measures having local optimum will be discussed in Section 2.6.

2.4 Bounding the Proximity

To develop the lower and upper bounds, we introduce three basic operations, i.e., deletion, restoration, and destination change of transition probability. We show how proximities change if we modify transition probabilities according to these operations. Then we discuss how to derive lower and upper bounds based on them.

(a) deletion and restoration (b) destination change

Figure 2.2: Basic operations on transition probability

2.4.1 Modifying Transition Probability

We first introduce the operation of deleting a transition probability. Figure 2.2(a) shows an example, in which the original graph is on the top, with node 1 being the query node and transition probabilities $p_{2,1} = p_{2,3} = 0.5$ and $p_{3,2} = 1$. After deleting $p_{2,3}$, the resulting graph is shown at the bottom. Note that deleting a transition probability is different from deleting an edge. Deleting an edge will change the transition probabilities on the remaining graph, while deleting a transition probability will not.

Theorem 3. *Deleting a transition probability will not increase the proximity of any node.*

Proof. Let $\mathbf{P}$ be the original transition probability matrix of PHP. Deleting $\mathbf{P}_{i,j}$ from $\mathbf{P}$ is the same as setting $\mathbf{P}_{i,j}$ to 0. Let $\mathbf{P}'$ represent the resulting matrix. The PHP proximity vector $\mathbf{r}$ is computed based on $\mathbf{P}$, and $\mathbf{r}'$ is based on $\mathbf{P}'$.

Let $\Delta\mathbf{r} = \mathbf{r} - \mathbf{r}'$, and $\Delta\mathbf{P} = \mathbf{P} - \mathbf{P}'$. Note that $\Delta\mathbf{P}$ has only one non-zero element $\Delta\mathbf{P}_{i,j} = \mathbf{P}_{i,j}$. We have $\Delta\mathbf{r} = c(\mathbf{P}' + \Delta\mathbf{P})\mathbf{r} - c\mathbf{P}'\mathbf{r}' = c\mathbf{P}'\Delta\mathbf{r} + c\Delta\mathbf{P}\mathbf{r} = c\mathbf{P}'\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}'$ is a vector with the only non-zero element $\mathbf{e}'_i = c\mathbf{P}_{i,j}\mathbf{r}_j$. The solution of the previous equation is $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}')^{-1}\mathbf{e}'$. The elements of $c\mathbf{P}'$ are non-negative and $||c\mathbf{P}'||_\infty < 1$. We have that $(\mathbf{I} - c\mathbf{P}')^{-1} = \sum_{l=0}^{\infty}(c\mathbf{P}')^l \geq 0$ [88]. Thus the solution $\Delta\mathbf{r}$ must be non-negative. This completes the proof. □

Continue with the example in Figure 2.2(a). Suppose that the decay factor $c = 0.5$. The original PHP proximity vector is $\mathbf{r} = [1, 2/7, 1/7]$. After deleting $p_{2,3}$, the new proximity vector is $\mathbf{r}' = [1, 1/4, 1/8]$.

It can be shown in a similar way that if we restore the transition probability as shown in Figure 2.2(a), the proximities will not decrease.

Theorem 4. *Restoring a deleted transition probability will not decrease the proximity of any node.*

Proof. Omitted. □

Figure 2.2(b) shows an example in which we change the destination of the original transition probability $p_{3,2}$ from node 2 to 1. Thus, $p_{3,1}$ is set to $p_{3,2}$, and $p_{3,2}$ is set to 0.

Theorem 5. *Changing the destination of transition pro-bability $p_{i,j}$ from node j to node u with $\mathbf{r}_u \geq \mathbf{r}_j$ ($\mathbf{r}_u \leq \mathbf{r}_j$) will not decrease (increase) the proximity of any node.*

Proof. Let $\mathbf{P}$ be the original transition probability matrix of PHP. Changing the destination of transition probability $p_{i,j}$ from node j to node u is the same as adding $\mathbf{P}_{i,j}$ to $\mathbf{P}_{i,u}$ and setting $\mathbf{P}_{i,j}$ to 0. Let $\mathbf{P}'$ represent the resulting matrix. PHP proximity $\mathbf{r}$ is computed based on $\mathbf{P}$, and $\mathbf{r}'$ is based on $\mathbf{P}'$.

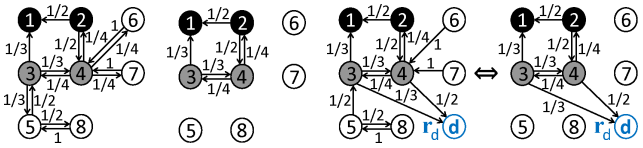
Let $\Delta\mathbf{r} = \mathbf{r}' - \mathbf{r}$, and $\Delta\mathbf{P} = \mathbf{P} - \mathbf{P}'$. Note that $\Delta\mathbf{P}$ has only two non-zero elements $\Delta\mathbf{P}_{i,j} = \mathbf{P}_{i,j}$ and $\Delta\mathbf{P}_{i,u} = -\mathbf{P}_{i,j}$. We have that $\Delta\mathbf{r} = c\mathbf{P}'\mathbf{r}' - c(\mathbf{P}' + \Delta\mathbf{P})\mathbf{r} = c\mathbf{P}'\Delta\mathbf{r} - c\Delta\mathbf{P}\mathbf{r} = c\mathbf{P}'\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}'$ is a vector with the only non-zero element $\mathbf{e}'_i = c\mathbf{P}_{i,j}(\mathbf{r}_u - \mathbf{r}_j)$. If $\mathbf{r}_u \geq \mathbf{r}_j$, $\Delta\mathbf{r}$ must be non-negative. Otherwise, it is non-positive. This completes the proof. □

Let us continue with the example in Figure 2.2(b), where node 1 is the query node. After we change the destination of $p_{3,2}$ from node 2 to 1, the proximity values should be non-decreasing. With a decay factor $c = 0.5$, the proximity vectors before and after the destination change are $\mathbf{r} = [1, 2/7, 1/7]$ and $\mathbf{r}' = [1, 3/8, 1/2]$ respectively.

The proofs for other measures having no local optimum are similar and omitted here.

2.4.2 Lower Bound

Recall that S, $\bar{S}$, δS and $\delta\bar{S}$ represent the set of visited nodes, the set of unvisited nodes, the boundary of S, and the boundary of $\bar{S}$, respectively. From Theorem 3,



(a) original $\mathbf{r}$ (b) LB $\underline{\mathbf{r}}$ (c) upper bound $\overline{\mathbf{r}}$

Figure 2.3: Lower and upper bounds based on basic operations

if we delete all transition probabilities $\{p_{i,j} : i \text{ or } j \in \bar{S}\}$ in the original graph, the proximity value of any node u computed using the resulting graph, $\underline{\mathbf{r}}_u$, will be less than or equal to its original value $\mathbf{r}_u$, i.e., $\underline{\mathbf{r}}_u \leq \mathbf{r}_u$. Thus, $\underline{\mathbf{r}}$ can be used as the lower bound of $\mathbf{r}$.

Let us take the undirected graph in Figure 2.1(a) for example. Its transition graph is shown in Figure 2.3(a), with node 1 being the query node and transition probabilities shown on the edges. Suppose that the current set of visited nodes $S = \{1, 2, 3, 4\}$. Thus $\bar{S} = \{5, 6, 7, 8\}$, $\delta S = \{3, 4\}$, and $\delta\bar{S} = \{5, 6, 7\}$. The nodes in S but not in δS are black. The nodes in δS are gray. The nodes in $\bar{S}$ are white.

Figure 2.3(b) shows the resulting transition graph after deleting all transition probabilities $\{p_{i,j} : i \text{ or } j \in \bar{S}\}$. The proximity values $\underline{\mathbf{r}}$ computed based on Figure 2.3(b) will lower bound the original proximity values $\mathbf{r}$ for the nodes in S.

2.4.3 Upper Bound

From Theorem 5, if we change the destination of the transition probabilities $\{p_{i,j} : i \in \delta S, j \in \delta\bar{S}\}$ to a newly added dummy node d with a constant proximity value $\mathbf{r}_\mathrm{d}$ and $\mathbf{r}_\mathrm{d} > \mathbf{r}_v$ $(\forall v \in \bar{S})$, the proximity value of any node u computed using the resulting graph, $\overline{\mathbf{r}}_u$, will be greater than or equal to its original value $\mathbf{r}_u$, i.e., $\overline{\mathbf{r}}_u \geq \mathbf{r}_u$. Thus, $\overline{\mathbf{r}}$ can be used as the upper bound of $\mathbf{r}$.

Continue with the example in Figure 2.3(a). In the original graph, $\delta S = \{3, 4\}$, $\delta\bar{S} = \{5, 6, 7\}$, $p_{3,5} = 1/3$, $p_{4,6} = p_{4,7} = 1/4$. The left figure in Figure 2.3(c) shows the resulting graph after we change all the transition probabilities going from δS to $\delta\bar{S}$ to the newly added dummy node d. Specifically, $p_{3,\mathrm{d}} = 1/3$ and $p_{4,\mathrm{d}} = p_{4,6} + p_{4,7} = 1/2$. Note that after changing the destination to d, there will be no transition probability

Algorithm 2 Fast top-k local search (FLoS)

Input: $G(V, E)$, query q, number k, decay factor c, value τ
Output: Top-k node set K

```
1: S^0 ← {q}; δS^0 ← {q}; r_q^0 ← 1; r̄_q^0 ← 1; P_{q,q}^0 ← 0;
2: bStop ← false; t ← 0;
3: while bStop = false do
4:     t ← t + 1;
5:     LocalExpansion();
6:     UpdateLowerBound();
7:     UpdateUpperBound();
8:     CheckTerminationCriterion();
9: return K;
```

from any node in S to any node in $\bar{S}$. Therefore, for the nodes in S, the upper bounds can be computed by the subgraph induced by the nodes in S and the dummy node d, as shown on the right in Figure 2.3(c).

Note that to get the upper bound, we need to add a dummy node d with constant proximity value $\mathbf{r}_{\mathrm{d}} \geq \mathbf{r}_v$ ($\forall v \in \bar{S}$). In the next section, we will present the fast local search algorithm, FLoS, and discuss how to choose the value $\mathbf{r}_{\mathrm{d}}$.

2.5 Fast Local Search

In this section, we present the FLoS algorithm, which utilizes the bounds developed in Section 2.4 to enable the local search. We show that the bounds can only change monotonically when more nodes are visited. A theoretical analysis on the number of visited nodes is also provided.

2.5.1 The FLoS Algorithm

Algorithm 2 describes the FLoS algorithm. It has four main steps. In the first step, the algorithm expands locally to the neighbors of a selected visited node. In the second and third steps, it updates the lower and upper bounds of the visited nodes. Finally, it checks whether the top-k nodes are identified.

The local expansion step is shown in Algorithm 3. It picks the node in δS having

Algorithm 3 LocalExpansion()

1: $u \leftarrow \text{argmax}_{i \in \delta S^{t-1}} (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;
2: $S^t \leftarrow S^{t-1} \cup N_u$;
3: Update δS^t;

Algorithm 4 UpdateLowerBound()

1: $\mathbf{P}_{i,j}^t \leftarrow w_{i,j} / \sum_{v \in N_i} w_{i,v}$, if node i or j are newly added;
2: $\mathbf{P}_{q,j}^t \leftarrow 0$, if node j is newly added;
3: $\mathbf{P}_{i,j}^t \leftarrow \mathbf{P}_{i,j}^{t-1}$, if nodes i and j exist in the last iteration;
4: $\underline{\mathbf{r}}_i^t \leftarrow 0$, if node i is newly added;
5: $\underline{\mathbf{r}}_i^t \leftarrow \underline{\mathbf{r}}_i^{t-1}$, if node i exists in the last iteration;
6: $\mathbf{e}_i \leftarrow 1$, if $i = q$; $\mathbf{e}_i \leftarrow 0$, otherwise;
7: $\underline{\mathbf{r}}^t \leftarrow$ IterativeMethod($\mathbf{P}^t$, $\underline{\mathbf{r}}^t$, $\mathbf{e}$, c, τ);

Algorithm 5 UpdateUpperBound()

1: Extend $\mathbf{P}^t$ with 1 column and 1 row for the dummy node d;
2: $\mathbf{P}_{i,\mathrm{d}}^t \leftarrow 1 - \sum_{j \in N_i} \mathbf{P}_{i,j}^t$, if node $i \in \delta S^t$;
3: $\mathbf{P}_{i,\mathrm{d}}^t \leftarrow 0$, if $i \in S^t \setminus \delta S^t$;
4: $\mathbf{P}_{\mathrm{d},i}^t \leftarrow 0$, for any node i;
5: $\overline{\mathbf{r}}_i^t \leftarrow 1$, if node i is newly added;
6: $\overline{\mathbf{r}}_i^t \leftarrow \overline{\mathbf{r}}_i^{t-1}$, if node i exists in the last iteration;
7: $\overline{\mathbf{r}}_{\mathrm{d}}^t \leftarrow \mathbf{r}_{\mathrm{d}}^t \leftarrow \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}_i^{t-1}$; `// dummy node value`
8: Extend $\mathbf{e}$ with 1 new element $\mathbf{e}_{\mathrm{d}} = \mathbf{r}_{\mathrm{d}}^t$ for the node d;
9: $\overline{\mathbf{r}}^t \leftarrow$ IterativeMethod($\mathbf{P}^t$, $\overline{\mathbf{r}}^t$, $\mathbf{e}$, c, τ);

the largest average lower and upper bound values, and expands to the neighbors of this node. S and δS are then updated accordingly. We take the average of the lower and upper bound value as an approximation of the exact proximity value. Expanding the node with the largest value is the best-first search strategy.

Algorithm 4 shows how to update the lower bound by using PHP as an example. It can also be applied to other measures with no local optimum. To update the bounds, we first construct the transition matrix $\mathbf{P}$. Note that the size of $\mathbf{P}$ is $|S| \times |S|$ instead of $|V| \times |V|$. We do not allocate memory for the full matrix $\mathbf{P}$, but only use adjacent list to represent it. The lower bound vector $\underline{\mathbf{r}}$ is initiated the same as in the previous iteration or 0 for the newly added nodes. We then use the standard iterative method, which is shown in Algorithm 7, to solve the linear equation $\underline{\mathbf{r}} = c\mathbf{P}\underline{\mathbf{r}} + \mathbf{e}$ and update $\underline{\mathbf{r}}$.

Algorithm 6 CheckTerminationCriterion()

1: **if** $|S^t \setminus \delta S^t \setminus \{q\}| \geq k$ **then**
2: $\quad K \leftarrow k$ nodes in $S^t \setminus \delta S^t \setminus \{q\}$ with largest $\underline{\mathbf{r}}^t$;
3: $\quad$ **if** $\min_{i \in K} \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \setminus K \setminus \{q\}} \overline{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true;

Algorithm 7 IterativeMethod()

Input: matrix $\mathbf{P}$, vector $\mathbf{r}_{in}$, vector $\mathbf{e}$, decay factor c, value τ
Output: proximity vector $\mathbf{r}_{out}$

1: $\mathbf{r}^0 \leftarrow \mathbf{r}_{in}$; $l \leftarrow 0$;
2: **repeat** $l \leftarrow l + 1$; $\mathbf{r}^l \leftarrow c\mathbf{P}\mathbf{r}^{l-1} + \mathbf{e}$; **until** $||\mathbf{r}^l - \mathbf{r}^{l-1}|| < \tau$;
3: **return** $\mathbf{r}^l$;

Algorithm 5 shows how to update the upper bound. The transition matrix $\mathbf{P}$ has one additional dummy node d and its related transition probabilities $\{p_{i,\mathrm{d}} : i \in \delta S\}$. The values in $\overline{\mathbf{r}}$ are initiated the same as the values in the previous iteration or 1 for the newly added nodes. The smaller the value of $\mathbf{r}_\mathrm{d}$, the tighter the upper bounds. On the other hand, we also need to make sure that the value $\mathbf{r}_\mathrm{d}$ is larger than the exact proximity value of any unvisited node. Therefore in line 7, we use the largest upper bound value in the boundary of the last iteration as $\mathbf{r}_\mathrm{d}^t$. We have $\mathbf{r}_\mathrm{d}^t = \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}_i^{t-1} \geq \max_{i \in \delta S^{t-1}} \mathbf{r}_i > \mathbf{r}_j (\forall j \in \bar{S}^{t-1})$, where the last inequality is based on Theorem 1. Thus $\mathbf{r}_\mathrm{d}^t > \mathbf{r}_j (\forall j \in \bar{S}^t)$, since $\bar{S}^t \subseteq \bar{S}^{t-1}$. This guarantees the correctness of the upper bound according to Theorem 5. Finally, we solve the linear equation $\overline{\mathbf{r}} = c\mathbf{P}\overline{\mathbf{r}} + \mathbf{e}$ to update $\overline{\mathbf{r}}$ by using Algorithm 7. Algorithm 7 is very efficient in practice. This is because when the initial values of the iterative method is close to the exact solution, the algorithm will converge very fast. In our method, between two adjacent iterations, the proximity values of the visited nodes are very close. Therefore, updating the proximity is very efficient.

Algorithm 6 shows the termination criterion. We select the k nodes in $S \setminus \delta S \setminus \{q\}$ with largest $\underline{\mathbf{r}}$ values. If the minimum lower bound of the selected nodes is greater than or equal to the maximum upper bound of the remaining visited nodes, the selected nodes will be the top-k nodes in the entire graph. This is because the maximum proximity of unvisited nodes is bounded by the maximum proximity in δS, which is

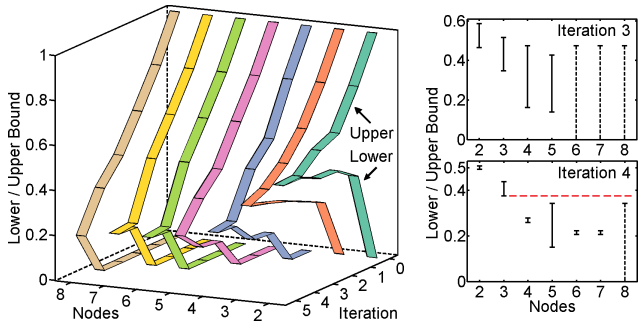


Figure 2.4: Lower and upper bounds in different iterations (PHP: $q = 1$, $c = 0.8$)

Table 2.3: Newly visited nodes in each iteration

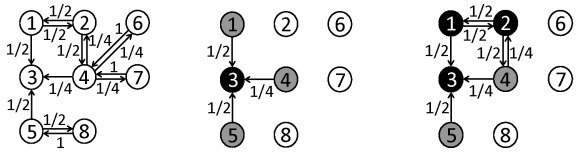
Iteration	1	2	3	4	5
Newly visited nodes	{2, 3}	{4}	{5}	{6, 7}	{8}

in turn bounded by the maximum upper bound in δS.

Figure 2.4 shows the lower and upper bounds at different iterations using the example graph in Figure 2.1(a). One iteration represents one local expansion process. The newly visited nodes in each iteration are listed in Table 2.3.

The left figure in Figure 2.4 shows how the lower and upper bounds change through local expansions. Query node 1 has constant proximity value 1.0 thus is not shown. It can be seen that the bounds monotonically change and eventually converge to the exact proximity value when all the nodes are visited. The monotonicity of the bounds is proved theoretically in next two sections.

The right figure in Figure 2.4 shows the lower and upper bounds in iteration 3 (at the top) and 4 (at the bottom). The interval from the lower to upper bounds is indicated by the vertical line segment. The interval of the unvisited node is indicated by the dashed line segment. In iteration 3, nodes $\{6, 7, 8\}$ are unvisited, and their upper bound is the upper bound for node 4, which is the maximum upper bound for the boundary nodes $\{4, 5\}$. In iteration 4, the bounds become tighter, and the minimum lower bound of nodes $\{2, 3\}$ is larger than the maximum upper bound of the remaining nodes $\{4, 5, 6, 7, 8\}$, which is indicated by the horizontal red dashed line. Therefore, nodes $\{2, 3\}$ are guaranteed to be the top-2 nodes after iteration 4, even though node 8 is still unvisited.



(a) original (b) 1st iteration (c) 2nd iteration

Figure 2.5: Example transition graphs between two adjacent iterations for analyzing lower bound monotonicity with query node 3.

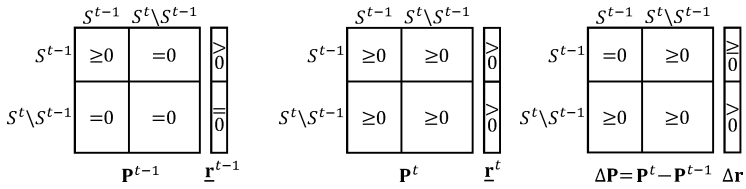
2.5.2 Monotonicity of the Lower Bound

We first consider the monotonicity of the lower bound. Let S^{t-1} and S^t represent the set of visited nodes in iterations $(t-1)$ and t respectively. In the next, we prove that the lower bound is monotonically non-decreasing when more nodes are visited, i.e., $\underline{\mathbf{r}}_i^t \geq \underline{\mathbf{r}}_i^{t-1} (i \in S^{t-1})$.

Given a directed transition graph, we say that a node u can *reach* a node v if there exists a sequence of adjacent nodes (i.e., a path) which starts from u and ends at v. For example, in the transition graph in Figure 2.5(a), node 1 can reach node 6, but node 5 cannot reach node 6. We use $u \rightsquigarrow v$ to denote that node u can reach v, and $u \not\rightsquigarrow v$ to denote that node u cannot reach v. We also use $u \rightsquigarrow S$ to denote that node u can reach at least one node in S, and $u \not\rightsquigarrow S$ to denote that node u cannot reach any node in S.

From iteration $(t-1)$ to t, we only restore some transition probabilities in $\{p_{i,j} : i$ or $j \in S^t \setminus S^{t-1}\}$. The following Theorem 6 says that if node i can reach at least one of the newly added nodes, the lower bound of node i is strictly increasing. If node i cannot reach any of the newly added nodes, the lower bound value of node i will not change during the iteration.

Figure 2.5 shows an example. Figure 2.5(a) shows the full transition graph when the query is node 3. Figures 2.5(b) and 2.5(c) show the transition graphs constructed in the first and second iteration of Algorithm 2 respectively. $S^1 = \{3, 1, 4, 5\}$, $S^2 = \{3, 1, 4, 5, 2\}$, and node 2 is the newly visited node in the second iteration. We can see that node 5 cannot reach node 2. Thus, the lower bound value $\underline{\mathbf{r}}_5$ is unchanged. The lower bound values of nodes $\{1, 4\}$ are strictly increasing.



(a) $(t-1)$-th iteration (b) t-th iteration (c) difference

Figure 2.6: Transition probability matrices between two adjacent iterations (lower bound)

Theorem 6. (Monotonicity of the lower bound) *For any node $i \in S^{t-1}$, we have that*

$$\begin{cases} \underline{\mathbf{r}}_i^t = \underline{\mathbf{r}}_i^{t-1}, & \text{if } i \nrightarrow S^t \setminus S^{t-1}, \\ \underline{\mathbf{r}}_i^t > \underline{\mathbf{r}}_i^{t-1}, & \text{if } i \rightsquigarrow S^t \setminus S^{t-1}, \end{cases}$$

where $\rightsquigarrow$ and $\nrightarrow$ represent the reachability in the transition graph at the t-th iteration.

Proof. Let $\mathbf{P}^{t-1}$ and $\mathbf{P}^t$ be the transition probability matrices at the $(t-1)$-th and t-th iterations respectively. The lower bound $\underline{\mathbf{r}}^{t-1}$ is computed based on $\mathbf{P}^{t-1}$, and $\underline{\mathbf{r}}^t$ is computed based on $\mathbf{P}^t$. Since the sizes of matrices $\mathbf{P}^{t-1}$ and $\mathbf{P}^t$ are $|S^{t-1}| \times |S^{t-1}|$ and $|S^t| \times |S^t|$ respectively, we extend $\mathbf{P}^{t-1}$ with $|S^t \setminus S^{t-1}|$ columns and $|S^t \setminus S^{t-1}|$ rows. The newly added elements are set to 0. After extending, $\mathbf{P}^{t-1}$ and $\mathbf{P}^t$ are of the same size. Similarly, we extend $\underline{\mathbf{r}}^{t-1}$ with $|S^t \setminus S^{t-1}|$ elements, which are also set to 0. Thus, $\underline{\mathbf{r}}^{t-1}$ and $\underline{\mathbf{r}}^t$ are of the same size $|S^t| \times 1$ after extending. For any node $i \in S^{t-1}$, we have $\underline{\mathbf{r}}_i^{t-1} > 0$. Figures 2.6(a) and 2.6(b) illustrate these matrices and vectors. We have the following two equations.

$$\begin{cases} \underline{\mathbf{r}}^{t-1} = c\mathbf{P}^{t-1}\underline{\mathbf{r}}^{t-1} + \mathbf{e}, \\ \underline{\mathbf{r}}^{t} \quad = c\mathbf{P}^{t}\underline{\mathbf{r}}^{t} \qquad + \mathbf{e}, \end{cases}$$

where $\mathbf{e}_q = 1$ and $\mathbf{e}_i = 0$ if $i \in S^t \setminus \{q\}$.

Let $\Delta\mathbf{r} = \underline{\mathbf{r}}^t - \underline{\mathbf{r}}^{t-1}$ and $\Delta\mathbf{P} = \mathbf{P}^t - \mathbf{P}^{t-1}$. Note that $\Delta\mathbf{P}_{i,j} = 0$ if $i, j \in S^{t-1}$ and $\Delta\mathbf{P}_{i,j} = \mathbf{P}^t_{i,j}$ otherwise. Figure 2.6(c) illustrates the matrix $\Delta\mathbf{P}$ and vector $\Delta\mathbf{r}$. We have that $\Delta\mathbf{r} = c\mathbf{P}^t\underline{\mathbf{r}}^t - c\mathbf{P}^{t-1}\underline{\mathbf{r}}^{t-1} = c\mathbf{P}^t\Delta\mathbf{r} + c\Delta\mathbf{P}\underline{\mathbf{r}}^{t-1} = c\mathbf{P}^t\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}' = c\Delta\mathbf{P}\underline{\mathbf{r}}^{t-1}$.

Note that $\Delta\mathbf{P}_{i,j} = 0$ if $i, j \in S^{t-1}$. We also have that $\underline{\mathbf{r}}_j^{t-1} = 0$ if $j \in S^t \setminus S^{t-1}$. Thus, $\mathbf{e}'_i = 0$ if $i \in S^{t-1}$. For any node $i \in S^t \setminus S^{t-1}$, it must connect to at least one node in S^{t-1}. Thus, for each node $i \in S^t \setminus S^{t-1}$, $\Delta\mathbf{P}_{i,j} > 0$ for some node $j \in S^{t-1}$. We also have

that $\underline{\mathbf{r}}_j^{t-1} > 0$ if $j \in S^{t-1}$. Therefore, $\mathbf{e}'_i > 0$ if $i \in S^t \setminus S^{t-1}$. Thus, we have

$$\begin{cases} \mathbf{e}'_i = 0, & \text{if } i \in S^{t-1}, \\ \mathbf{e}'_i > 0, & \text{if } i \in S^t \setminus S^{t-1}. \end{cases}$$

We can get that $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}^t)^{-1}\mathbf{e}'$. The elements of $c\mathbf{P}^t$ are non-negative and $||c\mathbf{P}^t||_\infty < 1$. We have that $(\mathbf{I} - c\mathbf{P}^t)^{-1} = \sum_{l=0}^{\infty} c^l(\mathbf{P}^t)^l$, where $(\mathbf{P}^t)^l$ represents the matrix $\mathbf{P}^t$ to the power of l. Each element $[(\mathbf{P}^t)^l]_{i,j}$ represents the probability of transiting from node i to j in the l-th step.

Consider a node $i \in S^{t-1}$. If $i \rightsquigarrow S^t \setminus S^{t-1}$, we have $[(\mathbf{P}^t)^l]_{i,j} > 0$ for some l and some node $j \in S^t \setminus S^{t-1}$. So, $[(\mathbf{P}^t)^l\mathbf{e}']_i > 0$ and $\Delta\mathbf{r}_i > 0$. If $i \not\rightsquigarrow S^t \setminus S^{t-1}$, we have $[(\mathbf{P}^t)^l]_{i,j} = 0$ for any l and any node $j \in S^t \setminus S^{t-1}$. So, $[(\mathbf{P}^t)^l\mathbf{e}']_i = 0$ and $\Delta\mathbf{r}_i = 0$. □

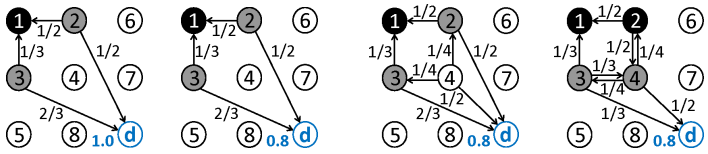
2.5.3 Monotonicity of the Upper Bound

In this subsection, we analyze the monotonicity of the upper bound. Specifically, we prove that the upper bound values are strictly increasing until they converge to the exact proximity values. That is, for any node $i \in S^{t-1}$, $\overline{\mathbf{r}}_i^t < \overline{\mathbf{r}}_i^{t-1}$ until $\overline{\mathbf{r}}_i^{t-1} = \mathbf{r}_i$.

From iteration $(t-1)$ to t, we decrease the proximity value of the dummy node and add new nodes in $S^t \setminus S^{t-1}$. After adding the new nodes, the transition probabilities need to be updated accordingly. Specifically, we need to

1. Decrease the proximity value of the dummy node from $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1}$ to $\overline{\mathbf{r}}_{\mathrm{d}}^t$;
2. Add the transition probabilities $\{p_{i,j}\}$ from the newly added nodes $i (\in S^t \setminus S^{t-1})$ to nodes $j (\in S^t)$, and $\{p_{i,\mathrm{d}}\}$ from i to the dummy node d;
3. Add the transition probabilities $\{p_{j,i}\}$ from nodes $j (\in \delta S^{t-1})$ to the newly added nodes $i (\in S^t \setminus S^{t-1})$, and remove their correspondences in $\{p_{j,\mathrm{d}}\}$.

An example is shown in Figure 2.7. Figure 2.7(a) shows the transition graph for the first iteration. Figures 2.7(b), 2.7(c) and 2.7(d) show the resulting graphs after applying steps 1, 2, and 3 respectively. The graph in Figure 2.7(d) is the final transition graph for the next iteration.



(a) 1st iter. (b) after step 1 (c) after step 2 (d) 2nd iter.

Figure 2.7: Transition graphs between two adjacent iterations (upper bound)

The upper bound values will monotonically change at each step. In step 1, reducing the value of $\mathbf{r}_{\mathrm{d}}$ will not increase the upper bound values. Applying step 2 will not change the upper bound values for the nodes in S^{t-1}, since all the newly added transition probabilities begin from nodes $i(\in S^t \setminus S^{t-1})$. In step 3, we resets the transition probabilities from nodes $j(\in \delta S^{t-1})$ to the newly added nodes $i(\in S^t \setminus S^{t-1})$. This is equivalent to destination change, i.e., changing $\{p_{j,\mathrm{d}}\}$ to $\{p_{j,i}\}$. Moreover, we have that $\overline{\mathbf{r}}_{\mathrm{d}}^t \geq \overline{\mathbf{r}}_i^t$. Thus, in step 3, the upper bound values will not increase.

We will analyze the three steps rigidly in Lemmas 5, 6, and 7 respectively. Only the process from iteration $(t-1)$ to t is considered in the three lemmas. After the discussion of the three lemmas, we will use mathematical induction on each iteration to prove the monotonicity of the upper bound at all the iterations. Thus, in Lemmas 5, 6, and 7, we assume that $t \geq 2$ and $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1} > \overline{\mathbf{r}}_i^{t-1}$ for any node $i \in \delta S^{t-1}$. The idea is that the condition is first assumed to be satisfied at the $(t-1)$-th iteration, and we will prove that it is also satisfied at the t-th iteration.

In Lemmas 5, 6, and 7, we use the following symbols. At the $(t-1)$-th iteration, we use $\mathbf{P}^{t-1}$ to denote the transition probability matrix, and $\overline{\mathbf{r}}^{t-1}$ to denote the upper bound vector. After applying step 1, the transition probability matrix does not change and we use $\mathbf{r}^a$ to represent the new upper bound vector. After applying step 2, the new transition probability matrix is denoted as $\mathbf{P}'$ and the new upper bound vector is denoted as $\mathbf{r}^b$. After applying step 3, we use $\mathbf{P}^t$ to denote the transition probability matrix, and $\overline{\mathbf{r}}^t$ to denote the upper bound vector.

Lemma 5. (Step 1) *For any node $i \in S^{t-1}$, we have*

$$\begin{cases} \overline{\mathbf{r}}_i^{t-1} = \mathbf{r}_i^a, & \text{if } i \not\rightsquigarrow \mathrm{d}, \\ \overline{\mathbf{r}}_i^{t-1} > \mathbf{r}_i^a, & \text{if } i \rightsquigarrow \mathrm{d}, \end{cases}$$

where $\rightsquigarrow$ *and* $\not\rightsquigarrow$ *represent the reachability in the transition graph at the* $(t-1)$*-th iteration.*

Proof. In step 1, we decrease the proximity value of the dummy node from $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1}$ to $\overline{\mathbf{r}}_{\mathrm{d}}^{t} = \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}_{i}^{t-1}$. By the assumption, $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1} > \overline{\mathbf{r}}_{i}^{t-1}$ for any node $i \in \delta S^{t-1}$. Thus, $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1} > \overline{\mathbf{r}}_{\mathrm{d}}^{t}$.

The upper bound vectors $\overline{\mathbf{r}}^{t-1}$ and $\mathbf{r}^{a}$ both are computed based on $\mathbf{P}^{t-1}$ but with different $\mathbf{e}$ vectors. The size of matrix $\mathbf{P}^{t-1}$ is $(|S^{t-1}|+1) \times (|S^{t-1}|+1)$. The sizes of vectors $\overline{\mathbf{r}}^{t-1}$ and $\mathbf{r}^{a}$ both are $(|S^{t-1}|+1) \times 1$. We have the following two equations.

$$\begin{cases} \overline{\mathbf{r}}^{t-1} = c\mathbf{P}^{t-1}\overline{\mathbf{r}}^{t-1} + \mathbf{e}^{t-1}, \\ \mathbf{r}^{a} \quad = c\mathbf{P}^{t-1}\mathbf{r}^{a} \quad + \mathbf{e}^{a}. \end{cases}$$

In the first equation, $\mathbf{e}_{q}^{t-1} = 1$, $\mathbf{e}_{\mathrm{d}}^{t-1} = \overline{\mathbf{r}}_{\mathrm{d}}^{t-1}$, and $\mathbf{e}_{i}^{t-1} = 0$ if $i \in S^{t-1} \setminus \{q\}$. In the second equation, $\mathbf{e}_{q}^{a} = 1$, $\mathbf{e}_{\mathrm{d}}^{a} = \overline{\mathbf{r}}_{\mathrm{d}}^{t}$, and $\mathbf{e}_{i}^{a} = 0$ if $i \in S^{t-1} \setminus \{q\}$.

Let $\Delta\mathbf{r} = \overline{\mathbf{r}}^{t-1} - \mathbf{r}^{a}$. We have that $\Delta\mathbf{r} = c\mathbf{P}^{t-1}\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}' = \mathbf{e}^{t-1} - \mathbf{e}^{a}$. Since $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1} > \overline{\mathbf{r}}_{\mathrm{d}}^{t}$, we have that

$$\begin{cases} \mathbf{e}'_{i} = 0, & \text{if } i \in S^{t-1}, \\ \mathbf{e}'_{i} > 0, & \text{if } i = \mathrm{d}. \end{cases}$$

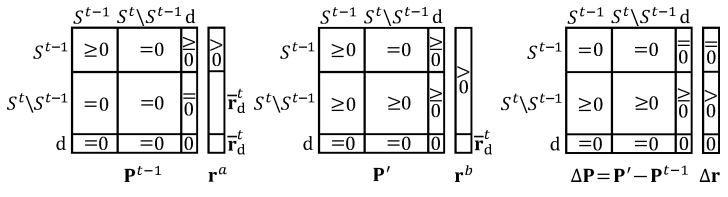
We can get that $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}^{t-1})^{-1}\mathbf{e}'$. The elements of $c\mathbf{P}^{t-1}$ are non-negative and $||c\mathbf{P}^{t-1}||_{\infty} < 1$. We have that $(\mathbf{I} - c\mathbf{P}^{t-1})^{-1} = \sum_{l=0}^{\infty} c^{l}(\mathbf{P}^{t-1})^{l}$, where $(\mathbf{P}^{t-1})^{l}$ denotes matrix $\mathbf{P}^{t-1}$ to the power of l. Each element $[(\mathbf{P}^{t-1})^{l}]_{i,\mathrm{d}}$ represents the probability of transiting from node i to d in the l-th step.

Consider a node $i \in S^{t-1}$. If $i \rightsquigarrow \mathrm{d}$, we have that $[(\mathbf{P}^{t-1})^{l}]_{i,\mathrm{d}} > 0$ for some l. Thus, $\Delta\mathbf{r}_{i} > 0$, i.e., $\overline{\mathbf{r}}_{i}^{t-1} > \mathbf{r}_{i}^{a}$. If $i \not\rightsquigarrow \mathrm{d}$, we have that $[(\mathbf{P}^{t-1})^{l}]_{i,\mathrm{d}} = 0$ for any l. Thus, $\Delta\mathbf{r}_{i} = 0$. □

Lemma 6. (Step 2)

$$\begin{cases} \mathbf{r}_{i}^{a} = \mathbf{r}_{i}^{b}, & \textit{if } i \in S^{t-1}, \\ \overline{\mathbf{r}}_{\mathrm{d}}^{t} > \mathbf{r}_{i}^{b}, & \textit{if } i \in S^{t} \setminus S^{t-1}. \end{cases}$$

Proof. In step 2, we add transition probabilities $\{p_{i,j}\}$ from the newly added nodes $i (\in S^{t} \setminus S^{t-1})$ to nodes $j (\in S^{t})$, and $\{p_{i,\mathrm{d}}\}$ from i to the dummy node d.



(a) after step 1 (b) after step 2 (c) difference

Figure 2.8: Transition probability matrices between two adjacent iterations (upper bound, step 2)

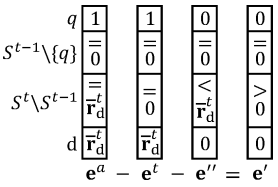


Figure 2.9: The vectors $\mathbf{e}^a$, $\mathbf{e}^t$, $\mathbf{e}''$, and $\mathbf{e}'$

Since the sizes of matrices $\mathbf{P}^{t-1}$ and $\mathbf{P}'$ are $(|S^{t-1}|+1)\times(|S^{t-1}|+1)$ and $(|S^t|+1)\times(|S^t|+1)$ respectively, we extend $\mathbf{P}^{t-1}$ with $|S^t \setminus S^{t-1}|$ columns and $|S^t \setminus S^{t-1}|$ rows as shown in Figure 2.8(a). The newly added elements are set to 0. Thus, matrices $\mathbf{P}^{t-1}$ and $\mathbf{P}'$ are of the same size after extension. Similarly, we extend $\mathbf{r}^a$ with $|S^t \setminus S^{t-1}|$ elements, which are set to $\overline{\mathbf{r}}^t_{\mathrm{d}}$. Thus, $\mathbf{r}^a$ and $\mathbf{r}^b$ are of the same size $(|S^t|+1)\times 1$ after extension. For any node $i \in S^{t-1}$, we must have that $\mathbf{r}^a_i > 0$. Figures 2.8(a) and 2.8(b) illustrate these matrices and vectors. We have the following two equations.

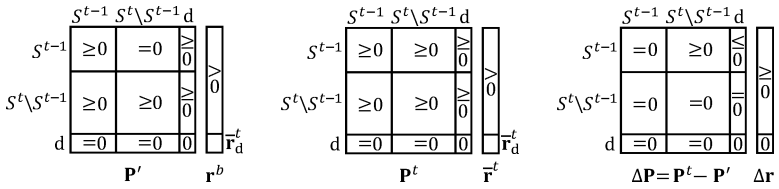
$$\begin{cases} \mathbf{r}^a = c\mathbf{P}^{t-1}\mathbf{r}^a + \mathbf{e}^a, \\ \mathbf{r}^b = c\mathbf{P}'\mathbf{r}^b \quad + \mathbf{e}^t. \end{cases}$$

In the first equation, $\mathbf{e}^a_q = 1$, $\mathbf{e}^a_i = 0$ if $i \in S^{t-1} \setminus \{q\}$ and $\mathbf{e}^a_i = \overline{\mathbf{r}}^t_{\mathrm{d}}$ if $i \in S^t \setminus S^{t-1} \cup \{\mathrm{d}\}$. In the second equation, $\mathbf{e}^t_q = 1$, $\mathbf{e}^t_{\mathrm{d}} = \overline{\mathbf{r}}^t_{\mathrm{d}}$, and $\mathbf{e}^t_i = 0$ if $i \in S^t \setminus \{q\}$.

Let $\Delta\mathbf{r} = \mathbf{r}^a - \mathbf{r}^b$ and $\Delta\mathbf{P} = \mathbf{P}' - \mathbf{P}^{t-1}$. Note that $\Delta\mathbf{P}_{i,j} = 0$ if $i \in S^{t-1} \cup \{\mathrm{d}\}$ and $\Delta\mathbf{P}_{i,j} = \mathbf{P}'_{i,j}$ otherwise. Figure 2.8(c) illustrates the matrix $\Delta\mathbf{P}$ and vector $\Delta\mathbf{r}$. We have that $\Delta\mathbf{r} = c\mathbf{P}^{t-1}\mathbf{r}^a - c\mathbf{P}'\mathbf{r}^b + \mathbf{e}^a - \mathbf{e}^t = c\mathbf{P}'\Delta\mathbf{r} - c\Delta\mathbf{P}\mathbf{r}^a + \mathbf{e}^a - \mathbf{e}^t = c\mathbf{P}'\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}' = \mathbf{e}^a - \mathbf{e}^t - c\Delta\mathbf{P}\mathbf{r}^a$.

Let $\mathbf{e}'' = c\Delta\mathbf{P}\mathbf{r}^a$. Note that $\Delta\mathbf{P}_{i,j} = 0$ if $i \in S^{t-1} \cup \{\mathrm{d}\}$. Thus, $\mathbf{e}''_i = 0$ if $i \in S^{t-1} \cup \{\mathrm{d}\}$. We have that $\Delta\mathbf{P}_{i,j} = 0$ if $i \in S^t \setminus S^{t-1}$ and $j \in S^{t-1} \setminus \delta S^{t-1}$ and $\Delta\mathbf{P}_{i,j} \geq 0$ if $i \in S^t \setminus S^{t-1}$ and $j \in \delta S^{t-1}$. For any node $j \in \delta S^{t-1}$, $j \rightsquigarrow \mathrm{d}$. By Lemma 5, we have that $\overline{\mathbf{r}}^{t-1}_j > \mathbf{r}^a_j$. Since $\overline{\mathbf{r}}^t_{\mathrm{d}} = \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}^{t-1}_i$, $\mathbf{r}^a_j < \overline{\mathbf{r}}^t_{\mathrm{d}}$ for any node $j \in \delta S^{t-1}$. We also have that $\mathbf{r}^a_j = \overline{\mathbf{r}}^t_{\mathrm{d}}$ if $j \in S^t \setminus S^{t-1} \cup \{\mathrm{d}\}$. The sum of the elements in the i-th ($i \in S^t \setminus S^{t-1}$) row of $\Delta\mathbf{P}$ equals 1. Thus, $\mathbf{e}''_i < \overline{\mathbf{r}}^t_{\mathrm{d}}$ if $i \in S^t \setminus S^{t-1}$. Since $\mathbf{e}' = \mathbf{e}^a - \mathbf{e}^t - \mathbf{e}''$, we have that

$$\begin{cases} \mathbf{e}'_i = 0, & \text{if } i \in S^{t-1} \cup \{\mathrm{d}\}, \\ \mathbf{e}'_i > 0, & \text{if } i \in S^t \setminus S^{t-1}. \end{cases}$$



(a) after step 2 (b) after step 3 (c) difference

Figure 2.10: Transition probability matrices between two adjacent iterations (upper bound, step 3)

Figure 2.9 illustrates the vectors $\mathbf{e}^a$, $\mathbf{e}^t$, $\mathbf{e}''$, and $\mathbf{e}'$.

We can get that $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}')^{-1}\mathbf{e}'$. The elements of $c\mathbf{P}'$ are non-negative and $||c\mathbf{P}'||_\infty < 1$. We have that $(\mathbf{I} - c\mathbf{P}')^{-1} = \sum_{l=0}^{\infty} c^l(\mathbf{P}')^l$, where $(\mathbf{P}')^l$ represents the matrix $\mathbf{P}'$ to the power of l. We have that $\Delta\mathbf{r} = (\mathbf{I} + \sum_{l=1}^{\infty} c^l(\mathbf{P}')^l)\mathbf{e}' = \mathbf{e}' + \sum_{l=1}^{\infty} c^l(\mathbf{P}')^l\mathbf{e}'$. Since all the elements in matrix $\mathbf{P}'$ are non-negative, we have that $[(\mathbf{P}')^l]_{i,j} \geq 0$ for any l and any nodes $i, j \in S^t \cup \{\mathrm{d}\}$.

For any node $i \in S^t \setminus S^{t-1}$, $\Delta\mathbf{r}_i > 0$, i.e., $\mathbf{r}^a_i = \bar{\mathbf{r}}^t_{\mathrm{d}} > \mathbf{r}^b_i$.

For any node $i \in S^{t-1}$, $i \nrightarrow S^t \setminus S^{t-1}$ in the transition graph corresponding to $\mathbf{P}'$. Thus, $[(\mathbf{P}')^l]_{i,j} = 0$ for any l and any node $j \in S^t \setminus S^{t-1}$. So, $\Delta\mathbf{r}_i = 0$. □

Lemma 7. (Step 3) *For any node $i \in S^t$, we have that*

$$\begin{cases} \mathbf{r}^b_i = \bar{\mathbf{r}}^t_i, & \text{if } i \nrightarrow S^t \setminus S^{t-1}, \\ \mathbf{r}^b_i > \bar{\mathbf{r}}^t_i, & \text{if } i \rightsquigarrow S^t \setminus S^{t-1}, \end{cases}$$

where $\rightsquigarrow$ and $\nrightarrow$ represent the reachability in the transition graph at the t-th iteration.

Proof. In step 3, we add the transition probabilities $\{p_{j,i}\}$ from nodes $j \in \delta S^{t-1}$ to the newly added nodes $i \in S^t \setminus S^{t-1}$, and remove their correspondences in $\{p_{j,\mathrm{d}}\}$.

