2012

A two-stage strategy for solving the connection subgraph problem

Heyong Wang
Iowa State University

A two-stage strategy for solving the connection subgraph problem

by

Heyong Wang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Carl Chang, Major Professor

Wensheng Zhang

Simanta Mitra

Iowa State University

Ames, Iowa

2012

DEDICATION

Dedicate to my wife Qiaolin, my parents, my brothers and to my sister without whose support I would not have been able to complete this work. I would also like to thank my friends for their loving guidance during the writing of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my gratitude to everyone who helped me in conducting the research leading to this thesis. Firstly, I would like to thank my advisor, Dr. Carl. K. Chang for his guidance, patience and support throughout the period of this research work. I would also like to thank my committee members, Dr. Wensheng Zhang and Dr. Simanta Mitra for their thoughtful insights and pointers. I want to give my sincere thanks to Dr. Hen-I Yang whose help, stimulating suggestions and encouragement helped in all the time of writing of this thesis. Further, I would like to acknowledge useful discussions with Kai-shin Lu, Liping Wu and other colleagues. Finally, I would like to warmly thank my wife for all those intangible bits of reassurance given at key moments.

ABSTRACT

A connection subgraph is a small subgraph of a large graph that best capture the relationship between two nodes. Formally, Connection Subgraph Problem is: Given: An edge-weighted undirected graph G, two query vertices s and t from G, the size of the subgraph b, and a goodness function. Find: A connected subgraph H containing s and t and at most b other vertices that maximizes the goodness function g(H).

Two challenging problems for Connection Subgraph Problem are i) Finding an good "goodness" function ii) and designing algorithms to quickly identify the connection subgraph.

The goal of this work is to provide a way to explore and discovery relationship between nodes on graphs, especially for social networks. We focus on connection subgraph problem on the graphs where the relationship between nodes is from all the contributions of their paths. In this research, we propose a measure called the Path Betweenness of a path to measure the contribution of a path between two nodes to these two nodes' relationship. Based on path betweenness, We introduce graph between of a graph as the "goodness" function for the Connection Subgraph Problem. Graph betweenness of a graph takes contributions of all paths in the graph into consideration. In order to quickly find the connection subgraph from a large graph, we propose a two-stage solution. The two-stage strategy first generates a small enough intermediate subgraph by eliminating the unimportant nodes in the graph. In the second stage, a candidate subgraph generation algorithm and the graph betweenness computational algorithm are designed to find the connection subgraph. We implemented a system to conduct experiments on real data. Our experimental results show the two-stage strategy can help us quickly find the connection subgraph in practice.

CHAPTER 1. Overview

This chapter provides an introduction to the main topics and the motivation behind the thesis. A brief overview of our contributions and the outline of the overall structure of the thesis is also provided.

1.1 Motivations

A graph is a very important data structure to represent entity and entity relationship in disciplines such as sociology, biology, mathematics, and computer science. For example, in a citation network, papers are linked by citations; in a coauthor network, authors are linked by co-authorship; the customers and products can be represented as bipartite graph customers and products; in a social network, friends are linked by friends. One of the most important factors in gaining insights into such representations is to evaluate the relationships (connections) between nodes in the graph.

Given a large graph, finding out how two vertices A and B are related has a lot of applications. For instance, in recent years, social networks have become more and more popular in peoples lives. In the simplest case, the relationship between two nodes is manifested as an edge in the graph. However, in most cases, social network graphs are sparse and have a large number of vertices. On the one hand using any single path to represent a relationship between two nodes is often insufficient; on the other hand, it is unrealistic and oftentimes meaningless to use the whole graph to represent the relationship.

As a result, Connection Subgraph [2] is defined as a small subgraph of a large graph that best captures the relationship between nodes, and it has a lot of applications in many domains. For example, in a social network setting, connection subgraphs will help us identify the few people

most likely to have been infected with a disease (or heard a piece of news, or other information). In a terrorist network maintained by Homeland Security, understanding how known terrorists interacted with others can help the security agents to find the other terrorists. In the protein network, capturing the connection subgraph can help find how proteins participate in pathways with other proteins. In some other domain, connection subgraph may help us summarize the relationship, or discover new important paths between two nodes.