The sizes of matrices $\mathbf{P}'$ and $\mathbf{P}^t$ both are $(|S^t|+1)\times(|S^t|+1)$. The sizes of vectors $\mathbf{r}^b$ and $\bar{\mathbf{r}}^t$ both are $(|S^t|+1)\times 1$. Figures 2.10(a) and 2.10(b) illustrate these matrices and vectors. We have the following two equations.

$$\begin{cases} \mathbf{r}^b = c\mathbf{P}'\mathbf{r}^b + \mathbf{e}^t, \\ \bar{\mathbf{r}}^t = c\mathbf{P}^t\bar{\mathbf{r}}^t + \mathbf{e}^t, \end{cases}$$

where $\mathbf{e}^t_q = 1$, $\mathbf{e}^t_{\mathrm{d}} = \bar{\mathbf{r}}^t_{\mathrm{d}}$, and $\mathbf{e}^t_i = 0$ if $i \in S^t \setminus \{q\}$.

Let $\Delta\mathbf{r} = \mathbf{r}^b - \bar{\mathbf{r}}^t$ and $\Delta\mathbf{P} = \mathbf{P}^t - \mathbf{P}'$. Note that $\Delta\mathbf{P}_{i,j} = 0$ if $i \in S^t \setminus S^{t-1} \cup \{\mathrm{d}\}$ or

$j \in S^{t-1}$, $\Delta\mathbf{P}_{i,j} \geq 0$ if $i \in S^{t-1}$ and $j \in S^t \setminus S^{t-1}$, and $\Delta\mathbf{P}_{i,\mathrm{d}} \leq 0$ if $i \in S^{t-1}$. The sum of elements in each row of $\Delta\mathbf{P}$ equals 0. Figure 2.10(c) illustrates the matrix $\Delta\mathbf{P}$ and vector $\Delta\mathbf{r}$. We have that $\Delta\mathbf{r} = c\mathbf{P}'\mathbf{r}^b - c\mathbf{P}^t\overline{\mathbf{r}}^t = c\mathbf{P}^t\Delta\mathbf{r} - c\Delta\mathbf{P}\mathbf{r}^b = c\mathbf{P}^t\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}' = -c\Delta\mathbf{P}\mathbf{r}^b$.

Consider the set of nodes $S' = \{i \,|\, i \in \delta S^{t-1} \text{and } \mathbf{P}^t_{i,j} > 0 \text{for some } j \in S^t \setminus S^{t-1}\}$. S' comprises the set of nodes in δS^{t-1} that are adjacent to the newly added nodes. If node $i \notin S'$, the transition probability $p_{i,j}$ does not change during step 3. Thus, we have that $\Delta\mathbf{P}_{i,j} = 0$ and $\mathbf{e}'_i = 0$ for any node $i \notin S'$ and $j \in S^t \cup \{\mathrm{d}\}$. If node $i \in S'$, the sum of elements in the i-th row of $\Delta\mathbf{P}$ is equal to 0. By Lemma 6, we have that $\mathbf{r}^b_j < \overline{\mathbf{r}}^t_{\mathrm{d}}$ for any node $j \in S^t \setminus S^{t-1}$. Thus, we have that $\mathbf{e}'_i > 0$. In conclusion, we have that

$$\begin{cases} \mathbf{e}'_i = 0, & \text{if } i \in S^t \cup \{\mathrm{d}\} \setminus S', \\ \mathbf{e}'_i > 0, & \text{if } i \in S'. \end{cases}$$

We can get that $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}^t)^{-1}\mathbf{e}'$. The elements of $c\mathbf{P}^t$ are non-negative and $||c\mathbf{P}^t||_\infty < 1$. We have that $(\mathbf{I} - c\mathbf{P}^t)^{-1} = \sum_{l=0}^{\infty} c^l(\mathbf{P}^t)^l$, where $(\mathbf{P}^t)^l$ represents the matrix $\mathbf{P}^t$ to the power of l. We have that $\Delta\mathbf{r} = \sum_{l=0}^{\infty} c^l(\mathbf{P}^t)^l\mathbf{e}'$. Since all the elements in matrix $\mathbf{P}^t$ are non-negative, we have that $[(\mathbf{P}^t)^l]_{i,j} \geq 0$ for any l and any nodes $i, j \in S^t \cup \{\mathrm{d}\}$.

Consider a node $i \in S^t$. If $i \rightsquigarrow S'$, $[(\mathbf{P}^t)^l]_{i,j} > 0$ for some l and some node $j \in S'$. Thus, $\Delta\mathbf{r}_i > 0$ and $\mathbf{r}^b_i > \overline{\mathbf{r}}^t_i$. If $i \not\rightsquigarrow S'$, $[(\mathbf{P}^t)^l]_{i,j} = 0$ for any l and any $j \in S'$. Thus, $\Delta\mathbf{r}_i = 0$ and $\mathbf{r}^b_i = \overline{\mathbf{r}}^t_i$. Since each node in S' is adjacent to at least one node in $S^t \setminus S^{t-1}$ and the graph is undirected, $i \rightsquigarrow S'$ if and only if $i \rightsquigarrow S^t \setminus S^{t-1}$. This completes the proof. □

Theorem 7. *For any node $i \in \delta S^t$, we have that $\overline{\mathbf{r}}^t_{\mathrm{d}} > \overline{\mathbf{r}}^t_i$.*

Proof. The theorem is proved by mathematical induction on the iteration. At the first iteration, for any node $i \in \delta S^1$, we have that $\overline{\mathbf{r}}^1_i < 1$ and $\overline{\mathbf{r}}^1_{\mathrm{d}} = 1$. So, the theorem is trivially satisfied.

Suppose that the theorem is satisfied at the $(t-1)$-th iteration with $t \geq 2$. That

is, $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1} > \overline{\mathbf{r}}_i^{t-1}$, for any node $i \in \delta S^{t-1}$. Next, we will prove that this theorem is still satisfied at the t-th iteration.

Consider a node $i \in \delta S^{t-1}$. Note that $i \rightsquigarrow \mathrm{d}$ in the transition graph corresponding to $\mathbf{P}^{t-1}$. By Lemma 5, $\overline{\mathbf{r}}_i^{t-1} > \mathbf{r}_i^a$. By Lemma 6, $\mathbf{r}_i^a = \mathbf{r}_i^b$. By Lemma 7, $\mathbf{r}_i^b \geq \overline{\mathbf{r}}_i^t$. Thus, $\overline{\mathbf{r}}_i^{t-1} > \overline{\mathbf{r}}_i^t$. Since $\overline{\mathbf{r}}_{\mathrm{d}}^t = \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}_i^{t-1}$, we have that $\overline{\mathbf{r}}_{\mathrm{d}}^t > \overline{\mathbf{r}}_i^t$.

Consider a node $i \in S^t \setminus S^{t-1}$. By Lemma 6, $\overline{\mathbf{r}}_{\mathrm{d}}^t > \mathbf{r}_i^b$. By Lemma 7, $\mathbf{r}_i^b > \overline{\mathbf{r}}_i^t$. Thus, we have that $\overline{\mathbf{r}}_{\mathrm{d}}^t > \overline{\mathbf{r}}_i^t$.

Since $\delta S^t \subset \delta S^{t-1} \cup (S^t \setminus S^{t-1})$, we have that $\overline{\mathbf{r}}_{\mathrm{d}}^t > \overline{\mathbf{r}}_i^t$ for any node $i \in \delta S^t$. This completes the proof. □

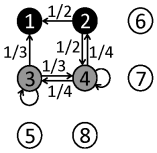
Theorem 8. (Monotonicity of the upper bound) *For any node $i \in S^{t-1}$, we have that*

$$\begin{cases} \overline{\mathbf{r}}_i^{t-1} = \overline{\mathbf{r}}_i^t, & \text{if } i \nrightarrow \mathrm{d}, \\ \overline{\mathbf{r}}_i^{t-1} > \overline{\mathbf{r}}_i^t, & \text{if } i \rightsquigarrow \mathrm{d}, \end{cases}$$

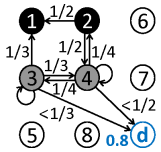
where $\rightsquigarrow$ and $\nrightarrow$ represent the reachability in the transition graph at the t-th iteration.

Proof. For any t ($t \geq 1$) and any node $i \in \delta S^t$, we have that $\overline{\mathbf{r}}_{\mathrm{d}}^t > \overline{\mathbf{r}}_i^t$ by Theorem 7. Thus, the assumption for Lemmas 5, 6, and 7 is satisfied. Consider a node $i \in S^{t-1}$. Suppose that $i \rightsquigarrow \mathrm{d}$. By Lemma 5, $\overline{\mathbf{r}}_i^{t-1} > \mathbf{r}_i^a$. By Lemma 6, $\mathbf{r}_i^a = \mathbf{r}_i^b$. By Lemma 7, $\mathbf{r}_i^b \geq \overline{\mathbf{r}}_i^t$. Thus, we have that $\overline{\mathbf{r}}_i^{t-1} > \overline{\mathbf{r}}_i^t$. Suppose that $i \nrightarrow \mathrm{d}$. By Lemma 5, $\overline{\mathbf{r}}_i^{t-1} = \mathbf{r}_i^a$. By Lemma 6, $\mathbf{r}_i^a = \mathbf{r}_i^b$. By Lemma 7, $\mathbf{r}_i^b = \overline{\mathbf{r}}_i^t$. Thus, we have that $\overline{\mathbf{r}}_i^{t-1} = \overline{\mathbf{r}}_i^t$. □

If $i \nrightarrow \mathrm{d}$ in the transition graph at the t-th iteration, $i \nrightarrow \mathrm{d}$ in the transition graph at any future iteration. Thus, $\overline{\mathbf{r}}_i^t$ will not change during the future iterations. Since $\overline{\mathbf{r}}_i^t$ converges to the exact proximity value $\mathbf{r}_i$ when the entire graph is visited. We must have that $\overline{\mathbf{r}}_i^t = \mathbf{r}_i$ when $i \nrightarrow \mathrm{d}$. In conclusion, the upper bound strictly increases until it converges to the exact proximity value.



(a) adding self-loop for LB $\underline{\mathbf{r}}$



(b) adding self-loop for UB $\overline{\mathbf{r}}$

Figure 2.11: Transition graphs with self-loops

2.5.4 Tightening the Bounds

The bounds used in FLoS can be further tightened by adding self-loop transition probabilities to the nodes in δS. We will still use PHP as an example to illustrate the process. We first define the star-to-mesh transformation on the transition graph, which is inspired by the star-mesh transformation in circuit theory [41].

Definition 3. [Star-to-mesh transformation] *(1) Delete a node $u \in V \setminus \{q\}$ and its incident transition probabilities; (2) For any pair of nodes $i, j \in N_u$, add transition probabilities $p'_{i,j} = cp_{i,u}p_{u,j}$.*

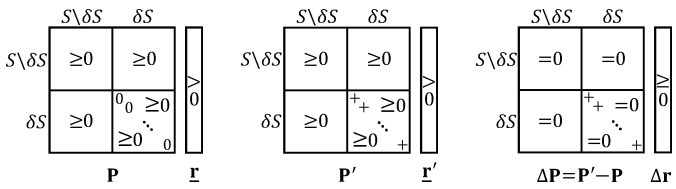
Note that if $i = j$, it becomes the self-loop transition probability $p'_{i,i} = cp_{i,u}p_{u,i}$. Applying the star-to-mesh transformation for a node will not change the PHP proximity values of the remaining nodes.

Lemma 8. *Applying the star-to-mesh transformation of node u will not change the* PHP *proximity values of nodes $V \setminus \{q, u\}$.*

Proof. This can be proved by plugging the equation $\mathbf{r}_u = c\sum_{i \in N_u} p_{u,i}\mathbf{r}_i$ into the recursive equations of neighbor nodes $i \in N_u$ in the original transition graph. □

The self-loop transition probabilities generated in the star-to-mesh transformation can be used to further tighten the lower and upper bounds. Lemmas 9 and 10 analyze the tighter lower and upper bounds respectively. Figure 2.11 shows the transition graphs with self-loop transition probabilities for computing the lower and upper bounds. The original ones are shown in Figure 2.3.

Lemma 9. *Adding self-loop transition probability $p_{i,i} = c\sum_{j \in N_i \cap \delta\bar{S}} p_{i,j}p_{j,i}$ $(\forall i \in \delta S)$ will tighten the lower bound.*



(a) original (b) with self-loop (c) difference

Figure 2.12: Transition probability matrices (lower bound)

Proof. First, we show the new bound values are still lower bounds. For the nodes in $\delta\bar{S}$, we apply star-to-mesh transformation sequentially in any order. After the star-to-mesh transformation of one node $j \in \delta\bar{S}$, we delete all the newly added transition probabilities except the self-loop transition probabilities of nodes in δS. After all the nodes in $\delta\bar{S}$ have been deleted, the self-loop transition probability of a node $i \in \delta S$ is $p_{i,i} = c\sum_{j\in N_i\cap\delta\bar{S}} p_{i,j}p_{j,i}$. During this process, we only apply star-to-mesh transformation and transition probability deletion. Therefore, the new bound values are still lower bounds.

Next, we prove that the new lower bound is tighter.

Let $\mathbf{P}$ be the transition probability matrix for computing the original lower bound vector $\underline{\mathbf{r}}$. Since we add some self-loop transition probabilities to the nodes in δS, we use $\mathbf{P}'$ to denote the new transition probability matrix and $\underline{\mathbf{r}}'$ to denote the corresponding lower bound vector. Figures 2.12(a) and 2.12(b) illustrate the matrices and vectors. We have the following two equations

$$\begin{cases} \underline{\mathbf{r}} = c\mathbf{P}\underline{\mathbf{r}} + \mathbf{e}, \\ \underline{\mathbf{r}}' = c\mathbf{P}'\underline{\mathbf{r}}' + \mathbf{e}. \end{cases}$$

Let $\Delta\mathbf{r} = \underline{\mathbf{r}}' - \underline{\mathbf{r}}$ and $\Delta\mathbf{P} = \mathbf{P}' - \mathbf{P}$. Note that $\Delta\mathbf{P}_{i,i} > 0$ if $i \in \delta S$ and $\Delta\mathbf{P}_{i,j} = 0$ otherwise. Figure 2.12(c) illustrates the matrix $\Delta\mathbf{P}$ and vector $\Delta\mathbf{r}$. We have that $\Delta\mathbf{r} = c\mathbf{P}'\underline{\mathbf{r}}' - c\mathbf{P}\underline{\mathbf{r}} = c\mathbf{P}'\Delta\mathbf{r} + c\Delta\mathbf{P}\underline{\mathbf{r}} = c\mathbf{P}'\Delta\mathbf{r} + \mathbf{e}'$, where $\mathbf{e}' = c\Delta\mathbf{P}\underline{\mathbf{r}}$. We have that

$$\begin{cases} \mathbf{e}'_i = 0, & \text{if } i \in S \setminus \delta S, \\ \mathbf{e}'_i > 0, & \text{if } i \in \delta S. \end{cases}$$

We can get that $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}')^{-1}\mathbf{e}'$. The elements of $c\mathbf{P}'$ are non-negative and $||c\mathbf{P}'||_\infty < 1$. We have that $(\mathbf{I} - c\mathbf{P}')^{-1} = \sum_{l=0}^{\infty} c^l(\mathbf{P}')^l$, where $(\mathbf{P}')^l$ represents the matrix

$\mathbf{P}'$ to the power of l. Each element $[(\mathbf{P}')^l]_{i,j}$ represents the probability of transiting from node i to j in the l-th step.

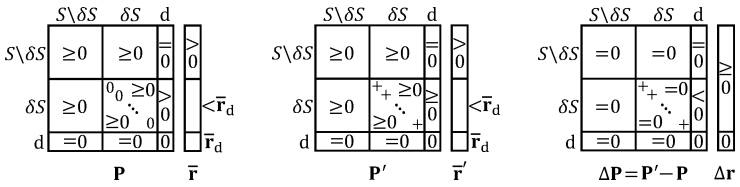
Consider a node $i \in S$. If $i \rightsquigarrow \delta S$ in the transition graph corresponding to $\mathbf{P}'$, $[(\mathbf{P}')^l]_{i,j} > 0$ for some l and some node $j \in \delta S$. So, $[(\mathbf{P}')^l \mathbf{e}']_i > 0$ and $\Delta \mathbf{r}_i > 0$. If $i \not\rightsquigarrow \delta S$ in the transition graph corresponding to $\mathbf{P}'$, $[(\mathbf{P}')^l]_{i,j} = 0$ for any l and any node $j \in \delta S$. So, $[(\mathbf{P}')^l \mathbf{e}']_i = 0$ and $\Delta \mathbf{r}_i = 0$. In conclusion, we have that $\underline{\mathbf{r}}_i \leq \underline{\mathbf{r}}'_i$ for any node $i \in S$. This completes the proof. □

Lemma 10. *Adding self-loop transition probability* $p_{i,i} = c \sum_{j \in N_i \cap \delta \bar{S}} p_{i,j} p_{j,i}$ *and setting the transition probability to dummy node as* $p_{i,\mathrm{d}} = c \sum_{j \in N_i \cap \delta \bar{S}} p_{i,j}(1 - p_{j,i})$ $(\forall i \in \delta S)$ *will tighten the upper bound.*

Proof. First, we show the new bound values are still upper bounds. For the nodes in $\delta \bar{S}$, we apply star-to-mesh transformation sequentially in any order. After the star-to-mesh transformation of one node $j \in \delta \bar{S}$, we change the destination of all the newly added transition probabilities except the self-loop transition probabilities of nodes in δS to the dummy node d. After all the nodes in $\delta \bar{S}$ have been deleted, the self-loop transition probability of a node $i \in \delta S$ is $p_{i,i} = c \sum_{j \in N_i \cap \delta \bar{S}} p_{i,j} p_{j,i}$, and the transition probability from a node $i \in \delta S$ to the dummy node is $p_{i,\mathrm{d}} = c \sum_{j \in N_i \cap \delta \bar{S}} p_{i,j}(1 - p_{j,i})$. During this process, we only apply star-to-mesh transformation and change the destination of transition probabilities from a node $i \in \delta S$ to the dummy node. The proximity value of the dummy node is set as the maximum upper bound value of the nodes in the boundary node, i.e., $\bar{\mathbf{r}}_{\mathrm{d}} = \max_{i \in \delta S} \bar{\mathbf{r}}_i$. Thus, we have that $\bar{\mathbf{r}}_{\mathrm{d}} > \mathbf{r}_i$, for any node $i \in \delta S$. Therefore, the new bound values are still upper bounds.

Next, we prove that the new upper bound is tighter.

Let $\mathbf{P}$ be the transition probability matrix for computing the original upper bound vector $\bar{\mathbf{r}}$. Let $\mathbf{P}'$ denote the new transition probability matrix and $\bar{\mathbf{r}}'$ denote the corresponding upper bound vector. Figures 2.13(a) and 2.13(b) illustrate the matrices and vectors. We have the following two equations



(a) original (b) with self-loop (c) difference

Figure 2.13: Transition probability matrices (upper bound)

$$\begin{cases} \overline{\mathbf{r}} = c\mathbf{P}\overline{\mathbf{r}} + \mathbf{e}', \\ \overline{\mathbf{r}}' = c\mathbf{P}'\overline{\mathbf{r}}' + \mathbf{e}', \end{cases}$$

where $\mathbf{e}'_q = 1$, $\mathbf{e}'_{\mathrm{d}} = \overline{\mathbf{r}}_{\mathrm{d}}$, and $\mathbf{e}'_i = 0$ if $i \in S \setminus \{q\}$.

Let $\Delta\mathbf{r} = \overline{\mathbf{r}} - \overline{\mathbf{r}}'$ and $\Delta\mathbf{P} = \mathbf{P}' - \mathbf{P}$. Comparing $\mathbf{P}$ and $\mathbf{P}'$, we add some self-loop transition probabilities to the nodes in δS and change the destination of transition probabilities from node $i(\in \delta S)$ to the dummy node d. Note that $\Delta\mathbf{P}_{i,i} = p_{i,i}$ if $i \in \delta S$, $\Delta\mathbf{P}_{i,\mathrm{d}} < 0$ if $i \in \delta S$ and $\Delta\mathbf{P}_{i,j} = 0$ otherwise. Moreover, $-\Delta\mathbf{P}_{i,\mathrm{d}} = \sum_{j \in N_i \cap \delta\bar{S}} p_{i,j} - c\sum_{j \in N_i \cap \delta\bar{S}} p_{i,j}(1 - p_{j,i}) > p_{i,i}$. Thus, for any node $i \in \delta S$, $\Delta\mathbf{P}_{i,i} + \Delta\mathbf{P}_{i,\mathrm{d}} < 0$. Figure 2.13(c) illustrates the matrix $\Delta\mathbf{P}$ and vector $\Delta\mathbf{r}$.

We have that $\Delta\mathbf{r} = c\mathbf{P}\overline{\mathbf{r}} - c\mathbf{P}'\overline{\mathbf{r}}' = c\mathbf{P}'\Delta\mathbf{r} - c\Delta\mathbf{P}\overline{\mathbf{r}} = c\mathbf{P}'\Delta\mathbf{r} + \mathbf{e}''$, where $\mathbf{e}'' = -c\Delta\mathbf{P}\overline{\mathbf{r}}$. Based on Theorem 7, we have that for any node $i \in \delta S^t$, $\overline{\mathbf{r}}^t_{\mathrm{d}} > \overline{\mathbf{r}}^t_i$. We get that

$$\begin{cases} \mathbf{e}''_i = 0, & \text{if } i \in S \setminus \delta S \cup \{\mathrm{d}\}, \\ \mathbf{e}''_i > 0, & \text{if } i \in \delta S. \end{cases}$$

We can get that $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{P}')^{-1}\mathbf{e}''$. The elements of $c\mathbf{P}'$ are non-negative and $||c\mathbf{P}'||_\infty < 1$. We have that $(\mathbf{I} - c\mathbf{P}')^{-1} = \sum_{l=0}^{\infty} c^l(\mathbf{P}')^l$, where $(\mathbf{P}')^l$ represents the matrix $\mathbf{P}'$ to the power of l. Each element $[(\mathbf{P}')^l]_{i,j}$ represents the probability of transiting from node i to j in the l-th step.

Consider a node $i \in S$. If $i \rightsquigarrow \delta S$ in the transition graph corresponding to $\mathbf{P}'$, $[(\mathbf{P}')^l]_{i,j} > 0$ for some l and some node $j \in \delta S$. So, $[(\mathbf{P}')^l\mathbf{e}'']_i > 0$ and $\Delta\mathbf{r}_i > 0$. If $i \not\rightsquigarrow \delta S$ in the transition graph corresponding to $\mathbf{P}'$, $[(\mathbf{P}')^l]_{i,j} = 0$ for any l and any node $j \in \delta S$. So, $[(\mathbf{P}')^l\mathbf{e}'']_i = 0$ and $\Delta\mathbf{r}_i = 0$. In conclusion, we have that $\overline{\mathbf{r}}_i \geq \overline{\mathbf{r}}'_i$ for any node $i \in S$. This completes the proof. □

2.5.5 Analysis on the Number of Visited Nodes

In this subsection, we analyze the number of visited nodes. Let h be the average number of neighbors of a node. Suppose that the nodes in the boundary of visited nodes are ρ hops away from the query node q. We assume that the number of visited nodes equals h^ρ. Note that this is an upper bound of the number of visited nodes. In Section 2.8, we show the actual number of visited nodes in real graphs.

The proximity of one node will decrease by a factor of c when it is one hop farther away from the query node. The nodes in the boundary of the visited nodes have proximity less than c^ρ because they are ρ hops away from the query.

What is the distribution of the gaps between the upper and lower bounds? The upper bound $\overline{\mathbf{r}}$ is computed based on the recursive equation $\overline{\mathbf{r}} = c\mathbf{P}\overline{\mathbf{r}} + \mathbf{e}''$, where vector $\mathbf{e}''$ has only two non-zero elements $\mathbf{e}''_q = 1$ and $\mathbf{e}''_{\mathrm{d}} = \mathbf{r}_{\mathrm{d}}$. When we set the proximity value of dummy node to 0, i.e., $\mathbf{e}''_{\mathrm{d}} = 0$, we will get the recursive equation $\underline{\mathbf{r}} = c\mathbf{P}\underline{\mathbf{r}} + \mathbf{e}$ for the lower bound $\underline{\mathbf{r}}$. Let $\mathbf{r}' = \overline{\mathbf{r}} - \underline{\mathbf{r}}$ be the gaps between the upper and lower bounds. We have that $\mathbf{r}' = c\mathbf{P}\mathbf{r}' + \mathbf{e}'$, where vector $\mathbf{e}'$ has only one non-zero element $\mathbf{e}'_{\mathrm{d}} = \mathbf{r}_{\mathrm{d}}$. Thus the gaps $\mathbf{r}'$ can be interpreted as the PHP proximity values when the query node is the dummy node d with constant proximity value $\mathbf{r}_{\mathrm{d}}$.

Based on this observation, the gap $\mathbf{r}'_u$ of one node u will decrease by a factor of c when it is one hop farther away from the boundary. Let ρ_u denote the number of hops that u is away from the query node. Node u is $\rho - \rho_u$ hops away from the boundary. So, the gap $\mathbf{r}'_u$ is less than $c^{\rho-\rho_u} \cdot c^\rho$, i.e., $\mathbf{r}'_u < c^{2\rho-\rho_u}$.

For any two nodes u and v, we can distinguish their rankings if $\mathbf{r}'_u + \mathbf{r}'_v < \epsilon$, where ϵ is the difference of the exact proximity values of nodes u and v. We already derive that $\mathbf{r}'_u < c^{2\rho-\rho_u}$ and $\mathbf{r}'_v < c^{2\rho-\rho_v}$. Thus if we have that $\rho > \frac{1}{2}\log_c \epsilon - \frac{1}{2}\log_c(c^{-\rho_u} + c^{-\rho_v})$, then the pair of nodes u and v can be distinguished.

Now we consider the case when u and v are the k-th and $(k+1)$-th nodes in the exact ranking list respectively. We have $h^{\rho_u} \geq k$ and $h^{\rho_v} \geq k+1$. Thus, the inequality $\rho > \frac{1}{2}\log_c \frac{\epsilon}{2} + \frac{1}{2}\log_h k$ should be satisfied. Therefore, we need to visit

$h^{\rho} = O((kh^{\log_c(\epsilon/2)})^{\frac{1}{2}})$ nodes to distinguish the k-th and $(k+1)$-th nodes.

2.5.6 Complexity

Assume Algorithm 2 executes in β iterations, which is proportional to $(kh^{\log_c(\epsilon/2)})^{\frac{1}{2}}$. Let h be the average number of neighbors of a node. The LocalExpansion step takes $O(ht)$ time to find the node to expand at the t-th iteration. To update the lower bound, updating $\mathbf{P}$ needs $O(h^2)$ operations, and updating $\underline{\mathbf{r}}$ and $\mathbf{e}$ needs $O(h)$ operations. Subgraph induced by S has $O(h^2t)$ edges, so matrix $\mathbf{P}$ has $O(h^2t)$ non-zero entries. Therefore using the iterative method to solve linear equations takes $O(\alpha h^2 t)$ time, where α is the number of iterations. Thus the overall complexity of UpdateLowerBound in the t-th iteration is $O(\alpha h^2 t)$. The complexity of UpdateUpperBound function is the same as that of UpdateLowerBound. In the CheckTerminationCriterion step, finding the nodes with largest lower bounds takes $O(ht)$ time. Therefore, the overall complexity of FLoS is $O(\sum_{t=1}^{\beta}(\alpha h^2 t + ht)) = O(\alpha h^2 \beta^2)$.

At each iteration, FLoS visits h new nodes on average. In the worst case, where the whole graph is visited, FLoS needs to run $\beta = n/h$ iterations. Thus, the worst case complexity of FLoS is $O(\alpha h^2 \beta^2) = O(\alpha n^2)$.

Note that so far, we have used PHP to illustrate the key principles underlying the fast local search method. EI and DHT are equivalent with PHP thus there is no need to develop algorithm for them. For THT, deleting a transition probability will not increase the proximity of any node. Therefore, when we delete all the transition probabilities $\{p_{i,j} : i \text{ or } j \in \bar{S}\}$ in the original transition graph, the proximity value of any node computed based on the modified transition graph will be the lower bound. For the upper bound, we add a dummy node with value L, which is the largest possible proximity value of THT. All other processes are similar to those of PHP and omitted here.

Algorithm 8 CheckTerminationCriterion2()

1: **if** $|S^t \setminus \delta S^t \setminus \{q\}| \geq k$ **then**
2: | **for** $k' = k$ **to** $\min\{|S^t \setminus \delta S^t \setminus \{q\}|, \eta k\}$ **do**
3: | | $K \leftarrow k'$ nodes in $S^t \setminus \delta S^t \setminus \{q\}$ with largest $\underline{\mathbf{r}}^t$;
4: | | **if** $\min_{i \in K} \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \setminus K \setminus \{q\}} \overline{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true;

2.5.7 Analysis on the Termination Criterion

Algorithm 2 is guaranteed to stop. If the whole graph has been visited, the lower and upper bound values must converge to the exact proximity values. Then, lines 2 and 3 of Algorithm 6 can be changed into

2: $K \leftarrow k$ nodes in $S^t \setminus \{q\}$ with largest $\mathbf{r}$;

3: **if** $\min_{i \in K} \mathbf{r}_i \geq \max_{i \in S^t \setminus K \setminus \{q\}} \mathbf{r}_i$ **then** bStop $\leftarrow$ true;

Since the whole graph has been visited, we have $\delta S^t = \emptyset$. So, K contains the top-k nodes in $S^t \setminus \{q\}$ with largest $\mathbf{r}$, and $\min_{i \in K} \mathbf{r}_i \geq \max_{i \in S^t \setminus K \setminus \{q\}} \mathbf{r}_i$. Thus the termination criterion is met and the algorithm will terminate.

If the proximity values of the k-th and $(k+1)$-th nodes are close, the algorithm may need to visit many nodes before the termination criterion is met. As suggested by [19, 61], a small relaxation of the termination criterion in Algorithm 6 often leads to even faster termination. When the user wants to view the top-k nodes, it is usually quite acceptable if a few more nodes are returned. For example, if the users wants $k = 20$ nodes, the system may limit the number of answers to a range $[k, \eta k]$, where $\eta (\eta \geq 1)$ is a constant, such as, $[k, 2k] = [20, 40]$. In this case, the termination criterion above should be modified to looking for a set of k' nodes $K = \{u_1, \cdots, u_{k'}\}$, where $k \leq k' \leq \eta k$, such that the nodes in K are the exact top-k' nodes. Algorithm 8 shows the relaxed termination criterion. When $\eta = 1$, Algorithm 8 degrades to the original Algorithm 6. Usually, we can set $\eta = 2$. Each iteration at lines 3-4 of Algorithm 8 can be implemented by the linear time selection algorithm. In the t-th iteration, the number of visited nodes is $O(ht)$, where h is the average node degree. Thus, Algorithm 8 runs in $O(\eta k h t)$.

2.6 Extensions of FLoS to Other Measures

In this section, we study how to extend the FLoS method to random walk with restart, RoundTripRank, Katz score, and absorption probability.

2.6.1 Extension to Random Walk with Restart

In this section, we discuss how to apply the FLoS method to random walk with restart (RWR) [112] which has local optimum. The key idea is to utilize the relationship between RWR and PHP. Utilizing such relationship, we can easily derive the lower and upper bounds for RWR based on the lower and upper bounds for PHP.

Random walk with restart (also known as personalized PageRank) [112] is a widely used proximity measure. RWR can be described as follows. From a node i, the random walker can walk to its neighbors with probabilities proportional to the edge weights. In each step, it has a probability of $(1-c)$ to return to the query node q, where c is a constant. The proximity of node i w.r.t. q is defined as the stationary probability that the random walker will finally stay at i. RWR can be defined recursively as

$$\mathbf{r}_i = \begin{cases} c\sum_{j\in N_i} p_{j,i}\mathbf{r}_j + (1-c), & \text{if } i = q, \\ c\sum_{j\in N_i} p_{j,i}\mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where c $(0<c<1)$ is a constant decay factor, and $p_{j,i}=\frac{w_{j,i}}{w_j}$ is the transition probability from node j to i.

Lemma 11. RWR *has local maximum.*

Proof. Examples can be constructed to show that RWR has local maximum, which are omitted. □

Let RWR(i) and PHP(i) represent the RWR and PHP proximity values of node i respectively. We first show that RWR and PHP have the following relationship.

Theorem 9. $\text{RWR}(i) \propto w_i \cdot \text{PHP}(i)$

Proof. Based on the recursive definition of RWR, we have $\frac{\text{RWR}(i)}{w_i} = c\sum_{j\in N_i}\frac{w_{j,i}}{w_j}\cdot\frac{\text{RWR}(j)}{w_i} = c\sum_{j\in N_i}\frac{w_{i,j}}{w_i}\cdot\frac{\text{RWR}(j)}{w_j}$, for any node $i \neq q$. Thus, for any node $i \neq q$, we have that $\frac{\text{RWR}(i)}{w_i} = c\sum_{j\in N_i} p_{i,j}\cdot\frac{\text{RWR}(j)}{w_j}$. This degree normalized RWR, $\frac{\text{RWR}(i)}{w_i}$, has the same recursive equation as PHP with decay factor c. So we have that $\frac{\text{RWR}(i)}{w_i\cdot\text{PHP}(i)} = \frac{\text{RWR}(q)}{w_q\cdot\text{PHP}(q)}$. When the query node q is fixed, we have that $\text{RWR}(i) \propto w_i \cdot \text{PHP}(i)$. □

Suppose that node $v \in \delta S^t$ has the largest PHP proximity value. Based on Theorem 1, for any node $i \in \bar{S}^t$, we have that $\text{PHP}(i) \leq \text{PHP}(v)$. Let $w(\bar{S}^t)$ denote the maximum degree of unvisited nodes in $\bar{S}^t$. We have that $w_i \cdot \text{PHP}(i) \leq w(\bar{S}^t) \cdot \text{PHP}(i) \leq w(\bar{S}^t) \cdot \text{PHP}(v)$. Therefore, if we maintain the maximum degree of the unvisited nodes, we can develop the upper bound for the proximity values of unvisited nodes.

Specifically, we can apply FLoS to RWR as follows. In Algorithm 3, we can change line 1 to the following line.

1: $u \leftarrow \text{argmax}_{i\in\delta S^{t-1}}\, w_i \cdot (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;

In Algorithm 6, we can change line 2 and 3 to the following two lines.

2: $K \leftarrow k$ nodes in $S^t \backslash \delta S^t \backslash \{q\}$ with largest $w_i \cdot \underline{\mathbf{r}}_i^t$ values;

3: **if** $\min_{i\in K} w_i \cdot \underline{\mathbf{r}}_i^t \geq \max_{i\in S^t\backslash K\backslash\{q\}} w_i \cdot \overline{\mathbf{r}}_i^t$ **and** $\min_{i\in K} w_i \cdot \underline{\mathbf{r}}_i^t \geq w(\bar{S}^t) \cdot \max_{i\in\delta S^t} \overline{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true;

All other processes remain the same.

2.6.2 Extension to RWR with Multiple Query Nodes

In RWR, there may be multiple query nodes. In this section, we study how to extend the FLoS method to handle multiple query nodes.

We use Q to denote the set of query nodes. We use α_i to denote the preference value of node i. We have that $\alpha_i > 0$ if $i \in Q$ and $\alpha_i = 0$ otherwise. These preference values satisfy that $\sum_{i\in Q}\alpha_i = 1$. Then, RWR can be defined recursively as

$$\mathbf{r}_i = \begin{cases} c\sum_{j\in N_i} p_{j,i}\mathbf{r}_j + (1-c)\alpha_i, & \text{if } i \in Q, \\ c\sum_{j\in N_i} p_{j,i}\mathbf{r}_j, & \text{if } i \in V \setminus Q, \end{cases}$$

where c $(0 < c < 1)$ is a constant decay factor.

We use $\mathrm{RWR}_i(j)$ and $\mathrm{PHP}_i(j)$ to denote the RWR and PHP proximity value of node j respectively when the query is node i. We use $\mathrm{RWR}_Q(j)$ to denote the RWR proximity value of node j when there is a set of query nodes Q and the preference value of node $i(\in Q)$ is α_i. We have the following linearity theorem [51].

Theorem 10. (Linearity) $\mathrm{RWR}_Q(j) = \sum_{i\in Q} \alpha_i \cdot \mathrm{RWR}_i(j)$.

In Theorem 9, we prove that there is a simple relationship between $\mathrm{RWR}_i(j)$ and $\mathrm{PHP}_i(j)$. By this relationship, we can develop the relationship between $\mathrm{RWR}_Q(j)$ and $\mathrm{PHP}_i(j)$ with $i \in Q$.

Theorem 11. $\mathrm{RWR}_Q(j) = \sum_{i\in Q} \frac{\alpha_i \cdot w_j \cdot \mathrm{RWR}_i(i)}{w_i} \cdot \mathrm{PHP}_i(j)$.

Proof. From Theorem 9, if node i is the query node, we have that $\frac{\mathrm{RWR}_i(j)}{w_j \cdot \mathrm{PHP}_i(j)} = \frac{\mathrm{RWR}_i(i)}{w_i \cdot \mathrm{PHP}_i(i)}$. Since $\mathrm{PHP}_i(i) = 1$, we have that $\mathrm{RWR}_i(j) = \frac{w_j \cdot \mathrm{RWR}_i(i)}{w_i} \cdot \mathrm{PHP}_i(j)$. Plugging it into the equation in Theorem 10 completes the proof. □

Theorem 11 gives a simple relationship between RWR with multiple query nodes and PHP. Based on this theorem, we can develop the lower and upper bounds for RWR with multiple query nodes.

Algorithm 9 shows the modified FLoS algorithm. We still maintain one set of visited nodes, S^t, at the t-th iteration. When we compute the upper bound, we set each node $i \in Q$ as the single query node individually, and compute the lower and upper bound of the non-query nodes $j \in S^t \setminus \{i\}$ with regard to the single query node i. Note that we pre-compute the node degree w_u and value $\mathrm{RWR}_u(u)$ for each node $u \in V$. Then, by Theorem 11, we can combine the bounds and get the lower and upper bounds for the nodes $j \in S^t \setminus Q$ with regard to the set of query nodes Q.

Consider each query node $i \in Q$. Suppose that node $v_i \in \delta S^t$ has the largest PHP proximity value. Based on Theorem 1, for any node $j \in \bar{S}^t$, we have that

Algorithm 9 FLoS_RWR for multiple query nodes

Input: $G(V, E)$, query Q, number k, decay factor c, value τ
Output: Top-k node set K

1: $S^0 \leftarrow Q$; $\delta S^0 \leftarrow \delta Q$; bStop $\leftarrow$ false; $t \leftarrow 0$;
2: **for** *each* $i \in Q$ **do** $\underline{\mathbf{r}}_i^0 \leftarrow 1$; $\overline{\mathbf{r}}_i^0 \leftarrow 1$;
3: **while** bStop = false **do**
4: $\quad t \leftarrow t + 1$; LocalExpansion();
5: $\quad$ **for** *each* $i \in Q$ **do**
6: $\quad\quad$ Set node i as the single query node;
7: $\quad\quad$ Update PHP LB and UB of each node $j \in S^t \setminus \{i\}$;
8: $\quad$ Update RWR LB($\underline{\mathbf{r}}_j^t$) and UB($\overline{\mathbf{r}}_j^t$) of each node $j \in S^t \setminus Q$;
9: $\quad$ Compute $\mathrm{UB}_Q(\bar{S}^t)$, the upper bound for $\bar{S}^t$;
10: $\quad$ CheckTerminationCriterion3();
11: **return** K;

Algorithm 10 CheckTerminationCriterion3()

1: **if** $|S^t \setminus \delta S^t \setminus Q| \geq k$ **then**
2: $\quad K \leftarrow k$ nodes in $S^t \setminus \delta S^t \setminus Q$ with largest $\underline{\mathbf{r}}^t$;
3: $\quad$ **if** $\min_{i \in K} \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \setminus K \setminus Q} \overline{\mathbf{r}}_i^t$ **and** $\min_{i \in K} \underline{\mathbf{r}}_i^t \geq \mathrm{UB}_Q(\bar{S}^t)$ **then** bStop $\leftarrow$ true;

$\mathrm{PHP}_i(j) \leq \mathrm{PHP}_i(v_i)$. Let $w(\bar{S}^t)$ denote the maximum degree of unvisited nodes in $\bar{S}^t$. We have $w_j \cdot \mathrm{PHP}_i(j) \leq w(\bar{S}^t) \cdot \mathrm{PHP}_i(j) \leq w(\bar{S}^t) \cdot \mathrm{PHP}_i(v_i)$. Plugging it into the equation in Theorem 11, we can get that for any node $j \in \bar{S}^t$,

$$\mathrm{RWR}_Q(j) \leq w(\bar{S}^t) \cdot \textstyle\sum_{i \in Q} \frac{\alpha_i \cdot \mathrm{RWR}_i(i)}{w_i} \cdot \mathrm{PHP}_i(v_i),$$

where $v_i (\in \delta S^t)$ is the node with the largest PHP proximity value in δS^t when node i is set as the single query node. Therefore, if we maintain the maximum degree of the unvisited nodes, we can develop the upper bound for the proximity values of unvisited nodes. We compute the following quantity as the upper bound of the unvisited nodes.

$$\mathrm{UB}_Q(\bar{S}^t) = w(\bar{S}^t) \cdot \textstyle\sum_{i \in Q} \frac{\alpha_i \cdot \mathrm{RWR}_i(i)}{w_i} \cdot \overline{\mathrm{PHP}}_i(v_i),$$

where $\overline{\mathrm{PHP}}_i(v_i) (\geq \mathrm{PHP}_i(v_i))$ denotes the upper bound.

The analysis of the complexity of Algorithm 9 is similar to that of Algorithm 2 in Section 2.5.6. The overall complexity of Algorithm 9 is $O(\alpha h^2 \beta^2 |Q|)$, where α is the number of iterations in the iterative method in Algorithm 7, h is the average number of neighbors of a node, β is the number of iterations in Algorithm 9, and $|Q|$ is the

number of query nodes. The worst case complexity is $O(\alpha n^2|Q|)$.

2.6.3 Extension to RoundTripRank

RoundTripRank (RT) [31] considers round trip paths between nodes. RT of node i is defined as the probability that a round trip from q contains node i. The round trip can be decoupled into two types of trips, i.e., the forward trip from node q to i and the backward trip from node i to q. Let $\mathbf{r}_i$ denote the RT proximity value of node i with regard to the query node q. Let $\mathbf{x}_i$ denote the proximity value defined in the forward direction from q to i, and $\mathbf{y}_i$ denote the proximity value defined in the backward direction from i to q. $\mathbf{x}_i$ and $\mathbf{y}_i$ can be defined as

$$\mathbf{x}_i = \begin{cases} c\sum_{j\in N_i} p_{j,i}\mathbf{x}_j + (1-c), & \text{if } i = q, \\ c\sum_{j\in N_i} p_{j,i}\mathbf{x}_j, & \text{if } i \neq q, \end{cases}$$

$$\mathbf{y}_i = \begin{cases} c\sum_{j\in N_i} p_{i,j}\mathbf{y}_j + (1-c), & \text{if } i = q, \\ c\sum_{j\in N_i} p_{i,j}\mathbf{y}_j, & \text{if } i \neq q, \end{cases}$$

where c $(0<c<1)$ is a constant decay factor, and $p_{i,j} = \frac{w_{i,j}}{w_i}$ is the transition probability from node i to j.

RT can be decomposed as the multiplication of these two values, i.e.,

$$\mathbf{r}_i \propto \mathbf{x}_i^{\beta} \cdot \mathbf{y}_i^{1-\beta},$$

where β $(0 \leq \beta \leq 1)$ is a constant and used to tune the trade-off between the forward and backward directions.

The relationship between RT and PHP is shown in the following theorem.

Theorem 12. $\mathrm{RT}(i) \propto w_i^{\beta} \cdot \mathrm{PHP}(i)$

Proof. We can see that $\mathbf{x}_i$ is exactly the RWR proximity value, and $\mathbf{y}_i$ has a linear relationship with the EI proximity value, i.e., $\mathbf{y}_i = w_q \cdot \mathrm{EI}(i)$. Thus, we have that $\mathrm{RT}(i) \propto \mathrm{RWR}(i)^{\beta} \cdot \mathrm{EI}(i)^{1-\beta}$. From Theorem 2, we have that $\mathrm{EI}(i) \propto \mathrm{PHP}(i)$. From Theorem 9, we have that $\mathrm{RWR}(i) \propto w_i \cdot \mathrm{PHP}(i)$. Thus, $\mathrm{RT}(i) \propto w_i^{\beta} \cdot \mathrm{PHP}(i)$. □

Lemma 12. RT *has no local maximum when* $\beta = 0$*, and has local maximum when* $0 < \beta \leq 1$.

Proof. Based on Theorem 12, when $\beta = 0$, we have that $\text{RT}(i) \propto \text{PHP}(i)$. Since PHP has no local maximum, RT has no local maximum. Examples can be constructed to show that RT has local maximum when $0 < \beta \leq 1$, which are omitted. □

Suppose that node $v \in \delta S^t$ has the largest PHP proximity value. Based on Theorem 1, for any $i \in \bar{S}^t$, we have that $\text{PHP}(i) \leq \text{PHP}(v)$. Let $w(\bar{S}^t)$ denote the maximum degree of unvisited nodes in $\bar{S}^t$. We have that $w_i^\beta \cdot \text{PHP}(i) \leq w(\bar{S}^t)^\beta \cdot \text{PHP}(i) \leq w(\bar{S}^t)^\beta \cdot \text{PHP}(v)$. Therefore, if we maintain the maximum degree of the unvisited nodes, we can develop the upper bound for the proximity values of unvisited nodes.

Specifically, we can apply FLoS to RT as follows. In Algorithm 3, we can change line 1 to the following line.

1: $u \leftarrow \text{argmax}_{i \in \delta S^{t-1}} w_i^\beta \cdot (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;

In Algorithm 6, we can change line 2 and 3 to the following two lines.

2: $K \leftarrow k$ nodes in $S^t \backslash \delta S^t \backslash \{q\}$ with largest $w_i^\beta \cdot \underline{\mathbf{r}}_i^t$ values;

3: **if** $\min_{i \in K} w_i^\beta \cdot \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \backslash K \backslash \{q\}} w_i^\beta \cdot \overline{\mathbf{r}}_i^t$ **and** $\min_{i \in K} w_i^\beta \cdot \underline{\mathbf{r}}_i^t \geq w(\bar{S}^t)^\beta \cdot \max_{i \in \delta S^t} \overline{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true;

All other processes remain the same.

2.6.4 Extension to the Katz Score

The Katz score (KZ) [58] measures the proximity between nodes via a weighted sum of the length of paths between them. Let $\mathbf{R}$ denote the matrix containing all pairwise KZ proximity values (with $\mathbf{R}_{i,j}$ denoting the KZ proximity value between nodes i and j). Let $\mathbf{W}$ be the symmetric adjacency matrix of an undirected and connected graph. We have that

$$\mathbf{R} = \kappa\mathbf{W} + \kappa^2\mathbf{W}^2 + \cdots = (\mathbf{I} - \kappa\mathbf{W})^{-1} - \mathbf{I},$$

where $\kappa\,(0 < \kappa < 1)$ is the decay factor.

In general, KZ has local maximum. But when κ is small, it does not have local maximum. To prove this, we first define a new proximity measure PHP$'$ as follows

$$\mathbf{r}_i = \begin{cases} 1, & \text{if } i = q, \\ c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where c $(0 < c < 1)$ is the decay factor in the random walk process, $p_{i,j} = w_{i,j}/w_{\max}$ denotes the transition probability from node i to j, and $w_{\max}$ denotes the maximum degree among all the nodes. Different from PHP, the transition probability in PHP$'$ is normalized by the maximum degree.

Theorem 13. *If* $\kappa < 1/w_{\max}$, $\mathrm{KZ}(i) \propto \mathrm{PHP}'(i)$.

Proof. For the KZ proximity measure, we already have that $\mathbf{R} = (\mathbf{I} - \kappa\mathbf{W})^{-1} - \mathbf{I}$. The elements in the q-th column of $\mathbf{R}$ represent the KZ proximity values when the query node is q. Thus, the KZ proximity vector is $\mathbf{r} = (\mathbf{I} - \kappa\mathbf{W})^{-1}\mathbf{e} - \mathbf{e}$. Then, we have that

$$\mathbf{r} + \mathbf{e} = \kappa\mathbf{W}(\mathbf{r} + \mathbf{e}) + \mathbf{e}.$$

To derive the relationship between KZ and PHP$'$, we define a new proximity measure KZ$'$ as

$$\mathrm{KZ}'(i) = \begin{cases} \frac{\mathrm{KZ}(i)+1}{w_{\max}}, & \text{if } i = q, \\ \frac{\mathrm{KZ}(i)}{w_{\max}}, & \text{if } i \neq q. \end{cases}$$

Let $\mathbf{r}'$ denote the proximity vector of KZ$'$ when the query node is q. The relationship between KZ and KZ$'$ can be expressed in a matrix form as

$$\mathbf{r}' = (\mathbf{r} + \mathbf{e})/w_{\max}.$$

Then, the proximity vector of KZ$'$ satisfies that

$$\mathbf{r}' = \kappa\mathbf{W}\mathbf{r}' + \mathbf{e}/w_{\max}.$$

Based on this matrix form, KZ$'$ has the following recursive definition

$$\mathbf{r}'_i = \begin{cases} \kappa \cdot w_{\max}\sum_{j\in N_i} p_{i,j}\mathbf{r}'_j + \frac{1}{w_{\max}}, & \text{if } i = q; \\ \kappa \cdot w_{\max}\sum_{j\in N_i} p_{i,j}\mathbf{r}'_j, & \text{if } i \neq q. \end{cases}$$

Since $\kappa < 1/w_{\max}$, we have that $\kappa \cdot w_{\max} < 1$. If we set the decay factor in PHP$'$ as $c = \kappa \cdot w_{\max}$, KZ$'$ and PHP$'$ have the same recursive definition for any node $i \neq q$, and we have that $\frac{\text{PHP}'(i)}{\text{KZ}'(i)} = \frac{\text{PHP}'(q)}{\text{KZ}'(q)}$. Thus, $\frac{\text{PHP}'(i)}{\text{KZ}(i)} = \frac{\text{PHP}'(q)}{\text{KZ}(q)+1}$. When the query node q is fixed, we have that $\text{KZ}(i) \propto \text{PHP}'(i)$. □

Lemma 13. KZ *does not have local maximum when* $\kappa < 1/w_{\max}$*, and has local maximum when* $\kappa \geq 1/w_{\max}$.

Proof. Based on Theorem 13, when $\kappa < 1/w_{\max}$, we have that $\text{KZ}(i) \propto \text{PHP}'(i)$. We can prove that PHP$'$ has no local maximum. The proof is similar to that in Lemma 1. Thus KZ has no local maximum. Examples can be constructed to show that KZ has local maximum when $\kappa \geq 1/w_{\max}$, which are omitted. □

Theorem 13 says that $\text{KZ}(i)$ is proportional to $\text{PHP}'(i)$ when κ is small. Therefore, ranking by KZ is equivalent to ranking by PHP$'$. Comparing with PHP, PHP$'$ only has different transition probabilities, which are normalized by the maximum degree instead of the degree of each node as in PHP. Therefore, the FLoS method is readily applicable to PHP$'$.

2.6.5 Extension to Absorption Probability

In the absorption probability (AP) [119], with probability $p_{i,i}$, the random walker will be absorbed at node i, and with probability $1 - p_{i,i}$, the random walker will follow a random edge out of it. Let $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$ be a diagonal matrix, where $\lambda_i > 0$ is a constant for any node $i = 1, \cdots, n$. The transition probability in AP is defined as

$$p_{i,j} = \begin{cases} \frac{\lambda_i}{\lambda_i + w_i}, & \text{if } i = j, \\ \frac{w_{i,j}}{\lambda_i + w_i}, & \text{if } i \neq j. \end{cases}$$

The AP proximity $r_{i,j}$ of node j with regard to node i is defined as the probability that a random walker starting from node i is absorbed at node j. AP can be defined recursively as

$$r_{i,j} = \begin{cases} \sum_{k \in N_i} p_{i,k} r_{k,j} + p_{i,i}, & \text{if } i = j, \\ \sum_{k \in N_i} p_{i,k} r_{k,j}, & \text{if } i \neq j. \end{cases}$$

Let $\mathbf{R}$ be the matrix of AP proximity values, i.e., $\mathbf{R}_{i,j} = r_{i,j}$. Let $\mathbf{W}$ be the adjacency matrix, and $\mathbf{D}$ be the diagonal matrix where each element $\mathbf{D}_{i,i}$ equals the degree w_i of node i. The above equations can be expressed in a matrix form

$$\mathbf{R} = (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{W}\mathbf{R} + (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{\Lambda}.$$

Thus, we have that

$$\mathbf{R} = (\mathbf{\Lambda} + \mathbf{D} - \mathbf{W})^{-1}\mathbf{\Lambda}.$$

$\mathbf{R}$ is a non-negative matrix with each row summing up to 1 [119]. The elements in the q-th row of $\mathbf{R}$ represent the AP proximity values with regard to the query q.

To extend FLoS to AP, we consider a variant of PHP, which is referred to as PHP$''$. PHP$''$ is defined as

$$\mathbf{r}_i = \begin{cases} 1, & \text{if } i = q, \\ \sum_{j \in N_i} p_{i,j} \mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where $p_{i,j}$ is the transition probability defined in AP.

Compared with PHP, the transition probability in PHP$''$ is changed. In PHP, the transition probability is normalized by degree w_i. In PHP$''$, the transition probability is normalized by $\lambda_i + w_i$. Another difference is that there is no decay factor in PHP$''$. Even though we do not have decay factor in PHP$''$, FLoS is still applicable to PHP$''$. Some key observations are shown as follows.

Lemma 14. PHP$''$ *has no local maximum.*

Proof. Suppose that node i is a local maximum. We have that $\mathbf{r}_i = \sum_{j \in N_i} p_{i,j} \mathbf{r}_j \leq \sum_{j \in N_i} p_{i,j} \mathbf{r}_i = \mathbf{r}_i \cdot \frac{w_i}{\lambda_i + w_i} < \mathbf{r}_i$. We get a contradiction that $\mathbf{r}_i < \mathbf{r}_i$. □

We still use $\mathbf{P}$ to denote the transition probability matrix with

$$\mathbf{P}_{i,j} = \begin{cases} 0, & \text{if } i = q \text{ or } i = j, \\ p_{i,j}, & \text{if } i \neq q \text{ and } i \neq j. \end{cases}$$

Then the proximity $\mathbf{r}$ based on PHP$''$ can be written in the following matrix form

$$\mathbf{r} = \mathbf{P}\mathbf{r} + \mathbf{e}.$$

We can see that the sum of each row of $\mathbf{P}$ is smaller than 1, i.e., $\sum_j \mathbf{P}_{i,j} = \sum_j p_{i,j} = \frac{w_i}{\lambda_i + w_i} < 1$. Thus, we have that $||\mathbf{P}||_\infty < 1$. Based on this, Theorems 3, 4 and 5 are still satisfied. Thus, we can develop the lower and upper bounds in a similar way as that for PHP. Therefore, the FLoS method is applicable to PHP$''$.

AP and PHP$''$ have the following relationship.

Theorem 14. $\mathrm{AP}(i) \propto \lambda_i \cdot \mathrm{PHP}''(i)$

Proof. The elements in the q-th row of $\mathbf{R}$ represent the AP proximity values when the query node is q. Thus, the AP proximity vector is $\mathbf{r} = \mathbf{R}^\top \mathbf{e} = \mathbf{\Lambda}(\mathbf{\Lambda} + \mathbf{D} - \mathbf{W})^{-1}\mathbf{e}$, where $\mathbf{R}^\top$ represents the transpose of $\mathbf{R}$. Then, $(\mathbf{\Lambda} + \mathbf{D} - \mathbf{W})\mathbf{\Lambda}^{-1}\mathbf{r} = \mathbf{e}$. Thus, the proximity vector of AP satisfies that

$$\mathbf{\Lambda}^{-1}\mathbf{r} = (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{W}\mathbf{\Lambda}^{-1}\mathbf{r} + (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{e}.$$

We define a new proximity measure AP$'$ as $\mathrm{AP}'(i) = \frac{\mathrm{AP}(i)}{\lambda_i}$, for any node $i \in V$.

Let $\mathbf{r}'$ denote the proximity vector of AP$'$ when the query is q. The relationship between AP and AP$'$ can be expressed in a matrix form as $\mathbf{r}' = \mathbf{\Lambda}^{-1}\mathbf{r}$. Thus, the proximity vector of AP$'$ satisfies the following equation

$$\mathbf{r}' = (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{W}\mathbf{r}' + (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{e}.$$

Based on this matrix form, AP$'$ has the following recursive definition

$$\mathbf{r}'_i = \begin{cases} \sum_{j \in N_i} p_{i,j}\mathbf{r}'_j + \frac{1}{\lambda_i + w_i}, & \text{if } i = q, \\ \sum_{j \in N_i} p_{i,j}\mathbf{r}'_j, & \text{if } i \neq q. \end{cases}$$

AP$'$ and PHP$''$ have the same recursive definition for any node $i \neq q$. Thus, $\frac{\mathrm{AP}'(i)}{\mathrm{PHP}''(i)} = \frac{\mathrm{AP}'(q)}{\mathrm{PHP}''(q)}$. Since $\mathrm{AP}'(i) = \frac{\mathrm{AP}(i)}{\lambda_i}$, we have that $\frac{\mathrm{AP}(i)}{\lambda_i \cdot \mathrm{PHP}''(i)} = \frac{\mathrm{AP}(q)}{\lambda_q \cdot \mathrm{PHP}''(q)}$. When q is fixed, we have that $\mathrm{AP}(i) \propto \lambda_i \cdot \mathrm{PHP}''(i)$. □

Lemma 15. AP *has local maximum.*

Proof. Examples can be constructed to show that AP has local maximum and are omitted here. □

Suppose that node $v \in \delta S^t$ has the largest PHP$''$ proximity value. Based on Theorem 1, for any node $i \in \bar{S}^t$, $\mathrm{PHP}''(i) \leq \mathrm{PHP}''(v)$. Let $\lambda(\bar{S}^t)$ denote the maximum

λ value of unvisited nodes in $\bar{S}^t$. We have that $\lambda_i \cdot \text{PHP}''(i) \leq \lambda(\bar{S}^t) \cdot \text{PHP}''(i) \leq \lambda(\bar{S}^t) \cdot \text{PHP}''(v)$. Therefore, if we maintain the maximum λ value of the unvisited nodes, we can develop the upper bound for the proximity values of unvisited nodes.

Specifically, we can apply FLoS to AP as follows. In Algorithm 3, we can change line 1 to the following line.

1: $u \leftarrow \text{argmax}_{i \in \delta S^{t-1}} \lambda_i \cdot (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;

In Algorithm 6, we can change line 2 and 3 to the following two lines.

2: $K \leftarrow k$ nodes in $S^t \backslash \delta S^t \backslash \{q\}$ with largest $\lambda_i \cdot \underline{\mathbf{r}}_i^t$ values;

3: **if** $\min_{i \in K} \lambda_i \cdot \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \backslash K \backslash \{q\}} \lambda_i \cdot \overline{\mathbf{r}}_i^t$ **and** $\min_{i \in K} \lambda_i \cdot \underline{\mathbf{r}}_i^t \geq \lambda(\bar{S}^t) \cdot \max_{i \in \delta S^t} \overline{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true;

All other processes remain the same.

2.7 Top-k Reverse-Proximity Query Problem

In this section, we study the top-k reverse-proximity query problem [10] and discuss how FLoS can be applied to solve it efficiently.

Given a query node q, we can compute the proximity values of all other nodes. We can also use each node i as the query, and compute the proximity value of q. We refer to this proximity of node q as the reverse proximity of node i. The top-k reverse-proximity query problem aims to find the top-k nodes that are ranked by the reverse proximity.

Note that the top-k reverse-proximity query problem is different from the reverse top-k problem studied in [126]. Given a query node q, the reverse top-k problem aims at finding all the nodes that have q in their top-k proximity sets. In this chapter, we study the top-k reverse-proximity query problem, which aims at finding the top-k nodes ranked by the reverse proximity [10].

The top-k reverse-proximity query problem has been studied when RWR is used as the proximity measure [10]. In a recent paper [31], the original and reverse proximity

values in RWR are interpreted as importance and specificity respectively. If node i has large RWR proximity value when the query node is q, node i is important for node q. On the other hand, if node q has large RWR proximity value when the query node is i, node i is specific for node q. The authors show that ranking by the combination of two directions performs better than ranking by one direction.

The naive method to solve the top-k reverse-proximity query problem is as follows. First, each node is used as the query node, and the proximity value of node q is computed by the iterative method. Then the top-k nodes with largest reverse proximity values are selected. Suppose that the iterative method takes $O(\alpha m)$ for each query node. The naive method takes time $O(\alpha mn)$, where α is the number of iterations in the iterative method, m is the number of edges, and n is the number of nodes. This is expensive and prohibitive for large graphs.

For the RWR proximity measure, it is shown that the reverse proximity vector can be computed using the iterative method, which has the same complexity $O(\alpha m)$ as computing the original proximity vector [126]. However, the iterative method is still expensive since it needs to iterate over the entire graph. Moreover, it is unclear how to compute the reverse proximity vectors for other measures in a similar way.

Now we show how to apply the FLoS method to solve the top-k reverse-proximity query problem. Let $\mathrm{RWR}_i(j)$, $\mathrm{EI}_i(j)$, $\mathrm{PHP}_i(j)$, $\mathrm{DHT}_i(j)$, $\mathrm{RT}_i(j)$, and $\mathrm{AP}_i(j)$ denote the RWR, EI, PHP, DHT, RT, and AP proximity values of node j when the query is node i.

2.7.1 Extension to Reverse RWR

In this subsection, we show that ranking by the reverse RWR proximity is the same as ranking by the PHP proximity. Thus the FLoS method for PHP can be directly applied to find the top-k nodes for reverse RWR.

Theorem 15. PHP *and reverse* RWR *give the same ranking results.*

Proof. From Theorem 2, we have that PHP and EI give the same ranking results. Thus, it is equivalent to prove that EI and reverse RWR give the same ranking results.

EI is defined as the degree normalized RWR, i.e., $\mathrm{EI}_q(i) = \frac{\mathrm{RWR}_q(i)}{w_i}$. Thus, we have that $\mathrm{RWR}_q(i) = w_i \cdot \mathrm{EI}_q(i)$. Therefore, we have that $\mathrm{RWR}_i(q) = w_q \cdot \mathrm{EI}_i(q) = w_q \cdot \mathrm{EI}_q(i)$, where we use the fact that EI is symmetric, i.e., $\mathrm{EI}_i(q) = \mathrm{EI}_q(i)$. Thus, $\mathrm{RWR}_i(q) \propto \mathrm{EI}_q(i)$, when the query q is fixed. This completes the proof. □

2.7.2 Extension to Reverse PHP and Reverse DHT

In this subsection, we first show how to solve the top-k reverse PHP problem. Then, we prove that reverse PHP and reverse DHT give the same ranking results.

To find the top-k nodes for reverse PHP, we use the following relationship between PHP and reverse PHP.

Theorem 16. $\mathrm{PHP}_i(q) \propto \frac{\mathrm{PHP}_q(i)}{\mathrm{EI}_i(i)}$

Proof. From Theorem 2, we have that $\mathrm{PHP}_q(i) = \frac{\mathrm{EI}_q(i)}{\mathrm{EI}_q(q)}$. Thus, $\mathrm{PHP}_i(q) = \frac{\mathrm{EI}_i(q)}{\mathrm{EI}_i(i)} = \frac{\mathrm{EI}_q(i)}{\mathrm{EI}_i(i)} = \mathrm{EI}_q(q) \cdot \frac{\mathrm{PHP}_q(i)}{\mathrm{EI}_i(i)}$, where we use the fact that EI is symmetric, i.e., $\mathrm{EI}_i(q) = \mathrm{EI}_q(i)$. Therefore, we have that $\mathrm{PHP}_i(q) \propto \frac{\mathrm{PHP}_q(i)}{\mathrm{EI}_i(i)}$ when the query node q is fixed. □

Suppose that we already pre-compute the values $\mathrm{EI}_i(i)$ for each node i, and node $v \in \delta S^t$ has the largest PHP proximity value. Based on Theorem 1, for any node $i \in \bar{S}^t$, $\mathrm{PHP}_q(i) \leq \mathrm{PHP}_q(v)$. Let $\mathrm{EI}(\bar{S}^t)$ denote the minimum $\mathrm{EI}_i(i)$ value of unvisited nodes i in $\bar{S}^t$. We have that $\frac{\mathrm{PHP}_q(i)}{\mathrm{EI}_i(i)} \leq \frac{\mathrm{PHP}_q(i)}{\mathrm{EI}(\bar{S}^t)} \leq \frac{\mathrm{PHP}_q(v)}{\mathrm{EI}(\bar{S}^t)}$. Therefore, if we maintain the minimum $\mathrm{EI}_i(i)$ value of the unvisited nodes, we can develop the upper bound for the proximity values of unvisited nodes.

Specifically, we can apply FLoS to reverse PHP as follows. In Algorithm 3, we can change line 1 to the following line.

1: $u \leftarrow \mathrm{argmax}_{i \in \delta S^{t-1}} \frac{1}{\mathrm{EI}_i(i)} \cdot (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;

In Algorithm 6, we can change line 2 and 3 to the following two lines.

2: $K \leftarrow k$ nodes in $S^t \setminus \delta S^t \setminus \{q\}$ with largest $\frac{\mathbf{r}_i^t}{\mathrm{EI}_i(i)}$ values;
3: **if** $\min_{i \in K} \frac{\mathbf{r}_i^t}{\mathrm{EI}_i(i)} \geq \max_{i \in S^t \setminus K \setminus \{q\}} \frac{\overline{\mathbf{r}}_i^t}{\mathrm{EI}_i(i)}$ **and** $\min_{i \in K} \frac{\mathbf{r}_i^t}{\mathrm{EI}_i(i)} \geq (\mathrm{EI}(\bar{S}^t))^{-1} \cdot \max_{i \in \delta S^t} \overline{\mathbf{r}}_i^t$ **then**
bStop $\leftarrow$ true;

All other processes remain the same.

Theorem 17. *Reverse* PHP *and reverse* DHT *give the same ranking results.*

Proof. From Theorem 2, DHT and PHP have the relationship $\mathrm{DHT}_q(i) = \frac{1}{1-c}(1 - \mathrm{PHP}_q(i))$. Thus, $\mathrm{DHT}_i(q) = \frac{1}{1-c}(1 - \mathrm{PHP}_i(q))$. It means that reverse DHT is a linear function of reverse PHP. Thus reverse DHT and reverse PHP give the same ranking results. □

2.7.3 Extension to Reverse RT

In this subsection, we show how to find the top-k nodes for reverse RT.

Theorem 18. $\mathrm{RT}_i(q) \propto w_i^{1-\beta} \cdot \mathrm{PHP}_q(i)$

Proof. Let $\mathrm{RTF}_i(j)$ represent the forward proximity value defined in the direction from i to j. Thus, we have that $\mathrm{RTF}_q(i) = \mathbf{x}_i$, where $\mathbf{x}_i$ is defined in Section 2.6.3. $\mathrm{RTF}_q(i)$ is exactly the RWR proximity value, i.e., $\mathrm{RTF}_q(i) = \mathrm{RWR}_q(i)$. Let $\mathrm{RTB}_i(j)$ represent the backward proximity value defined in the direction from j to i. Thus, we have that $\mathrm{RTB}_q(i) = \mathbf{y}_i$, where $\mathbf{y}_i$ is defined in Section 2.6.3. $\mathrm{RTB}_q(i)$ has a linear relationship with the EI proximity value, i.e., $\mathrm{RTB}_q(i) = w_q \cdot \mathrm{EI}_q(i)$.

RT can be decomposed as

$$\mathrm{RT}_q(i) \propto \mathrm{RTF}_q(i)^\beta \cdot \mathrm{RTB}_q(i)^{1-\beta}.$$

Reverse RT can be decomposed as

$$\begin{aligned}\mathrm{RT}_i(q) &\propto \mathrm{RTF}_i(q)^\beta \cdot \mathrm{RTB}_i(q)^{1-\beta} \\ &\propto \mathrm{RWR}_i(q)^\beta \cdot w_i^{1-\beta} \cdot \mathrm{EI}_i(q)^{1-\beta}.\end{aligned}$$

From Theorem 15, we have that $\mathrm{RWR}_i(q) \propto \mathrm{EI}_q(i)$. Since EI is symmetric, we have that $\mathrm{EI}_i(q) = \mathrm{EI}_q(i)$. From Theorem 2, we have that $\mathrm{EI}_q(i) \propto \mathrm{PHP}_q(i)$. Thus, we have that $\mathrm{RT}_i(q) \propto w_i^{1-\beta} \cdot \mathrm{PHP}_q(i)$. □

The relationship between PHP and reverse RT in Theorem 18 is quite similar to the relationship between PHP and RT in Theorem 12. We only need to change β to $(1-\beta)$ in the FLoS method for RT to apply it for reverse RT.

2.7.4 Extension to Reverse AP

In this subsection, we show how to find the top-k nodes for reverse AP.

Theorem 19. PHP$''$ *and reverse* AP *give the same ranking results.*

Proof. In Section 2.6.5, we have that the proximity matrix of AP is $\mathbf{R} = (\mathbf{\Lambda} + \mathbf{D} - \mathbf{W})^{-1}\mathbf{\Lambda}$. The elements in the q-th column of $\mathbf{R}$ represent the reverse AP proximity values when the query node is q. Thus, the reverse AP proximity vector is $\mathbf{r} = \mathbf{R}\mathbf{e} = (\mathbf{\Lambda}+\mathbf{D}-\mathbf{W})^{-1}\mathbf{\Lambda}\mathbf{e}$. Then, $(\mathbf{\Lambda}+\mathbf{D}-\mathbf{W})\mathbf{r} = \mathbf{\Lambda}\mathbf{e}$. Thus, the proximity vector of reverse AP satisfies the following equation

$$\mathbf{r} = (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{W}\mathbf{r} + (\mathbf{\Lambda} + \mathbf{D})^{-1}\mathbf{\Lambda}\mathbf{e}.$$

Based on this matrix form, reverse AP has the following recursive definition

$$\mathbf{r}_i = \begin{cases} \sum_{j\in N_i} p_{i,j}\mathbf{r}_j + \frac{\lambda_i}{\lambda_i+w_i}, & \text{if } i = q, \\ \sum_{j\in N_i} p_{i,j}\mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where $p_{i,j} = \frac{w_{i,j}}{\lambda_i+w_i}$ is the transition probability from node i to j.

PHP$''$ and reverse AP have the same recursive definition for any node $i \neq q$, thus $\frac{\mathrm{AP}_i(q)}{\mathrm{PHP}''_q(i)} = \frac{\mathrm{AP}_q(q)}{\mathrm{PHP}''_q(q)}$. When the query node q is fixed, $\mathrm{AP}_i(q) \propto \mathrm{PHP}''_q(i)$. This completes the proof. □

EI and KZ both are symmetric, thus the top-k results for reverse proximity are the same as the top-k results for the original proximity.

2.8 Experimental Results

In this section, we present extensive experimental results on evaluating the performance of the FLoS algorithm. The datasets are shown in Table 2.4. The real datasets

Table 2.4: Datasets used in the experiments

Datasets		Abbr.	#Nodes	#Edges
Real	Amazon	AZ	334,863	925,872
	DBLP	DP	317,080	1,049,866
	Youtube	YT	1,134,890	2,987,624
	LiveJournal	LJ	3,997,962	34,681,189
Synthetic	In-memory	–	Varying size	
		–	Varying density	
	Disk-resident	–	Varying size	

are publicly available at the website [72]. The synthetic datasets are generated using the Erdös-Rényi random graph (RAND) model [29] and R-MAT model [18] with different parameters. All programs are written in C++. All experiments are performed on a server with 32G memory, Intel Xeon 3.2GHz CPU, and Redhat 4.1.2 OS.

2.8.1 State-of-the-Art Methods

The measures we use include PHP, EI, RWR, RT, KZ, THT, and AP. We compare FLoS with the state-of-the-art methods for each measure as summarized in Table 2.5. These methods are categorized into global and local methods.

The global iteration (GI) method directly applies the iterative method on the entire graph [95]. It guarantees to find the exact top-k nodes. The graph embedding (GE) method can answer the query in constant time after embedding [129]. It can only be applied to RWR. However, the embedding process is very time consuming. Moreover, it only returns approximate results. The Castanet algorithm is specifically designed for RWR. It improves the GI method and guarantees the exactness of the results [35]. AA_KZ improves the global iteration method by prioritized execution of the iterative computation and also guarantees the exactness of the results [61]. K-dash is the state-of-the-art matrix-based method for RWR which guarantees result exactness [34]. Note that K-dash and GE can only be applied on two medium-sized real graphs because of the expensive preprocessing step.

Dynamic neighborhood expansion (DNE) method applies a best-first expansion strategy to find the top-k nodes using PHP [128]. This strategy is heuristic and

Table 2.5: State-of-the-art methods used for comparison

Our methods (Exact)	State-of-the-art methods			
	Abbr.	Key idea	Ref.	Exactness
FLoS_PHP	GI_PHP	Global iteration	[95]	Exact
	DNE	Local search	[128]	Approx.
	NN_EI	Local search	[14]	Exact
	LS_EI	Local search	[98]	Approx.
FLoS_RWR	GI_RWR	Global iteration	[95]	Exact
	GE_RWR	Graph embedding	[129]	Approx.
	Castanet	Improved GI	[35]	Exact
	K-dash	Matrix inversion	[34]	Exact
	LS_RWR	Local search	[98]	Approx.
FLoS_RT	GI_RT	Global iteration	[95]	Exact
	LS_RT	Local search	[31]	Approx.
FLoS_KZ	GI_KZ	Global iteration	[95]	Exact
	LS_KZ	Local search	[30]	Approx.
	AA_KZ	Improved GI	[61]	Exact
FLoS_THT	GI_THT	Global iteration	[95]	Exact
	LS_THT	Local search	[97]	Approx.
FLoS_AP	GI_AP	Global iteration	[95]	Exact
FLoS_rPHP	GI_rPHP	Global iteration	[95]	Exact

does not guarantee to find the exact solution. The number of visited nodes is fixed to $4,000$ in the experiments. NN_EI applies the push style method [11, 19] in local search, and guarantees the exactness of the top-k results [14]. Since PHP and EI are equivalent in terms of ranking, we can compare the methods for PHP and EI directly. LS_RWR applies the dynamic programming technique [51] to develop bounds in local search [98]. It returns approximate results. LS_EI is based on LS_RWR and has similar performance [98]. LS_RT leverages the push style method [11] developed for RWR to estimate the bounds and find the approximate top-k nodes with largest RoundTripRank proximity values. LS_KZ locally searches a small portion of the graph and adapts the push style method [11] to find the approximate top-k results for the Katz score. LS_THT is a local search method for THT [97].

The decay factors in PHP, RWR, EI, and RT are all set to 0.5. The decay factor in KZ is set to $0.99/w_{\max}$. In RT, we set the parameter $\beta = 0.4$. In AP, we set the parameter $\lambda_i = 10$ for any node i. The truncated length in THT is set to 10.

We use FLoS_rPHP to denote the FLoS method for reverse PHP. Reverse RWR gives the same ranking as PHP, so we only evaluate FLoS_PHP. The FLoS method for reverse RT is quite similar to that for RT, thus we only evaluate FLoS_RT. When we set the parameter $\lambda_i = 10$ for any node i, AP becomes symmetric. Thus AP and reverse AP give the same ranking results, and we only evaluate FLoS_AP.

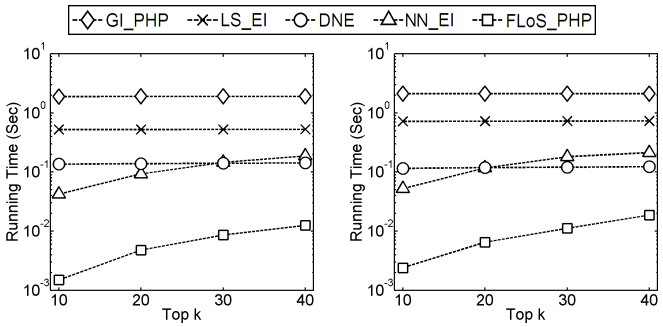
2.8.2 Evaluation on Real Graphs

We study the efficiency of the selected methods on real graphs when varying the number of returned nodes k. For each k, we repeat the experiments 10^3 times, each with a randomly picked query node. The average running time is reported. For methods using the iteration procedure in Algorithm 7, the termination threshold is set to $\tau = 10^{-5}$. We also perform experiments using a fixed number of 10 iterations. The results are similar and omitted here.

2.8.2.1 Evaluation of FLoS_PHP

Figure 2.14 shows the running time of different methods for PHP. The running time of DNE is almost a constant for different k, because it visits a fixed number of nodes. The running time of NN_EI increases when k increases. FLoS_PHP is more efficient than NN_EI, which demonstrates that the bounds of FLoS are tighter. LS_EI has a constant running time. This is because it extracts the cluster containing the query node. Note that LS_EI takes tens of hours in the preprocessing step to cluster the graphs.

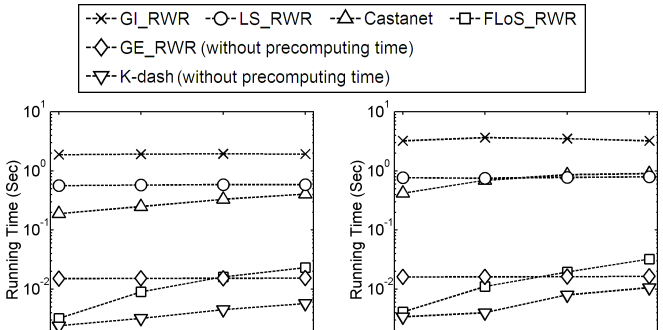
Figure 2.16(a) shows the ratio between the number of visited nodes using F-LoS_PHP and total number of nodes in the graph. The value indicated by the bar is the average ratio of 10^3 queries. The minimum and maximum ratios are also shown in the figure. As can be seen from the figure, only a very small part of the graph is needed for FLoS to find the exact solution. Moreover, the ratio decreases when the graph size increases. This indicates that FLoS is more effective for larger graphs.



(a) AZ (b) DP

(c) YT (d) LJ

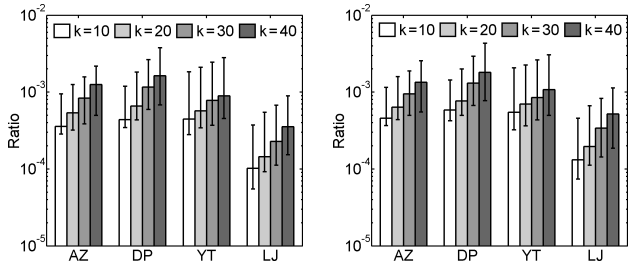
Figure 2.14: Running time of different methods for PHP on real graphs



(a) AZ (b) DP

(c) YT (d) LJ

Figure 2.15: Running time of different methods for RWR on real graphs



(a) FLoS_PHP (b) FLoS_RWR

Figure 2.16: Ratio between the number of visited nodes and the total number of nodes on real graphs

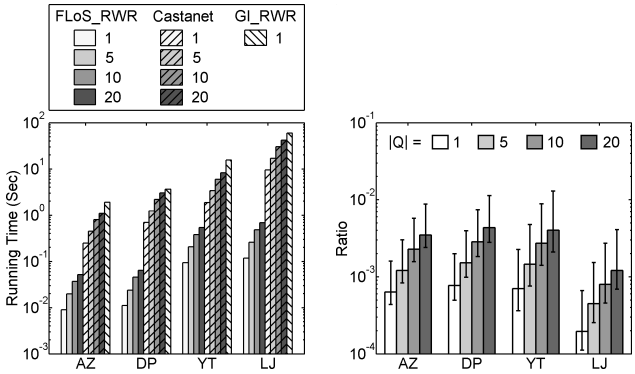
2.8.2.2 Evaluation of FLoS_RWR

Figure 2.15 shows the running time for RWR. K-dash has the best performance after precomputing the matrix inversion as shown in Figures 2.15(a) and 2.15(b). The precomputing step of K-dash takes tens of hours for the medium-sized AZ and DP graphs and cannot be applied to the other two larger graphs. GE_RWR also has fast response time. However, as discussed before, its embedding step is time consuming and not applicable to larger graphs. Moreover, it does not find the exact solution. Castanet method cuts the running time from the GI method by 72% to 91%. LS_RWR method has constant running time, and it needs tens of hours in the precomputing step to cluster the graphs.

Figure 2.16(b) shows the ratio of the number of visited nodes of the FLoS_RWR method. The results are similar to that of Figure 2.16(a).

2.8.2.3 Evaluation of FLoS_RWR (Multiple Query Nodes)

We randomly set 1, 5, 10 or 20 nodes as the query nodes. The preference scores are set to the same value. For each number of query nodes, we repeat the experiments 10^3 times. The average running time is reported. For the AZ and DP graphs, we pre-compute the exact values $\text{RWR}_i(i)$ for each node i by the K-dash method [34]. For the YT and LJ graphs, we only pre-compute the exact values $\text{RWR}_i(i)$ for some nodes, which will be used as query nodes, by the GI_RWR method. Among the baseline methods for RWR, only GI_RWR and Castanet can handle several query



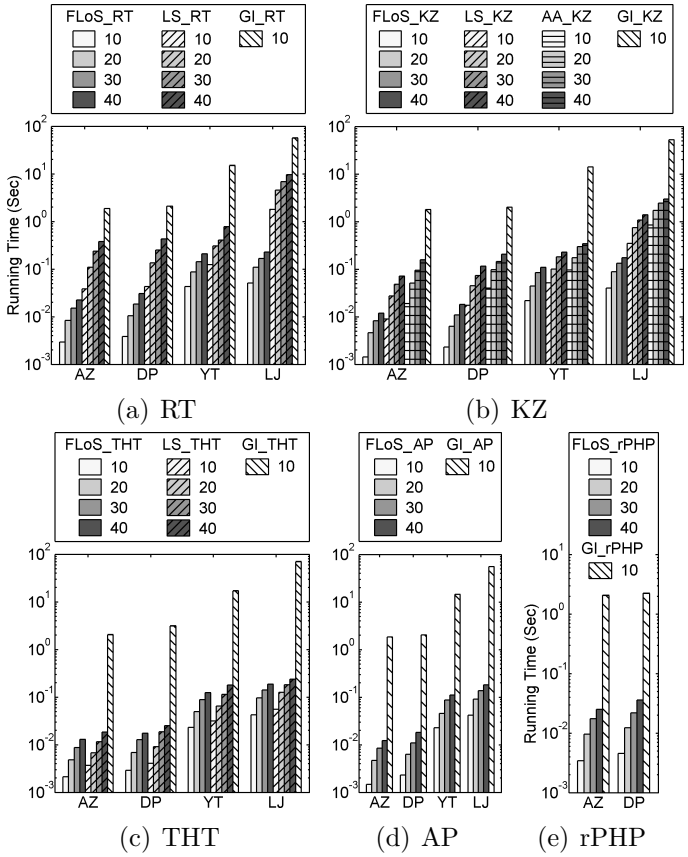
(a) running time (b) visited nodes

Figure 2.17: Results of FLoS_RWR for multiple query nodes on real graphs ($k = 20$, number of query nodes $|Q| = 1, 5, 10, 20$)

nodes. Figure 2.17(a) shows the running time of selected methods for RWR. We only show the running time when there is only one query node for the GI_RWR method since it has almost constant running time for different number of query nodes. FLoS_RWR and Castanet both have increasing running time when increasing the number of query nodes. We can see that FLoS_RWR still runs efficiently and is about 1-2 orders of magnitude faster than Castanet and GI_RWR. Figure 2.17(b) shows the ratio of the number of visited nodes of the FLoS_RWR method. We can see that the number of visited nodes is increasing when increasing the number of query nodes.

2.8.2.4 Evaluation of FLoS_RT

Figure 2.18(a) shows the running time of different methods for RT. The number on the right side of the rectangle legend indicates the value of k. Since the GI_RT method has almost constant running time for different k, we only show the result when $k = 10$. The running time of FLoS_RT and LS_RT increases when k increases. FLoS_RT is the most efficient method. FLoS_RT is about 1 order of magnitude faster than LS_RT, and 2 orders of magnitude faster than GI_RT. LS_RT uses the push style method to develop the bounds, which are looser than those of FLoS_RT.



(a) RT (b) KZ

(c) THT (d) AP (e) rPHP

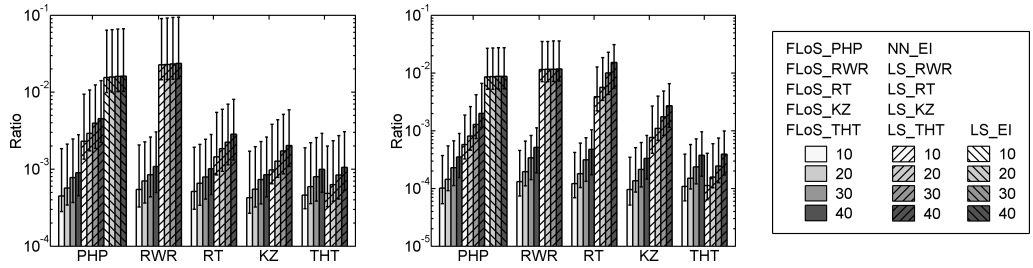
Figure 2.18: Running time of different methods for RT, KZ, THT, AP, and rPHP on real graphs

2.8.2.5 Evaluation of FLoS_KZ

Figure 2.18(b) shows the running time of different methods for KZ. We also only show the running time when $k = 10$ for the GI_KZ method since it has almost constant running time for different k. FLoS_KZ, LS_KZ, and AA_KZ methods all have increasing running time when increasing k. FLoS_KZ is about 1-2 orders of magnitude faster than LS_KZ and AA_KZ. LS_KZ uses the push style method to develop the bounds, which are not as tight as those of FLoS_KZ. The results also demonstrate that the bounds in AA_KZ are looser than those of FLoS_KZ.

2.8.2.6 Evaluation of FLoS_THT

Figure 2.18(c) shows the running time for THT. As we can see, FLoS_THT runs faster than LS_THT, which is specifically designed to speed up the computation for THT.



(a) YT (b) LJ

Figure 2.19: Ratio between the number of visited nodes and the total number of nodes on real graphs for the local search methods

This is because the lower and upper bounds of FLoS_THT are tighter than those of LS_THT. Both of the two local search methods are 2 to 3 orders of magnitude faster than GI_THT.

2.8.2.7 Evaluation of FLoS_AP

Figure 2.18(d) shows the running time for AP. Similar to the results of other proximity measures, FLoS_AP runs 2-3 orders of magnitude faster than the GI_AP method.

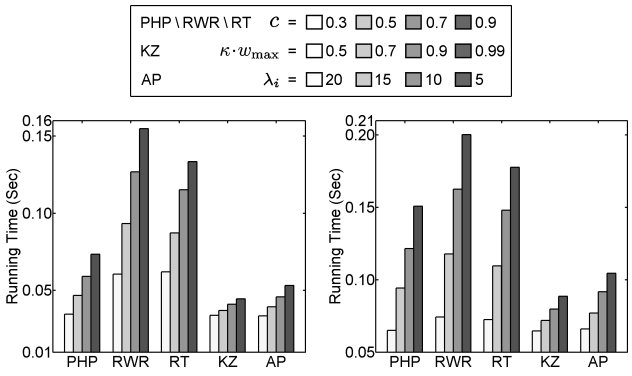
2.8.2.8 Evaluation of FLoS_rPHP

In FLoS_rPHP, we pre-compute the exact values $\mathrm{EI}_i(i)$ for each node i by the K-dash method [34]. The precomputation step takes 28.5 and 34.6 hours for two medium-sized graphs, AZ and DP. Thus we did not apply FLoS_rPHP on the large graphs.

Figure 2.18(e) shows the running time of our local search method and the global iteration method for reverse PHP. Similar to the results for other proximity measures, FLoS_rPHP runs 2-3 orders of magnitude faster than the GI_rPHP method.

2.8.2.9 Number of Visited Nodes (Local Search Methods)

In this section, we study the number of visited nodes of different local search methods on real graphs. The number of visited nodes in the DNE method is fixed, thus it is not included. Figure 2.19(a) shows the ratio between the number of visited nodes using different local search methods and total number of nodes in the YT graph. Figure



(a) YT (b) LJ

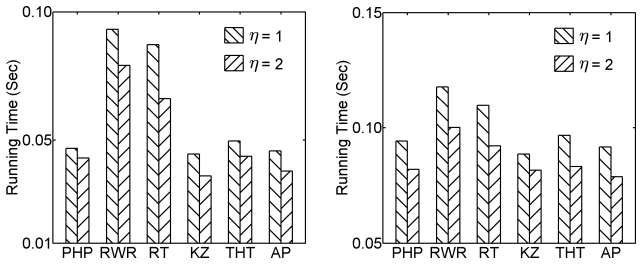
Figure 2.20: Running time of different parameters on real graphs ($k = 20$)

2.19(b) shows that in the LJ graph. The value indicated by the bar is the average ratio of 10^3 queries. The minimum and maximum ratios are also shown in the figure. As can be seen from the figure, other local search methods need to visit larger number of nodes than the FLoS methods do. This demonstrates the tightness of the bounds in the FLoS methods. We also can observe that the LS_EI and LS_RWR methods visit relatively large number of nodes and the ratio is stable when the number k changes. This is because in each expansion of the LS_EI and LS_RWR methods, all the nodes in one cluster will be visited. Thus, they need to visit larger number of nodes.

2.8.2.10 Effect of Different Parameters

In this section, we study how the different settings of parameters in the proximity measures will affect the running time. For the PHP, RWR and RT proximity measures, the decay factor c is set to 0.3, 0.5, 0.7, or 0.9. For the KZ proximity measure, the parameter κ is set to $0.5/w_{\max}$, $0.7/w_{\max}$, $0.9/w_{\max}$, or $0.99/w_{\max}$. For the AP proximity measure, the parameter λ_i is set to 20, 15, 10, or 5.

Figures 2.20(a) and 2.20(b) show the running time on the YT and LJ graphs respectively. We can see that the running time is changing monotonically when we tune the parameters in different proximity measures. For example, in PHP, when the decay factor is larger, FLoS takes longer time. When the decay factor is larger, the bounds become looser, thus, more nodes need to be visited to distinguish the top-k



(a) YT (b) LJ

Figure 2.21: Effect of η in the relaxed termination criterion on two large real graphs ($k = 20$)

nodes from the remaining nodes. Similar analysis is applicable for the other proximity measures. We also can observe that the increasing rate in RWR and RT is larger. It may be because of the multiplication of the node degree in deriving the bounds.

2.8.2.11 Evaluation of the Relaxed Termination Criterion

The parameter η controls the maximum number of remaining candidate nodes allowed when the computation is terminated. A larger η helps to decrease the running time in exchange for a less precise output, i.e., the output set is larger than k. Thus, we study how the value η can affect the running time. Figure 2.21(a) shows the results on the YT graph. We can see that with value $\eta = 2$, the running time can be reduced to 70%-90% of the original running time. Figure 2.21(b) shows the results on the LJ graph. Similar trends can be observed.

2.8.3 Evaluation on In-Memory Synthetic Graphs

We generate synthetic graphs with different parameters to evaluate the selected methods. More specifically, we study two types of graphs: Erdös-Rényi random graph (RAND) [29] and scale-free graph based on the R-MAT model [18]. There are two parameters, the size and density of the graphs. We study how these two parameters affect the running time of different methods for PHP, RWR, RT, and KZ.

We download the graph generator available from the website `https://github.com/dhruvbird/GTgraph` and use the default parameters to generate two series of graphs with varying size and varying density, using RAND and R-MAT respectively. The

Table 2.6: Statistics of in-memory synthetic graphs

Varying size	#Nodes	1×2^{20}	2×2^{20}	4×2^{20}	8×2^{20}
	#Edges	1×10^7	2×10^7	4×10^7	8×10^7
	Density	9.5	9.5	9.5	9.5
Varying density	#Nodes	1×2^{20}	1×2^{20}	1×2^{20}	1×2^{20}
	#Edges	5×10^6	10×10^6	15×10^6	20×10^6
	Density	4.8	9.5	14.3	19.1

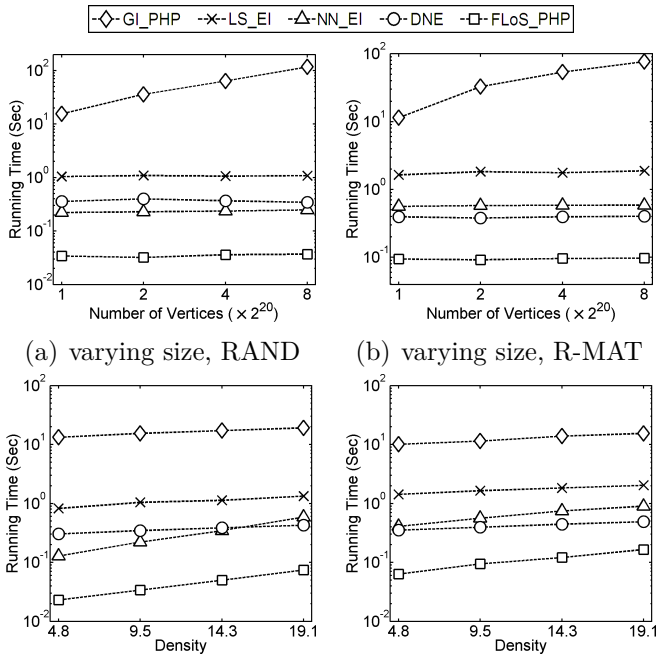
graphs with varying size have the same density but different number of nodes. The graphs with varying density have the same number of nodes but different densities. The statistics are shown in Table 2.6.

We apply the selected methods for PHP, RWR, RT and KZ on these graphs with $k = 20$. For each graph, we repeat the query 10^3 times with randomly picked query nodes, and report the average running time.

2.8.3.1 Evaluation of FLoS_PHP

Figure 2.22(a) shows the running time of the selected methods for PHP on the series of RAND graphs with varying size. The running time of GI_PHP increases as the number of nodes increases. FLoS_PHP, DNE, NN_EI and LS_EI all have almost constant running time when the number of nodes increases. This is because these methods only search locally. When the density of the graph is fixed, adding more nodes to the graph will not change the size of the search space of these methods. Figure 2.22(b) shows the running time on the series of R-MAT graphs with varying size. Similar trends are observed. Comparing Figure 2.22(a) and 2.22(b), GI_PHP has less running time on R-MAT than on RAND graphs, while other methods have more. The reason is that R-MAT graphs have the power-law distribution, thus it is easier for FLoS_PHP, DNE, NN_EI and LS_EI to encounter hub nodes with larger degree when expanding subgraph. The faster performance of GI_PHP on R-MAT may be because of the greater data locality due to the hub node.

Figure 2.22(c) shows the running time of the selected methods for PHP on the series of RAND graphs with varying density. The running time of all the methods



(a) varying size, RAND (b) varying size, R-MAT

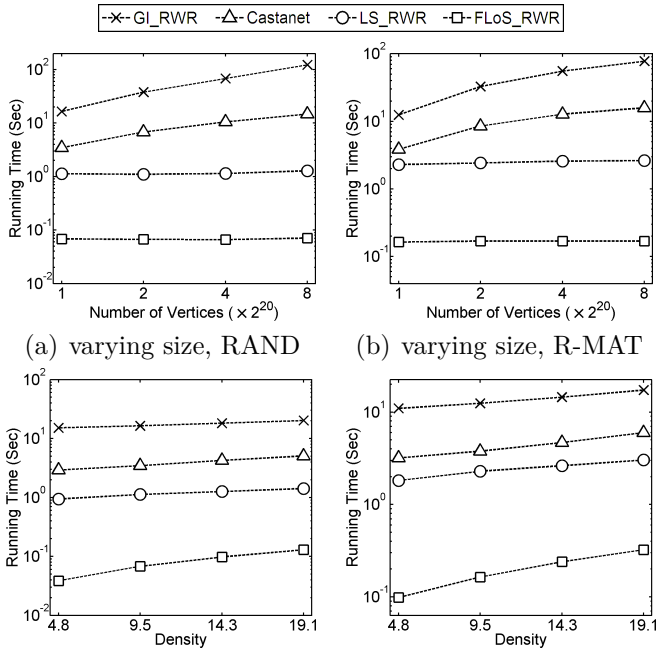
(c) varying density, RAND (d) varying density, R-MAT

Figure 2.22: Running time of different methods for PHP on in-memory synthetic graphs ($k = 20$)

increases as the density increases. FLoS_PHP and NN_EI have increasing running time because the number of visited nodes in these two methods increases when the density becomes larger. LS_EI has increasing running time because the number of nodes and edges increases in local clusters. Figure 2.22(d) shows the running time on the series of R-MAT graphs with varying density. Similar trends are observed.

2.8.3.2 Evaluation of FLoS_RWR

Figure 2.23(a) shows the running time of the selected methods for RWR on the series of RAND graphs with varying size. The running time of GI_RWR and Castanet increases as the number of nodes increases. Castanet method cuts the running time from the GI method by 69% to 88%. FLoS_RWR and LS_RWR both have almost constant running time when the number of nodes increases. This is because FLoS_RWR and LS_RWR only search locally. Figure 2.23(b) shows the running time on the series of R-MAT graphs with varying size. Similar trends are observed. Comparing Figure 2.23(a) and 2.23(b), GI_RWR has less running time on the R-MAT graphs than on



(a) varying size, RAND (b) varying size, R-MAT

(c) varying density, RAND (d) varying density, R-MAT

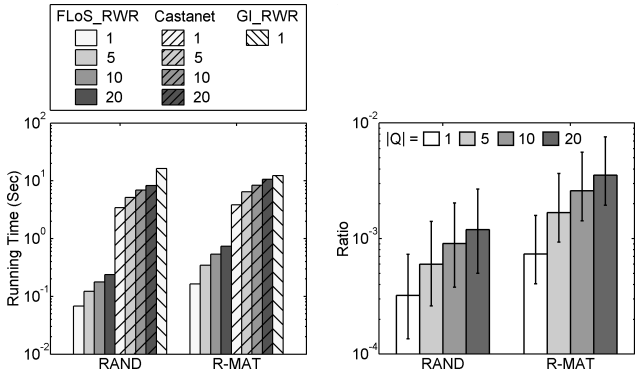
Figure 2.23: Running time of different methods for RWR on in-memory synthetic graphs ($k = 20$)

the RAND graphs, while other methods have more. The reason is similar as what discussed previously.

Figure 2.23(c) shows the running time on the series of RAND graphs with varying density. The running time of all the methods increases as the density increases. Figure 2.23(d) shows the running time on the series of R-MAT graphs with varying density. Similar trends are observed.

2.8.3.3 Evaluation of FLoS_RWR (Multiple Query Nodes)

The settings are the same as that in real graphs in Section 2.8.2.3. Since the graph is large, we only pre-compute the values $\text{RWR}_i(i)$ for some nodes, which will be used as query nodes, by the GI_RWR method. We use one RAND graph and one R-MAT graph both with 2^{20} nodes and 10^7 edges. Figure 2.24(a) shows the running time of selected methods for RWR. FLoS_RWR and Castanet both have increasing running time when increasing the number of query nodes. We can see that FLoS_RWR still runs efficiently and is about 1-2 orders of magnitude faster than Castanet and



(a) running time (b) visited nodes

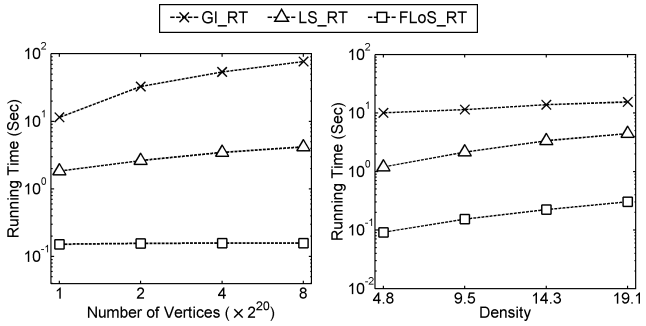
Figure 2.24: Results of FLoS_RWR for multiple query nodes on synthetic graphs (2^{20} nodes and 10^7 edges, k=20, number of query nodes $|Q|$=1,5,10,20)

GI_RWR. Comparing the running time on the RAND and R-MAT graphs, we can see that the increasing rate on the R-MAT graph is larger than that on the RAND graph. The reason is that the node degree of the R-MAT graph follows power-law distribution, thus FLoS will visit many nodes if high degree nodes are visited. On the other hand, the RAND graph has uniform degree distribution, thus there is no such kind of hub node problem. Figure 2.24(b) shows the ratio of the number of visited nodes of the FLoS_RWR method. We can see that the number of visited nodes is increasing when increasing the number of query nodes. The increasing rate on the R-MAT graph is also larger than that on the RAND graph.

2.8.3.4 Evaluation of FLoS_RT

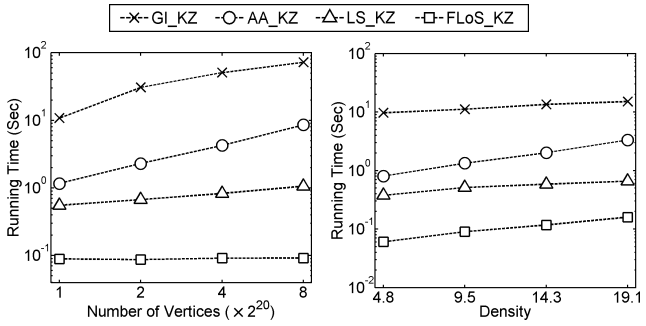
Figure 2.25(a) shows the running time of the selected methods for RT on the series of R-MAT graph with varying size. The running time of GI_RT increases as the number of nodes increases. FLoS_RT has almost constant running time when the number of nodes increases. Because it only searches locally. LS_RT has increasing running time. LS_RT needs to find the node with the largest residual proximity value in each iteration. When the number of nodes in the graph increases, the search space may also increase. This may be the reason why it has a slightly increasing running time.

Figure 2.25(b) shows the running time of the selected methods for RT on the series



(a) varying size (b) varying density

Figure 2.25: Running time of different methods for RT on in-memory synthetic graphs (R-MAT, $k = 20$)



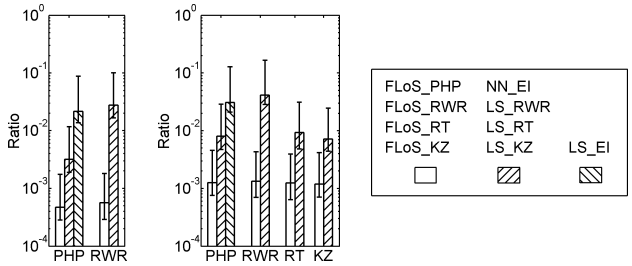
(a) varying size (b) varying density

Figure 2.26: Running time of different methods for KZ on in-memory synthetic graphs (R-MAT, $k = 20$)

of R-MAT graph with varying density. The running time of all the methods increases as the density increases. Both FLoS_RT and LS_RT have increasing running time because they will visit more nodes in a graph with larger density.

2.8.3.5 Evaluation of FLoS_KZ

Figure 2.26(a) shows the running time of the selected methods for KZ on the series of R-MAT graph with varying size. The running time of GI_KZ increases as the number of nodes increases. FLoS_KZ has almost constant running time when the number of nodes increases. LS_KZ and AA_KZ both have increasing running time when increasing graph size. LS_KZ needs to update the node with the largest residual proximity value in each iteration, thus it has a slightly increasing running time when the graph size increases. In AA_KZ, computing the upper bound of each node requires linear time $O(m)$. Thus it has increasing running time.



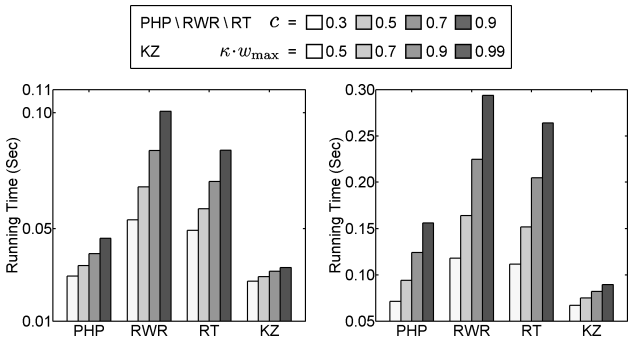
(a) RAND (b) R-MAT

Figure 2.27: Ratio between the number of visited nodes and the total number of nodes on synthetic graphs for the local search methods (2^{20} nodes and 10^7 edges, $k=20$)

Figure 2.26(b) shows the running time of the selected methods for KZ on the series of R-MAT graph with varying density. The running time of all the methods increases as the density increases. The reason why FLoS_KZ, LS_KZ and AA_KZ have increasing running time is that they need to visit more nodes when increasing the graph density.

2.8.3.6 Number of Visited Nodes (Local Search Methods)

In this section, we study the number of visited nodes using different local search methods on synthetic graphs. We use the synthetic graphs with 2^{20} nodes and 10^7 edges. The number of query nodes is fixed to $k=20$. Figure 2.27(a) shows the ratio between the number of visited nodes using different local search methods for PHP and RWR and total number of nodes in the RAND graph. Figure 2.27(b) shows the ratio between the number of visited nodes using different local search methods for PHP, RWR, RT and KZ and total number of nodes in the R-MAT graph. The value indicated by the bar is the average ratio of 10^3 queries. The minimum and maximum ratios are also shown in the figure. As can be seen from the figure, other local search methods need to visit larger number of nodes than the FLoS methods do. This demonstrates the tightness of the bounds in the FLoS methods.



(a) RAND (b) R-MAT

Figure 2.28: Running time of different parameters on synthetic graphs (2^{20} nodes and 10^7 edges, $k=20$)

Table 2.7: Statistics of disk-resident synthetic graphs

`#Nodes`	16×2^{20}	32×2^{20}	48×2^{20}	64×2^{20}
`#Edges`	16×10^7	32×10^7	48×10^7	64×10^7
Disk size	3.1 G	6.5 G	9.9 G	13.2 G

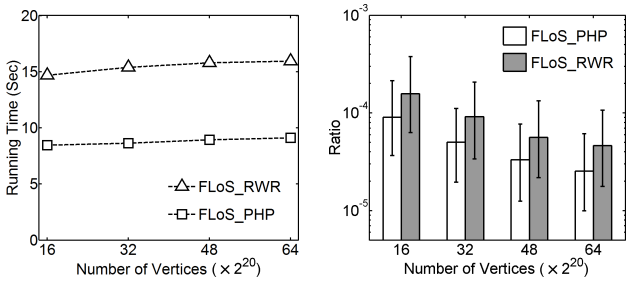
2.8.3.7 Effect of Different Parameters

In this section, we study how the different settings of parameters in the proximity measures will affect the running time on synthetic graphs. The settings of the parameters are the same as that in Section 2.8.2.10. We use the synthetic graphs with 2^{20} nodes and 10^7 edges. The number of query nodes is fixed to $k=20$.

Figures 2.28(a) and 2.28(b) show the running time on the RAND and R-MAT graphs respectively. The running time is changing monotonically when the parameters in different measures are tuned. These results are similar to those in Section 2.8.2.10. We can also observe that the changing rate in RWR and RT is larger than that in PHP and KZ.

2.8.4 Evaluation on Disk-Resident Synthetic Graphs

What if the graphs are too large to fit into memory? To test the performance of FLoS on disk-resident graphs, we generate disk-resident R-MAT graphs, whose statistics are in Table 2.7. We use the open source Neo4j (available from `http://www.neo4j.org`) version 2.0 graph database. The FLoS method for disk-resident graphs only calls



(a) running time (b) visited nodes

Figure 2.29: Results of FLoS_PHP and FLoS_RWR on disk-resident synthetic graphs ($k=20$)

some basic query functions provided by Neo4j, such as, querying the neighbors of one node. And the remaining work is the same as that for in-memory graphs. We apply the FLoS_PHP and FLoS_RWR methods on the disk-resident graphs with $k = 20$. We repeat the query 10^3 times with randomly picked query nodes and report the average running time. In the experiments, we restrict the memory usage to 2 GB.

Figure 2.29(a) shows the running time of FLoS_PHP and FLoS_RWR. From the figure, we can see that FLoS can process disk-resident graphs in tens of seconds. The reason is that FLoS only needs to find the neighbors of visited nodes and the transition probabilities on the edges. These results also verify that FLoS has almost constant running time when the number of nodes increases. Figure 2.29(b) shows the ratio of the number of visited nodes to the total number of nodes in the graph. FLoS only needs to explore a small portion of the whole graph to return the top-k nodes. When the graph size becomes larger, the portion of visited nodes becomes smaller.

2.9 Conclusion

Top-k nodes query in large graphs is a fundamental problem that has attracted intensive research interests. Existing methods need expensive preprocessing steps or are designed for specific proximity measures. In this chapter, we propose a unified method, FLoS, which adopts a local search strategy to find the exact top-k nodes efficiently. FLoS is based on the no local optimum property of proximity measures. By exploiting the relationship among different proximity measures, we can also extend

FLoS to the proximity measures having local optimum. Extensive experimental results demonstrate that FLoS enables efficient and exact query for a variety of random walk based proximity measures.

Chapter 3

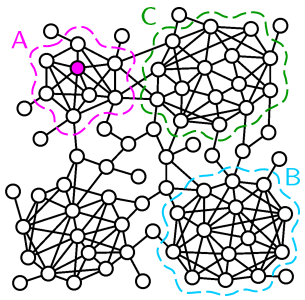
Robust Local Community Detection

3.1 Introduction

Community detection is a fundamental problem in network analysis [33]. Traditional community detection methods aim at finding all communities in the entire network.

With the increasing size of the networks, local community detection has recently drawn intensive research interest [64, 2, 24, 96, 103]. Given a set of query nodes, the local community detection problem aims at finding the community near the query nodes. It has a wide range of applications in analyzing social networks, collaborative tagging systems, query-logs, and biological networks [103, 28, 64, 63].

In local community detection, a goodness metric is usually used to measure whether a subgraph forms a community. The existing goodness metrics for local community detection can be categorized into three classes. The first class optimizes the internal denseness of a subgraph, i.e., the set of nodes in a community should be densely connected with each other. Such metrics include the classic density definition [96], edge-surplus [113], and minimum degree [103]. The second class optimizes both the internal denseness and the external sparseness. That is, the set of nodes



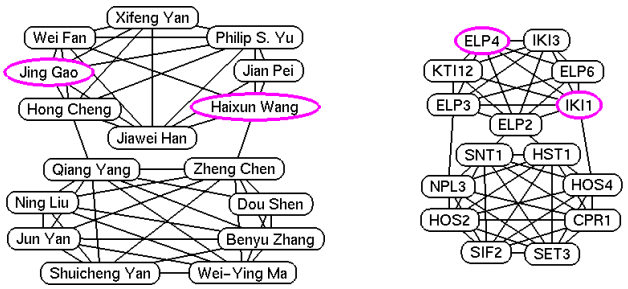
Goodness metrics	A	A∪B	A∪C
Classic density	2.50	2.95	2.83
Edge-surplus	15.3	26.5	22.8
Minimum degree	4	4	4
Subg. modularity	2.0	3.6	4.6
Density-isolation	–2.6	3.8	1.5
Ext. conductance	0.25	0.14	0.11
Local modularity	0.63	0.70	0.78

Figure 3.1: Left: a network with 4 communities, the query node is the purple node in community A; right: goodness values of different metrics on subgraphs A, A∪B, and A∪C

in the community are not only densely connected with each other, but also sparsely connected with the nodes that are not in the community. Such metrics include subgraph modularity [77], density-isolation [69], and external conductance [2]. The local modularity measures the sharpness of the community boundary and belongs to the third class [24]. Using this metric, the set of nodes in the boundary of the community are highly connected to the nodes in the community but sparsely connected to the nodes outside the community.

Given a goodness metric, the generic local community detection problem aims at finding a subgraph such that: (1) the subgraph contains the query nodes, and (2) the goodness metric is maximized (or minimized). This formulation has been explicitly adopted in many existing works [103, 28, 80]. Other works can also fit into this form [96, 113, 24, 77, 2]. The local community detection problem has also been studied as the local clustering problem or the community search problem in the literature [105, 103].

Most of the existing goodness metrics tend to include irrelevant subgraphs in the identified local communities. For example, Figure 3.1 shows a network containing four communities. Suppose that the query node is the purple node in community A. Intuitively, subgraph A should be the target local community. Suppose that we use the classic density definition, which measures the average degree of the nodes in the subgraph. If we merge subgraphs B or C into A, the density will increase: the densities of subgraphs A, A∪B, and A∪C are 2.50, 2.95, and 2.83 respectively. We refer to the



(a) Co-author networks (b) Biological networks

Figure 3.2: Two local communities with free riders identified from the real-world networks using the classic density definition. The query nodes are in purple ellipses.

irrelevant subgraphs, such as B and C, as free riders in local community detection.

Other goodness metrics also cause the free rider effect. The table on the right in Figure 3.1 shows the goodness values for subgraphs A, A∪B, and A∪C for different metrics. For all these metrics, merging B or C into A will result in better goodness values. Note that the external conductance tries to minimize the goodness value instead of maximizing it.

In the example shown in Figure 3.1, subgraph B is the densest subgraph in the entire network. We refer to such a subgraph with the largest goodness value as the global optimal subgraph. If a goodness metric will include the global optimal subgraph in the identified local community, we say this metric causes the global free rider effect. We refer to a subgraph whose goodness value is greater than that of any subgraph of it as a local optimal subgraph. In this figure, subgraph C is a local dense subgraph, which is irrelevant to the query node. If a goodness metric will include a local optimal subgraph in the identified local community, we say that this metric causes the local free rider effect.

In addition to the running example in Figure 3.1, Figure 3.2 shows two examples of free riders from real-world networks. Figure 3.2(a) shows the local community identified in the co-author network extracted from the DBLP dataset. The query nodes are in purple ellipses. The classic density is used as the goodness metric. It is clear from this figure that the lower part is a free rider that should not have been included in the result. Figure 3.2(b) shows the local community identified in the protein interaction

network downloaded from BioGRID (thebiogrid.org). The query proteins {ELP4, IKI1} are from the protein complex transcription elongation factor [38]. The lower part in the figure does not belong to the complex and clearly is a free rider. Similar results can be observed when using other metrics listed in the table in Figure 3.1.

In this chapter [120], we systematically study the existing goodness metrics and provide theoretical explanations on why they will cause global or local free rider effects. To reduce the free rider effect, we propose a node weighting scheme based on random walk. The nodes far away from the query nodes will have large weights and high penalties to be included in the target local community. This query biased weighting scheme forces the global densest subgraph shift to the neighborhood of the query nodes. We use the query biased density as the new metric and formulate the query biased densest connected subgraph (QDC) problem. Its goal is to find the query biased densest subgraph that contains the query nodes and is also connected.

The original QDC problem is NP-hard. To solve it efficiently, we study two related problems QDC′ and QDC″. In QDC″, we only maximize the query biased density. In QDC′, in addition to maximizing the query biased density, we also have the constraint that the query nodes must be included in the resulting subgraph. These two problems can be solved in polynomial time. Moreover, the solutions to these two problems can often be used as the optimal or approximate solutions to the QDC problem. We perform extensive experimental studies on a variety of real and synthetic networks to evaluate the performance of the proposed method. The results show that our method achieves much higher accuracy compared to the state-of-the-art methods and runs efficiently.

3.2 Related Work

Global graph partitioning: Global graph partitioning has been extensively studied [33]. Representative methods include modularity clustering [89] and multi-level graph

partitioning [57]. Some existing methods use edge weighting schemes to enhance the performance. For example, one method [60] strengthens the intracluster edges by rewarding edges with high common neighbor ratio, and weakens the intercluster edges by penalizing edges with high betweenness centrality. Other methods reward edges that have short cycles connecting their endpoints [12] or edges whose endpoints have similar degrees [23].

Local community detection: Given a set of query nodes, the local community detection problem aims at finding the community near the query nodes. Most existing methods optimize a goodness metric. The classic density definition [96], edge-surplus [113], and minimum degree [103, 28] maximize the internal denseness of the community. The subgraph modularity [77], density-isolation [69], and external conductance [105, 2, 63] try to maximize the internal denseness and minimize the external sparseness at the same time. The local modularity metric [24] optimizes the sharpness of the boundary of the community. Other methods enumerate different types of communities, such as the α-adjacency-γ-quasi-k-clique [27] and the k-truss community [46]. The α-adjacency-γ-quasi-k-clique is based on the γ-quasi-k-clique, which contains k vertices and at least $\lfloor \gamma \frac{k(k-1)}{2} \rfloor$ edges. The k-truss community is based on the k-truss subgraph, where every edge is contained in at least $(k-2)$ triangles within the subgraph.

3.3 The Free Rider Effect

In this section, we first review some representative goodness metrics for local communities, and then discuss the free rider effect caused by them. Table 3.1 lists the main symbols used in this chapter and their definitions.

Table 3.1: Main symbols

Symbols	Definitions
$G(V, E)$	undirected graph G with node set V and edge set E
$Q; q$	a set of query nodes Q; a query node q
S; $\|S\|$	a set of nodes $S \subseteq V$; number of nodes in S
S^*	node set of the global optimal subgraph
S_l^*	node set of a local optimal subgraph
$G[S]$	subgraph induced by S
$f(S)$	a goodness metric defined on subgraph $G[S]$
$w(u, v)$	weight of an edge (u, v)
$e(S)$	sum of edge weights, $e(S) = \frac{1}{2}\sum_{u,v \in S} w(u, v)$
$e(S, T)$	sum of edge weights, $e(S, T) = \sum_{u \in S, v \in T} w(u, v)$
c	decay factor in the penalized hitting probability
$r(u)$	penalized hitting probability of u w.r.t. the query
$r(S)$	sum of proximity values, $r(S) = \sum_{u \in S} r(u)$
$\pi(u)$	query biased node weight, $\pi(u) = 1/r(u)$
$\pi(S)$	sum of query biased node weights, $\pi(S) = \sum_{u \in S} \pi(u)$
$\rho(S)$	query biased density, $\rho(S) = e(S)/\pi(S)$
N_u	the set of neighbor nodes of node u in graph G
$w_S(u)$	degree of u in $G[S]$, $w_S(u) = \sum_{v \in N_u \cap S} w(u, v)$
$w'_S(u)$	query biased degree in $G[S]$, $w'_S(u) = w_S(u)/\pi(u)$
$w(u); w'(u)$	degree of u in G; query biased degree of u in G
$\phi(S)$	volume of S, $\phi(S) = \sum_{u \in S} w(u)$
$\overline{S}$	complement of S, $\overline{S} = V \setminus S$
δS	boundary of S, $\delta S = \{u \in S \mid \exists v \in N_u \cap \overline{S}\}$

3.3.1 Representative Goodness Metrics

Let $G(V, E)$ be an undirected graph and $G[S]$ be the subgraph induced by a set of nodes $S \subseteq V$. Let $f(S)$ be a goodness metric defined on subgraph $G[S]$. Most of the existing methods on local community detection can fit in the following generic formulation [24, 2, 77, 96, 103, 80, 113].

Problem 4. [Local Community Detection] *Given an undirected graph $G(V, E)$, a set of query nodes Q, and a goodness metric $f(S)$, find the subgraph $G[S]$ such that*

1) $Q \subseteq S$;

2) $f(S)$ is optimized.

Table 3.2 lists some representative goodness metrics and their formulas. The

Table 3.2: Goodness metrics and their free rider effects

Goodness metrics	Ref.	Formulas $f(S)$		Glo.	Loc.
Classic density	[96]	$e(S)/\|S\|$		✓	✓
Edge-surplus	[113]	$e(S)-\alpha h(\|S\|)$	concave $h(x)$	✓	✓
			$h(x)=\binom{x}{2}$	×	✓
Minimum degree	[103]	$\min_{u\in S} w_S(u)$		✓	✓
Subgraph modularity	[77]	$e(S)/e(S,\overline{S})$		✓	✓
Density-isolation	[69]	$e(S)-\alpha e(S,\overline{S})-\beta\|S\|$		✓	✓
External conductance	[2]	$e(S,\overline{S})/\min\{\phi(S),\phi(\overline{S})\}$		×	✓
Local modularity	[24]	$e(\delta S,S)/e(\delta S,V)$		×	✓

classic density definition [96], edge-surplus [113], and minimum degree [103] measure the internal denseness of the local community. In the edge-surplus metric, h is a strictly-increasing function, and α is a constant. As proposed in [113], usually $h(x)$ is concave, or $h(x)=\binom{x}{2}$.

The subgraph modularity [77], density-isolation [69], and external conductance [2] measure both the internal denseness and external sparseness of the community. Note that α and β in the density-isolation formula are user specified constants [69]. In the external conductance metric [2], we have that $\phi(S)=2e(S)+e(S,\overline{S})$. The total internal edge weight $e(S)$ measures the internal denseness, and the total external edge weight $e(S,\overline{S})$ measures the external sparseness.

The local modularity metric [24] measures the sharpness of the boundary of the community. In this metric, the numerator represents the total weight of the internal edges incident to δS, and the denominator represents the total weight of all edges incident to δS.

Note that among the metrics discussed above, the external conductance needs to be minimized, while other metrics need to be maximized.

3.3.2 Global Free Rider Effect

In this section, we discuss the global free rider effect. In this case, for any subgraph, its goodness value will increase if the global optimal subgraph is merged into it.

Definition 4. [Global Optimal Subgraph] *The global optimal subgraph is the subgraph* $G[S^*]$ *whose goodness value* $f(S^*) \geq f(S)$*, for any* $S \subseteq V$*.*

A goodness metric f causes the global free rider effect if for any $S \subseteq V$, $f(S) \leq f(S \cup S^*)$. In this case, the optimal solution to Problem 4 must contain the global optimal subgraph $G[S^*]$.

In the next, we formally prove that a set of goodness metrics cause the global free rider effect. First, we review some definitions [100]: f is *submodular* if $f(S)+f(T) \geq f(S \cap T)+f(S \cup T)$, *supermodular* if $f(S)+f(T) \leq f(S \cap T)+f(S \cup T)$, and *modular* if it is both submodular and supermodular.

For example, $f(S)=e(S,\overline{S})$ is submodular; $f(S)=e(S)$ is supermodular; $f(S)=|S|$ is modular. The edge-surplus metric with a concave $h(x)$ and the density-isolation metric are both supermodular.

Theorem 20. *If the goodness metric* f *is supermodular, then for any* $S \subseteq V$*,* $f(S) \leq f(S \cup S^*)$*.*

Proof. Since f is supermodular, we have that $f(S)+f(S^*) \leq f(S \cap S^*)+f(S \cup S^*)$. Since $G[S^*]$ is the global optimal subgraph, we have that $f(S^*) \geq f(S \cap S^*)$. Combining these two inequalities, we have that $f(S) \leq f(S \cup S^*)$. □

Since the edge-surplus metric with a concave $h(x)$ and the density-isolation metric are both supermodular, we have the following lemma.

Lemma 16. *The edge-surplus metric with a concave* $h(x)$ *and the density-isolation metric satisfy that for any* $S \subseteq V$*,* $f(S) \leq f(S \cup S^*)$*.*

We say that f is *monotonically increasing*, if for any $S \subseteq V$ and $S' \subseteq \overline{S}$, $f(S \cup S') \geq f(S)$.

Theorem 21. *If* $f(S) = \frac{g(S)}{h(S)}$*, where* $g(S) > 0$ *is supermodular and monotonically increasing, and* $h(S) > 0$ *is submodular, then for any* $S \subseteq V$*,* $f(S) \leq f(S \cup S^*)$*.*

Proof. Let $f'(S, \lambda) = g(S) - \lambda h(S)$, where λ is a real-valued constant. Let $\lambda = f(S)$. Proving that $f(S \cup S^*) \geq f(S)$ is equivalent to proving that $f'(S \cup S^*, \lambda) \geq 0$. Note that $g(S) - \lambda h(S) = 0$ since $\lambda = f(S)$. We have that

$$\begin{aligned} & f'(S \cup S^*, \lambda) = g(S \cup S^*) - \lambda h(S \cup S^*) \\ \geq\ & g(S) + g(S^*) - g(S \cap S^*) - \lambda(h(S) + h(S^*) - h(S \cap S^*)) \\ =\ & g(S^*) - g(S \cap S^*) - \lambda(h(S^*) - h(S \cap S^*)). \end{aligned}$$

Since g is monotonically increasing, $g(S^*) - g(S \cap S^*) \geq 0$. If $h(S^*) - h(S \cap S^*) \leq 0$, we have that $f'(S \cup S^*, \lambda) \geq 0$. Now suppose that $h(S^*) - h(S \cap S^*) > 0$. Since $G[S^*]$ is the global optimal subgraph, we have that $f(S^*) \geq f(S \cap S^*)$. Thus we can derive that

$$\frac{g(S^*) - g(S \cap S^*)}{h(S^*) - h(S \cap S^*)} \geq \frac{g(S^*)}{h(S^*)} = f(S^*) \geq \lambda.$$

So, we have that $f'(S \cup S^*, \lambda) \geq 0$, i.e., $f(S \cup S^*) \geq f(S)$. □

The classic density definition and subgraph modularity both have the ratio form as shown in Theorem 21. Thus we have the following lemma.

Lemma 17. *The classic density definition and subgraph modularity satisfy that for any $S \subseteq V$, $f(S) \leq f(S \cup S^*)$.*

Lemma 18. *The minimum degree metric satisfies that for any $S \subseteq V$, $f(S) \leq f(S \cup S^*)$.*

Proof. If we merge $G[S^*]$ into $G[S]$, its minimum degree will not decrease. □

In summary, the classic density, edge-surplus with a concave $h(x)$, density-isolation, subgraph modularity, and minimum degree metrics all cause the global free rider effect.

3.3.3 Local Free Rider Effect

In this section, we discuss the free rider effects caused by the local optimal subgraph.

Definition 5. [Local Optimal Subgraph] *A local optimal subgraph is the subgraph* $G[S_l^*]$ *whose goodness value* $f(S_l^*) \geq f(S)$*, for any* $S \subseteq S_l^*$.

That is, deleting any node(s) from a local optimal subgraph will decrease its goodness value. Note that by definition, the global optimal subgraph is also a local optimal subgraph.

Let $G[S]$ denote the target local community that contains the query nodes. A local optimal subgraph $G[S_l^*]$ could be irrelevant to the query nodes. A goodness metric f causes the local free rider effect if $f(S) \leq f(S \cup S_l^*)$ for an irrelevant local optimal subgraph $G[S_l^*]$. We can generalize Theorems 1 and 2 to the local optimal subgraphs.

Theorem 22. *If the goodness metric* f *is supermodular, then for any* $S \subseteq V$, $f(S) \leq f(S \cup S_l^*)$.

Proof. Since f is supermodular, we have that $f(S) + f(S_l^*) \leq f(S \cap S_l^*) + f(S \cup S_l^*)$. Since $G[S_l^*]$ is the local optimal subgraph, we have that $f(S_l^*) \geq f(S \cap S_l^*)$. Combining these two inequalities, we have that $f(S) \leq f(S \cup S_l^*)$. □

Theorem 23. *Suppose that* $f(S) = \frac{g(S)}{h(S)}$*, where* $g(S) > 0$ *is supermodular and monotonically increasing, and* $h(S) > 0$ *is submodular. If* $f(S_l^*) \geq f(S)$*, the goodness metric* f *satisfies that* $f(S) \leq f(S \cup S_l^*)$.

Proof. Let $f'(S, \lambda) = g(S) - \lambda h(S)$, where λ is a real-valued constant. Let $\lambda = f(S)$. Proving that $f(S \cup S_l^*) \geq f(S)$ is equivalent to proving that $f'(S \cup S_l^*, \lambda) \geq 0$. Note that $g(S) - \lambda h(S) = 0$ since $\lambda = f(S)$. We have that

$$\begin{aligned} & f'(S \cup S_l^*, \lambda) = g(S \cup S_l^*) - \lambda h(S \cup S_l^*) \\ \geq\ & g(S) + g(S_l^*) - g(S \cap S_l^*) - \lambda(h(S) + h(S_l^*) - h(S \cap S_l^*)) \\ =\ & g(S_l^*) - g(S \cap S_l^*) - \lambda(h(S_l^*) - h(S \cap S_l^*)). \end{aligned}$$

Since g is monotonically increasing, $g(S_l^*) - g(S \cap S_l^*) \geq 0$. If $h(S_l^*) - h(S \cap S_l^*) \leq 0$, we have that $f'(S \cup S_l^*, \lambda) \geq 0$. Now suppose that $h(S_l^*) - h(S \cap S_l^*) > 0$. Since $G[S_l^*]$

is a local optimal subgraph, we have that $f(S_l^*) \geq f(S \cap S_l^*)$. Thus we can derive that

$$\frac{g(S_l^*) - g(S \cap S_l^*)}{h(S_l^*) - h(S \cap S_l^*)} \geq \frac{g(S_l^*)}{h(S_l^*)} = f(S_l^*) \geq \lambda.$$

So, we have that $f'(S \cup S_l^*, \lambda) \geq 0$, i.e., $f(S \cup S_l^*) \geq f(S)$. □

The proofs of Theorems 22 and 23 are similar to those of Theorems 20 and 21 respectively. Based on Theorems 22 and 23, we have the following lemmas.

Lemma 19. *The edge-surplus metric with a concave $h(x)$ and the density-isolation metric satisfy that for any $S \subseteq V$, $f(S) \leq f(S \cup S_l^*)$.*

Lemma 20. *If $f(S_l^*) \geq f(S)$, the classic density definition and the subgraph modularity metric satisfy that $f(S) \leq f(S \cup S_l^*)$.*

Next, we identify conditions that will cause the local free rider effects for the remaining goodness metrics.

Lemma 21. *If $f(S_l^*) \geq f(S)$, the minimum degree metric satisfies that $f(S) \leq f(S \cup S_l^*)$.*

Proof. If we merge $G[S_l^*]$ into $G[S]$, its minimum degree will not decrease. □

The following lemma says that if the target community and the irrelevant local optimal subgraph are node disjoint, and the goodness value of the irrelevant subgraph is large, then the edge-surplus metric with $h(x) = \binom{x}{2}$ causes the local free rider effect.

Lemma 22. *If $S \cap S_l^* = \emptyset$ and $f(S_l^*) \geq \alpha \cdot |S| \cdot |S_l^*|$, where α is used in the definition of edge-surplus, then the edge-surplus with $h(x) = \binom{x}{2}$ satisfies that $f(S) \leq f(S \cup S_l^*)$.*

Proof. Since $S \cap S_l^* = \emptyset$, we have that $e(S) + e(S_l^*) \leq e(S \cup S_l^*)$. Since $f(S_l^*) = e(S_l^*) - \alpha\binom{|S_l^*|}{2} \geq \alpha \cdot |S| \cdot |S_l^*| = \alpha[\binom{|S|+|S_l^*|}{2} - \binom{|S|}{2} - \binom{|S_l^*|}{2}]$, we have that $e(S_l^*) \geq \alpha\binom{|S|+|S_l^*|}{2} - \alpha\binom{|S|}{2}$. Combining these two inequalities, we have that $e(S) - \alpha\binom{|S|}{2} \leq e(S \cup S_l^*) - \alpha\binom{|S|+|S_l^*|}{2}$ thus $f(S) \leq f(S \cup S_l^*)$. □

The following lemma says that if any pair of nodes u and v, where u is in the target community and v is in the irrelevant local optimal subgraph, are at least two hops away from each other, and the local optimal subgraph has a larger goodness value, then the local modularity metric causes the local free rider effect.

Lemma 23. *If $S_l^* \cap (S \cup \delta\overline{S}) = \emptyset$ and $f(S_l^*) \geq f(S)$, the local modularity metric satisfies that $f(S) \leq f(S \cup S_l^*)$.*

Proof. Since $S_l^* \cap (S \cup \delta\overline{S})) = \emptyset$, $\delta(S \cup S_l^*) = \delta S \cup \delta S_l^*$ and $\delta S \cap \delta S_l^* = \emptyset$. Thus, we have that $e(\delta(S \cup S_l^*), S \cup S_l^*) = e(\delta S, S) + e(\delta S_l^*, S_l^*)$ and $e(\delta(S \cup S_l^*), V) = e(\delta S, V) + e(\delta S_l^*, V)$. So, $f(S \cup S_l^*) = \frac{e(\delta S, S) + e(\delta S_l^*, S_l^*)}{e(\delta S, V) + e(\delta S_l^*, V)} \geq f(S)$. □

Different from other metrics, the external conductance value needs to be minimized. In this case, a local optimal subgraph is the subgraph $G[S_l^*]$ whose goodness value $f(S_l^*) \leq f(S)$, for any $S \subseteq S_l^*$. A goodness metric f causes the local free rider effect if $f(S) \geq f(S \cup S_l^*)$ for an irrelevant local optimal subgraph $G[S_l^*]$. The following lemma says that if the volumes of the target community and the irrelevant local optimal subgraph are both small, and the irrelevant subgraph has a smaller goodness value, the external conductance metric causes the local free rider effect.

Lemma 24. *If $\phi(S \cup S_l^*) \leq \phi(V)/2$ and $f(S_l^*) \leq f(S)$, the external conductance metric satisfies that $f(S) \geq f(S \cup S_l^*)$.*

Proof. Since $\phi(S \cup S_l^*) \leq \phi(V)/2$, subgraphs $G[S]$, $G[S_l^*]$ and $G[S \cup S_l^*]$ all have smaller volumes than the complements. In this case, external conductance degenerates to $f(S) = \frac{e(S,\overline{S})}{\phi(S)} = \frac{e(S,\overline{S})}{2e(S) + e(S,\overline{S})}$. Minimizing $f(S)$ is equivalent to maximizing $f'(S) = \frac{e(S)}{e(S,\overline{S})}$, which is subgraph modularity. So we have that $f'(S) \leq f'(S \cup S_l^*)$ and $f(S) \geq f(S \cup S_l^*)$. □

In real-world networks, we can often find local optimal subgraphs that are irrelevant to the query nodes and satisfy the conditions discussed above. These irrelevant subgraphs will reduce the accuracy of the local community detection methods. Table

3.2 lists whether the goodness metrics cause the global or local free rider effect in the last two columns.

3.4 Node Weighting

In this section, we first use the random walk based proximity values to weight the nodes, and then define a new goodness metric, the query biased density. We show that after node weighting, the query biased densest subgraph is in the neighborhood of the query nodes.

3.4.1 Node Weighting by Random Walk

We first compute the proximity value of each node with regard to the query nodes. The reciprocal of the proximity value is used as the node weight. Thus the nodes closer to the query nodes will have smaller weights.

Many proximity measures can be used, such as random walk with restart [112] and the degree normalized penalized hitting probability [121]. In this chapter, we use a variant of the degree normalized penalized hitting probability, which is referred to as the penalized hitting probability.

Let $w(u, v)$ be the edge weight between u and v, $w(u)$ be the degree of node u, and $w_{\max}$ be the maximum degree. The transition probability from u to v is $w(u, v)/w_{\max}$, which is normalized by the maximum degree. The penalized hitting probability penalizes the random walk for each additional step. The probability of hitting the query nodes for the first time is used as the proximity value. Let $r(u)$ denote the proximity value with regard to the query nodes Q. The penalized hitting probability can be defined as follows

$$r(u) = \begin{cases} 1, & \text{if } u \in Q; \\ c \sum_{v \in N_u} \frac{w(u,v)}{w_{\max}} \cdot r(v), & \text{if } u \in V \setminus Q, \end{cases}$$

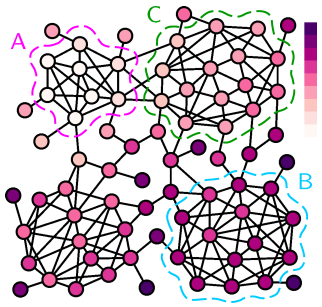


Figure 3.3: Effect of node weighting with $c = 0.9$ (darker color represents higher node weight; subgraph A is the query biased densest subgraph)

where c $(0 < c < 1)$ is the decay factor. The power iteration method can solve this linear system in $O(\kappa m)$ time, where κ is the number of iterations [95].

The query biased node weight $\pi(u)$ of node u is defined as the reciprocal of $r(u)$, i.e., $\pi(u) = 1/r(u)$. We always have that $0 \leq r(u) \leq 1$ and $\pi(u) \geq 1$.

Consider the example in Figure 3.1. The nodes in community A are densely connected to the query node through multiple short paths. Thus the random walker will have high probability to hit the query node starting from any node in A. On the other hand, there are only a few long paths connecting the query node and the nodes not in A. Starting from these nodes, the random walker will have low probabilities to hit the query node, since the probabilities are penalized by the path lengths. Thus the nodes in A will have higher proximity values than the nodes not in A. The distribution of the node weights, i.e., the reciprocal of the proximity values, are shown in Figure 3.3. A lighter color indicates a higher proximity value.

3.4.2 The Query Biased Density

Based on the query biased node weights, the query biased density is defined as follows.

Definition 6. [Query Biased Density]

$$\rho(S) = \frac{e(S)}{\pi(S)},$$

where $\pi(S) = \sum_{u \in S} \pi(u)$ *is the sum of the query biased weights of nodes in* S.

If the node weights $\pi(u) = 1$, the query biased density degenerates to the classic

density $e(S)/|S|$. In the next, we will use the query biased density and density interchangeably if there is no ambiguity.

After node weighting, the densest subgraph is shifted to the neighborhood of the query nodes. For example, in the graph shown in Figure 3.1, before node weighting, the densest subgraph is B. Figure 3.3 shows the node weights after applying our node weighting scheme. A darker color represents a larger node weight. After node weighting, subgraph A becomes the query biased densest subgraph.

In the next, we show why the query biased densest subgraph will shift to the neighborhood of the query nodes. In particular, we prove that the query biased densest subgraph always (1) contains the query node, and (2) is connected, if the decay factor is small.

Let $|N_u|$ be the number of neighbors of u, and $N_{\max}$ be the maximum number of neighbors among all the nodes. The following lemma says that for any non-query node u, there exists a neighbor node v whose weight is smaller than that of node u times $w(u,v)/w(u)$.

Lemma 25. *If $c < (N_{\max})^{-1}$, for any node $u \in V \setminus Q$, there exists a neighbor node v of node u (i.e., $v \in N_u$), such that $\pi(v) < \frac{w(u,v)}{w(u)} \cdot \pi(u)$.*

Proof. It is equivalent to prove that $w(u,v) \cdot r(v) > w(u) \cdot r(u)$. Suppose that node u violates this lemma, i.e., $\forall v \in N_u$, $w(u,v) \cdot r(v) \leq w(u) \cdot r(u)$. We have that $r(u) = c\sum_{v \in N_u} \frac{w(u,v)}{w_{\max}} r(v) \leq c\sum_{v \in N_u} \frac{w(u)}{w_{\max}} r(u) \leq c \cdot |N_u| \cdot r(u) < r(u)$. We get a contradiction that $r(u) < r(u)$. □

Based on this lemma, we can prove that if node u belongs to the query biased densest subgraph, the neighbor node v with small node weight also belongs to the query biased densest subgraph. Next, we show that the query biased densest subgraph is connected and contains the query node.

Theorem 24. *If $c < (N_{\max})^{-1}$ and there is one query node, the query biased densest subgraph is connected and contains the query node.*

Proof. Let $G[T]$ denote one connected component of the densest subgraph $G[S]$. We have that $\rho(T) \leq \rho(S)$. We also have that $\rho(T) \geq \rho(S)$, since otherwise $\rho(S \setminus T) > \rho(S)$. Thus $G[T]$ has equal density with $G[S]$. Suppose that $G[T]$ does not contain the query node.

First, let $u \in T$ be the node with the smallest value $\frac{\pi(u)}{w(u)}$, i.e., $\forall v \in N_u \cap T, \frac{\pi(v)}{w(v)} \geq \frac{\pi(u)}{w(u)}$. Since $w(v) \geq w(u,v)$, we have that $\forall v \in N_u \cap T, \pi(v) \geq \frac{w(u,v)}{w(u)} \cdot \pi(u)$. There exists a node $y \in N_u \cap \overline{T}$ such that $\pi(y) < \frac{w(u,y)}{w(u)} \cdot \pi(u)$. Otherwise, node u violates Lemma 25.

Next, we will prove that $\rho(T \cup \{y\}) > \rho(T)$. We have that $\frac{w(u)}{\pi(u)} \geq \frac{e(T)}{\pi(T)}$ because otherwise $\rho(T \setminus \{u\}) > \rho(T)$. Since $\pi(y) < \frac{w(u,y)}{w(u)} \cdot \pi(u)$, we have that $\frac{w(u,y)}{\pi(y)} > \frac{w(u)}{\pi(u)} \geq \frac{e(T)}{\pi(T)}$. So, we have that $\rho(T \cup \{y\}) \geq \frac{e(T)+w(u,y)}{\pi(T)+\pi(y)} > \rho(T)$.

In conclusion, if $G[T]$ does not contain the query node, there must exist a node in $\delta\overline{T}$, whose addition increases the density. This contradicts the assumption that $G[T]$ has the largest density. This completes the proof. □

Theorem 24 can be generalized to multiple query nodes.

Theorem 25. *If $c < (N_{\max})^{-1}$ and there are multiple query nodes, each connected component of the query biased densest subgraph contains at least one query node.*

Proof. Let $G[T]$ denote one connected component of the densest subgraph $G[S]$. We have that $\rho(T) \leq \rho(S)$. We also have that $\rho(T) \geq \rho(S)$, since otherwise $\rho(S \setminus T) > \rho(S)$. Thus $G[T]$ has equal density with $G[S]$. Suppose that $G[T]$ does not contain any query node.

First, let $u \in T$ be the node with the smallest value $\frac{\pi(u)}{w(u)}$, i.e., $\forall v \in N_u \cap T, \frac{\pi(v)}{w(v)} \geq \frac{\pi(u)}{w(u)}$. Since $w(v) \geq w(u,v)$, we have that $\forall v \in N_u \cap T, \pi(v) \geq \frac{w(u,v)}{w(u)} \cdot \pi(u)$. There exists a node $y \in N_u \cap \overline{T}$ such that $\pi(y) < \frac{w(u,y)}{w(u)} \cdot \pi(u)$. Otherwise, node u violates Lemma 25.

Next, we will prove that $\rho(T \cup \{y\}) > \rho(T)$. We have that $\frac{w(u)}{\pi(u)} \geq \frac{e(T)}{\pi(T)}$ because otherwise $\rho(T \setminus \{u\}) > \rho(T)$. Since $\pi(y) < \frac{w(u,y)}{w(u)} \cdot \pi(u)$, we have that $\frac{w(u,y)}{\pi(y)} > \frac{w(u)}{\pi(u)} \geq \frac{e(T)}{\pi(T)}$. So, we have that $\rho(T \cup \{y\}) \geq \frac{e(T)+w(u,y)}{\pi(T)+\pi(y)} > \rho(T)$.

In conclusion, if $G[T]$ does not contain any query node, there must exist a node in $\delta\overline{T}$, whose addition increases the density. This contradicts the assumption that $G[T]$ has the largest density. This completes the proof. □

3.4.3 Intuition behind the Query Biased Density

In this subsection, we further discuss the intuition on why the query biased density can help to reduce the free rider effect. The key observation is that the local community should not include the nodes that are well separated from it.

We examine two different ways to model the separateness between the local community and the irrelevant nodes. One model utilizes the conductance. As discussed before, the conductance of a subgraph can measure how well it is separated from the remaining graph. In this model, we assume that the local community has low conductance and show that the irrelevant nodes are unlikely to be included in the result using the query biased density. In another model, we assume that the local community is separated from the remaining graph by a set of low degree nodes. We show that in this case, irrelevant local optimal subgraphs are guaranteed to be not included in the result. These theoretical results help explain why the query biased density method can reduce the free rider effect.

We still use $G[S]$ to denote the target local community. Let $r(\overline{S})$ be the sum of the proximity values of the nodes outside $G[S]$. The following theorem says that the expected value of $r(\overline{S})$ is upper bounded by the conductance of $G[S]$.

To derive Theorems 26 and 27 shown shortly, we first define a new proximity measure, PHP$''$, as follows.

$$r(u) = \begin{cases} c\sum_{v\in N_u} \frac{w(u,v)}{w_{\max}} r(v) + 1, & \text{if } u = q; \\ c\sum_{v\in N_u} \frac{w(u,v)}{w_{\max}} r(v), & \text{if } u \neq q. \end{cases}$$

Let PHP(u) and PHP$''(u)$ be the PHP and PHP$''$ proximity values of node u with regard to the query node q. PHP and PHP$''$ have the following relationship.

Lemma 26. $\mathrm{PHP}(u) = \frac{\mathrm{PHP}''(u)}{\mathrm{PHP}''(q)}$

Proof. We can see that PHP and PHP″ have the same recursive equation for any node $u \neq q$. Thus we have that $\frac{\mathrm{PHP}(u)}{\mathrm{PHP}''(u)} = \frac{\mathrm{PHP}(q)}{\mathrm{PHP}''(q)}$. This completes the proof. □

Based on this, we can derive the PHP proximity matrix. Let $\mathbf{W}$ be the adjacency matrix with $\mathbf{W}_{u,v} = w(u,v)$, $\mathbf{P} = \mathbf{W}/w_{\max}$ be the transition probability matrix, $\mathbf{r}$ be the PHP″ proximity vector, and $\mathbf{1}_q$ be a vector with only one non-zero element $\mathbf{1}_q(q) = 1$. The recursive equation of PHP″ can be expressed as $\mathbf{r} = c\mathbf{P}\mathbf{r} + \mathbf{1}_q$. Thus the PHP″ proximity vector is $\mathbf{r} = (\mathbf{I} - c\mathbf{P})^{-1}\mathbf{1}_q$. So, the PHP proximity vector is $\mathbf{r} = \frac{1}{\mathrm{PHP}''(q)}(\mathbf{I} - c\mathbf{P})^{-1}\mathbf{1}_q$ from Lemma 26. Let $\mathbf{\Lambda}$ be a diagonal matrix with $\mathbf{\Lambda}_{u,u} = (\mathrm{PHP}''_u(u))^{-1}$, where $\mathrm{PHP}''_u(u)$ denotes the PHP″ proximity value of node u when the query is u. Then, the PHP proximity matrix is

$$\mathbf{R} = (\mathbf{I} - c\mathbf{P})^{-1}\mathbf{\Lambda}.$$

The qth column of $\mathbf{R}$ represents the PHP proximity vector when the query is node q.

Suppose that the query node q belongs to a local community $G[S]$. Recall that the conductance of $G[S]$ is defined as

$$\frac{e(S,\overline{S})}{\min\{\phi(S),\phi(\overline{S})\}}.$$

When the volume $\phi(S) < \phi(V)/2$, the conductance value is equal to $e(S,\overline{S})/\phi(S)$.

Let $\mathbf{d}$ be a vector with $\mathbf{d}(u) = w(u)$ for any node $u \in V$. Let $\mathbf{d}_S$ be a vector with $\mathbf{d}_S(u) = w(u)$ if $u \in S$, and $\mathbf{d}_S(u) = 0$ otherwise. Let $\mathbf{1}$ be a vector whose elements all are 1. Let $\mathbf{1}_S$ be a vector with $\mathbf{1}_S(u) = 1$ if $u \in S$, and $\mathbf{1}_S(u) = 0$ otherwise. Let $\mathbf{1}_S^{\mathrm{T}}$ be the transpose of $\mathbf{1}_S$. Let $\mathbf{R}$ be the proximity matrix with $\mathbf{R}_{u,v}$ denoting the PHP proximity value of node u when the query is node v.

Lemma 27. *Suppose that $\phi(S) < \phi(V)/2$. We have that $\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}^k\mathbf{d}_S \leq k \cdot e(S,\overline{S})$, for any integer k.*

Proof. Since $\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}\mathbf{d}_S = \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{W}\mathbf{d}_S/w_{\max} \leq \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{W}\mathbf{1}_S = e(S,\overline{S})$, this lemma holds when $k = 1$.

Suppose that it holds for $k = i$. By induction, it suffices to show it holds for $k = i + 1$. Let $\mathbf{x} = \mathbf{P}^i\mathbf{d}_S$, and $\mathbf{x}_S(u) = \mathbf{x}(u)$ if $u \in S$ and $\mathbf{x}_S(u) = 0$ otherwise. We have that

$$\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}^{i+1}\mathbf{d}_S = \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}\mathbf{x} = \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}(\mathbf{x}_S + \mathbf{x}_{\overline{S}}) \leq \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}\mathbf{x}_S + \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{x}.$$

Since $\mathbf{d}_S \leq \mathbf{d}$, $\mathbf{P}\mathbf{d}_S \leq \mathbf{P}\mathbf{d} = \mathbf{W}\mathbf{d}/w_{\max} \leq \mathbf{W}\mathbf{1} = \mathbf{d}$. Thus, we have that $\mathbf{P}^i\mathbf{d}_S \leq \mathbf{d}$. So, $\mathbf{x}_S \leq \mathbf{d}_S$ because $\mathbf{d}_S$ equals $\mathbf{d}$ restricted on S. Therefore, $\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}\mathbf{x}_S \leq \mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}\mathbf{d}_S \leq e(S, \overline{S})$. Thus, $\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}^{i+1}\mathbf{d}_S \leq e(S, \overline{S}) + i \cdot e(S, \overline{S}) = (i + 1)e(S, \overline{S})$. □

Theorem 26. *Suppose that $\phi(S) < \phi(V)/2$. If we randomly pick a node in $G[S]$ with a probability proportional to its degree and use it as the query node, the expected value of $r(\overline{S})$ is no greater than the conductance of $G[S]$ times $\frac{c}{(1-c)^2}$.*

Proof. If we randomly pick a node from $G[S]$ with a probability proportional to its degree and use this node as the query node, the expected value of $r(\overline{S})$ can be expressed as $\frac{1}{\phi(S)}\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{R}\mathbf{d}_S$. Thus, it is equivalient to prove that $\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{R}\mathbf{d}_S \leq \frac{c}{(1-c)^2}e(S, \overline{S})$.

For any node u, since $\mathrm{PHP}''_u(u) \geq 1$, $\mathbf{\Lambda}_{\mathbf{u},\mathbf{u}} \leq 1$. Thus,

$$\begin{aligned}\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{R}\mathbf{d}_S &= \mathbf{1}_{\overline{S}}^{\mathrm{T}}(\mathbf{I} - c\mathbf{P})^{-1}\mathbf{\Lambda}\mathbf{d}_S \leq \mathbf{1}_{\overline{S}}^{\mathrm{T}}(\mathbf{I} - c\mathbf{P})^{-1}\mathbf{d}_S \\ &= \mathbf{1}_{\overline{S}}^{\mathrm{T}}\textstyle\sum_{k=0}^{\infty}(c\mathbf{P})^k\mathbf{d}_S = \sum_{k=0}^{\infty} c^k\mathbf{1}_{\overline{S}}^{\mathrm{T}}\mathbf{P}^k\mathbf{d}_S \\ &\leq \textstyle\sum_{k=1}^{\infty} c^k k \cdot e(S, \overline{S}) = \frac{c}{(1-c)^2}e(S, \overline{S}),\end{aligned}$$

where the fifth step is based on Lemma 27. □

Theorem 26 says that if the conductance of $G[S]$ is small, the sum of the proximity values of the nodes outside $G[S]$ is also expected to be small. Thus these nodes are unlikely to be included in the result as a free rider to $G[S]$ because of their large weights (reciprocals of their proximity values).

The next theorem is based on the second model, i.e., the target local community is separated from the remaining graph by a set of low degree nodes. We still use $G[S_l^*]$ to denote a local optimal subgraph.

Theorem 27. *Suppose that (1) $G[S]$ and $G[S_l^*]$ are separated by a set of nodes L with low degree $w(v) < \frac{e(S)}{|S|}$ (for any node $v \in L$); and (2) for any pair of nodes*

$u \in S$ and $v \in L$, $\pi(u) < \pi(v)$. If $c < (N_{\max})^{-1}$, the query biased density satisfies that $f(S) \geq f(S \cup S_l^*)$.

Proof. First, we prove that the query biased densest subgraph does not contain any node in L. For any node $v \in L$, $w'(v) = \frac{w(v)}{\pi(v)} < \frac{e(S)}{|S| \cdot \pi(v)} < \frac{e(S)}{\pi(S)} = \rho(S)$. The query biased densest subgraph must have a density larger than $\rho(S)$. We can prove that any node in the densest subgraph must have query biased degree greater than $\rho(S)$. Thus, any node $u \in L$ does not belong to the densest subgraph.

Since $c < (N_{\max})^{-1}$, the query biased densest subgraph is connected and containing the query node based on Theorem 25. So, if a node y in S_l^* belongs to the densest subgraph, there must exist a path in the densest subgraph linking y to one query node. From the assumption, this path contains at least one node $v \in L$. However, we already prove that the nodes in L do not belong to the optimal subgraph. Thus, the local optimal subgraph $G[S_l^*]$ cannot exist in the densest subgraph. □

The theorem assumes the degree of the nodes in L is less than half of the average node degree in $G[S]$. This is a reasonable assumption to model a sparse region separating $G[S]$ and $G[S_l^*]$. The second assumption is that the node weight of any node $u \in S$ is smaller than the node weight of any node $v \in L$, i.e., $\pi(u) < \pi(v)$. This is also a reasonable assumption, because in Theorem 26, we already show that the nodes outside $G[S]$ usually have smaller proximity values than the nodes in $G[S]$. Once these two assumptions are satisfied, the query biased densest subgraph will not have the free rider effect, no matter how dense the local optimal subgraph is.

3.5 Our Problem Formulation

In this section, we introduce a new formulation for the local community detection problem, i.e., the query biased densest connected subgraph (QDC) problem. We further study two related problems QDC′ and QDC″ with fewer constraints. The QDC problem is NP-hard, while QDC′ and QDC″ can be solved in polynomial time.

We can often obtain the optimal or approximate solution for the QDC problem by solving QDC′ and QDC″.

3.5.1 The QDC Problem

We use the following problem formulation for the local community detection. The problem aims at finding the query biased densest subgraph that contains the query nodes and is also connected.

Problem 5. [Query Biased Densest Connected Subgraph (QDC)] *Given an undirected graph $G(V, E)$ and a set of query nodes Q, find the subgraph $G[S]$ such that*

1) $Q \subseteq S$;

2) $\rho(S)$ is maximized;

3) $G[S]$ is connected.

Theorem 28. *The QDC problem is NP-hard.*

Proof. The QDC problem can be reduced from the set cover problem [56]. Let $X = \{X_1, \cdots, X_l\}$ be a family of sets with $Y = \{y_1, \cdots, y_k\} = \bigcup_{i=1}^{l} X_i$ being the elements. The set cover problem consists of finding a minimum subset $X_{\text{opt}} \subseteq X$, such that each element y_j is contained in at least one set in X_{opt}.

The graph is constructed as follows. For each element $y_j \in Y$, we create two nodes y_j and z_j. Let node set $Z = \{z_1, \cdots, z_k\}$. Let node set $V = \{q\} \cup X \cup Y \cup Z$. Node q is connected to any node $X_i \in X$. Node $y_j \in Y$ is connected to node X_i if $y_j \in X_i$ in the set cover problem. Node $z_j \in Z$ is connected to node $y_j \in Y$. The edge (y_j, z_j) has weight $2 + \frac{1}{k} + \epsilon$, where $0 < \epsilon < \frac{1}{k^2}$ is a real number. The edges incident to X_i have equal weight $\frac{1}{|X_i|+1}$. Let the set of query nodes $Q = \{q\} \cup Q'$, where $Q' \subseteq Y \cup Z$. If Q' is empty, there is one query node q. The node weight $\pi(u) = 1, \forall u \in V$.

Figure 3.4 gives an example of the graph constructed from an instance of the set cover problem with $X_1 = \{y_1, y_2\}, X_2 = \{y_1\}, X_3 = \{y_2, y_4\}, X_4 = \{y_2, y_3\}$, and $X_5 = \{y_4\}$.

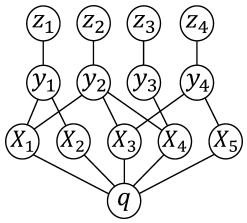


Figure 3.4: Graph construction from an instance of the set cover problem

Consider a subgraph $G[S]$ which contains Q ($Q \subseteq S$) and is connected. We must have that $S = \{q\} \cup X' \cup Y' \cup Z'$, where $X' \subseteq X$, $Y' \subseteq Y$, $Z' \subseteq Z$, and $|Y'| \geq |Z'|$.

Suppose that at least one node z_j does not exist in S, i.e., $|Z'| \leq k-1$. Let $S' = \{q\} \cup X' \cup Y'$. We have that $e(S') \leq |X'|$ since the node degree $w(X_i) = 1$. Consider the density $\rho(S) = \frac{\sum_{z_j \in Z'}(2+\frac{1}{k}+\epsilon)+e(S')}{|X'|+|Y'|+|Z'|+1} \leq \frac{2|Z'|+|X'|+(\frac{1}{k}+\epsilon)|Z'|}{2|Z'|+|X'|+1}$. We have that $(\frac{1}{k}+\epsilon)|Z'| < (\frac{1}{k}+\frac{1}{k^2})(k-1) < 1$. Thus $\rho(S) < 1$.

Suppose that all the nodes Z exist in S. The density $\rho(S) = \frac{\sum_{z_j \in Z}(2+\frac{1}{k}+\epsilon)+|X'|}{2k+|X'|+1} > \frac{(2+\frac{1}{k})k+|X'|}{2k+|X'|+1} = 1$. So, the subgraph containing Z always has larger density than that containing a proper subset of Z. Thus we only consider the subgraph containing Z in the following.

Next, let $X' \subseteq X$ be a feasible solution to the set cover problem. Let $S = \{q\} \cup X' \cup Y \cup Z$. Then $G[S]$ is connected and $\rho(S) = 1 + \frac{\sum_{z_j \in Z}(2+\frac{1}{k}+\epsilon)-2k-1}{2k+|X'|+1}$. We have that $\sum_{z_j \in Z}(2+\frac{1}{k}+\epsilon)-2k-1 > (2+\frac{1}{k})k-2k-1 = 0$. So $\rho(S)$ is monotonically decreasing with regard to $|X'|$. Since $|X_{\text{opt}}| \leq |X'|$, the subgraph induced from $S = \{q\} \cup X_{\text{opt}} \cup Y \cup Z$ contains the query nodes, has the largest density, and is connected.

So, X_{opt} solves the set cover problem, and the subgraph induced from $S = \{q\} \cup X_{\text{opt}} \cup Y \cup Z$ solves the QDC problem.

Continue with our example in Figure 3.4. $X_{\text{opt}} = \{X_1, X_3, X_4\}$ solves the set cover problem. The subgraph induced from $S = \{q\} \cup X_{\text{opt}} \cup Y \cup Z$ solves the QDC problem. □

3.5.2 Two Related Problems

In this section, we study two related problems that have fewer constraints than the QDC problem. The solutions to these two problems can be used to obtain the optimal

or approximate solution to the QDC problem.

Problem 6. [QDC′] *Given an undirected graph $G(V, E)$ and a set of query nodes Q, find the subgraph $G[S]$ such that*

1) $Q \subseteq S$;

2) $\rho(S)$ is maximized.

Problem 7. [QDC″] *Given an undirected graph $G(V, E)$, find the subgraph $G[S]$ such that $\rho(S)$ is maximized.*

Compared to the QDC problem, QDC′ removes the connectivity constraint, and QDC″ further removes the constraint that $Q \subseteq S$. It is easy to see that these three problems have the following relationships.

Lemma 28. *Let $G[S'']$, $G[S']$, and $G[S]$ be the solutions to* QDC″, QDC′, *and* QDC *respectively. We have that*

1) $\rho(S'') \geq \rho(S') \geq \rho(S)$;

2) If $Q \subseteq S''$, $G[S'']$ is also the solution to QDC′;

3) If $G[S']$ is connected, $G[S']$ is also the solution to QDC.

Lemma 29. *Suppose that $G[S']$ is the solution to* QDC′ *and it is disconnected. If $G[S']$ has a connected component $G[T]$ containing all the query nodes and at least one non-query node, then $G[T]$ is a $\frac{\pi(T)}{\pi(T)-\pi(Q)}$-approximation of the solution to the* QDC *problem.*

Proof. Let $G[S]$ be the solution to QDC. $G[S' \setminus (T \setminus Q)]$ is a feasible solution to QDC′ thus has smaller density than $G[S']$ does. Since $\frac{e(S')-e(T)}{\pi(S')-(\pi(T)-\pi(Q))} \leq \rho(S' \setminus (T \setminus Q)) \leq \rho(S')$, $\frac{e(T)}{\pi(T)-\pi(Q)} \geq \rho(S') \geq \rho(S)$. Thus $\rho(S) \leq \frac{\pi(T)}{\pi(T)-\pi(Q)}\rho(T)$. □

For any $u \in \overline{Q}$, $\pi(u) > 1$. If there is one query node, we have that $\pi(Q) = 1$, $\pi(T) > 2$. Thus the approximation ratio is smaller than 2.

As will be discussed in the next section, QDC′ and QDC″ can be solved in polynomial time. Although QDC is NP-hard, based on Lemmas 28 and 29, the solutions

Algorithm 11: The overall procedure for the QDC problem

Input: $G(V, E)$, query nodes Q, decay factor c

```
Compute the node weights with the decay factor c;
Compute the optimal solution G[S] to the QDC′ problem;
if G[S] is connected then return G[S]; break;
if G[S] has a connected component G[T] containing all the
    query nodes Q and at least one non-query node then
    return G[T]; break;
Apply the heuristic algorithms to find a solution G[S] to QDC;
return G[S];
```

to QDC′ and QDC″ can be used to obtain the optimal or approximate solution to the QDC problem.

The overall procedure for the QDC problem: Algorithm 11 outlines the overall procedure for solving QDC. We first compute the optimal solution to QDC′. If it is connected, it is also the solution to QDC. If it is disconnected but it has one connected component containing all the query nodes and at least one non-query node, this connected component is returned as an approximate solution to QDC. Otherwise, we apply the heuristic algorithms to find a solution to QDC.

3.6 Algorithms

In this section, we describe the main components of the overall procedure outlined in Algorithm 11. We start with the algorithm for QDC″.

3.6.1 Algorithm for the QDC″ Problem

The algorithm to find the optimal solution of the QDC″ problem consists of the following steps.

1. Find a density threshold d by applying the greedy node deletion algorithm [20, 5] on G;

2. Find the d-core of G using the value d from step 1;

3. Find the densest subgraph from the d-core by the parametric maximum flow algorithm [37].

The first two steps are used to reduce the size of the original graph. The optimal solution to the QDC″ problem is guaranteed to be retained in the pruned graph. The exact parametric maximum flow algorithm is then applied to find the optimal solution in the pruned graph.

In step 1, the greedy node deletion algorithm keeps deleting the node with the lowest query biased degree: $w'(u) = w(u)/\pi(u)$. The subgraph with the maximum query biased density during the node deletion process is returned as the approximate solution. This algorithm outputs a 2-approximation to the QDC″ problem. The proof can be extended from that in [5] and is omitted here. The density of the identified subgraph will be used as the threshold d.

In step 2, the d-core is the maximal subgraph of G with all query biased node degrees no less than d. Any subgraph in which every node's query biased degree is no less than d is part of the d-core. Let $G[S]$ be the optimal solution of the QDC″ problem. Since any node $u \in S$ has query biased degree $w'_S(u) \geq \rho(S)$, $G[S]$ is a subgraph of d-core if $d \leq \rho(S)$. Thus using the d value identified in the previous step, we guarantee that the optimal solution is retained in the d-core.

In step 3, we apply the parametric maximum flow algor-ithm to find the densest subgraph in the d-core from step 2.

Let n and m be the number of nodes and edges in the original graph G, and n' and m' be the number of nodes and edges in the d-core. The greedy node deletion algorithm runs in $O(m+n\log n)$ [20]; the d-core is computed in $O(m)$ [5]; the parametric maximum flow algorithm runs in $O(n'm'\log(n'^2/m'))$ [37]. Experimental results show that n' (m') is orders of magnitude smaller than n (m).

Algorithm 12: The algorithm for the QDC′ problem

Input: $G(V, E)$, query nodes Q, node weights π

1: $P_0 \leftarrow Q$; $G_0 \leftarrow$ contract $G[P_0]$ into a supernode p_0; $i \leftarrow 0$;
2: **while** *true* **do**
3: Compute the query biased densest subgraph $G_i[S_i]$ in G_i;
4: **if** S_i *contains the supernode* p_i **then break**;
5: $P_{i+1} \leftarrow S_i \cup P_i$; $G_{i+1} \leftarrow$ contract $G[P_{i+1}]$ into p_{i+1}; $i \leftarrow i+1$;
6: **return** $G[S_i \setminus \{p_i\} \cup P_i]$;

3.6.2 Algorithm for the QDC′ Problem

In this section, we develop an exact polynomial time algorithm for the QDC′ problem. The algorithm uses the following *subgraph contraction* operation.

Contracting a subgraph $G[P]$ of $G(V, E)$ into a supernode p results in a new graph $G'(V', E')$ with $V' = \overline{P} \cup \{p\}$. The supernode p has weight $r(P)$, i.e., the sum of query biased weights of all nodes in P. Other nodes keep their original weights. The edge set E' is constructed as follows.

1. Keep edge (u,v) and its original weight $w(u,v)$ if $(u,v) \in E$ and $u, v \in \overline{P}$;
2. Add an edge (u, p) with weight $e(\{u\}, P)$ if $u \in \delta\overline{P}$;
3. Add a self-loop edge (p, p) with weight $e(P)$ if $e(P) > 0$.

Subgraph contraction operation preserves the density. That is, for any $S \subseteq \overline{P}$, subgraphs $G[P \cup S]$ and $G'[\{p\} \cup S]$ have the same density, so do subgraphs $G[S]$ and $G'[S]$.

The procedure is outlined in Algorithm 12. Initially, the subgraph induced by the query nodes is contracted into a supernode. In each iteration in lines 3-5, we find the query biased densest subgraph $G_i[S_i]$ by solving the QDC″ problem in G_i. If $G_i[S_i]$ does not contain the supernode p_i, the optimal solution of the QDC′ problem must contain $G[S_i]$, since $G[S_i]$ is the optimal solution for the QDC″ problem in G_i and adding it will increase the density of the current subgraph $G[P_i]$. Thus we can contract $G[S_i \cup P_i]$ into a supernode and repeat this process until the query biased densest subgraph $G_i[S_i]$ in G_i contains the supernode.

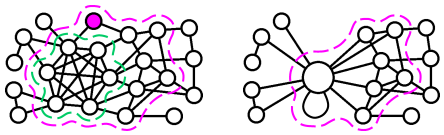


Figure 3.5: Subgraph contraction

Figure 3.5 shows an example. In the left figure, all nodes and edges have unit weights. The purple node is the query node. The densest subgraph is the 6-clique in the green curve with density 2.5. Since it does not contain the query node, we contract it together with the query node into a supernode. The densest subgraph with density 2.38 in the new graph is indicated by the purple curve, and it contains the supernode. Thus, it is the optimal subgraph and highlighted by the purple curve in both figures.

Algorithm 12 runs in $O(\kappa t)$, where κ is the number of iterations and t is the running time of solving the QDC$''$ problem. At least one node is newly contracted into the supernode in each iteration, thus $\kappa \leq n$.

3.6.3 Algorithm for the QDC Problem

In this section, we develop two heuristic algorithms for the QDC problem if solving QDC$'$ does not give the desired solution. Note that our experimental results show that the probability of getting an optimal or approximate solution of QDC is very high by using the polynomial time algorithms introduced in the previous two subsections. With more than 90% probability, we get the optimal solution of QDC by solving QDC$'$. With more than 5% probability, we get an approximate solution of QDC by solving QDC$'$ (approximation ratio is shown in Lemma 29). Only with less than 5% probability, we need to apply the following heuristics to find a solution of QDC.

Heuristic1: Greedy node deletion with connectivity constraint. The heuristic is outlined in Algorithm 13. It iteratively deletes a set of non-articulation nodes [109] which have low query biased degrees. Here, the non-articulation nodes are the nodes whose deletion will not disconnect the graph [109]. The subgraph with the largest density during the node deletion process is returned.

Algorithm 13: Greedy node deletion with connectivity constraint

Input: $G(V, E)$, query nodes Q, node weights π, parameter η

1: $V_0 \leftarrow V$; $i \leftarrow 0$;
2: **while** *true* **do**
3: Compute the articulation nodes S in $G[V_i]$; $S \leftarrow V_i \setminus S \setminus Q$;
4: **if** $|S| = 0$ **then break**;
5: Select a set of nodes $T \subseteq S$ that can be deleted together and has low query biased degree $w'_{V_i}(u) \leq \eta \cdot \rho(V_i)$, $\forall u \in T$;
6: **if** $|T| = 0$ **then** $T \leftarrow \{u \,|\, u = \text{argmin}_{v \in S}\, w'_{V_i}(v)\}$;
7: $V_{i+1} \leftarrow V_i \setminus T$; $i \leftarrow i + 1$;
8: $j \leftarrow \text{argmax}_i\, \rho(V_i)$; **return** $G[V_j]$;

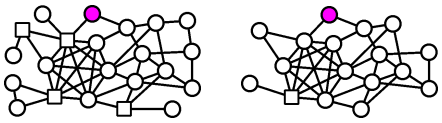


Figure 3.6: Articulation nodes

Suppose that we have a set of biconnected components of graph G. In line 5, if we select at most one non-articulation node from each biconnected component, the deletion of this set of nodes will not disconnect the graph. This is because the deletion of any non-articulation node in one biconnected component does not disconnect this biconnected component.

Parameter η controls the degree of the non-articulation nodes to be deleted. η is usually set between $0 \sim 2$. Since $2\rho(V_i)$ is the average node degree, there are about half of the nodes whose query biased degree is smaller than the threshold $2\rho(V_i)$. More nodes are deleted in each iteration when larger η value is used. If all non-articulation non-query nodes have query biased degree greater than $\eta \cdot \rho(V_i)$, the algorithm picks the node with the minimum degree to delete. The algorithm runs in $O(nm)$ for finding the articulation nodes and biconnected components by depth first search [109].

Figure 3.6 shows an example. The articulation nodes are represented by squares, and the non-articulations nodes are represented by circles. The density of the original graph is 2.05. Suppose that $\eta = 1$. Then, the low degree node threshold is 2.05. After deleting some low degree non-articulation nodes, we get the subgraph as shown on the right in Figure 3.6.

Heuristic2: The maximum adjacency search algorithm. The heuristic is outlined

Algorithm 14: The maximum adjacency search algorithm

Input: $G(V, E)$, query nodes Q, node weights π, parameter K

1: Compute the Steiner tree of Q, and let S be its node set;
2: $V_0 \leftarrow S$; $i \leftarrow 0$;
3: **while** $|V_i| \leq K$ **do**
4: $\quad u \leftarrow \text{argmax}_{v \in \delta\overline{V_i}} e(\{v\}, V_i)/\pi(v)$;
5: $\quad V_{i+1} \leftarrow V_i \cup \{u\}$; $i \leftarrow i + 1$;
6: $j \leftarrow \text{argmax}_i \rho(V_i)$; **return** $G[V_j]$;

in Algorithm 14. Mehlhorn's algorithm [86] is used to compute the Steiner tree given the query nodes. Thus the query nodes become connected. When computing the Steiner tree, the edge weight is set to the reciprocal of the original edge weight.

Next, the algorithm begins a local search process. The Steiner tree is used as the initial subgraph. In each iteration (lines 4-5), the algorithm adds the node $u \in \delta\overline{V_i}$ with the maximum adjacency value $e(\{u\}, V_i)/\pi(u)$ to V_i. The subgraph with the maximum density during the local search process is returned. Parameter K is used to control the search space. When the number of nodes in V_i is greater than K, the algorithm will terminate.

The local search process needs at most K iterations. Let N_{avg} be the average number of neighbors. Then, at the ith iteration, it takes $O(i \cdot N_{\text{avg}})$ time to find the node with the maximum adjacency value in line 4. Thus the local search process runs in $O(\sum_{i=0}^{K} i \cdot N_{\text{avg}}) = O(K^2 N_{\text{avg}})$. The Steiner tree can be computed in $O(m+n \log n)$ [86].

3.7 Problem Variants

In this section, we extend the algorithm for QDC to solve two variations of the basic QDC problem: (1) finding overlapping local communities, and (2) finding multiple disjoint local communities.

Finding overlapping local communities: In real applications, the same set of query nodes may belong to multiple different overlapping communities. We can extend the algorithm for QDC to solve this problem: after finding one community, we remove the nodes except the query nodes in the community; the algorithm can then be applied

to find the next community. The algorithm runs in $O(k\tau)$, where k is the number of local communities and τ is the running time for finding one community.

Finding multiple disjoint local communities: Given a set of query nodes Q, a subset of Q may belong to a community, while other subsets may belong to entirely different communities. These communities are disjoint with each other. Our algorithm can also be extended to solve this problem. For each query node, we find its local community. Then we select the community with the maximum density. If this community contains several query nodes, then they are classified into this community. We remove the nodes in this community from the original graph. By applying this procedure repeatedly, we can partition the query nodes and identify their local communities. The algorithm runs in $O(|Q|^2\tau)$, where τ is the running time for finding one community.

3.8 Experimental Results

We perform extensive experiments to evaluate the effectiveness and efficiency of the proposed methods using a variety of real and synthetic datasets. All the programs are written in C++. All experiments are performed on a server with 32G memory, Intel Xeon 3.2GHz CPU, and Redhat OS.

3.8.1 Datasets and State-of-the-Art Methods

The statistics of the real networks used in the experiments are shown in Table 3.3. These datasets are provided with ground-truth community memberships and are publicly available at the website [72].

We compare our QDC method with several state-of-the-art local community detection methods, which are summarized in Table 3.4. These methods are categorized into three classes. The first class optimizes the internal denseness. The second class optimizes both the internal denseness and external sparseness. The third class opti-

Table 3.3: Statistics of the real networks

Datasets	Abbr.	#Nodes	#Edges	#Communities
Amazon	AZ	334,863	925,872	151,037
DBLP	DP	317,080	1,049,866	13,477
Youtube	YT	1,134,890	2,987,624	8,385
Orkut	OR	3,072,441	117,185,083	6,288,363
LiveJournal	LJ	3,997,962	34,681,189	287,512
Friendster	FS	65,608,366	1,806,067,135	957,154

Table 3.4: State-of-the-art methods used for comparison

Classes	Abbr.	Ref.	Key idea
Internal	DS	[96]	Densest subgraph with query constraint
	OQC	[113]	Optimal quasi-clique; edge-surplus
	MDG	[103]	Minimum degree
Internal & External	PRN	[2]	External conductance
	LS	[81]	Local spectral
	EMC	[32]	More internal edges than external edges
	SM	[77]	Subgraph modularity
Boundary	LM	[24]	Local modularity

mizes the sharpness of the community boundary.

The densest subgraph with query constraint (DS) method finds the densest subgraph containing the query nodes [96]. The optimal quasi-clique with query constraint (OQC) meth-od maximizes the edge-surplus goodness metric and also has the query constraint [113]. The minimum degree with global search (MDG) method maximizes the minimum degree and also has the query, size and distance constraints [103].

PageRank-Nibble (PRN) minimizes the external conductance [2]. The local spectral (LS) method is a revised version of the classic spectral method, and it is biased to the query node [81]. We implement the Spielman-Teng linear equation solver [104] in the LS method. The external minimum cut (EMC) method defines a community as a subgraph whose internal edges are more than external edges [32], and designs a minimum cut based algorithm to find local communities. The subgraph modularity (SM) and local modularity (LM) methods both use a heuristic local search procedure but have different goodness metrics [77, 24].

3.8.2 Evaluation Criteria

We use three different types of criteria to evaluate the selected methods: F-score, a set of community goodness metrics, and consistency.

F-score: It measures the accuracy of the detected community with regard to the ground-truth community labels. Given the discovered community $G[S]$ and the ground-truth community $G[T]$, F-score is defined as

$$F(S,T) = 2 \cdot \frac{\text{prec}(S,T) \times \text{rec}(S,T)}{\text{prec}(S,T) + \text{rec}(S,T)},$$

where $\text{prec}(S,T) = \frac{|S \cap T|}{|S|}$ is the precision and $\text{rec}(S,T) = \frac{|S \cap T|}{|T|}$ is the recall.

Community goodness metrics: We use three goodness metrics: *density*, *separability*, and *cohesiveness* [125]. *Density* adopts the classic density definition. Subgraph modularity is used to measure *separability* [125]. *Cohesiveness* is defined as the minimum internal conductance,

$$\min_{S' \subset S} \frac{e(S', S \setminus S')}{\min\{\phi_S(S'), \phi_S(S \setminus S')\}},$$

where $\phi_S(S')$ is the volume of S' in $G[S]$ [55]. Intuitively, a good local community should have high density, high separability, and high cohesiveness.

Consistency: Intuitively, using different sets of nodes in the same community as the query nodes, we should find the same community. Let $G[S]$ be the detected community when the query is Q, and $G[S']$ be the detected community when the query is $Q' \subseteq S$. The consistency of an algorithm is defined as

$$1 - \sqrt{\frac{1}{\binom{|S|}{|Q|}} \sum_{Q' \subseteq S, |Q'| = |Q|} (F(S,S') - F_{\text{mean}})^2},$$

where F_{mean} is the mean value of $F(S,S')$ over all the S' [80]. That is, the consistency is defined as one minus the standard deviation of F-scores of the identified communities using different sets of query nodes from the community. We randomly take $\min\{10^3, \binom{|S|}{|Q|}\}$ subsets $Q' \subseteq S$ as the queries to estimate the consistency.

Table 3.5: F-scores on real networks

F-score	QDC	DS	OQC	MDG	PRN	LS	EMC	SM	LM
AZ	**0.83**	0.52	0.54	0.46	0.69	0.66	0.61	0.60	0.58
DP	0.46	0.31	0.33	0.32	**0.48**	0.42	0.34	0.36	0.37
YT	**0.43**	0.23	0.22	0.17	0.26	0.24	0.21	0.21	0.22
OR	**0.47**	0.15	0.16	0.13	0.21	0.17	0.19	0.16	0.18
LJ	**0.64**	0.48	0.47	0.40	0.52	0.51	0.47	0.48	0.49
FS	**0.32**	–	0.14	0.12	0.17	0.16	–	0.14	0.13
$\overline{F}$	**0.53**	0.30	0.31	0.27	0.39	0.36	0.33	0.33	0.33
$\overline{\text{Prec}}$	**0.67**	0.46	0.45	0.29	0.51	0.41	0.34	0.38	0.48
$\overline{\text{Rec}}$	**0.72**	0.61	0.58	0.69	0.67	0.64	0.66	0.63	0.59

3.8.3 Evaluation on Real Networks

3.8.3.1 Effectiveness Evaluation

We first evaluate the effectiveness of the selected methods on real networks. We randomly pick 1000 sets of query nodes with size ranging from 1 to 40. Each set of query nodes is picked from a random ground-truth community. The decay factor is set to 0.9 in all experiments unless stated otherwise.

Table 3.5 shows the F-scores of the selected methods on different datasets. It can be seen that the QDC method outperforms other methods on most datasets except the DP dataset. The ground-truth community of DP network is based on conferences and the authors in one conference may not be densely connected [125]. The QDC method significantly improves the accuracy of the density based methods compared to DS and OQC. The PRN method has the second best performance. QDC outperforms PRN for about 10% on most datasets in terms of the F-score. Note that LS is also biased to the query node. However, its threshold constraint scheme cannot eliminate the free rider effect effectively. Please see Subsection 3.9.1 for more detailed discussions. The average F-score $\overline{F}$, precision $\overline{\text{Prec}}$, and recall $\overline{\text{Rec}}$ over all datasets are also shown in this table. We can observe that the existing methods have high recall but low precision. This may be caused by the free rider effect.

Figure 3.7 shows the three goodness metrics, density, separability, and cohesive-

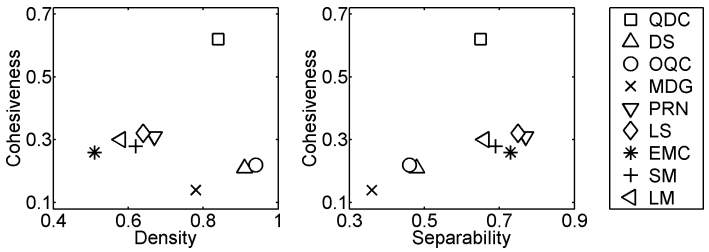


Figure 3.7: Goodness metrics on the LJ network

Table 3.6: Consistency on real networks

Consistency	QDC	DS	OQC	MDG	PRN	LS	EMC	SM	LM
AZ	**0.94**	0.77	0.76	0.58	0.79	0.69	0.74	0.67	0.61
DP	**0.88**	0.62	0.64	0.37	0.65	0.53	0.56	0.43	0.56
YT	**0.85**	0.61	0.54	0.46	0.71	0.41	0.57	0.37	0.36
OR	**0.83**	0.56	0.52	0.32	0.68	0.43	0.51	0.54	0.47
LJ	**0.93**	0.74	0.67	0.43	0.84	0.64	0.73	0.58	0.52
FS	**0.78**	–	0.56	0.45	0.65	0.49	–	0.32	0.39
Average	**0.87**	0.64	0.62	0.44	0.72	0.53	0.61	0.49	0.49

Table 3.7: F-scores for different node weighting schemes

F-score	AZ	DP	YT	OR	LJ	FS	$\overline{F}$	$\overline{\text{Prec}}$	$\overline{\text{Rec}}$
PHP	0.83	0.46	0.43	0.47	0.64	0.32	0.53	0.65	0.78
RWR	0.73	0.39	0.33	0.35	0.56	0.28	0.45	0.52	0.69
PHP′	0.76	0.37	0.35	0.38	0.58	0.24	0.46	0.54	0.71

ness, of the detected communities on the LJ network. These metrics are normalized so that their maximum values are 1. The QDC method has the best overall performance. It has high cohesiveness, high density, and high separability. DS, OQC, and MDG have high density, but low cohesiveness and separability. PRN, LS, EMC, SM, and LM have high separability but low cohesiveness and density. All the selected existing methods suffer from the free rider effect. This is why their cohesiveness scores are low. Similar results can be observed in other datasets.

Table 3.6 shows the consistency of the detected communities using different methods. We can see that the consistency value of QDC is about 30% to 60% higher than those of other methods. The free rider effect causes the low consistency of other methods. If the nodes in the irrelevant free rider subgraph are selected as the query nodes, these methods will detect irrelevant communities.

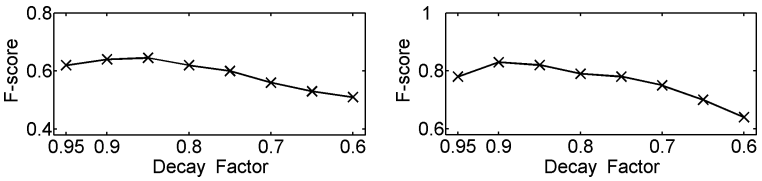


Figure 3.8: Tuning the decay factor (left: LJ; right: AZ)

In addition to penalized hitting probability (PHP), we also apply random walk with restart (RWR) [112] and degree normalized penalized hitting probability (PHP′) [121]. Table 3.7 shows the F-scores using different weighting schemes. We can see from the table that these three strategies provide similar performance, with PHP being slightly better. The differences between these three proximity measures are subtle. Please see Subsection 3.9.2 for further discussions. Note that QDC with RWR or PHP′ still provides high F-scores compared to other state-of-the-art methods.

We further test the sensitivity of QDC with respect to the decay factor c. Figure 3.8 shows the F-scores on LJ and AZ datasets when varying the value of c from 0.95 to 0.6. It can be seen that the performance of QDC is stable for different values. When the decay factor is 0.9, the algorithm has the best performance. F-score slightly decreases when decreasing the decay factor.

3.8.3.2 Efficiency Evaluation

In this section, we first compare the overall running time of different methods, and then study the efficiency of each step of the QDC method.

Figure 3.9 shows the overall running time averaged over 1000 random queries. The QDC method can process large graphs with millions of nodes in tens of seconds. Note that even though some heuristic local search methods, such as LM and SM, run faster, their accuracies are low. The PRN and LS methods have similar performance, because both of them are based on the Spielman-Teng's method [104, 105]. The MDG method takes long running time because of the size and distance constraints in the problem formulation. The DS and EMC methods compute the maximum flow on the whole graph thus have long running time.

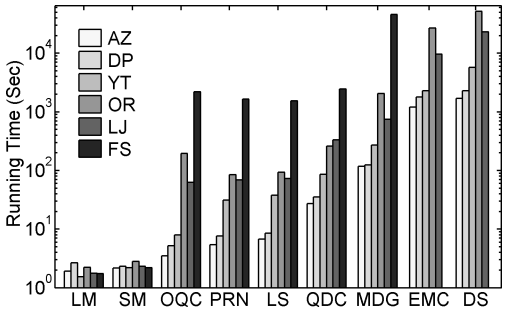


Figure 3.9: The overall running time of different methods

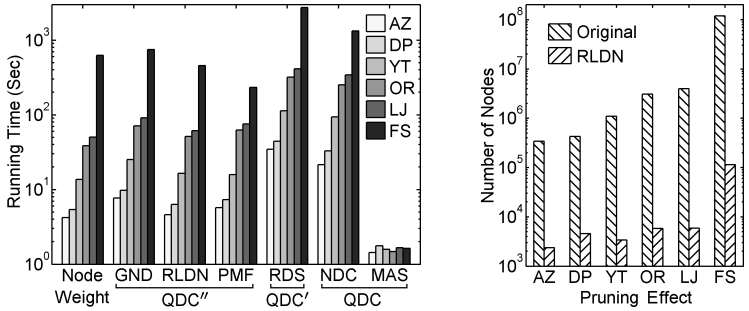


Figure 3.10: Left figure: running time of each step in QDC; right figure: pruning effect of the RLDN step

Recall that there are three major steps in solving QDC after node weighting. We first solve QDC′ exactly by repeatedly calling the algorithm for solving QDC″. If solving QDC′ does not give the optimal solution to QDC, then we try to find an approximate solution with approximation ratio shown in Lemma 29. If both previous steps do not give the solution to QDC, then we apply two heuristics, the greedy node deletion with connectivity constraint (NDC) method or the maximum adjacency search (MAS) method, to find a solution to QDC.

Among the 1000 random queries, we get the optimal solution of QDC by solving QDC′ for 915 queries. We get an approximate solution of QDC by solving QDC′ for 54 queries. We only need to apply heuristics NDC or MAS in 31 queries. Among the 915 queries, the query biased densest subgraph is the optimal solution to QDC for 813 queries.

The left figure in Figure 3.10 shows the running time of each step of the QDC method. Weighting a network with millions of nodes can be done in about 10 seconds. There are three steps in solving the QDC″ problem: greedy node deleting (GND) for

finding the density threshold d, removing low degree node (RLDN) to find the d-core, and applying the parametric maximum flow (PMF) algorithm to find the densest subgraph. In solving QDC, heuristic MAS is more efficient than NDC, since NDC searches over the entire graph, while MAS only searches locally. Note that node weighting and solving QDC$''$ dominate the overall running time, since in most cases solving QDC$''$ will solve QDC.

The right figure in Figure 3.10 shows the pruning effect of the RLDN step in solving QDC$''$. The RLDN step reduces the number of nodes by 2 to 3 orders of magnitude. The parametric maximum flow step runs efficiently, since the RLDN step has significantly reduced the number of nodes.

3.8.4 Evaluation on Synthetic Networks

We generate a collection of synthetic networks as proposed in [68] to evaluate the performance of the selected methods. The network generating model has three parameters γ, β, and μ. The degree and community size follow the power-law distribution with exponents γ and β respectively. We set $\gamma = 2$ and $\beta = 1$. The mixing parameter μ indicates the proportion of a vertex's neighbors that reside in other communities. By tuning μ, we can vary the clearness of the community structure: the boundaries between different communities become less clear for larger μ values.

We first evaluate the F-score when varying the mixing parameter μ from 0.1 to 0.6. The number of nodes in the network is 2^{20} and number of edges is 10^7. Figure 3.11 shows the F-scores. QDC-NDC (QDC-MAS) indicates that we use heuristic NDC (MAS). As we can see, the F-score decreases when increasing μ, i.e., lowering the clearness of the community structure. The QDC-NDC and QDC-MAS methods both have high F-scores, and clearly outperform other methods. Moreover, the QDC methods are very robust to the parameter μ with slight drop in accuracy for large μ values. The accuracies of all existing methods drop significantly after certain threshold. This is because when the boundary between the communities becomes less clear,

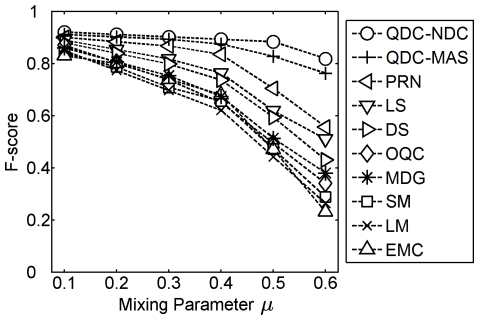


Figure 3.11: F-scores on the synthetic networks

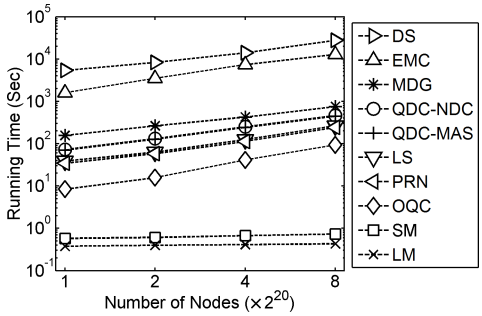


Figure 3.12: Running time on the synthetic networks

it is easier for the existing methods to include irrelevant subgraphs in the identified communities due to the free rider effect.

We then evaluate the scalability when varying the graph size from 2^{20} to 2^{23} nodes. The edge density of each network is 9.53, and the mixing parameter $\mu = 0.3$. Figure 3.12 shows the running time. The local search methods SM and LM have almost constant running time. The OQC method uses a local search procedure but has increasing running time for larger networks. Other methods search the whole graph, thus the running time increases when increasing the graph size. The DS and EMC methods compute the maximum flow on the whole graph, thus they have long running time. The QDC method also computes the parametric maximum flow. Even so, the RLDN step in it significantly prunes the nodes. Thus the QDC method runs efficiently.

3.8.5 Case Study in Co-Author Networks and Biological Networks

Since the previous DBLP dataset does not include author names, we construct an author collaboration network from `http://dblp.uni-trier.de/xml/` for case study. A node in the network represents an author and the edge weight represents the number of papers that the two connected authors have co-authored. We remove the edges with weights less than 3. The network contains $236,497$ nodes and $557,753$ edges. There is no ground-truth community in this co-author network.

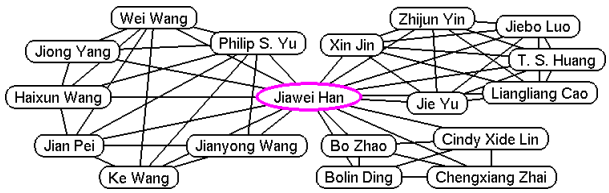
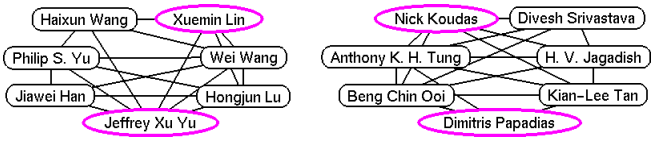
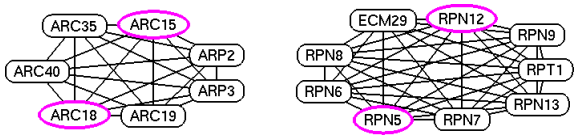


Figure 3.13: Three overlapping communities identified by QDC with Jiawei Han as the query author



(a) Co-author networks



(b) Protein-protein interaction networks

Figure 3.14: Finding multiple disjoint local communities

Figure 3.2(a) shows the communities detected by the existing DS method [96]. Clearly, the communities detected by DS have free riders. On the other hand, our QDC method only returns the communities in the upper part. Experimental results show that other methods also suffer from the free rider effect. The results are omitted due to space limitation.

We further evaluate the results of QDC on solving the two problem variants, i.e., finding overlapping local communities and finding multiple disjoint local communities.

Figure 3.13 shows three overlapping communities detected by QDC when using Jiawei Han as the query author. The authors in the left community are senior researchers in the core data mining research areas. The authors in the two communities on the right contain students and two senior researchers T. S. Huang and Chengxiang Zhai. The authors in the upper right community have published many works on social media mining. The authors in the lower right community mostly collaborate on information retrieval.

To find multiple disjoint local communities, in the co-author network, we use the set of authors {Jeffrey Xu Yu, Xuemin Lin, Dimitris Papadias, Nick Koudas} as the query authors.

Figure 3.14(a) shows two detected communities. The left community is more about data mining, while the right one is more about database. Note that given a set of query nodes, no existing work can be directly applied to find multiple disjoint local communities.

We further apply QDC to protein-protein interaction networks to detect multiple protein complexes for a given set of query proteins. We download the S. cerevisiae (yeast) protein-protein interaction network from BioGRID (*thebiogrid.org*). It contains 6,582 proteins and 224,724 unique physical bonding interactions. Many databases have curated the protein complexes, which can be used as the ground truth communities. We query the proteins {ARC15, ARC18, RPN5, RPN12}. Figure 3.14(b) shows the two detected communities. These two communities correspond to two protein complexes. The left community corresponds to the ARP2/3 complex, which contains the actin-related proteins. The right community is part of the 19/22s regulator complex, which is the proteasome regulatory particle [38].

3.9 Further Discussions About the Experimental Results

In this section, we provide further discussions about the following two questions arised in the experimental results.

1) Why does QDC perform better than LS?

2) Why is PHP better than RWR and PHP′?

3.9.1 Why does QDC Perform Better than LS?

Here we provide more detailed explanations on why our QDC method outperforms the LS method [81]. In LS, the objective is to minimize the conductance $e(S,\overline{S})/(\phi(S)\cdot\phi(\overline{S}))$.

The following lemma says that if (1) any pair of nodes u and v, where u is in the target local community and v is in the irrelevant local optimal subgraph, are at least two hops away from each other, (2) the local optimal subgraph has smaller conductance than the target local community, and (3) the volume of the whole graph is large enough, the goodness metric $f(S) = e(S, \overline{S})/(\phi(S) \cdot \phi(\overline{S}))$ causes the local free rider effect.

Lemma 30. *If* $S_l^* \cap (S \cup \delta\overline{S}) = \emptyset$, $\frac{e(S,\overline{S})}{\phi(S)} > \frac{e(S_l^*,\overline{S_l^*})}{\phi(S_l^*)}$, *and*

$$\phi(V) \geq \phi(S) + \frac{e(S,\overline{S})\phi(S_l^*)(\phi(S) + \phi(S_l^*))}{e(S,\overline{S})\phi(S_l^*) - e(S_l^*,\overline{S_l^*})\phi(S)},$$

the conductance defined in LS satisfies that $f(S) \geq f(S \cup S_l^*)$.

Proof. Let $T = S_l^*$. Since $\frac{e(S,\overline{S})}{\phi(S)} > \frac{e(T,\overline{T})}{\phi(T)}$, we have that $e(S,\overline{S})\phi(T) - e(T,\overline{T})\phi(S) > 0$. Then, from the third condition in the lemma, we have that

$$\frac{e(S,\overline{S})}{\phi(S)\phi(\overline{S})} \geq \frac{e(S,\overline{S}) + e(T,\overline{T})}{(\phi(S) + \phi(T))(\phi(V) - \phi(S) - \phi(T))}.$$

Since $T \cap (S \cup \delta\overline{S}) = \emptyset$, we have that $\phi(S \cup T) = \phi(S) + \phi(T)$ and $e(S \cup T, \overline{S \cup T}) = e(S,\overline{S}) + e(T,\overline{T})$. Thus, we have that

$$\frac{e(S,\overline{S})}{\phi(S)\phi(\overline{S})} \geq \frac{e(S \cup T, \overline{S \cup T})}{\phi(S \cup T)\phi(\overline{S \cup T})}.$$

This completes the proof. □

The LS method has an additional constraint which requires the correlation between the solution vector and the input preference vector to be greater than a threshold. The intension of this constraint is to force the solution subgraph to be near the query node. However, this problem formulation still suffers from the local free rider effect.

Let's first consider the problem formulation [81]. Let $G[S]$ be the target local community that contains the set of query nodes. Let $G[S_l^*]$ be a local optimal subgraph. Suppose that $G[S]$ and $G[S_l^*]$ satisfy the conditions in Lemma 30. Then, $G[S \cup S_l^*]$ has smaller goodness value than $G[S]$. Let $\mathbf{x}_S$ and $\mathbf{x}_{S \cup S_l^*}$ denote the solution vectors of the local spectral optimization program corresponding to the solution subgraphs

$G[S]$ and $G[S \cup S_l^*]$ respectively. Since both $G[S]$ and $G[S_l^*]$ are small subgraphs, $\mathbf{x}_S$ and $\mathbf{x}_{S \cup S_l^*}$ may both have high correlation with the preference vector. So, both $\mathbf{x}_S$ and $\mathbf{x}_{S \cup S_l^*}$ satisfy the constraint with a threshold. However, $G[S \cup S_l^*]$ has smaller goodness value than $G[S]$. Then, $G[S \cup S_l^*]$ is a better solution subgraph. Thus, the problem formulation still suffers from the local free rider effect.

To solve the problem, the algorithm developed in [81] contains two steps. First, it computes the vector $\mathbf{y}$ with regard to the set of query nodes. The vector $\mathbf{y}$ is similar to the RWR proximity vector with regard to the same set of query nodes. Second, for each $i = 1, \cdots, n$, it computes the conductance value of the set S_i composed of the nodes corresponding to the first i elements in $\mathbf{y}$ with largest values, and selects the set S_j with the minimum conductance value. We can see that this procedure is similar to the algorithm in the PRN method [2]. Thus, the LS and PRN methods have similar performance as shown in Table 3.5. They both suffer from the free rider effect.

In conclusion, the constraint with a threshold in the local spectral problem formulation cannot eliminate the free rider effect. The algorithm in [81] is similar to that of PRN and still suffers from the free rider effect.

3.9.2 Why is PHP Better than RWR and PHP′?

Even though the PHP, RWR, and PHP′ are all random walk based proximity measures, there are some subtle differences. Here we discuss why PHP has better performance than RWR and PHP′.

3.9.2.1 PHP and RWR

Random walk with restart (also known as personalized PageRank) [112] is a widely used proximity measure. RWR can be described as follows. From a node u, the random walker can walk to its neighbors with probabilities proportional to the edge weights. In each step, it has a probability of c to return to the query node q, where c

is a constant. The proximity of node u w.r.t. q is defined as the stationary probability that the random walker will finally stay at u. RWR can be defined recursively as

$$r(u) = \begin{cases} (1-c)\sum_{v \in N_u} \frac{w(v,u)}{w(v)} r(v) + c, & \text{if } u = q; \\ (1-c)\sum_{v \in N_u} \frac{w(v,u)}{w(v)} r(v), & \text{if } u \neq q, \end{cases}$$

where c $(0 < c < 1)$ is the constant restart probability.

The key reason why PHP performs better than RWR is that PHP does not have local maximum while RWR does. Thus RWR may favor the high degree nodes in irrelevant subgraphs.

Definition 7. [No Local Maximum] *A proximity measure has no local maximum if for any node $u \in V \setminus Q$, there exists a neighbor node v of u (i.e., $v \in N_u$), such that $r(v) > r(u)$.*

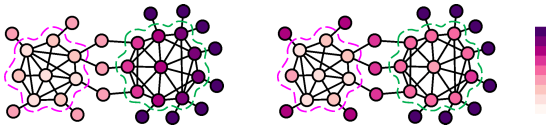
Lemma 31. *PHP has no local maximum.*

Proof. Suppose that node u is a local maximum. Thus, $r(u) = c\sum_{v \in N_u} \frac{w(u,v)}{w_{\max}} r(v) \leq c\sum_{v \in N_u} \frac{w(u,v)}{w_{\max}} r(u) = \frac{c \cdot w(u)}{w_{\max}} r(u) < r(u)$. We get a contradiction that $r(u) < r(u)$. □

No local maximum property in Definition 7 says that for any non-query node u, there exists a neighbor node v of u, i.e., $v \in N_u$, such that node v has larger proximity value than node u. If one proximity measure has no local maximum, the maximum proximity value in the boundary of a set of nodes S containing all the query nodes is the upper bound of the proximity values of nodes in $\overline{S}$. This is shown in the following theorem.

Theorem 29. *Let S be a node set containing all the query nodes, and u be the node with the largest proximity in δS. If a proximity has no local maximum, we have that $r(u) > r(v)$ $(\forall v \in \overline{S})$.*

Proof. Suppose otherwise. We have that $\exists v \in \overline{S}$, such that $r(u) \leq r(v)$. Now suppose that node y is the node with the largest proximity in $\overline{S}$. We have that $\forall z \in \overline{S} \cup \delta S$, $r(y) \geq r(z)$. The neighbors of node y must exist in $\overline{S} \cup \delta S$, i.e., $N_y \subseteq \overline{S} \cup \delta S$.



(a) PHP (b) RWR

Figure 3.15: Comparison between PHP and RWR (The query node is the central node in the community on the left)

Therefore, we have $r(y) \geq r(z)$ $(\forall z \in N_y)$, which means node y is a local maximum. This contradicts the assumption. □

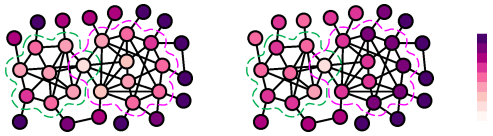
Based on Theorem 29, for a given local community $G[S]$, any node outside $G[S]$ has smaller proximity value than the maximum proximity value in the boundary δS. This shows that the nodes outside $G[S]$ generally have small proximity values than the nodes in $G[S]$.

In contrast, RWR has local maxima. Thus the high degree nodes in the irrelevant subgraph may have large RWR proximity values. This is an undesirable property of RWR.

Figure 3.15(a) shows the node weight distribution of PHP with decay factor $c = 0.9$, where the query node is the central node in the left community. PHP has no local maximum as shown in Lemma 31. Figure 3.15(b) shows the node weight distribution of RWR with restart probability $c = 0.1$. We can see that RWR has local maxima, i.e., the central node in the right community has larger RWR proximity value thus smaller node weight than all its neighbors. In RWR, the nodes with large degree in the irrelevant subgraph will have large proximity values, thus low node weights. Therefore, if the nodes with large degree also compose a dense subgraph, it is likely that this irrelevant subgraph has large query biased density and becomes the free rider subgraph.

3.9.2.2 PHP and PHP′

The degree normalized penalized hitting probability (PHP′) penalizes the random walk for each additional step. PHP′ can be defined recursively as



(a) PHP (b) PHP′

Figure 3.16: Comparision between PHP and PHP′ (The query node is the overlapping node of the two communities)

$$r(u) = \begin{cases} 1, & \text{if } u = q; \\ c\sum_{v \in N_u} \frac{w(u,v)}{w(u)} r(v), & \text{if } u \neq q, \end{cases}$$

where c $(0 < c < 1)$ is the decay factor in the random walk process.

In PHP′, the transition probability is normalized by the node degree. Therefore, the large degree node will have small transition probabilities to its neighbors, thus the probability of hitting the query node will be small. When we use PHP′ to weight the nodes, a dense subgraph will get small query biased density if it contains many large degree nodes. On the other hand, in PHP, the transition probability is normalized by the maximum degree, which is a constant value. So, the hitting probability of the large degree node will not be degraded. When we use PHP to weight the nodes, a dense subgraph still has high query biased density even it contains large degree nodes. This is the key reason why PHP performs better than PHP′.

Let us use the example in Figure 3.16 to further explain why PHP performs better than PHP′. The example graph in Figure 3.16(a) contains two communities, which have one overlapping node. We use this overlapping node as the query node. Figures 3.16(a) and 3.16(b) show the node weight distributions when using PHP and PHP′ respectively. The decay factor $c = 0.9$. Intuitively, the nodes in the right community should have larger proximity values than the nodes in the left community, since the nodes in the right community are more densely connected to the query node than the nodes in the left community are. However, from Figure 3.16(b), we can see that using PHP′, the nodes in the left community have smaller node weights (larger proximity values) than the nodes in the right community. The reason is that the nodes in the right community have large degrees, which reduce their hitting probabilities. The

query biased densities of the left and right communities are 2.04 and 1.66 respectively when using PHP′. This means that the left community has larger query biased density than the right one does. In Figure 3.16(a), we can see that the nodes in the right community have smaller node weights (larger proximity values) than the nodes in the left community. The query biased densities of the left and right communities are 0.41 and 0.45 respectively when using PHP to weight the nodes.

3.10 Conclusion

Local community detection is a fundamental problem in network analysis and has attracted intensive research interests. Most existing local community detection methods optimize a goodness metric but suffer from the free rider effect. In this chapter, we propose query biased node weighting, which shifts the query biased densest subgraph to the neighborhood of the query, to reduce the free rider effect. We study the QDC problem and two related problems, and develop efficient algorithms for them. Extensive experimental results demonstrate that the proposed method not only effectively reduces the free rider effect and achieves high accuracy, but also runs efficiently.

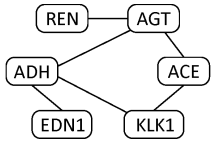
Chapter 4

Mining Dual Networks: Models, Algorithms and Applications

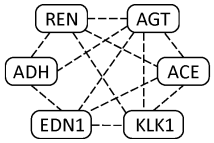
4.1 Introduction

Finding the densest subgraph in a single graph is a key primitive with a wide range of applications, such as modularity detection in biological networks [96] and community detection in social networks [21]. Given a graph $G(V, E)$, the goal is to find the subgraph with maximum average edge weight [71]. This problem can be solved in polynomial time [37]. For large graphs, approximation algorithms have also been developed [3, 5, 20].

In many real-life applications, we can often observe dual networks representing *physical* and *conceptual* interactions among a set of nodes respectively. For example, in genetics, it is crucial to examine interaction between genetic variants since many diseases are caused by the joint effect of multiple genetic factors [91]. The interacting strength between two genetic variants is usually measured by statistical tests such as likelihood ratio test or analysis of variance [92]. These statistical interactions are conceptual. Even two genetic variants have strong statistical interaction, their corresponding protein products may not have direct physical interaction. On the oth-



(a) Protein interaction network



(b) Genetic interaction network

Figure 4.1: An example of dual biological networks

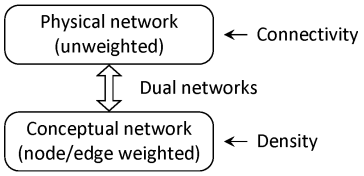


Figure 4.2: Finding DCS in dual networks

er hand, protein-protein interaction network represents physical interactions among proteins and can be used to uncover physical mechanisms behind statistical genetic interactions [106].

Figure 4.1 shows an example of dual biological networks, where Figure 4.1(a) shows the physical protein interaction network among a set of nodes and Figure 4.1(b) shows the statistical genetic interaction network. The set of nodes have high *density* in the genetic interaction network demonstrating that the statistical interactions among them are strong. Moreover, this set of nodes are *connected* in the protein-protein interaction network, which provides biological interpretation on how they interact with each other through signal transduction among proteins [114]. Note that we will use solid (dotted) lines to represent the edges in the physical (conceptual) network throughout the paper.

Dual research interest and collaboration networks can be constructed using bibliographic information such as the DBLP dataset [108]. Dual networks can also be found in social recommender systems. Please see Section 4.10 for a detailed description of various real-life dual networks.

In this chapter, we study the problem of finding the densest connected subgraph (DCS) in dual networks. Given two graphs $G_a(V, E_a)$ and $G_b(V, E_b)$ representing the physical and conceptual networks respectively, the DCS consists of a subset of nodes $S \subseteq V$ such that the induced subgraph $G_a[S]$ is connected and the density of

$G_b[S]$ is maximized. Density in the conceptual network indicates strong interacting signals among the nodes, and connectivity in the physical network explains the signal transduction process.

Figure 4.2 summarizes our DCS problem setting. Note that our problem is different from finding co-dense subgraphs [59, 90] or coherent dense subgraphs [45, 75], whose goal is to find the dense subgraphs preserved across multiple networks of the *same* type. In our problem, the physical and conceptual networks complement each other and require different treatments.

We show that the DCS problem is NP-hard and develop a two-step approach to solve the DCS problem. In the first step, we effectively prune the dual networks while guarantee that the optimal solution is contained in the remaining networks. For the second step, we develop two efficient greedy approaches based on different search strategies to find the DCS. The first approach finds the densest subgraph in the conceptual network first and then refines it according to the physical network to make it connected. Although finding the densest subgraph in a single graph can be solved in polynomial time, its actual computational cost is still high and becomes prohibitive for large graphs. We show how to effectively remove nodes in the conceptual network while still retaining the densest subgraph in it. The second approach keeps the target subgraph connected in the physical network while deleting low degree nodes in the conceptual network. We further study two variations of the DCS problem. One problem aims to find the DCS with fixed number of nodes. Another problem requires a set of input seed nodes to be included in the identified subgraph. Both problems are of practical interests. For the DCS problem with input seed nodes, we design an efficient heuristic local search algorithm.

Based on the basic DCS problem, we further study several extensions. First, we observe that the conceptual network has node weights in some applications. Thus we study the DCS problem with both node and edge weights in the conceptual network. All the developed algorithms above can be readily extended to solve the new prob-

lem. Second, we formulate a more general problem, where there are multiple types of networks. Third, we provide the MapReduce implementation of the proposed algorithms.

We perform extensive empirical study using real-life biological, social and co-author networks to demonstrate the usefulness of the identified patterns and evaluate the efficiency of the developed algorithms.

4.2 Related Work

Finding the densest subgraph is an important problem with a wide range of applications [96, 21] and has attracted intensive research interests. Most of the existing work focuses on a single network, i.e., given a graph $G(V, E)$, find the subgraph with maximum density (average edge weight) [71]. This problem can be solved in polynomial time using parametric maximum flow [37]. However, its complexity $O(nm \log(n^2/m))$ is prohibitive for large graphs, where n is the number of nodes and m is the number of edges. For large graphs, efficient approximation algorithms have been developed. A 2-approximation algorithm is proposed in [3, 20]. The basic strategy is deleting the node with minimum degree. This idea can be traced back to [67], which shows that the density of the maximum core of a graph is at least half of the density of the densest subgraph. Recently, an improved $2(1+\epsilon)$ approximation greedy node deletion algorithm has been proposed [5]. The algorithm takes $O(\log_{1+\epsilon} n)$ iterations. In each iteration, it deletes a set of nodes with degree smaller than $2(1+\epsilon)$ times the density of the remaining subgraph.

Variations of the densest subgraph problem have also been studied. The densest k subgraph problem aims to find the densest subgraph with exactly k nodes, which has been shown to be NP-hard [13]. The problem of finding the densest subgraph with seed nodes requires that a set of input nodes must be included in the resulting subgraph, which can be solved in polynomial time [96].

In biomedical domain, the densest subgraph has been used to analyze the gene annotation graph [96]. The idea can be generalized to analyze multiple networks. For example, in [45, 75], the authors aim to find coherent dense subgraphs whose edges are not only densely connected but also frequently occur in multiple gene co-expression networks. Finding co-dense subgraphs that exist in multiple gene co-expression or protein interaction networks are studied in [59, 90]. The underlying assumption of these works is that the set of networks under study are of the same type.

The network-based methods have shown to be promising in integrating different datasets in systems biology. In [48], the authors use the gene expression data to weight the nodes in the protein interaction networks. Their goal is to find the maximum score connected subgraph. The maximum score connected subgraph requires that the subgraph is connected and also maximizes certain objective function. This approach has also been applied to integrate genome-wide association study datasets and protein interaction networks [7, 53]. Since the problem of finding the maximum score connected subgraph is NP-hard, heuristics are developed to find approximate solutions [7, 53]. All these methods aim to find dense subgraphs from a single network.

4.3 The DCS Problem

We adopt the classic graph density definition [3, 5, 20, 37] to formulate the DCS problem. Table 4.1 lists the main symbols and their definitions.

Definition 8. *Given a graph $G(V, E)$ and $S \subseteq V$, density $\rho(S)$ is defined as*

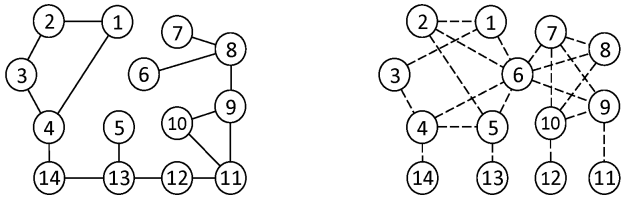
$$\rho(S) = \frac{e(S)}{|S|},$$

where $e(S)$ is the sum of the weights of edges in subgraph $G[S]$, and $|S|$ is the number of nodes in $G[S]$.

Let $G_a(V, E_a)$ be an unweighted graph representing the physical network and $G_b(V, E_b)$ be an edge weighted graph representing the conceptual network. We denote

Table 4.1: Main symbols

Symbols	Definitions
$G(V, E)$	graph G with node set V and edge set E
$G(V, E_a, E_b)$	dual networks $G_a(V, E_a)$ and $G_b(V, E_b)$
$n; m_a; m_b$	number of nodes; number of edges in G_a; number of edges in G_b
S	node set $S \subseteq V$
$G[S]$	subgraph induced by S in graph G
$G_a[S], G_b[S]$	subgraph induced by S in graph G_a, G_b
$\|S\|$	number of nodes in S
$e(u, v)$	weight of edge (u, v)
$e(S)$	sum of the weights of edges in subgraph $G[S]$
$e_b(S)$	sum of the weights of edges in subgraph $G_b[S]$
$e(S, T)$	sum of edge weights, $e(S,T) = \sum_{u \in S, v \in T} e(u, v)$
$r(u)$	weight of node u
$r(S)$	sum of node weights, $r(S) = \sum_{u \in S} r(u)$
$\rho(S)$	density of subgraph $G[S]$
$N(u)$	the set of neighbor nodes of node u in graph G
$\delta(S)$	boundary of S, $\delta(S) = \{u \in S \mid \exists v \in N(u) \cap (V \setminus S)\}$
$w_G(u)$	degree of node u in graph G
$w'_G(u)$	sum of node degree and node weight, $w'_G(u) = w_G(u) + r(u)$



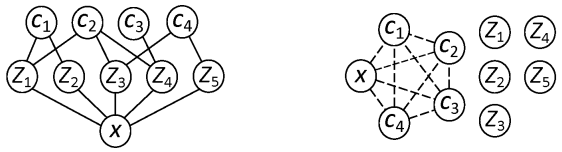
(a) Physical network G_a (b) Conceptual network G_b

Figure 4.3: An example of dual networks

the subgraphs induced by node set $S \subseteq V$ in the physical and conceptual networks as $G_a[S]$ and $G_b[S]$ respectively. For brevity, we also use $G(V, E_a, E_b)$ to represent the dual networks. Let $e_b(S)$ denote the sum of the weights of edges in subgraph $G_b[S]$.

Definition 9. *Given dual networks $G(V, E_a, E_b)$, the densest connected subgraph (DCS) consists of a set of nodes $S \subseteq V$ such that $G_a[S]$ is connected and the density of $G_b[S]$ is maximized.*

An example is shown in Figure 4.3. In this example, the DCS consists of nodes $S = \{6, 7, 8, 9, 10\}$. Its induced subgraph $G_a[S]$ is connected in the physical network



(a) Physical network G_a (b) Conceptual network G_b

Figure 4.4: Dual networks construction from an instance of the set cover problem

and $G_b[S]$ has the largest density in the conceptual network. Note that the dense component consisting of nodes $\{1, 2, 3, 4, 5, 6\}$ in G_b is not connected in G_a.

Theorem 30. *Finding the DCS in dual networks is NP-hard.*

Proof. We show that the DCS problem can be reduced from the set cover problem [56]. Let $Z = \{Z_1, \cdots, Z_l\}$ be a family of sets with $C = \{c_1, \cdots, c_h\} = \bigcup_{i=1}^{l} Z_i$ being the elements. The set cover problem aims to find a minimum subset $Z_{\text{opt}} \subseteq Z$, such that each element c_j is contained in at least one set in Z_{opt}.

The dual networks can be constructed as follows. Let the node set $V = \{x, c_1, \cdots, c_h, Z_1, \cdots, Z_l\}$. In the physical network G_a, node x is connected to every node $Z_i \in Z$, and every node $c_j \in C$ is connected to node Z_i if $c_j \in Z_i$ in the set cover problem. The conceptual network G_b is constructed by creating a unit edge weight clique among nodes $\{x, c_1, \cdots, c_h\}$ and leaving nodes $\{Z_1, \cdots, Z_l\}$ isolated.

Figure 4.4 gives an example of the dual networks constructed from an instance of the set cover problem with $Z_1 = \{c_1, c_2\}, Z_2 = \{c_1\}, Z_3 = \{c_2, c_4\}, Z_4 = \{c_2, c_3\}, Z_5 = \{c_4\}$.

Let $Z_{\text{opt}} \subseteq Z$ be the optimal solution to the set cover problem and $|Z_{\text{opt}}| = l^* \leq l$. Denote $X = \{x, c_1, \cdots, c_h\}$. The subgraph induced from $S = X \cup Z_{\text{opt}}$ is connected in G_a, and has density $\frac{h(h+1)/2}{h+l^*+1}$ in G_b. Let S' denote any node set, where $G_a[S']$ is connected. Next, we prove that the density of $G_b[S]$ is no less than that of $G_b[S']$.

First, we consider the case when S' contains all nodes in X. S' must contain a set of nodes $Z' \subseteq Z$ to be connected in G_a. Thus $S' = X \cup Z'$, $|S'| = h + 1 + |Z'|$, and $e_b(S') = h(h+1)/2$. Since Z_{opt} has the minimum number of sets (nodes) among all subsets of Z that cover all elements in C, the density of $G_b[S]$ is no less than that of $G_b[S']$.

Second, we consider the case when S' contains a subset of nodes $X' \subset X$. S' must contain a set of nodes $Z' \subseteq Z$ to be connected in G_a. Thus $S' = X' \cup Z'$. Let $|X'| = h'$ and $|Z'| = l' \geq 1$. The density of $G_b[S']$ is $\frac{h'(h'-1)/2}{h'+l'}$. Next, we show that adding nodes in $X \setminus X'$ to S' will only increase its density.

If $x \notin S'$, after adding x to S', the resulting subgraph has density $\frac{h'(h'-1)/2+h'}{h'+l'+1} > \frac{h'(h'-1)/2}{h'+l'}$ in G_b, and is also connected in G_a since x is connected to every $Z_i \in Z$. To add a node $c_j \in X \setminus X'$ to S' and make it still connected, we need to add at most one node Z_i, where $c_j \in Z_i$. The density of the resulting subgraph is at least $\frac{h'(h'-1)/2+h'}{h'+l'+2} > \frac{h'(h'-1)/2}{h'+l'}$. We can repeat this process by adding remaining nodes to S' until it contains all the nodes in X. During this process, the density of the resulting subgraph will keep increasing. In the first case, we already prove that the density of $G_b[S]$ is no less than that of $G_b[S']$ when $X \subset S'$. This completes the proof for the second case.

Therefore, the subgraph induced from $S = X \cup Z_{\text{opt}}$ is the DCS, and it gives an optimal solution to the set cover problem.

Let's continue the example in Figure 4.4. The subgraph induced from $S = \{x, c_1, c_2, c_3, c_4, Z_1, Z_3, Z_4\}$ is the DCS which is connected in G_a and has maximum density 1.25 in G_b. $Z_{\text{opt}} = \{Z_1, Z_3, Z_4\}$ is an optimal solution to the set cover problem. □

The DCS with size constraint (DCS_k) and input seed nodes (DCS_*seed*) can be defined as follows.

Definition 10. *Given dual networks $G(V, E_a, E_b)$ and an integer k, the DCS_k consists of a set of nodes $S \subseteq V$ such that $|S| = k$, $G_a[S]$ is connected and the density of $G_b[S]$ is maximized.*

Definition 11. *Given dual networks $G(V, E_a, E_b)$ and an input query node set $Q \subseteq V$, the DCS_*seed *consists of a set of nodes $S \subseteq V$ such that $Q \subseteq S$, $G_a[S]$ is connected and the density of $G_b[S]$ is maximized.*

The DCS_k and DCS_*seed* problems are also NP-hard. The proofs are omitted.

4.4 Optimality Preserving Pruning

In this section, we introduce a pruning step, which removes the *low degree leaf nodes* from the dual networks and still guarantees that the optimal DCS is contained in the resulting networks.

Definition 12. *Given dual networks $G(V, E_a, E_b)$, suppose that its DCS consists of a set of nodes S. Let $\rho(S)$ represent its density in G_b, i.e., $\rho(S) = \rho(G_b[S])$. A node $u \in V$ is a low degree leaf node if (1) u is a leaf node in G_a, i.e., $w_{G_a}(u) = 1$, and (2) its degree in G_b is less than $\rho(S)$, i.e., $w_{G_b}(u) < \rho(S)$.*

Lemma 32. *The DCS in dual networks does not contain any low degree leaf node.*

Proof. Suppose otherwise. We remove u from S and let S' be the remaining set of nodes. Since $G_a[S]$ is connected and u is a leaf node in G_a, so after deleting u, $G_a[S']$ is still connected. However, its density $\rho(S') = \frac{e_b(S')}{|S'|} = \frac{e_b(S) - w_{G_b[S]}(u)}{|S|-1} > \frac{e_b(S)}{|S|} = \rho(S)$, since $w_{G_b[S]}(u) \leq w_{G_b}(u) < \rho(S) = \frac{e_b(S)}{|S|}$. This contradicts the assumption. □

Even though the density of DCS ($\rho(S)$) is unknown beforehand, we can still effectively prune many low degree leaf nodes as follows. Let $G_0 = G$ be the original dual networks. We remove all low degree leaf nodes (using density $\rho(G_b[V])$) in the physical network G_0^a and conceptual network G_0^b respectively. That is, we remove all the nodes that have degree one in G_a and have degree less than $\rho(G_b[V])$ in G_b from the dual networks. Let the resulting dual networks be $G_1(V_1, E_a(V_1), E_b(V_1))$, where $E_a(V_1)$ and $E_b(V_1)$ represents the edge sets induced by V_1 in network G_a and G_b respectively. We then continue to remove the low degree leaf nodes using density $\rho(V_1)$ in G_1. That is, we remove all the nodes that have degree one in $G_a[V_1]$ and have degree less than $\rho(G_b[V_1])$ in G_b from the dual networks. We repeat this process until no such nodes left.

Let $\{G_0, G_1, \cdots, G_l\}$ represent the sequence of dual networks generated by this process and $\{v_j^i\}$ represent the set of nodes deleted in iteration i ($0 \leq i \leq l$). The following theorem shows that the DCS is retained in this process.

Theorem 31. *Iteratively removing low degree leaf nodes will not delete any node in the DCS.*

Proof. Consider two adjacent dual networks G_i and G_{i+1} in the sequence $\{G_0, G_1, \cdots, G_l\}$. From G_i to G_{i+1}, we delete a set of nodes $\{v_j^i\}$. For a node $u \in \{v_j^i\}$, it is a leaf node in G_i^a, and its degree in G_i^b is $w_{G_i^b}(u) < \rho(G_i^b)$. Let S_i be the node set of the DCS in G_i. We have that $\rho(G_i^b) \leq \rho(S_i)$. Thus $w_{G_i^b}(u) < \rho(S_i)$. Therefore, node u is a low degree leaf node with respect to the DCS in G_i. From the proof of Lemma 32, node u must not exist in S_i. By induction, we have that the DCS of the original dual networks is retained in the low degree leaf nodes removing process. □

Using this pruning strategy, we can safely remove the nodes that are not in the DCS, thus reduce the overall search space. Experimental results on real graphs show that 40% to 60% of the nodes can be pruned using this method.

Complexity: Let m_a and m_b be the numbers of edges in the graphs G_a and G_b respectively. At each iteration, the algorithm will delete a set of nodes and their adjacent edges in both G_a and G_b. For each deleted edge, the algorithm needs to update the node degree if the other endpoint still exists. Each update takes $O(1)$ time. Thus, the running time of the algorithm is $O(m_a + m_b)$.

Next, we introduce two greedy algorithms, DCS_RDS (Refining the Densest Subgraph) and DCS_GND (Greedy Node Deletion), to find the DCS from the size reduced dual networks.

4.5 The DCS_RDS Algorithm

The DCS_RDS algorithm first finds the densest subgraph in G_b, which usually is disconnected in G_a. It then refines the subgraph by connecting its disconnected components in G_a. Although the densest subgraph can be identified in polynomial time by the parametric maximum flow method [37], its actual complexity $O(nm \log(n^2/m))$ is

prohibitive for large graphs (n and m are the number of nodes and edges in the graph respectively). Next, we first introduce an effective procedure that can dramatically reduce the cost of finding the densest subgraph in a single graph.

4.5.1 Fast Densest Subgraph Finding in Conceptual Network

To find the densest subgraph in a single network, greedy node deletion algorithms [3, 20] and peeling algorithms [1, 5, 8] keep deleting the nodes with low degree. However, these methods do not guarantee that the densest subgraph is contained in the identified subgraph.

We introduce an approach which effectively removes nodes in G_b and still guarantees to retain the densest subgraph. Our node removal procedure is based on the following key observation.

Lemma 33. *Let $\rho(T)$ be the density of the densest subgraph $G[T]$. Any node $u \in T$ has degree $w_{G[T]}(u) \geq \rho(T)$.*

Proof. Suppose there exists a node $u \in T$ with $w_{G[T]}(u) < \rho(T)$. Then the subgraph $G[T'] = G[T \setminus \{u\}]$ has density $\rho(T') = \frac{e(T)-w_{G[T]}(u)}{|T|-1} > \frac{e(T)}{|T|} = \rho(T)$. Thus we find a subgraph $G[T']$ whose density is larger than that of $G[T]$. This contradicts the assumption that $G[T]$ is the densest subgraph. □

The lemma says that the degree of any node in the densest subgraph $G[T]$ must be no less than its density $\rho(T)$. Since the density of $G[T]$ is also equivalent to half of the average degree in $G[T]$, i.e., $\rho(T) = \overline{w}_{G[T]}/2$, this is equivalent to say that any node should have degree more than $\overline{w}_{G[T]}/2$. Note that this is a necessary condition for characterizing the densest subgraph. It is also related to the concept of d-core.

Definition 13. *The d-core D of G is the maximal subgraph of G such that for any node u in D, $w_D(u) \geq d$.*

Note that the d-core of a graph is unique and may consist of multiple connected components. It is easy to see that any subgraph in which every node's degree is no less than d is part of the d-core.

Theorem 32. *The densest subgraph $G[T]$ of G is a subgraph of the d-core D of G ($G[T] \subseteq D$) when $d \leq \rho(T)$.*

Proof. From Lemma 33, any node $u \in T$ has degree $w_{G[T]}(u) \geq \rho(T)$. Since $d \leq \rho(T)$, any node $u \in T$ has degree $w_{G[T]}(u) \geq d$. Thus $G[T]$ is a subgraph of the d-core D. □

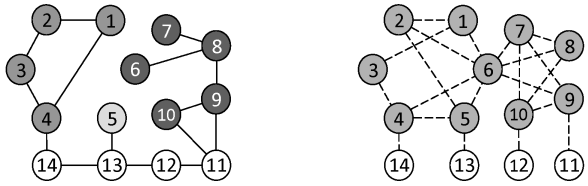
Lemma 34. *Let $\alpha = \rho(T)/d$. The d-core subgraph D is a 2α-approximation of the densest subgraph T.*

Proof. Let $D.V$ represent the node set in D. Since the density of the d-core is $\rho(D) = \frac{e(D.V)}{|D.V|} = \frac{\sum_{u \in D.V} w_D(u)}{2|D.V|} \geq \frac{\sum_{u \in D.V} d}{2|D.V|} = \frac{d}{2} = \frac{\rho(T)}{2\alpha}$, we have that $\rho(T) \leq 2\alpha\rho(D)$. □

From Theorem 32 and Lemma 34, if we can find a density value d ($d \leq \rho(T)$), then we have both: 1) $G[T] \subseteq D$, and 2) D is a $2\rho(T)/d$ approximation of the densest subgraph $G[T]$. Therefore, if we use the density of the 2-approximation subgraph generated by the greedy node deletion algorithm [3, 20] for d-core, we obtain a 4-approximation ratio ($2\alpha \leq 2(2d)/d = 4$) of the densest subgraph $G[T]$. Note that d-core can be generated by iteratively removing all nodes with degree less than d until every node in the remaining graph has degree no less than d [5].

To sum up, we use the following three-step procedure to find the exact densest subgraph from G: (1) Find a 2-approximation of the densest subgraph in G, where the density of the discovered subgraph $d \geq \rho(T)/2$; (2) Find the d-core D of G; (3) Compute the exact densest subgraph from D.

Empirical results show that after applying this approach, the remaining subgraph can be orders of magnitude smaller than the original graphs. It can significantly speed up the process of finding the exact densest subgraph. Moreover, we have shown that the density of the remaining subgraph is a 4-approximation of the density of the densest subgraph.



(a) Physical network G_a (b) Conceptual network G_b

Figure 4.5: Refining the densest subgraph

Complexity: Let n and m_b be the number of nodes and edges in the original graph G_b, and n' and m'_b be the number of nodes and edges in the d-core D. The first and second steps run in $O(m_b + n \log n)$ and $O(m_b)$ respectively. To find the exact densest subgraph from D, the parametric maximum flow algorithm runs in $O(n'm'_b \log(n'^2/m'_b))$. Note that n' (m'_b) can be orders of magnitude smaller than n (m_b).

4.5.2 Refining Subgraph in Physical Network

Suppose that the densest subgraph of G_b consists of node set T and is denoted as $G_b[T]$. The induced subgraph in the physical network $G_a[T]$ is typically disconnected. Given dual networks $G(V, E_a, E_b)$ and the densest subgraph $G_b[T]$, we use $\{G_a[V_1], G_a[V_2], \cdots, G_a[V_\kappa]\}$ to represent all connected components in $G_a[T]$, where $T = V_1 \cup V_2 \cup \cdots \cup V_\kappa$.

Example 1. *In Figure 4.5, the densest subgraph in the conceptual network consists of nodes* $T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. *Its corresponding connected components in the physical network are* $V_1 = \{6, 7, 8, 9, 10\}, V_2 = \{1, 2, 3, 4\}$, *and* $V_3 = \{5\}$.

In the next, we discuss how to refine the subgraph $G_a[T]$ to make it connected in the physical network G_a while still preserving its high density in G_b. Specifically, we consider the following dense subgraph refinement problem.

Definition 14. *Given dual networks* $G(V, E_a, E_b)$ *and the densest subgraph* $G_b[T]$ *of* G_b, *the problem of refining the densest subgraph aims to find a nonempty subset of*

Algorithm 15: Refining the densest subgraph

Input: $G(V, E_a, E_b)$, nodes T (densest subgraph in G_b)
Output: node set $\hat{S}$ of DCS

1: Find all κ connected components $\{G_a[V_i]\}$ in $G_a[T]$;
2: Sort $G_a[V_i]$ by the density $\rho(G_b[V_i])$ in descending order;
3: Weigh the node u in G_a by $(w_{G_b}(u))^{-1}$;
4: $S_1 \leftarrow V_1$;
5: **for** $i \leftarrow 1$ to $\kappa - 1$ **do**
6: Compute the shortest path $H_i(S_i, V_{i+1})$ in G_a;
7: $S_{i+1} \leftarrow S_i \cup V_{i+1} \cup H_i$;
8: $j \leftarrow \text{argmax}_i\, \rho(G_b[S_i])$; **return** S_j;

$\{G_a[V_1], G_a[V_2], \cdots, G_a[V_\kappa]\}$ *with node set* Y *and a node set* $X \subseteq V \setminus T$*, such that* $G_a[Y \cup X]$ *is connected and the density of* $G_b[Y \cup X]$ *is maximized.*

The problem of refining the densest subgraph is also NP-hard, which can be proved using similar reduction method as in the proof of Theorem 30.

We introduce a greedy heuristic procedure to refine the densest subgraph as outlined in Algorithm 15. The algorithm puts the node set T to G_a, and finds all the connected components $\{G_a[V_i]\}$. It then sorts $\{G_a[V_i]\}$ by their density $\rho(G_b[V_i])$ in descending order. It weights the nodes in G_a by the reciprocal of its degree in G_b. The intuition is that we want to select nodes that have high degree in G_b to connect $\{G_a[V_i]\}$. The algorithm merges the connected components in G_a iteratively. In each iteration, it merges two components by adding the nodes on the node weighted shortest path connecting two components. The density of the newly merged component is calculated after each iteration. The component with the largest density is returned as the DCS.

Example 2. *Continue the example in Figure 4.5. The densities of the connected components in the physical network are* $\rho(V_1 = \{6, 7, 8, 9, 10\}) = 1.6$*,* $\rho(V_2 = \{1, 2, 3, 4\}) = 0.75$*, and* $\rho(V_3 = \{5\}) = 0$*. Initially, the subgraph induced by* $S_1 = V_1$ *has density* $\rho(S_1) = 1.6$*. Algorithm 15 first connects* S_1 *and* V_2 *through the shortest path* $H_1 = \{11, 12, 13, 14\}$*. The subgraph induced by* $S_2 = S_1 \cup V_2 \cup H_1$ *has density* $\rho(S_2) = 1.31$*. After merging* V_3*, the subgraph induced by* S_3 *has density* $\rho(S_3) = 1.5$*.*

Therefore, the subgraph induced by S_1 has the largest density in G_b and is returned as the DCS.

The approximation ratio of the DCS_RDS algorithm can be estimated as $\alpha = \rho(T)/\rho(\hat{S})$, where $\rho(T)$ is the density of the densest subgraph in the conceptual network. Experimental results show that the approximation ratio is usually around 1.5∼2 using real networks.

Complexity: Algorithm 15 runs in $O(m_a + n \log n)$ as we can easily modify Dijkstra's algorithm to find the shortest path in node weighted graph by transforming each node as an edge.

4.6 The DCS_GND Algorithm

The basic DCS_GND algorithm keeps deleting nodes with low degree in the conceptual network, while avoiding disconnecting the physical network.

Definition 15. *A node is an articulation node if removing this node and the edges incident to it disconnects the graph.*

Articulation nodes can be identified in linear time by the depth first search [109]. The basic DCS_GND algorithm deletes one node in each iteration. The deleted node has the minimum degree in the conceptual network among all the non-articula-tion nodes in the physical network. Since in each iteration, only one non-articulation node is deleted, the remaining physical network will keep connected. Note that as long as the graph is not empty, there always exists a non-articulation node in the graph. Thus, the DCS_GND algorithm can always find a non-articulation node to delete until the graph becomes empty. Density of the subgraphs generated in this process is recorded and the subgraph with the largest density is returned as the identified DCS.

Example 3. *Suppose that the input physical and conceptual networks are as shown in Figures 4.6(a) and 4.6(b) respectively. Nodes $\{3, 7\}$ in gray color are articulation*

nodes and the remaining ones are non-articulation nodes. Node 6*, which has the minimum degree* 2 *among all the non-articulation nodes, will be deleted. The resulting dual networks are shown in Figures 4.6(c) and 4.6(d), where node 3 is the only articulation node.*

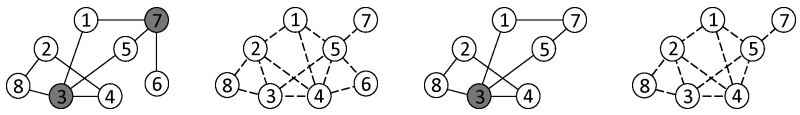
To further improve the efficiency, we can delete a set of low degree non-articulation nodes in each iteration. However, not all non-articulation nodes can be deleted simultaneously, since deleting one non-articulation node may make another non-articulation node to become an articulation node. Thus we need to find the subset of non-articulation nodes that can be deleted together.

Definition 16. *A set of non-articulation nodes are independent if the deletion of them does not disconnect the graph.*

Lemma 35. *Let* $\{B_i\}$ *represent the set of biconnected components of graph* G *such that each* B_i *has at least one non-articulation node. If we select one non-articulation node from each* B_i*, the set of selected nodes are independent non-articulation nodes.*

Proof. Suppose that we delete one non-articulation node $v_i \in B_i$. The deletion of node v_i does not disconnect B_i since it is biconnected. Since two distinct biconnected components share at most one articulation node, deleting node v_i does not disconnect any other biconnected components. So if we delete one non-articulation node from each component in $\{B_i\}$, every component is still connected. Therefore, the remaining subgraph is still connected. □

Algorithm 16 illustrates the algorithm based on deleting independent non-articulation nodes iteratively. Parameter γ is used to control the degree of the non-articulation nodes to be deleted. γ is usually set between $0 \sim 2$. Since $2\rho(G_b)$ is the average node degree, there are about half of the nodes whose degree is smaller than the threshold $2\rho(G_b)$. More nodes are deleted in each iteration when larger γ value is used. Please refer to experimental evaluation for further discussion on the effect of γ. If all low



(a) $G_a[V_0]$ (b) $G_b[V_0]$ (c) $G_a[V_1]$ (d) $G_b[V_1]$

Figure 4.6: Greedy node deletion example

Algorithm 16: Fast DCS_GND algorithm

Input: $G(V, E_a, E_b)$, parameter $\gamma > 0$
Output: node set $\hat{S}$ of DCS

1: $V_0 \leftarrow V$; $i \leftarrow 0$;
2: **while** $|V_i| > 0$ **do**
3: Compute the articulation nodes A in $G_a[V_i]$; $A \leftarrow V_i \setminus A$;
4: Select a set of nodes $L \subseteq A$ such that the nodes in L are independent non-articulation nodes and have low degrees, i.e., for any $u \in L$, $w_{G_b[V_i]}(u) \leq \gamma \cdot \rho(G_b[V_i])$;
5: **if** $|L| = 0$ **then** $L \leftarrow \{u | u = \text{argmin}_{v \in A}\, w_{G_b[V_i]}(v)\}$;
6: $V_{i+1} \leftarrow V_i \setminus L$; $i \leftarrow i + 1$;
7: $j \leftarrow \text{argmax}_i\, \rho(G_b[V_i])$; **return** V_j;

degree nodes are articulations nodes, the algorithm picks the non-articulation node with the minimum degree to delete.

Example 4. *Let's continue the example in Figure 4.6. Suppose that* $\gamma = 1.5$. *We have* $\gamma \cdot \rho(G_b[V_0]) = 2.44$. *The fast DCS_GND method will delete nodes* $\{6, 8\}$ *simultaneously, since they have degree* $2 < 2.44$ *and are independent non-articulation nodes.*

Complexity: It takes $O(nm_a)$ time to find the non-articulation nodes and biconnected components in line 3 by depth first search. It takes $O(m_b)$ time to find the low degree nodes in line 4. It takes $O(m_b + n \log n)$ time to select the node with minimum degree in line 5. Thus, the DCS_GND algorithm runs in $O(nm_a + m_b + n \log n)$.

We can estimate the approximation ratio of DCS_GND as follows. When deleting a node v, we assign its incident edges in G_b to it. Let $edg(v)$ denote the sum of the edge weights, and $edg_{\max}$ represent the maximum $edg(v)$ among all nodes (deleted in order by the algorithm). Let S be the node set of the optimal DCS, T be the node set of the densest subgraph in the conceptual network G_b. We have the following inequality.

Lemma 36. $\rho(S) \leq \rho(T) \leq edg_{\max}$.

Proof. It is easy to see that $\rho(S) \leq \rho(T)$. Next, we show that $\rho(T) \leq edg_{\max}$. Each edge in $G_b[T]$ must be assigned to a node in T in the node deletion process. Thus we have that $e_b(T) \leq \sum_{u \in T} edg(u) \leq \sum_{u \in T} edg_{\max} = |T| \cdot edg_{\max}$. This means that $\rho(T) = \frac{e_b(T)}{|T|} \leq edg_{\max}$. □

Lemma 37. *Let $\hat{S}$ be the node set of the DCS identified by the DCS_GND algorithm. The approximation ratio of the algorithm is $\alpha = edg_{\max}/\rho(\hat{S}) \geq \rho(S)/\rho(\hat{S})$.*

Based on Lemma 37, we can estimate the approximation ratio α from the results returned by the algorithm. Empirical study shows that α is usually around 2 in real networks.

The DCS_GND algorithm can be easily extended to solve the DCS_k and DCS_*seed* problems. For the DCS_k problem, we can keep deleting low degree non-articulation nodes until there are k nodes left. For the DCS_*seed* problem, we avoid deleting the seed nodes during the process. The approximation ratio analysis discussed above also applies to these variants.

4.7 The DCS_MAS Algorithm for the DCS_*seed* Problem

In this section, we introduce a heuristic algorithm, DCS_MAS, for the DCS_*seed* problem. DCS_MAS uses a local search procedure, which iteratively include one more node with the maximum adjacency value to the visited nodes.

The algorithm is outlined in Algorithm 17. Given the query nodes, we first compute the Steiner tree in the physical network by the Mehlhorn's algorithm [86]. Thus the query nodes become connected.

Next, the algorithm begins a local search process. The Steiner tree is used as the initial subgraph. In each iteration (lines 4-5), the algorithm adds the node $u \in$

Algorithm 17: Maximum adjacency search (DCS_MAS) algorithm for the DCS_*seed* problem

Input: $G(V, E_a, E_b)$, query nodes Q, parameter K
Output: node set $\hat{S}$ of DCS_*seed*

1: Compute the Steiner tree of Q in G_a, and let S be its node set;
2: $V_0 \leftarrow S$; $i \leftarrow 0$;
3: **while** $|V_i| < K$ **do**
4: $\quad u \leftarrow \text{argmax}_{v \in \delta_a(V \setminus V_i)} e_b(\{v\}, V_i)$;
5: $\quad V_{i+1} \leftarrow V_i \cup \{u\}$; $i \leftarrow i + 1$;
6: $j \leftarrow \text{argmax}_i \rho(G_b[V_i])$; **return** V_j;

$\delta_a(V \setminus V_i)$ with the maximum adjacency value $e_b(\{v\}, V_i)$ to V_i. Here, $\delta_a(S)$ denotes the boundary of the node set S in the physical network. The intuition is that the node u is adjacent to V_i in the physical network, and has the maximum adjacency value to V_i in the conceptual network. The subgraph during the local search process is always connected in the physical network. The subgraph with the maximum density in the conceptual network during the local search process is returned. Parameter K is used to control the search space. When the number of nodes in V_i is equal to K, the algorithm will terminate.

The local search process needs at most K iterations. Let $\overline{N}_a$ be the average number of neighbors in the physical network and $\overline{N}_b$ be the average number of neighbors in the conceptual network. Then, in the ith iteration, it takes $O(|V_i| \cdot \overline{N}_a)$ to find the node with the maximum adjacency value in line 4, and it takes $O(\overline{N}_b)$ to add this node and update the adjacency values of its neighborhood nodes in the conceptual network. Thus the local search process runs in $O(\sum_i(|V_i| \cdot \overline{N}_a + \overline{N}_b)) = O(K^2\overline{N}_a + K\overline{N}_b)$. Mehlhorn's algorithm runs in $O(m_a + n \log n)$ [86].

4.8 Node Weights in the Conceptual Network

In the conceptual network, in addition to edge weights, we can often have node weights as well. For example, in the genetic interaction network, in addition to measuring genetic interactions, i.e., the edge weights, we can also apply statistical

tests to measure node weights [52]. A node weight represents the strength of the association between a single genetic factor and the disease trait. In this section, we study the DCS problem when there are both edge and node weights in the conceptual network. To integrate both node and edge weights, we revise the classic density definition as follows.

Definition 17. *Given a graph $G(V, E)$ and a set of nodes $S \subseteq V$, the node and edge weighted density $\rho(S)$ is defined as*

$$\rho(S) = \frac{e(S) + r(S)}{|S|},$$

where $e(S)$ is the sum of the weights of edges in the subgraph $G[S]$, $r(S) = \sum_{u \in S} r(u)$ is the sum of the node weights, and $r(u)$ is the weight of node u. For brevity, the node and edge weighted density is also referred to as density when there is no ambiguity.

If we need to tune the composition ratio of the node and edge weights, we can reweight the nodes and edges in a pre-processing step. Let $\eta (0 \leq \eta \leq 1)$ be a constant. For each node u, its weight is changed to $(1-\eta) \cdot r(u)$. For each edge (u, v), its weight is changed to $\eta \cdot e(u, v)$. Then, the node and edge weighted density of subgraph $G[S]$ is changed to $(\eta \cdot e(S) + (1-\eta) \cdot r(S))/|S|$. Thus, we can tune the composition ratio of the two parts through the node and edge reweighting step.

The node and edge weighted density is adopted in the work [39]. When all nodes have unit weights, we have that $r(S) = |S|$ and $\rho(S) = \frac{e(S)}{|S|} + 1$, which degrades to the classic density as in Definition 8. The DCS problem with both node and edge weights is also NP-hard.

Next, we show how to extend the techniques developed before to solve the DCS problem following Definition 17.

4.8.1 Optimality Preserving Pruning

In this section, we extend the optimality preserving pruning step. We need to re-define the *low degree leaf nodes*. We still use $w_G(u)$ to denote the degree of node u in

graph G. Let $w'_G(u)$ denote the sum of node degree and node weight of node u, i.e., $w'_G(u) = w_G(u) + r(u)$.

Definition 18. *Given dual networks $G(V, E_a, E_b)$, suppose that its DCS consists of a set of nodes S. Let $\rho(S)$ represent the node and edge weighted density in G_b, i.e., $\rho(S) = \rho(G_b[S])$. A node $u \in V$ is a low degree leaf node if (1) u is a leaf node in G_a, i.e., $w_{G_a}(u) = 1$, and (2) u satisfies that $w'_{G_b}(u) < \rho(S)$.*

Lemma 38. *The DCS in dual networks does not contain any low degree leaf node.*

Proof. Suppose otherwise. We remove u from S and let S' be the remaining set of nodes. Since $G_a[S]$ is connected and u is a leaf node in G_a, after deleting u, $G_a[S']$ is still connected. However, its density $\rho(S') = \frac{e_b(S')+r(S')}{|S'|} = \frac{e_b(S)+r(S)-w'_{G_b[S]}(u)}{|S|-1} > \frac{e_b(S)+r(S)}{|S|} = \rho(S)$, since $w'_{G_b[S]}(u) \leq w'_{G_b}(u) < \rho(S) = \frac{e_b(S)+r(S)}{|S|}$. This contradicts the assumption that $G[S]$ is the optimal solution to the DCS problem. □

Therefore, when we iteratively remove the low degree leaf nodes from the dual networks, the DCS is retained in the resulting networks.

Theorem 33. *Iteratively removing low degree leaf nodes will not delete any node in the DCS.*

Proof. The proof is similar to that of Theorem 31. □

The optimality preserving pruning procedure still runs in $O(m_a + m_b)$.

4.8.2 The DCS_RDS Algorithm

4.8.2.1 The Greedy Node Deletion Algorithm

In this section, we extend the greedy node deletion algorithm to find a 2-approximation for the densest subgraph problem with Definition 17.

Algorithm 18 shows the greedy node deletion algorithm. It keeps deleting the node with the minimum $w'_{G[V_i]}(u)$ value. After deleting one node, we compute the density

Algorithm 18: Greedy node deletion algorithm for the densest subgraph problem with the node and edge weighted density

Input: $G(V, E)$
Output: node set $\hat{S}$

1: $V_0 \leftarrow V$; $i \leftarrow 0$;
2: **while** $|V_i| > 0$ **do**
3: $\quad u \leftarrow \text{argmin}_{v \in V_i} w'_{G[V_i]}(v)$;
4: $\quad V_{i+1} \leftarrow V_i \setminus \{u\}$; $i \leftarrow i + 1$;
5: $j \leftarrow \text{argmax}_i \rho(G[V_i])$; **return** V_j;

of the remaining subgraph. Then the subgraph with the maximum density during the node deletion process is returned. This algorithm still finds a 2-approximation solution.

Lemma 39. *For any node u in the densest subgraph $G[T]$, $w'_{G[T]}(u) \geq \rho(T)$.*

Proof. Since $G[T]$ is the densest subgraph, $\rho(T) \geq \rho(T \setminus \{u\})$. Therefore, we have that $\frac{e(T)+r(T)}{|T|} \geq \frac{e(T)+r(T)-w'_{G[T]}(u)}{|T|-1}$. Then we can prove this lemma. □

Lemma 40. *Algorithm 18 obtains a 2-approximation solution to the densest subgraph problem with Definition 17.*

Proof. Let $S = V_i$. That is, S represents the remaining nodes in the ith iteration. We have that $\sum_{u \in S} w'_{G[S]}(u) = 2|S| \cdot \rho(S) - r(S) \leq 2|S| \cdot \rho(S)$, which means that the average $w'_{G[S]}(u)$ value is no greater than $2\rho(S)$. If a node u has the minimum $w'_{G[S]}(u)$ value, we have that $w'_{G[S]}(u) \leq 2\rho(S)$.

Now, consider the first time in the iteration when a node u from the densest subgraph $G[T]$ is deleted. Clearly, $S \supseteq T$. Thus we have that $\rho(T) \leq w'_{G[T]}(u) \leq w'_{G[S]}(u) \leq 2\rho(S)$, where we use Lemma 39 in the first inequality. This implies that $\rho(S) \geq \rho(T)/2$ and hence the algorithm gives a 2-approximation solution. □

Algorithm 18 runs in $O(m_b + n \log n)$ time when using Fibonacci heaps [26].

4.8.2.2 Removing Low Degree Nodes

In this section, we show that the densest subgraph is contained in the node and edge weighted d-core. Thus, we can prune the search space by first finding the node and edge weighted d-core.

Definition 19. *The node and edge weighted d-core D of G is the maximal subgraph of G such that for any node u in D, $w'_D(u) \geq d$.*

Theorem 34. *The densest subgraph $G[T]$ of G is a subgraph of the node and edge weighted d-core D of G ($G[T] \subseteq D$) when $d \leq \rho(T)$.*

Proof. The proof is similar to that of Theorem 32. □

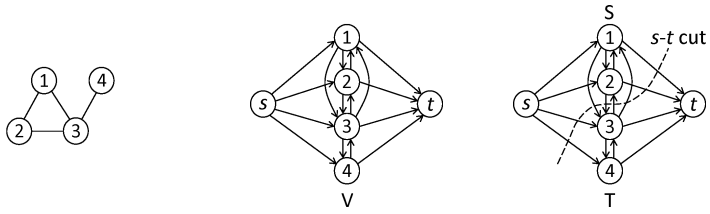
To compute the node and edge weighted d-core, we can iteratively remove the nodes with low $w'_G(u)$ values, i.e., $w'_G(u) < d$. This procedure still runs in $O(m_b)$.

4.8.2.3 The Parametric Maximum Flow Method

In this section, we show how to apply the parametric maximum flow method to exactly solve the densest subgraph problem with the node and edge weighted density. Given an undirected graph G, the flow network can be constructed as follows.

1. Replace each edge (u, v) of G by two oppositely directed edges $\langle u, v\rangle$ and $\langle v, u\rangle$ of capacity $e(u, v)$;
2. Add a source node s and a sink node t;
3. Create directed edge $\langle s, u\rangle$ of capacity $w_G(u) + 2r(u)$ for each node u;
4. Create directed edge $\langle u, t\rangle$ of capacity 2λ for each node u;

Figure 4.7 shows one example of constructing the flow network. Figure 4.7(a) shows the original undirected graph, and Figure 4.7(b) shows the flow network. We change the undirected edge into two directed edges, add the source node s and the edges from s to each node, and add the sink node t and the edges from each node to t.



(a) Original network (b) Flow network (c) s-t cut

Figure 4.7: Constructing the flow network from the original network

By computing the parametric maximum flow on the flow network, we can solve the densest subgraph problem exactly. Suppose that any s-t cut partitions the node set V into two parts S and T, where S is on the s side and T is on the t side. For example, in Figure 4.7(c), the s-t cut is indicated by the dotted curve, and we have that $S = \{1, 2\}$ and $T = \{3, 4\}$. The capacity of an s-t cut is

$$\sum_{u \in T}(w_G(u) + 2r(u)) + e(S, T) + 2\lambda|S|$$
$$= 2e(V) + 2r(V) - 2(e(S) + r(S) - \lambda|S|).$$

Therefore, minimizing the capacity of an s-t cut is equivalent to maximizing the quantity $e(S) + r(S) - \lambda|S|$. Thus we can maximize the ratio $\frac{e(S)+r(S)}{|S|}$ by searching for the largest λ value [37].

4.8.2.4 Refining Subgraph in Physical Network

We can simply change the line 3 of Algorithm 15 to the following line.

3: Weigh the node u in G_a by $(w'_{G_b}(u))^{-1}$;

The analysis of the approximation ratio and complexity is the same as that of the original algorithm.

4.8.3 The DCS_GND Algorithm

To extend the DCS_GND algorithm to handle the node and edge weighted conceptual network, we can change the lines 4 and 5 in Algorithm 16 to the following two lines.

4: Select a set of nodes $L \subseteq A$ such that the nodes in L are independent non-articulation nodes and have low $w'_{G_b[V_i]}(u)$ values, i.e., for any $u \in L$, $w'_{G_b[V_i]}(u) \leq \gamma \cdot \rho(G_b[V_i])$;

Table 4.2: Complexities of the methods

Methods (with/without node weights)		Complexities
Optimality preserving pruning		$O(m_a + m_b)$
DCS_RDS	Greedy node deletion	$O(m_b + n \log n)$
	Removing low degree nodes	$O(m_b)$
	Parametric maximum flow	$O(n' m'_b \log(n'^2/m'_b))$
	Refining densest subgraph	$O(m_a + n \log n)$
Basic/fast DCS_GND		$O(nm_a + m_b + n \log n)$
DCS_MAS	Maximum adjacency search	$O(K^2 \overline{N}_a + K \overline{N}_b)$
	Mehlhorn's algorithm	$O(m_a + n \log n)$

5: **If** $|L| = 0$ **then** $L \leftarrow \{u | u = \text{argmin}_{v \in A} w'_{G_b[V_i]}(v)\}$;

Next, we derive the approximation ratio. We still use S to denote the node set of the optimal DCS, T to denote the node set of the densest subgraph in the conceptual network G_b, and $edg(u)$ to denote the sum of the weights of edges, which are deleted together with node u during the node deletion process. We have the following lemmas.

Lemma 41. $\rho(S) \leq \rho(T) \leq \max_{u \in V}(edg(u) + r(u))$.

Proof. It is easy to see that $\rho(S) \leq \rho(T)$. Next, we show that $\rho(T) \leq \max_{u \in V}(edg(u) + r(u))$. Each edge in $G_b[T]$ must be assigned to a node in T in the node deletion process. Thus we have that $e_b(T) \leq \sum_{u \in T} edg(u)$. Therefore, $e_b(T) + r(T) \leq \sum_{u \in T} edg(u) + \sum_{u \in T} r(u) \leq |T| \cdot \max_{u \in V}(edg(u) + r(u))$. This means that $\rho(T) = \frac{e_b(T) + r(T)}{|T|} \leq \max_{u \in V}(edg(u) + r(u))$. □

Lemma 42. *Let $\hat{S}$ be the node set of the DCS identified by the DCS_GND algorithm. The approximation ratio of the algorithm is $\alpha = \max_{u \in V}(edg(u) + r(u))/\rho(\hat{S}) \geq \rho(S)/\rho(\hat{S})$.*

The complexity of the modified DCS_GND algorithm is the same as that of the original algorithm.

4.8.4 The DCS_MAS Algorithm for the DCS_*seed* Problem

We can simply change line 4 in Algorithm 17 to the following line.

4: $u \leftarrow \text{argmax}_{v \in \delta_a(V \setminus V_i)}(e_b(\{v\}, V_i) + r(v))$;

The complexity is the same as that of the original algorithm.

Table 4.2 lists the complexities of the methods. The complexity of each method for the networks with node weights is the same as that of the corresponding method for the networks without node weights.

4.9 Further Extensions

In this section, we discuss two extensions to the DCS problem: one from the theoretical perspective, and one from the implementation perspective.

4.9.1 A More General Problem Formulation

The basic DCS problem can be treated as a special case of a more general problem formulation, where the edges in the network may have different labels [62]. Specifically, we can define the graph with multiple edge labels as a triple $G(V, E, L)$, where V is a set of nodes, L is a set of labels, and E is a set of labeled edges. One triple $(u, v, l) \in E$ with $u, v \in V$ and $l \in L$ represents one edge (u, v) with edge label l. An edge label induced subgraph, $G_l(V, E_l)$, consists of the edges E_l of a particular label $l \in L$. Different optimization objective functions or constraints (such as edge density, k-edge or k-vertex connectivity, diameter of the subgraph, etc.) can be defined on different edge label induced subgraphs.

In the DCS problem, there are two edge labels, i.e., conceptual and physical edges. On the conceptual edge induced subgraph, the objective function is to maximize the density among a subset of nodes. On the physical edge induced subgraph, the constraint is the connectivity among the nodes. In the future work, we will explore the generalized subgraph discovery problem and their applications in the real-world.

4.9.2 MapReduce Implementations

Recently, there have been a lot of interests using MapReduce for processing large graphs [5]. In the following, we show that the DCS_RDS and DCS_GND methods can be easily implemented on top of the MapReduce framework. The DCS_MAS method is a local search heuristic, thus we do not discuss its implementation on MapReduce. In the discussion, we assume that the classic density in Definition 8 is used in the DCS problem. All the implementations can also be readily extended to solve the DCS problem with the node and edge weighted density in Definition 17.

4.9.2.1 DCS_RDS on MapReduce

The DCS_RDS method has four steps: computing the 2-approximation densest subgraph; finding d-core; computing the exact densest subgraph from the d-core; refining the densest subgraph in the physical network.

In the first step, we can directly apply the MapReduce algorithm developed in [5] to compute a $2(1+\epsilon)$-approximation densest subgraph, where ϵ is a small constant. Note that this algorithm gives a relaxed approximation ratio. The density of the discovered subgraph will be used as the threshold for finding d-core in the second step. The densest subgraph is still guaranteed to be contained in the d-core, since we are using a slightly smaller d value. In the second step, we can apply the same strategy when removing low degree nodes to find the d-core. In the third step, we can use the MapReduce implementation of the Ford-Fulkson method [43] as a subroutine of the parametric maximum flow method [37] to compute the densest subgraph. In the fourth step, we can apply the MapReduce implementation of single-source shortest path algorithm [76] to connect the disconnected components in the physical network.

4.9.2.2 DCS_GND on MapReduce

The DCS_GND method has four steps: computing the articulation nodes of G_a; computing the density $\rho(G_b)$; selecting the low degree independent non-articulation

nodes; removing those nodes.

To compute the density $\rho(G_b)$ and remove nodes, we can still apply the method in [5]. To compute the articulation nodes, we can use the algorithm in [4]. Let $\pi(u)$ denote whether node u is an articulation node. Let $\theta(u)$ denote the identifier of the biconnected component which contains the node u. Note that $\theta(u)$ may contain multiple identifiers if node u is an articulation node. The output of this step has the form $\langle u; \pi(u), \theta(u) \rangle$.

To select the low degree independent non-articulation nodes, we need two passes on MapReduce. In the first pass, we duplicate each edge (u, v) and its weight $e(u, v)$ to two $\langle key; value \rangle$ pairs $\langle u; e(u, v) \rangle$ and $\langle v; e(v, u) \rangle$ in the mapping step. We also add $\langle u; \pi(u), \theta(u) \rangle$ for each node in the mapping step. The input to the reduce task is of the form $\langle u; \pi(u), \theta(u), e(u, v_1), e(u, v_2), \ldots, e(u, v_k) \rangle$ where $v_1, v_2, \ldots, v_k$ are the neighbors of u. The reducer will do nothing if node u is an articulation node. Otherwise, the reducer will sum up the associated edge weights for each key, and output $\langle \theta(u); u, w_{G_b}(u) \rangle$ if $w_{G_b}(u) \leq \gamma \cdot \rho(G_b)$. In the second pass, we just emit the key-value pairs in the form of $\langle \theta(u); u, w_{G_b}(u) \rangle$ in the mapping step. The reducer will pick one node v with the minimum degree from each $\theta(u)$, and output $\langle v; \$ \rangle$, which denotes that the node v will be deleted.

4.10 Experimental Results

In this section, we perform comprehensive experiments to evaluate the effectiveness and efficiency of the proposed methods using a variety of real and synthetic datasets. All the programs are written in C++. All experiments are performed on a server with 32G memory, Intel Xeon 3.2GHz CPU, and Redhat OS.

Table 4.3: Statistics of the dual biological networks

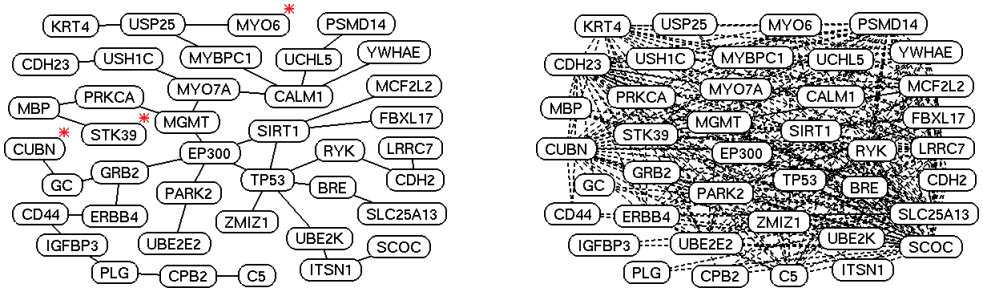
Dual networks	Abbr.	#nodes	#edges in G_a	#edges in G_b
WTCCC	WT	8,468	25,715	67,744
ARIC	AR	8,468	25,715	81,810

4.10.1 Effectiveness Evaluation in the Biological Application Domain

We first evaluate the effectiveness of the DCS method in the biological application domain. The dual biological networks include the physical protein interaction network and the conceptual genetic interaction network. The protein interaction network is downloaded from the BioGRID database (`http://thebiogrid.org/`). After filtering out duplicate interactions, the network contains 8,468 proteins and 25,715 unique physical bonding interactions.

The first genetic interaction network is generated by performing chi-square test on genetic marker pairs in the Wellcome Trust Case Control Consortium (WTCCC) hypertension dataset [16]. The WTCCC dataset includes 4,890 European adults. The most significant interactions between genes are used to weight the edges in the genetic interaction network, which has 67,744 edges. Note that we use half of the samples in the WTCCC dataset to construct the dual networks. Another half is used for significance evaluation of the identified DCS.

The second genetic interaction network is generated in a similar way from the atherosclerosis risk in communities (ARIC) study dataset downloaded from dbGaP [73, 111]. The ARIC dataset includes 15,792 African American and European American adults. We focus on the 9,319 European American adults. We study hypertension and calculate genetic interaction using the chi-square test. The resulting genetic interaction network has 81,810 edges. Note that the WTCCC and ARIC datasets are independent. We use the WTCCC dataset to evaluate the significance of the DCS identified from the ARIC dataset. Table 4.3 shows the basic statistics of the dual biological networks constructed from the WTCCC and ARIC datasets.



(a) Subgraph in protein interaction network (b) Subgraph in genetic interaction network

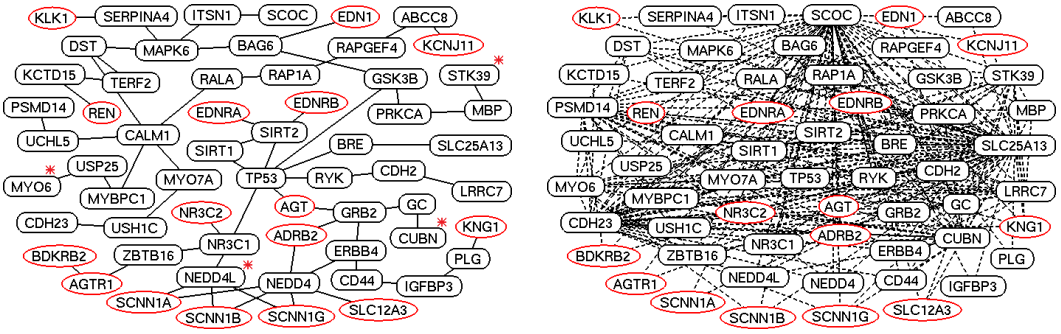
Figure 4.8: The DCS_k ($k = 40$) identified from the WTCCC dataset

4.10.1.1 The DCSs Identified from the WTCCC Dataset

The DCS identified in the dual biological networks has 211 nodes. The figures are omitted because of the large size. The set of nodes are sparsely connected in the protein interaction network, while the subgraph in the genetic interaction network has high density. Specifically, the DCS has 282 edges in the protein interaction network and $4,258$ edges in the genetic interaction network.

Note that the densest subgraph of the genetic interaction network is not connected in the protein interaction network. There are 73 nodes in the densest subgraph of the genetic interaction network. Only 2 of them are connected in the protein interaction network. This demonstrates that dual networks can help to uncover pattern that cannot be identified in individual networks. Such pattern cannot be identified by finding dense subgraphs preserved in both networks either. There are 68 overlapping nodes between the densest subgraph in the genetic interaction network and the DCS identified by DCS_RDS.

Figure 4.8 shows the identified DCS_k with $k = 40$. From the figure, it is clear that the identified subgraph is connected in the protein interaction network and highly dense in the genetic interaction network. Several genes in this subgraph have been reported to be associated with hypertension. For example, MYO6 encodes an actin-based molecular motor involved in intracellular vesicle and organelle transport, and has been shown to have association with hypertension [101]. The CUBN gene is associated with albuminuria, which is an important factor for cardiovascular disease



(a) Subgraph in protein interaction network (b) Subgraph in genetic interaction network

Figure 4.9: The DCS_*seed* identified from the WTCCC dataset (renin pathway genes are in red ellipses)

[85]. The STK39 gene has been reported many times as a hypertension susceptibility gene [117]. This gene encodes a serine/threonine kinase that is thought to function in the cellular stress response pathway. These genes are highlighted by stars in the figure. Other genes in the identified subgraph are potential hypertension candidate genes or important for signal transduction in hypertension related pathways.

To identify the DCS_*seed*, we use a set of 16 genes in renin pathways known to be associated with hypertension as the input seed nodes [127]. Renin pathway, also called renin-angiotensin system, is a hormone system that regulates blood pressure. The resulting subgraphs are shown in Figure 4.9. The input seed genes are in red ellipses and the remaining nodes represent the newly added genes. As can be seen from the figure, the seed nodes are originally not directly connected in the protein interaction network. The newly added genes tend to have large degree in the genetic interaction network. In addition to the genes discussed above, we can see the NEDD4L gene is connected to multiple seed genes. It has been reported that NEDD4L is involved in the regulation of plasma volume and blood pressure by controlling cell surface expression of the kidney epithelial Na^+ channel [78].

To evaluate the statistical significance of the discovered DCSs, we apply 4 widely used pathway evaluation methods: the GenGen method, the gene set ridge regression (GRASS) method, the Plink set-based test method, and the hybrid set-based test (HYST) method [116, 74]. Given a set of genes, these methods evaluate the

Table 4.4: P-values of the DCSs identified from the WTCCC dataset (without node weights, tested on half of the WTCCC dataset)

Methods	GenGen	GRASS	Plink	HYST
DCS (1)	2.4×10^{-6}	1.0×10^{-6}	2.3×10^{-6}	1.1×10^{-9}
DCS (2)	1.6×10^{-5}	2.8×10^{-5}	4.6×10^{-5}	5.6×10^{-7}
DCS (3)	4.8×10^{-5}	7.4×10^{-5}	9.5×10^{-5}	8.2×10^{-7}
DCS_k	5.6×10^{-5}	1.3×10^{-6}	4.6×10^{-6}	3.7×10^{-8}
DCS_*seed*	8.5×10^{-5}	4.9×10^{-6}	1.5×10^{-5}	2.4×10^{-6}
DS	0.36	0.47	0.33	0.17
MSCS	0.15	0.13	0.21	0.12

significance of the association between the set of genes and the disease phenotype. These methods adopt the null hypothesis that none of the genes in a gene set harbor genetic markers associated with the disease risk. The alternative hypothesis is that at least one gene harbors genetic markers associated with the disease risk. Different methods adopt different strategies to perform the tests. The GenGen method assigns the best test statistic among genetic markers in or near a gene to represent the gene level signal, then calculates the Kolmogorov-Smirnov-like enrichment score for a pathway [115]. The GRASS method first uses the regularized regression to select representative genetic markers for each gene, then assesses their joint association with the disease risk [22]. The Plink set-based test method selects the independent and significant genetic markers in the pathway, and then calculates the average of the test statistics as the pathway enrichment score [94]. The HYST method combines the extended Simes' test and the scaled chi-square test to assess the overall significance of the association in a set of genetic markers [74]. Note that the test dataset consists of the samples that are not used for constructing the dual networks to ensure the independence between pattern discovering and significance evaluation.

Table 4.4 shows the p-value of the identified DCSs. DCS (1), (2) and (3) represent the top-3 DCSs. The top DCSs are identified iteratively: after the top-1 DCS is identified, we remove its nodes and edges from the dual networks; the DCS_GND algorithm is then applied to each connected component to find the next DCS in the remaining graph. As can be seen from the table, the DCSs are highly significant.

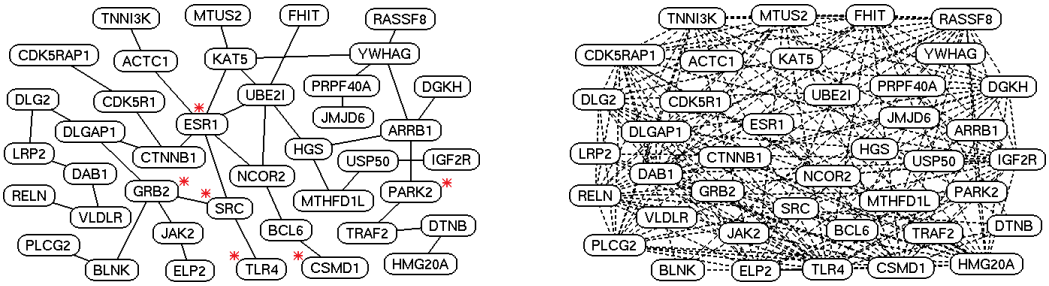
In the table, we also show the results of two other methods for finding pathways in biological networks. One method finds the densest subgraph (DS) in the protein interaction network. Another method aims to find the maximum-score connected subgraph (MSCS) in the protein interaction network [48]. The DS method uses the most significant genetic interactions between genes to weight the edges in the protein interaction network. The MSCS method uses the most significant chi-square test statistics to weight the nodes in the protein interaction network. As we can see, the subgraphs identified by these two methods are not as significant as the DCSs. This indicates the importance of integrating the complementary information encoded in the physical protein interaction network and the conceptual genetic interaction network.

4.10.1.2 The DCSs Identified from the ARIC Dataset

Using the genetic interaction network generated from the ARIC dataset, the identified DCS has 184 nodes. The figures are omitted because of the large size. The set of nodes are sparsely connected in the protein interaction network, while the subgraph in the genetic interaction network has high density. Specifically, the DCS has 246 edges in the protein interaction network and $4,135$ edges in the genetic interaction network.

Similar to the results from the WTCCC dataset, the densest subgraph of the genetic interaction network is not connected in the protein interaction network. The densest subgraph of the genetic interaction network consists of 89 nodes, however, the induced subgraph in the protein interaction network only contains 6 edges.

Figure 4.10 shows the identified DCS_k with $k = 40$. Several genes in the identified DCS have been reported to be associated with hypertension. The CSMD1 gene encodes a transmembrane protein and is a potential tumor suppressor. It has been shown to have association with hypertension [44]. The ESR1 gene encodes an estrogen receptor and has been shown to have association with pregnancy-induced hyperten-

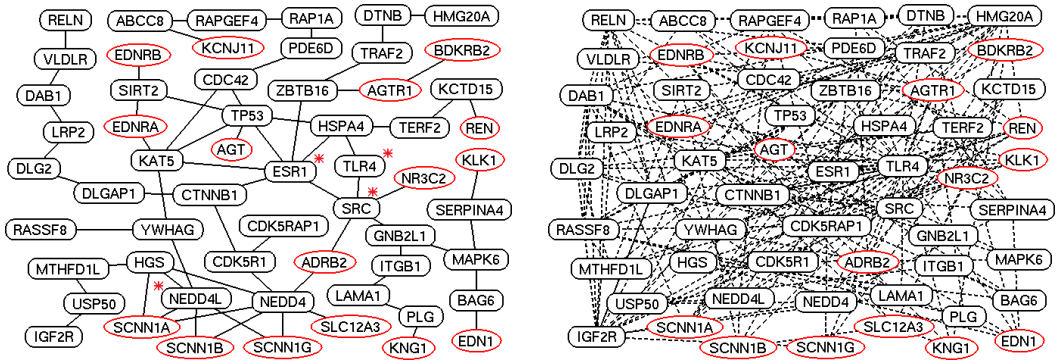


(a) Subgraph in protein interaction network (b) Subgraph in genetic interaction network

Figure 4.10: The DCS_k ($k = 40$) identified from the ARIC dataset

sion [107]. Considerable evidence has been accumulated suggesting the involvement of receptor tyrosine kinases in the pathogenesis of pulmonary arterial hypertension [93]. The SRC gene encodes a tyrosine-protein kinase, and has been shown to be associated with pulmonary arterial hypertension [93]. The TLR4 gene encodes toll-like receptor 4, which contributes to blood pressure regulation and vascular contraction in spontaneously hypertensive rats [15] and was also reported to be associated with hypertension in human [130]. The association study in 199 Nigerian families reveals that the PARK2 gene is significantly associated with the risk for hypertension [110]. This result is replicated in the Korean population [54]. The PARK2 and GRB2 genes exist in both the DCS_*seed*s identified from the WTCCC and ARIC datasets, which are shown in Figure 4.8 and 4.10 respectively. The GRB2 gene, together with the SRC gene, is interacting with the platelet-derived growth factor, which has been implicated in the pathobiology of vascular remodeling [47]. These genes are highlighted by stars in the figure. Other genes in the identified subgraph are potential hypertension candidate genes or important for signal transduction in hypertension related pathways.

To identify the DCS_*seed*, we still use the set of 16 genes in the renin pathways as the input seed nodes [127]. The resulting subgraphs are shown in Figure 4.11. The newly added genes tend to have large degree in the genetic interaction network. We observe the ESR1, TLR4, and SRC genes, which have been discussed above. We also observe the NEDD4L gene, which was observed in Figure 4.9. In Figure 4.9,



(a) Subgraph in protein interaction network (b) Subgraph in genetic interaction network

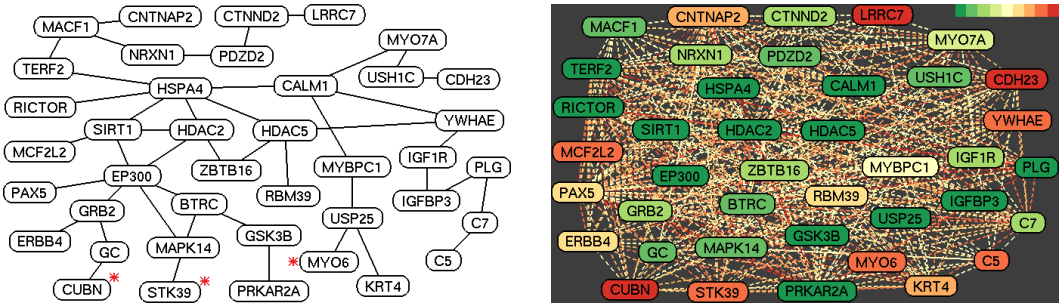
Figure 4.11: The DCS_*seed* identified from the ARIC dataset (renin pathway genes are in red ellipses)

Table 4.5: *P*-values of the DCSs identified from the ARIC dataset (without node weights, tested on the WTCCC dataset)

Methods	GenGen	GRASS	Plink	HYST
DCS (1)	7.9×10^{-6}	5.7×10^{-6}	8.3×10^{-6}	3.1×10^{-8}
DCS (2)	5.2×10^{-5}	6.9×10^{-5}	9.4×10^{-5}	1.3×10^{-6}
DCS (3)	8.3×10^{-5}	1.2×10^{-4}	2.6×10^{-4}	6.4×10^{-6}
DCS_k	9.4×10^{-5}	2.1×10^{-5}	2.0×10^{-5}	4.5×10^{-7}
DCS_*seed*	7.3×10^{-4}	1.7×10^{-5}	6.8×10^{-4}	7.2×10^{-5}
DS	0.21	0.32	0.37	0.26
MSCS	0.12	0.14	0.23	0.09

the DCS_*seed* contains 44 newly added genes in addition to the 16 seed genes from the renin pathways; in Figure 4.11, the DCS_*seed* contains 42 newly added genes. Between these two sets of newly added genes, there are 14 overlapping genes, such as NEDD4L, NEDD4, TP53, SIRT2, etc.

We also apply the GenGen, GRASS, Plink, and HYST methods to evaluate the statistical significance of the discovered DCSs. Note that we use the WTCCC dataset as the test dataset, which is independent of the ARIC dataset. Table 4.5 shows the p-value of the identified DCSs. DCS (1), (2) and (3) represent the top-3 DCSs. As can be seen from the table, the DCSs are highly significant. In the table, we also show the results of the DS and MSCS methods. As we can see, the subgraphs identified by these two methods are not as significant as the DCSs.



(a) Subgraph in protein interaction network (b) Subgraph in genetic interaction network

Figure 4.12: The DCS_k ($k = 40$) with node and edge weighted density identified from the WTCCC dataset

4.10.1.3 The DCSs with Node and Edge Weighted Density

In this section, we study the effectiveness of the DCS method when the conceptual network has both node and edge weights as discussed in Section 4.8.

We perform single-marker chi-square test on the genetic markers in the WTCCC dataset. The single-marker test statistics are used as the node weights. Then, we apply the algorithms developed for the DCS problem with node and edge weighted density. The DCS identified in the dual networks has 176 nodes. The figures are omitted because of the large size. The set of nodes are sparsely connected in the protein interaction network, while the subgraph in the genetic interaction network has high node and edge weighted density. Specifically, the DCS has 217 edges in the protein interaction network and $3,928$ edges in the genetic interaction network.

Figure 4.12 shows the DCS_k with $k = 40$ identified from the WTCCC dataset. To better visualize the weights, in Figure 4.12(b), the node and edge weights are indicated by the colors in the color bar. The red color represents the maximum weight and the green color represents the minimum weight. The CUBN, STK39, MYO6 genes are also observed in this subgraph, which have been discussed before since they are also observed in Figure 4.8. These genes are highlighted by stars in the figure. In Figure 4.12(b), we can see that some genes, such as the PRKAR2A gene, have green node color, which means that they have weak single-marker association. However, they have strong interaction with other genes. If only the node weights are

Table 4.6: *P*-values of the DCSs identified from the WTCCC dataset (with node weights, tested on the ARIC dataset)

Methods	GenGen	GRASS	Plink	HYST
DCS	5.8×10^{-6}	4.6×10^{-6}	6.7×10^{-6}	1.4×10^{-8}
DCS_k	8.2×10^{-5}	8.7×10^{-6}	9.4×10^{-6}	1.5×10^{-7}
DCS_*seed*	4.1×10^{-4}	7.7×10^{-6}	7.4×10^{-5}	9.1×10^{-6}
DS	0.15	0.26	0.23	0.25
MSCS	0.14	0.07	0.25	0.14

Table 4.7: *P*-values of the DCSs identified from the ARIC dataset (with node weights, tested on the WTCCC dataset)

Methods	GenGen	GRASS	Plink	HYST
DCS	4.2×10^{-6}	2.9×10^{-6}	3.6×10^{-6}	2.3×10^{-8}
DCS_k	6.9×10^{-5}	7.4×10^{-6}	6.6×10^{-6}	1.8×10^{-7}
DCS_*seed*	5.1×10^{-4}	8.5×10^{-6}	6.5×10^{-5}	8.3×10^{-6}
DS	0.21	0.32	0.18	0.35
MSCS	0.06	0.09	0.14	0.23

used, such as in the MSCS method, we will miss these important interactions.

We also use the set of 16 genes in the renin pathways as the input seed nodes, and discover the DCS_*seed* with the node and edge weighted density. The results are similar to that in Figure 4.9 and omitted here.

We further evaluate the statistical significance of the discovered DCSs. Note that to ensure the independence between the training and test datasets, when the DCSs are discovered from the WTCCC dataset, we use the ARIC dataset as the test dataset; when the DCSs are discovered from the ARIC dataset, we use the WTCCC dataset as the test dataset.

Tables 4.6 and 4.7 show the p-values of the DCSs identified from the WTCCC and ARIC datasets respectively. As can be seen from the table, the DCSs are highly significant. The subgraphs identified by the DS and MSCS methods are not as significant as the DCSs.

To compare the methods with and without node weights, we evaluate the significance of the DCSs, which are discovered from the WTCCC dataset, using the ARIC dataset. The results are shown in Table 4.8. Previously, in Table 4.4, the DCSs dis-

Table 4.8: P-values of the DCSs identified from the WTCCC dataset (without node weights, tested on the ARIC dataset)

Methods	GenGen	GRASS	Plink	HYST
DCS (1)	7.2×10^{-6}	8.2×10^{-6}	7.6×10^{-6}	4.8×10^{-8}
DCS (2)	3.8×10^{-5}	3.2×10^{-5}	6.1×10^{-5}	3.5×10^{-7}
DCS (3)	6.5×10^{-5}	8.1×10^{-5}	9.8×10^{-5}	7.9×10^{-7}
DCS_k	8.9×10^{-5}	1.6×10^{-5}	2.2×10^{-5}	4.5×10^{-7}
DCS_$seed$	6.3×10^{-4}	2.1×10^{-5}	9.3×10^{-5}	1.8×10^{-5}
DS	0.27	0.36	0.38	0.21
MSCS	0.13	0.15	0.19	0.16

covered from the WTCCC dataset are evaluated using the other half of the WTCCC dataset. However, in Table 4.6, the results with node weights are evaluated using the ARIC dataset. To make a fair comparison, we compare the results evaluated using the same ARIC dataset. Comparing Tables 4.6 and 4.8, we can see that the p-values are smaller when node weights are integrated in the method. Comparing Tables 4.7 and 4.5, we also observe that the p-values are smaller when node weights are integrated. These results demonstrate that the integration of node weights gives better performance.

4.10.1.4 Gene Set Enrichment Analysis

To further understand the biological meaning of the discovered DCSs, we apply the standard gene set enrichment analysis [9] to evaluate their significance. In particular, for each DCS (gene set) S, we identify the most significantly enriched KEGG (Kyoto Encyclopedia of Genes and Genomes) pathways (downloaded from `http://www.genome.jp/kegg/`). The significance (p-value) is determined by the Fisher's exact test. The raw p-values are further calibrated to correct for the multiple testing problem [118]. To compute calibrated p-values for each S, we perform a randomization test, wherein we apply the same test to 10^7 randomly created gene sets that have the same number of genes as S.

Tables 4.9 and 4.10 show the most significantly enriched pathways and the corresponding p-values for DCSs identified from the WTCCC and ARIC datasets re-

Table 4.9: Gene set enrichment analysis (WTCCC dataset)

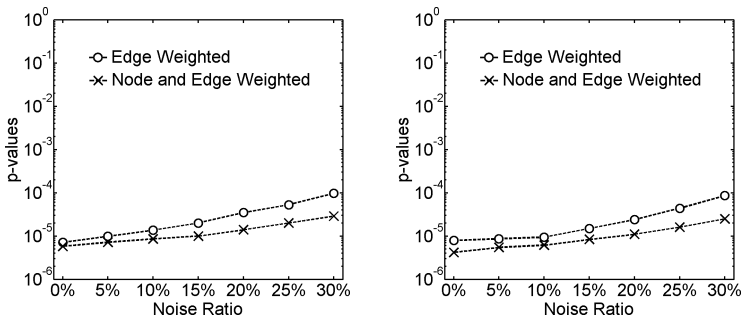
DCSs		KEGG pathways	p-values	Ref.
Only edge weights	DCS (1)	Neurotrophin signaling pathway	3.2×10^{-6}	[102]
	DCS (2)	ErbB signaling pathway	2.6×10^{-5}	[84]
	DCS (3)	Glioma	7.5×10^{-5}	–
	DCS_k	Neurotrophin signaling pathway	9.3×10^{-6}	[102]
	DCS_*seed*	Renin-angiotensin system	2.8×10^{-5}	[65]
Node and edge weights	DCS	Neurotrophin signaling pathway	1.4×10^{-6}	[102]
	DCS_k	ErbB signaling pathway	5.7×10^{-6}	[84]
	DCS_*seed*	Renin-angiotensin system	8.3×10^{-6}	[65]

Table 4.10: Gene set enrichment analysis (ARIC dataset)

DCSs		KEGG pathways	p-values	Ref.
Only edge weights	DCS (1)	Calcium signaling pathway	4.7×10^{-6}	[82]
	DCS (2)	Neurotrophin signaling pathway	8.4×10^{-6}	[102]
	DCS (3)	ErbB signaling pathway	6.3×10^{-5}	[84]
	DCS_k	MAPK signaling pathway	7.2×10^{-6}	[6]
	DCS_*seed*	Renin-angiotensin system	1.8×10^{-5}	[65]
Node and edge weights	DCS	Calcium signaling pathway	1.6×10^{-6}	[82]
	DCS_k	Insulin signaling pathway	3.9×10^{-6}	[17]
	DCS_*seed*	Renin-angiotensin system	8.5×10^{-6}	[65]

spectively. We can see that all the pathways have low p-values and are significantly enriched. We further study the existing literature and find that most of these pathways have been previously reported to be associated with hypertension. For example, the renin-angiotensin system is known to be associated with hypertension [65]. The MAPK signaling pathway interacts with the angiotensin system [6]. The Neuregulin-1/ErbB signaling in rostral ventrolateral medulla is involved in blood pressure regulation as an antihypertensive system [84]. The brain-derived neurotrophic factor may be a compensatory mechanism for the high blood pressure in Africans [102]. The calcium signaling pathway is reported to interact with the renin-angiotensin system [82], and it may also contribute to the hypertension pathogenesis. The insulin signaling pathway is also reported to interact with the renin-angiotensin system [17].

Moreover, we can observe that the DCSs identified from the node and edge weighted dual networks have more significant p-values than those identified from the dual



(a) WTCCC dataset (b) ARIC dataset

Figure 4.13: Robustness analysis

networks with only edge weights do. This indicates that integrating node weights can further increase the significance of the detected patterns.

4.10.1.5 Robustness Evaluation

The protein interaction network is usually noisy. In this section, we perform simulation study to evaluate the robustness of our method. Specifically, given a noise ratio $\tau\%$, we randomly remove $\tau\%$ edges from the protein interaction network, and then randomly add the same number of edges. Thus we get a noisy protein interaction network. We then find the DCS from these dual networks, and evaluate the significance of the discovered DCS by the GenGen method. Note that to ensure the independence between the training and test datasets, when the DCSs are discovered from the WTCCC dataset, we use the ARIC dataset as the test dataset; when the DCSs are discovered from the ARIC dataset, we use the WTCCC dataset as the test dataset.

Figure 4.13(a) shows the p-values of the DCSs identified from the WTCCC dataset. We can see that the p-values of the discovered DCSs slightly increase when we increase the noise ratio. When the noise ratio is 30%, our method can still find significant patterns. Figure 4.13(b) shows the p-values of the DCSs identified from the ARIC dataset. A similar trend can be observed. These results demonstrate that our method is robust to the noises in the protein interaction network. We can also observe that the method with node and edge weights is more robust than the method with only

edge weights. This indicates that integrating node weights can further increase the robustness of our method.

4.10.2 Effectiveness Evaluation in Other Application Domains

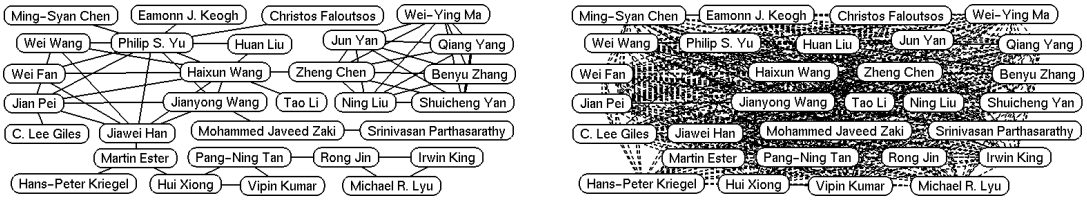
In addition to the biological application, we further evaluate the effectiveness of the DCS method in two other application domains. One application is about the bibliographic information analysis, and the other is about the social recommender system.

We use the DBLP dataset [108] to build two dual networks, one for data mining research community and one for database research community. To construct the dual networks for the data mining community, we extract a set of papers published in 5 data mining conferences: KDD, ICDM, SDM, PKDD and CIKM. The dataset contains 4,284 papers and 7,169 authors. The physical network is the co-author network with authors being the nodes and edges representing two authors have co-authored a paper. The conceptual research interest similarity network among authors is constructed based on the similarity of the terms in the paper titles of different authors. The shrunk Pearson correlation coefficient is used to compute the research interest similarity between authors [66]. The dual networks for the database community are constructed in a similar way based on papers published in SIGMOD, VLDB and ICDE.

We construct two dual networks using recommender system datasets, Flixster [49] and Epinions [83]. In the original Flixster dataset, the physical network has 786,936 nodes (users) and 7,058,819 edges representing their social connectivity. The user-item rating matrix consists of 8,184,462 user ratings for 48,791 items with rating scale from 1 to 5 with 0.5 increment. We construct the conceptual interest similarity network by measuring the correlation coefficients of the common ratings between users [79]. Note that we only calculate the correlation coefficient between two users with more than 5 common ratings. The constructed interest similarity network has 2,713,671 edges. The trust network in Epinions dataset has 49,288 nodes and 487,002 edges. The user-item rating matrix consists of 664,811 user ratings for 139,737 items

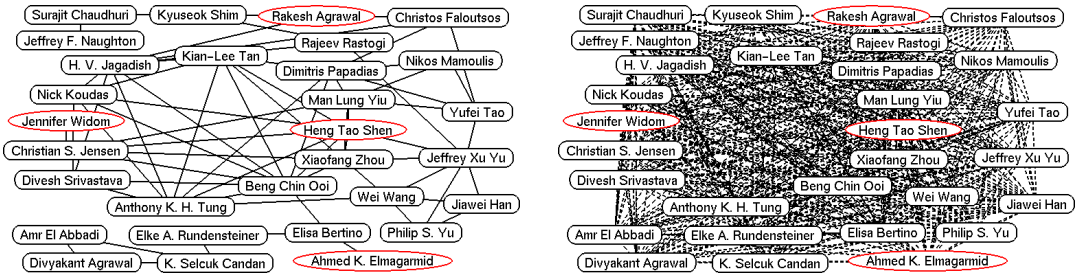
Table 4.11: Statistics of the dual networks in other application domains

Dual networks	Abbr.	#nodes	#edges in G_a	#edges in G_b
Research-DM	DM	7,169	14,526	30,000
Research-DB	DB	6,131	17,940	30,000
Recom-Epinions	EP	49,288	487,002	313,432
Recom-Flixster	FX	786,936	7,058,819	2,713,671



(a) Subgraph in co-author network (b) Subgraph in research interest similarity network

Figure 4.14: The DCS_k ($k = 30$) identified from the dual co-author (data mining) networks



(a) Subgraph in co-author network (b) Subgraph in research interest similarity network

Figure 4.15: The DCS_*seed* identified from the dual co-author (database) networks

with rating scale from 1 to 5 with 1 increment. The interest similarity network is constructed in a similar way as the one in the Flixster dataset. It has 313,432 edges. Table 4.11 shows the basic statistics of the dual networks in these applications.

4.10.2.1 Research Interest Similarity and Co-Author Dual Networks

The DCS identified in the dual co-author networks consists of hundreds of nodes. Here we only show the identified DCS_k for data mining in Figure 4.14 and DCS_*seed* for database in Figure 4.15.

Figures 4.14(a) and 4.14(b) shows the DCS_k ($k = 30$) identified in the dual networks of the data mining research community. The subgraph in the co-author network is sparsely connected and highly dense in the research interest similarity network. This indicates that the set of researchers have very close research interest.

(a) Induced subgraph in co-author network (b) Dense subgraph in research interest similarity network

Figure 4.16: The dense subgraph in the research interest similarity network of the dual co-author (data mining) networks

The subgraph in the co-author network shows their collaboration pattern.

Figures 4.15(a) and 4.15(b) shows the DCS_*seed* identified in the dual networks of the database research community. The names of the 4 input seed authors are in red ellipses. The researchers with similar interest and their collaboration patterns are clearly shown in the two subgraphs. The 4 seed authors do not have direct co-authorship with each other. Through the resulting DCS_*seed*, we uncover the connected community of common interests.

Note that a dense subgraph in the research interest similarity network may not be connected in the co-author network. One example is shown in Figure 4.16. Figure 4.16(b) shows a dense subgraph identified from the research interest similarity network for data mining. Figure 4.16(a) shows the induced subgraph in the co-author network. We can see that very few authors are connected. Thus finding dense subgraphs in a single network may miss important information presented in the other network.

4.10.2.2 User Interest Similarity and Social Connectivity Dual Networks

The DCS_k ($k = 40$) identified in the dual network constructed from the Epinions dataset is shown in Figure 4.17. The subgraph in the interest similarity network is a dense component and not shown here. In this figure, each node is a user, whose name is not shown because of the privacy issue. Because this group of users have high interest similarity and also have social connection, if one of the users receives an advertisement of an interested product, this information is likely to be propagated to the rest of the group.

To demonstrate the effectiveness of the DCS pattern, we compare it with the dense

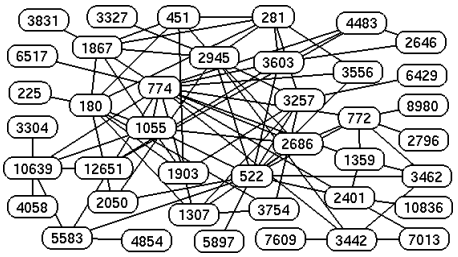
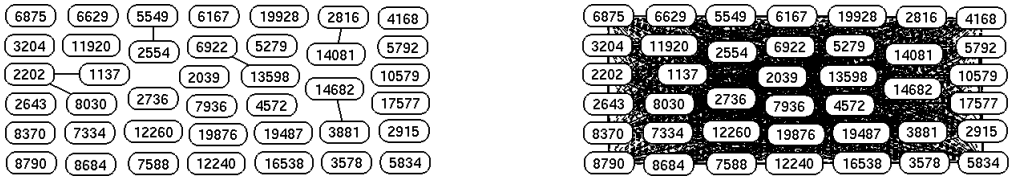
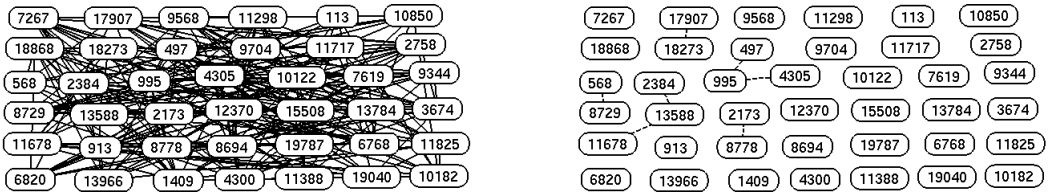


Figure 4.17: The social connectivity network of the DCS_k (k = 40) identified in the Epinions dataset



(a) Induced subgraph in social connectivity network (b) Dense subgraph in interest similarity network

Figure 4.18: The dense subgraph in the interest similarity network of the Epinions dataset



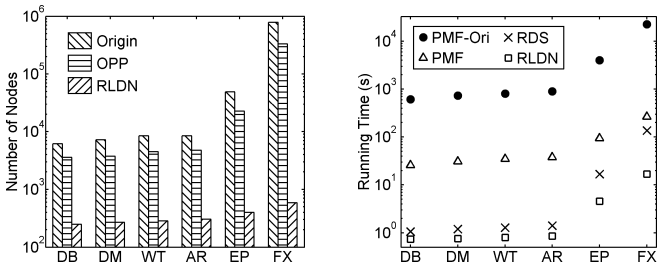
(a) Dense subgraph in social connectivity network (b) Induced subgraph in interest similarity network

Figure 4.19: The dense subgraph in the social connectivity network of the Epinions dataset

subgraphs discovered from a single network. Figure 4.18(b) shows a dense subgraph in the interest similarity network. Figure 4.18(a) shows its induced subgraph in the social connectivity network. We can see that this set of users have no social connectivity even though they have high interest similarity. Figure 4.19(a) shows a dense subgraph in the social connectivity network. Figure 4.19(b) shows its induced subgraph in the interest similarity network. We can see that the subgraph in the interest similarity network is very sparse. This indicates that a group of users having high social connectivity may not have similar interest. Similar observations can be made in the dual networks constructed from the Flixster dataset.

4.10.3 Efficiency Evaluation on Real Networks

In this subsection, we evaluate the efficiency of the proposed DCS_RDS, DCS_GND and DCS_MAS algorithms using real networks.



(a) Pruning effect (b) Running time

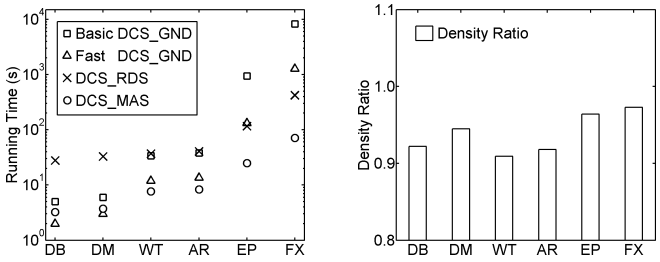
Figure 4.20: Pruning effect and running time of the DCS_RDS algorithm

The DCS_RDS algorithm has three major components: removing low degree nodes (RLDN) in the conceptual network, finding the densest subgraph in the remaining graph by parametric maximum flow (PMF), and refining the densest subgraph (RDS) to make it connected in the physical network.

We first evaluate the pruning effect of the RLDN step. Figure 4.20(a) shows the number of nodes in the original graph and the number of nodes remained after pruning. It can be seen that the RLDN step can reduce the number of nodes by 2 to 4 orders of magnitude. Moreover, the pruning effect becomes larger for larger graphs. This indicates that the RLDN step is more effective when graph size increases. The effect of the optimality preserved pruning (OPP) approach discussed in Section 4.4 is also shown in this figure. We can see that the OPP step can prune 40% to 60% nodes in the 6 real graphs. Since the real graphs are scale-free, there are many leaf nodes in the physical networks and the OPP step has large pruning ratio.

Figure 4.20(b) shows the running time of each step in DCS_RDS. We also run the parametric maximum flow method on the original graph (PMF-Ori) to see the performance improvement of our method. From the results, we can see that the RLDN and RDS steps run efficiently. The most time consuming part is to use parametric maximum flow to find the densest subgraph. Because of the pruning effect of RLDN, finding the densest subgraph after the RLDN step is about 2 orders of magnitude faster than directly finding it in the original graph.

Figure 4.21(a) shows the running time of the basic and fast DCS_GND methods. We can see that the fast DCS_GND method runs about 1 order of magnitude faster



(a) Running time (b) Density ratio

Figure 4.21: (a) Running time of DCS_RDS, basic and fast DCS_GND, and DCS_MAS; (b) Density ratio of the subgraphs identified by DCS_RDS and basic DCS_GND

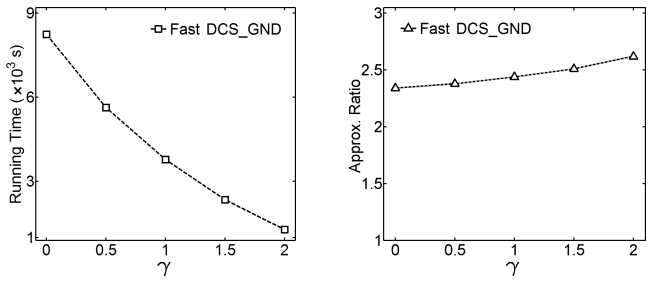
than the basic method, even though they have the same theoretical complexity. This demonstrates the effectiveness of simultaneously deleting independent non-articulation nodes. The running time of DCS_RDS is also shown in the figure for comparison. We can observe that DCS_GND runs faster on smaller graphs and DCS_RDS runs faster on larger graphs. The reason is that when the graph becomes larger, the depth first search procedure in DCS_GND will take longer time. On the other hand, more nodes will be removed by DCS_RDS for larger graphs as demonstrated in Figure 4.20(a). The running time of DCS_MAS is also shown in the figure. In the DCS_MAS method, we randomly select 8 seed nodes initially and set the parameter $K = 2000$. We can see that DCS_MAS takes about one hundred seconds on large graphs. Since DCS_MAS searches the whole graph to compute the Steiner tree, DCS_MAS has increasing running time when the graph size increases.

Figure 4.21(b) shows the ratio of the density of the subgraphs identified by DCS_RDS and DCS_GND. It can be seen that the densities of the subgraphs identified by these two methods are very similar. The DCS_GND method always results slightly larger density value. The reason is that the densest subgraph in the conceptual network may not have large overlap with the DCS. The exact DCS solution may contain other dense components instead of the densest subgraph because of the connectivity constraint in the physical network.

Table 4.12 shows the estimated approximation ratio of the proposed methods on different datasets. In the fast DCS_GND method, we set $\gamma = 2.0$. From the table, we

Table 4.12: Approximation ratios on real networks

Datasets →	DB	DM	WT	AR	EP	FX
DCS_RDS	1.48	1.42	1.94	1.83	1.23	2.25
Basic DCS_GND	1.53	1.44	2.11	1.97	1.26	2.34
Fast DCS_GND	2.21	2.10	2.35	2.14	1.87	2.62
DCS_MAS	3.45	4.21	5.16	5.28	6.39	6.72

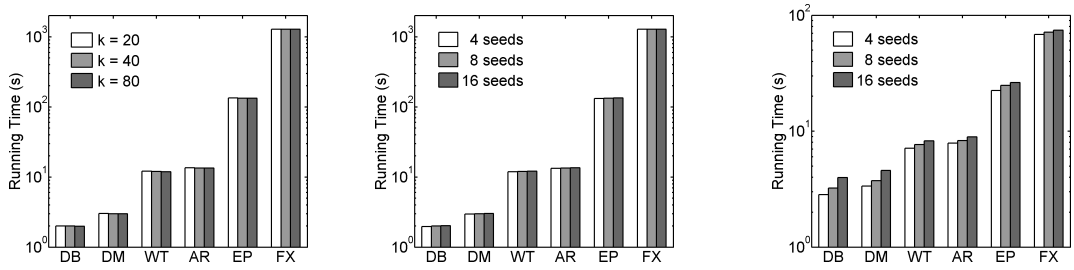


(a) Running time (b) Approx. ratio

Figure 4.22: Effects of γ in fast DCS_GND

can see that the approximation ratio of DCS_RDS is tighter than that of DCS_GND. The reason is that DCS_RDS uses the exact densest subgraph in its first step. We can also observe that the approximation ratio of the basic DCS_GND method is always smaller than that of the fast DCS_GND method. This is because the fast DCS_GND method is greedier, which deletes a set of low degree nodes in each iteration. The basic DCS_GND method only deletes the node with the minimum degree. From the table, we can also see that the approximation ratio of both methods is around 2, which is the theoretical approximation ratio of the greedy node deletion algorithm for finding the densest subgraph in a single graph. Since we compute the density of the densest subgraph in the DCS_RDS method, we can compute the approximation ratio of the DCS_MAS method. The approximation ratios of DCS_MAS are also shown in Table 4.12. Compared with the approximation ratios of the other methods, the approximation ratios of the DCS_MAS method are larger. The reason is that the local dense subgraph, which DCS_MAS aims to search for, may have smaller density than the global densest subgraph.

Figure 4.22(a) shows the running time of the fast DCS_GND method on the Flixster dataset when varying γ. When $\gamma = 0$, it degrades to the basic DCS_GND



(a) Varying the size k (b) Varying the number of seeds (c) Varying the number of seeds

Figure 4.23: Running time of fast DCS_GND (a,b) and DCS_MAS (c)

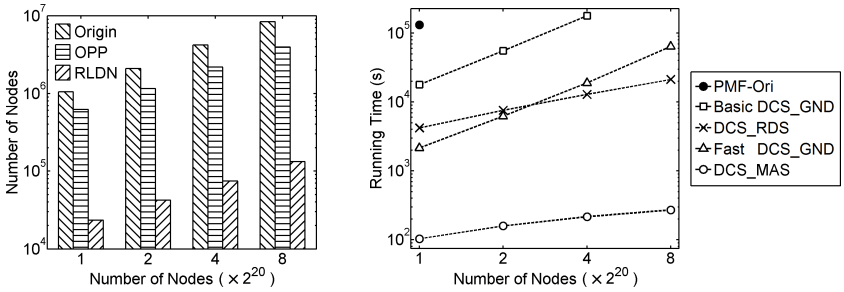
method. When increasing γ, the fast DCS_GND method will delete more nodes in each iteration. Thus the running time decreases. Figure 4.22(b) shows the approximation ratio when varying γ. We can see that the approximation ratio slightly increases when increasing γ. This indicates that the approximation ratio of the fast DCS_GND method is not very sensitive to γ.

Figure 4.23(a) and Figure 4.23(b) show the running time of the fast DCS_GND method when varying the output size k in DCS_k problem and varying the number of seeds in DCS_*seed* problem respectively. From the results, we can see that fast DCS_GND has almost constant running time when varying k and the number of seeds. This is because the DCS_GND method keeps deleting nodes from the dual networks and is not sensitive to k and the number of seeds. Figure 4.23(c) shows the running time of the DCS_MAS method when varying the number of seeds in the DCS_*seed* problem. We can see that the running time slightly increases when increasing the number of seeds. It may costs more time to find the Steiner tree when there are more seed nodes.

We further evaluate the DCS_RDS, DCS_GND and DCS_MAS methods for the DCS problem following the node and edge weighted density in Definition 17. The running time results are similar to that in Figure 4.21(a). The approximation ratio results are similar to that in Table 4.12. These results are omitted.

Table 4.13: Statistics of synthetic dual networks

#nodes	1×2^{20}	2×2^{20}	4×2^{20}	8×2^{20}
#edges in G_a, G_b	1×10^7	2×10^7	4×10^7	8×10^7



(a) Pruning ratio (b) Running time of different methods

Figure 4.24: Results on synthetic dual networks

4.10.4 Efficiency Evaluation on Large Synthetic Networks

To further evaluate the scalability of the proposed methods, we generate a series of synthetic dual networks. Both the physical and conceptual networks are scale-free graphs based on the R-MAT model [18]. We use the graph generator from `https://github.com/dhruvbird/GTgraph`. The statistics of the generated graphs are shown in Table 4.13.

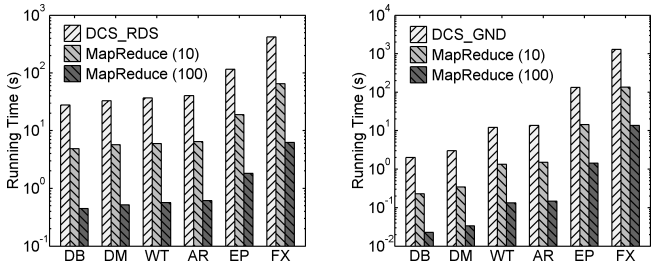
Figure 4.24(a) shows the pruning ratio of the OPP and RLDN steps in the DCS_RDS method. The OPP step can prune about 50% nodes, while the RLDN step can further reduce the number of nodes by 1∼2 orders of magnitude.

Figure 4.24(b) shows the running time of the DCS_RDS, DCS_GND and DCS_MAS methods. DCS_RDS has slower increasing rate. The reason is that the RLDN step has larger pruning ratio on larger graphs. The fast DCS_GND method runs about 1 order of magnitude faster than the basic DCS_GND method. This figure also shows the running time of the PMF method on the original conceptual network. PMF cannot be applied to large networks because of its long running time. DCS_MAS has increasing running time when the graph size increases. This is because it will take more time to find the Steiner tree when the graph size increases.

Table 4.14 shows the approximation ratios of the DCS_RDS, DCS_GND and D-

Table 4.14: Approximation ratios on synthetic networks

#nodes	1×2^{20}	2×2^{20}	4×2^{20}	8×2^{20}
DCS_RDS	1.53	1.48	1.44	1.41
Basic DCS_GND	1.58	1.54	1.51	–
Fast DCS_GND	1.72	1.69	1.67	1.63
DCS_MAS	5.86	5.63	5.42	5.16



(a) DCS_RDS (b) DCS_GND

Figure 4.25: Running time on real networks

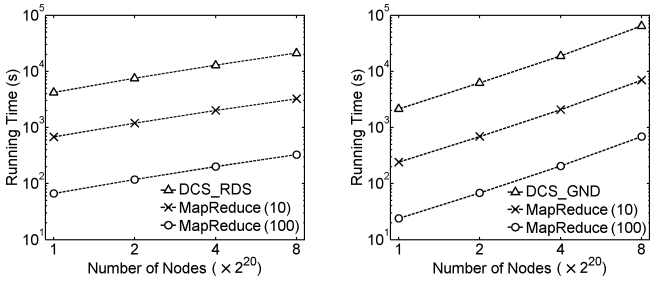
CS_MAS methods. The approximation ratio becomes tighter when the graph size increases.

To evaluate the scalability of the methods for the DCS problem following the node and edge weighted density in Definition 17, we randomly generate node weights on the synthetic conceptual networks. The running time results are similar to that in Figure 4.24(b). The approximation ratio results are similar to that in Table 4.14. These results are omitted.

4.10.5 Efficiency Evaluation of MapReduce Implementations

In this subsection, we evaluate the efficiency of the MapReduce implementation of the DCS_RDS and DCS_GND algorithms. We rent 101 nodes from the Amazon's Elastic Compute Cloud. The parameter γ in the DCS_GND method is set to 2.0.

Figure 4.25(a) shows the running time of the DCS_RDS method on real networks. The legends "MapReduce (10)" and "MapReduce (100)" denote that 10 and 100 worker nodes are used in the MapReduce implementation respectively. The MapReduce implementation with 10 worker nodes is about 6.1 times faster than the sequential algorithm. The MapReduce implementation with 100 worker nodes is about 64.5



(a) DCS_RDS (b) DCS_GND

Figure 4.26: Running time on synthetic networks

times faster than the sequential algorithm. Figure 4.25(b) shows the running time of the DCS_GND method on real networks. The MapReduce implementation with 10 worker nodes is about 9.2 times faster than the sequential algorithm. The MapReduce implementation with 100 worker nodes is about 91.5 times faster than the sequential algorithm. Comparing Figures 4.25(a) and 4.25(b), we can see that the MapReduce implementation of the DCS_GND method gets larger speed improvement than that of the DCS_RDS method does. The reason is that the DCS_GND method adopts the simple greedy node deletion strategy, which is quite suitable for the distributed computing environment.

Figures 4.26(a) and 4.26(b) show the running time of the DCS_RDS and DCS_GND methods on large synthetic networks respectively, whose statistics are shown in Table 4.13. Similar patterns can be observed.

4.11 Conclusion

Dual networks exist in many real-life applications, where the physical and conceptual networks encode complementary information. In this chapter, we study the problem of finding the densest connected subgraph in dual networks. A dense subgraph in the conceptual network that is also connected in the physical network can unravel interesting patterns that are invisible to the existing methods. We formulate the DCS problem and prove it is NP-hard. To find the DCS, we first introduce an effective optimality pruning strategy to remove the nodes that are not in the optimal solution.

Then, we develop two efficient greedy algorithms to find the DCS. We also develop an efficient local search heuristic for the DCS problem with input seed nodes. We further study the DCS problem when there are node weights in the conceptual network, and extend the algorithms to solve this new problem. Extensive experimental results on real and synthetic datasets demonstrate the interestingness of the identified patterns and the efficiency of the proposed algorithms.

Chapter 5

Conclusion and Future Work

Graph is a ubiquitous data structure, and the local patterns in graphs have numerous applications. In this dissertation, we focus on local patterns and related local algorithms. We study three local problems: the top-k proximity query problem, local community detection problem, and densest connected subgraph problem. For the top-k proximity query problem, we develop a simple, unified, and exact local search algorithm for the random walk based proximity measures. For the local community detection problem, we discover that the existing methods suffer from the free rider effect, and propose the query biased node weighting scheme to eliminate the free rider effect. We observe the dual networks in real applications, and propose the densest connected subgraph problem, which is a natural extension of the densest subgraph problem from a single network to dual networks. In conclusion, this dissertation provides a more efficient local search algorithm for the top-k proximity query problem, a more effective algorithm for the local community detection problem, and a more interesting densest connected subgraph pattern in dual networks.

In future, we plan to extend the works for the random walk based proximity measure and local community detection described in Chapters 1 and 2 respectively. In the work for the random walk based proximity measure, we observe that the existing random walk based proximity measures are all based on the first-order Markov chain,

i.e., the next step only depends on the current state. We plan to study whether the proximity measure defined on the high-order Markov chain can give more accurate ranking results. Because the high-order Markov chain may capture more information about the graph structure, such as local communities, intuitively it can provide more accurate ranking results. In the work for the local community detection, we would like to study the node weighting scheme for the local spectral algorithm. Spectral algorithm is a classic graph clustering algorithm, and it is important to extend the spectral clustering algorithm to the local community detection problem.

Bibliography

[1] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. K-core decomposition: a tool for the visualization of large scale networks. *arXiv preprint cs/0504107*, 2005.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *FOCS*, pages 475–486, 2006.

[3] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[4] G. Ausiello, D. Firmani, L. Laura, and E. Paracone. *Large-scale graph biconnectivity in MapReduce*. Technical Report, 2012.

[5] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and MapReduce. *PVLDB*, 5(5):454–465, 2012.

[6] W. Bao et al. Effects of p38 mapk inhibitor on angiotensin ii-dependent hypertension, organ damage, and superoxide anion production. *Journal of cardiovascular pharmacology*, 49(6):362–368, 2007.

[7] S. E. Baranzini, N. W. Galwey, J. Wang, et al. Pathway and network-based analysis of genome-wide association studies in multiple sclerosis. *Human Molecular Genetics*, 18(11):2078–2090, 2009.

[8] V. Batagelj and M. Zaversnik. An $O(m)$ algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[9] S. Bauer, S. Grossmann, M. Vingron, and P. N. Robinson. Ontologizer 2.0 – a multifunctional tool for go term enrichment analysis and data exploration. *Bioinformatics*, 24(14):1650–1651, 2008.

[10] A. A. Benczur, K. Csalogany, T. Sarlos, and M. Uher. SpamRank - Fully automatic link spam detection work in progress. In *AIRWeb*, 2005.

[11] P. Berkhin. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics*, 3(1):41–62, 2006.

[12] J. W. Berry, B. Hendrickson, R. A. LaViolette, et al. Tolerating the community detection resolution limit with edge weighting. *Physical Review E*, 83(5):056119, 2011.

[13] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

[14] P. Bogdanov and A. Singh. Accurate and scalable nearest neighbors in large networks based on effective importance. In *CIKM*, pages 523–528, 2013.

[15] G. F. Bomfim, R. A. Dos Santos, M. A. Oliveira, et al. Toll-like receptor 4 contributes to blood pressure regulation and vascular contraction in spontaneously hypertensive rats. *Clinical Science*, 122(11):535–543, 2012.

[16] P. R. Burton, D. G. Clayton, L. R. Cardon, et al. Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. *Nature*, 447(7145):661–678, 2007.

[17] M. A. d. Carvalho-Filho et al. Insulin and angiotensin ii signaling pathways cross-talk: implications with the association between diabetes mellitus, arterial hypertension and cardiovascular disease. *Arquivos Brasileiros de Endocrinologia & Metabologia*, 51(2):195–203, 2007.

[18] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SDM*, pages 442–446, 2004.

[19] S. Chakrabarti, A. Pathak, and M. Gupta. Index design and query processing for graph conductance search. *VLDB J.*, 20(3):445–470, 2011.

[20] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 139–152, 2000.

[21] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *TKDE*, 24(7):1216–1230, 2012.

[22] L. S. Chen, C. M. Hutter, J. D. Potter, Y. Liu, R. L. Prentice, U. Peters, and L. Hsu. Insights into colon cancer etiology via a regularized approach to gene set analysis of gwas data. *The American Journal of Human Genetics*, 86(6):860–871, 2010.

[23] M. Ciglan, M. Laclavík, and K. Nørvåg. On community detection in real-world networks and the importance of degree assortativity. In *KDD*, pages 1007–1015, 2013.

[24] A. Clauset. Finding local community structure in networks. *Physical Review E*, 72(2):026132, 2005.

[25] S. Cohen, B. Kimelfeld, and G. Koutrika. A survey on proximity measures for social networks. In *Search Computing*, pages 191–206, 2012.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2001.

[27] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

[28] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[29] P. Erdős and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 5:17–61, 1960.

[30] P. Esfandiar, F. Bonchi, D. F. Gleich, et al. Fast Katz and commuters: Efficient estimation of social relatedness in large networks. In *Algorithms and Models for the Web-Graph*, pages 132–145. Springer, 2010.

[31] Y. Fang, K.-C. Chang, and H. W. Lauw. RoundTripRank: Graph-based proximity with importance and specificity. In *ICDE*, pages 613–624, 2013.

[32] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identi- fication of web communities. In *KDD*, pages 150–160, 2000.

[33] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[34] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for random walk with restart. *PVLDB*, 5(5):442–453, 2012.

[35] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Efficient ad-hoc search for personalized PageRank. In *SIGMOD*, pages 445–456, 2013.

[36] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient personalized PageRank with accuracy assurance. In *KDD*, pages 15–23, 2012.

[37] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.

[38] A.-C. Gavin et al. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440(7084):631–636, 2006.

[39] A. V. Goldberg. *Finding a maximum density subgraph.* Technical Report, 1984.

[40] Z. Guan, J. Wu, Q. Zhang, A. Singh, and X. Yan. Assessing and ranking structural correlations in graphs. In *SIGMOD*, pages 937–948, 2011.

[41] E. A. Guillemin. *Introductory circuit theory.* John Wiley & Sons, 1953.

[42] M. Gupta, A. Pathak, and S. Chakrabarti. Fast algorithms for top-k personalized PageRank queries. In *WWW*, pages 1225–1226, 2008.

[43] F. Halim, R. H. Yap, and Y. Wu. A MapReduce-based maximum-flow algorithm for large small-world network graphs. In *ICDCS*, pages 192–202, 2011.

[44] K. Hong, M. Go, H. Jin, et al. Genetic variations in atp2b1, csk, arsg and csmd1 loci are related to blood pressure and/or hypertension in two korean cohorts. *Journal of Human Hypertension*, 24(6):367–372, 2009.

[45] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):i213–i221, 2005.

[46] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[47] M. Humbert, O. V. Evgenov, and J.-P. Stasch. *Pharmacotherapy of pulmonary hypertension*, volume 218. Springer, 2013.

[48] T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(suppl 1):S233–S240, 2002.

[49] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *RecSys*, pages 135–142, 2010.

[50] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *SIGKDD*, pages 538–543, 2002.

[51] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[52] P. Jia and Z. Zhao. Network-assisted analysis to prioritize gwas results: principles, methods and perspectives. *Human Genetics*, 133(2):125–138, 2014.

[53] P. Jia, S. Zheng, J. Long, W. Zheng, and Z. Zhao. dmGWAS: dense module searching for genome-wide association studies in protein–protein interaction networks. *Bioinformatics*, 27(1):95–102, 2011.

[54] H.-S. Jin, K.-W. Hong, B.-Y. Kim, et al. Replicated association between genetic variation in the park2 gene and blood pressure. *Clinica Chimica Acta*, 412(17):1673–1677, 2011.

[55] R. Kannan, S. Vempala, and A. Vetta. On clusterings: good, bad and spectral. *JACM*, 51(3):497–515, 2004.

[56] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[57] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[58] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[59] R. Kelley and T. Ideker. Systematic interpretation of genetic interactions using protein networks. *Nature Biotechnology*, 23(5):561–566, 2005.

[60] A. Khadivi, A. A. Rad, and M. Hasler. Network community-detection enhancement by proper weighting. *Physical Review E*, 83(4):046104, 2011.

[61] S. Khemmarat and L. Gao. Fast top-k path-based relevance query on massive graphs. In *ICDE*, pages 316–327, 2014.

[62] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *arXiv preprint arXiv:1309.7233*, 2013.

[63] K. Kloster and D. F. Gleich. Heat kernel based community detection. In *KDD*, pages 1386–1395, 2014.

[64] I. Kloumann and J. Kleinberg. Community membership identification from small seed sets. In *KDD*, pages 1366–1375, 2014.

[65] H. Kobori et al. The intrarenal renin-angiotensin system: from physiology to the pathobiology of hypertension and kidney disease. *Pharmacological reviews*, 59(3):251–287, 2007.

[66] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008.

[67] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994.

[68] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[69] K. J. Lang and R. Andersen. Finding dense and isolated submarkets in a sponsored search spending graph. In *CIKM*, pages 613–622, 2007.

[70] P. Lee, L. V. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *ICDE*, pages 774–785, 2012.

[71] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[72] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

[73] D. Levy, G. B. Ehret, K. Rice, et al. Genome-wide association study of blood pressure and hypertension. *Nature Genetics*, 41(6):677–687, 2009.

[74] M.-X. Li, J. S. Kwan, and P. C. Sham. HYST: A hybrid set-based test for genome-wide association studies, with application to protein-protein interaction-based association analysis. *The American Journal of Human Genetics*, 91(3):478–488, 2012.

[75] W. Li, H. Hu, Y. Huang, H. Li, M. R. Mehan, J. Nunez-Iglesias, M. Xu, X. Yan, and X. J. Zhou. Pattern mining across many massive biological networks. In *Functional Coherence of Molecular Networks in Bioinformatics*, pages 137–170. Springer, 2012.

[76] J. Lin and C. Dyer. *Data-intensive text processing with MapReduce*. Morgan & Claypool Publishers, 2010.

[77] F. Luo, J. Z. Wang, and E. Promislow. Exploring local community structures in large networks. *Web Intelligence and Agent Systems*, 6(4):387–400, 2008.

[78] F. Luo, Y. Wang, X. Wang, K. Sun, X. Zhou, and R. Hui. A functional variant of NEDD4L is associated with hypertension, antihypertensive response, and orthostatic hypotension. *Hypertension*, 54(4):796–801, 2009.

[79] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *WSDM*, pages 287–296, 2011.

[80] L. Ma, H. Huang, Q. He, K. Chiew, J. Wu, and Y. Che. GMAC: a seed-insensitive approach to local community detection. In *DaWak*, pages 297–308, 2013.

[81] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi. A local spectral method for graphs: with applications to improving graph partitions and exploring data graphs locally. *JMLR*, 13(1):2339–2365, 2012.

[82] H. Makani et al. Effect of renin-angiotensin system blockade on calcium channel blocker-associated peripheral edema. *The American journal of medicine*, 124(2):128–135, 2011.

[83] P. Massa and P. Avesani. Trust-aware recommender systems. In *RecSys*, pages 17–24, 2007.

[84] R. Matsukawa et al. Neuregulin-1/erbb signaling in rostral ventrolateral medulla is involved in blood pressure regulation as an antihypertensive system. *Journal of hypertension*, 29(9):1735–1742, 2011.

[85] G. M. McMahon, C. M. O'Seaghdha, S.-J. Hwang, J. B. Meigs, and C. S. Fox. The association of a single-nucleotide polymorphism in CUBN and the risk of albuminuria and cardiovascular disease. *Nephrology Dialysis Transplantation*, page gft386, 2013.

[86] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *IPL*, 27(3):125–128, 1988.

[87] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM*, pages 469–478, 2008.

[88] C. Meyer. *Matrix analysis and applied linear algebra*. SIAM, 2000.

[89] M. E. Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, 2006.

[90] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

[91] P. Phillips. Epistasis - the essential role of gene interactions in the structure and evolution of genetic systems. *Nature Review Genetics*, 9(11):855–867, 2008.

[92] S. Prabhu and I. Pe'er. Ultrafast genome-wide scan for SNP-SNP interactions in common complex disease. *Genome Research*, 22(11):2230–2240, 2012.

[93] S. S. Pullamsetti, E. M. Berghausen, S. Dabral, et al. Role of src tyrosine kinases in experimental pulmonary hypertension. *Arteriosclerosis, Thrombosis, and Vascular Biology*, 32(6):1354–1365, 2012.

[94] S. Purcell et al. Plink: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007.

[95] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[96] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, pages 456–472, 2010.

[97] P. Sarkar and A. W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *UAI*, pages 335–343, 2007.

[98] P. Sarkar and A. W. Moore. Fast nearest-neighbor search in disk-resident graphs. In *KDD*, pages 513–522, 2010.

[99] P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *ICML*, pages 896–903, 2008.

[100] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003.

[101] T. P. Slavin, T. Feng, A. Schnell, X. Zhu, and R. C. Elston. Two-marker association tests yield new disease associations for coronary artery disease and hypertension. *Human Genetics*, 130(6):725–733, 2011.

[102] A. Smith et al. Attenuated brain-derived neurotrophic factor and hypertrophic remodelling: the sabpa study. *Journal of human hypertension*, 29(1):33–39, 2015.

[103] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[104] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90, 2004.

[105] D. A. Spielman and S.-H. Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.

[106] Y. V. Sun and S. L. Kardia. Identification of epistatic effects using a protein-protein interaction database. *Human Molecular Genetics*, 19(22):4345–4352, 2010.

[107] M. Tamura, T. Nakayama, I. Sato, et al. Haplotype-based case-control study of estrogen receptor α (esr1) gene and pregnancy-induced hypertension. *Hypertension Research*, 31(2):221–228, 2008.

[108] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. ArnetMiner: extraction and mining of academic social networks. In *KDD*, pages 990–998, 2008.

[109] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[110] B. O. Tayo, A. Luke, X. Zhu, A. Adeyemo, and R. S. Cooper. Association of regions on chromosomes 6 and 7 with blood pressure in nigerian families. *Circulation: Cardiovascular Genetics*, 2(1):38–45, 2009.

[111] the ARIC Investigators. The atherosclerosis risk in communities (aric) study: design and objectives. *American Journal of Epidemiology*, 129(4):687–702, 1989.

[112] H. Tong, C. Faloutsos, and J. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[113] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112, 2013.

[114] I. Ulitsky and R. Shamir. Pathway redundancy and protein essentiality revealed in the Saccharomyces cerevisiae interaction networks. *Molecular Systems Biology*, 3:104, 2007.

[115] K. Wang, M. Li, and M. Bucan. Pathway-based approaches for analysis of genomewide association studies. *The American Journal of Human Genetics*, 81(6):1278–1283, 2007.

[116] K. Wang, M. Li, and H. Hakonarson. Analysing biological pathways in genome-wide association studies. *Nature Review Genetics*, 11(12):843–854, 2010.

[117] Y. Wang, J. R. O'Connell, P. F. McArdle, et al. Whole-genome association study identifies STK39 as a hypertension susceptibility gene. *PNAS*, 106(1):226–231, 2009.

[118] P. H. Westfall and S. S. Young. *Resampling-based multiple testing*. Wiley, New York, 1993.

[119] X.-M. Wu, Z. Li, A. M. So, J. Wright, and S.-F. Chang. Learning with partially absorbing random walks. In *NIPS*, pages 3077–3085, 2012.

[120] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[121] Y. Wu, R. Jin, and X. Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *SIGMOD*, pages 1139–1150, 2014.

[122] Y. Wu, R. Jin, and X. Zhang. Efficient and exact local search for random walk based top-k proximity query in large graphs. *TKDE*, 2016.

[123] Y. Wu, R. Jin, X. Zhu, and X. Zhang. Finding dense and connected subgraphs in dual networks. In *ICDE*, pages 915–926, 2015.

[124] Y. Wu, X. Zhu, L. Li, W. Fan, R. Jin, and X. Zhang. Mining dual networks: Models, algorithms and applications. *TKDD*, 2016.

[125] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pages 745–754, 2012.

[126] A. W. Yu, N. Mamoulis, and H. Su. Reverse top-k search using random walk with restart. *PVLDB*, 7(5):401–412, 2014.

[127] P. Yue, E. Melamud, and J. Moult. SNPs3D: candidate gene and SNP selection for association studies. *BMC Bioinformatics*, 7(1):166, 2006.

[128] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. Evaluating geo-social influence in location-based social networks. In *CIKM*, pages 1442–1451, 2012.

[129] X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao. On the embeddability of random walk distances. *PVLDB*, 6(14):1690–1701, 2013.

[130] X. Zhu, T. Feng, Y. Li, Q. Lu, and R. C. Elston. Detecting rare variants for complex traits using family and unrelated data. *Genetic Epidemiology*, 34(2):171–187, 2010.