Formally, the *Connection Subgraph Problem* is defined as [2]: Given: An edge-weighted undirected graph G, vertices s and t from G, and an integer budget b. Find: A connected subgraph H containing s and t and at most b other vertices that maximizes a goodness function g (H).

Finding a goodness function is critical for the *Connection Subgraph Problem.* It is not difficult to see that shortest path is not a good goodness function because it fails to capture other paths which also play a role in connecting the two nodes. And flow [15] can not be used as a good goodness function neither because it might eliminate some paths between two nodes due to the capacity constraints of edges. In [2], Christos Faloutsos et al. proposed the electric current as a goodness function. By assigning 1 voltage to one source node s and 0 voltage to the other node t, they tried to extract a connected subgraph which maximizes the electric current carried from s to t. But this approach has a constraint that the electric current has to flow from the vertex with higher voltage along the edge to the lower one which may eliminate some important paths.

In many networks, the relationship between nodes depends on the contributions from all the paths between them. For instance, the spread of disease, information, or news between two nodes may follow any path between these two nodes in the network. We focus on such networks. The goals of this research are to devise a method to find the connection subgraph quickly from a large graph with millions of nodes.

1.2 Research Problem and Challenges

1.2.1 Issues in Finding an Appropriate Goodness Function

In graph mining and network analysis, how to measure the relationship between two vertices is a critical part. An appropriate goodness function will help the user to find the desired subgraph. In many applications of graphs such as social networks and biology network, the spread of information, news, or diseases could follow any path from one node to another node in the graph [6] [21]. There are several measures people may think of as a goodness function to measure the relationship between two query nodes.

Shortest path is the simplest way to measure the relationship between two vertices. But obviously shortest path is not a good measure because it ignores the importance of other paths which also play some role in building the relationship between the two vertices. Flow may be another intuitive way to be thought of as the goodness function [15]. The more flow a subgraph can carry, the more important the subgraph is for capturing the relationship between the two query nodes. But using flow as the measure also has problem for the reason that flow only emphasizes the capacity of each path, and it ignores the length of the path. And we know that for many applications the longer a path is, the less important this path is in connecting the relationship between two query nodes.

Electric current was also proposed as a goodness function by C. Faloutsos [2] to measure the relationship between two vertices. Electric-current approach is a random walk based approach. It does consider certain number of paths and the length of paths. However it does not take all the paths into consideration, thus can not be a good goodness function. See the following example:

Using voltages and electric current, there are only 5 paths: s-a-b-t, s-a-c-t, s-a-b-c-t, s-b-t, s-b-c-t. However, it does not include the paths s-b-a-c-t and s-a-c-b-t which also connect the nodes s and t, and may play an important role in the relationship between s and t.

The above analysis gives us some insight of the goodness function. Since paths really matter, to build the connection between the two query vertices, a good goodness function should not only take all the paths of the two query vertices into consideration, it should also capture the

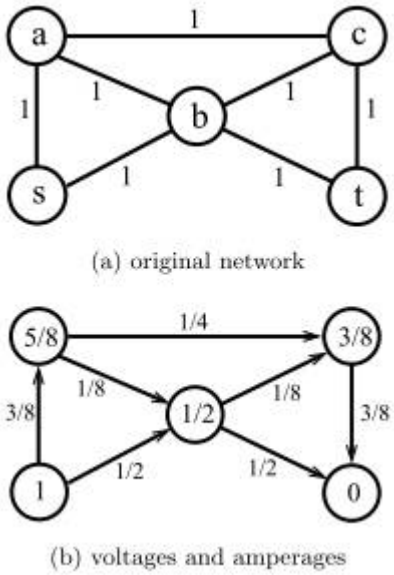

Figure 1.1 Electric Current as a Goodness Function

difference of importance among the paths.

1.2.2 Issues in Graph Computation

In graph theory and network analysis, one of the biggest challenges is scalability. One important characteristic of communication networks, link graph, social networks, or biology network is that most of them have huge number of nodes, usually more than millions of nodes. For instance, the social network website Facebook has more than eight hundred millions users. In addition, a lot of graph problems are NP-Hard problem or worse. For example, finding the number of paths between two query nodes s and t takes super-exponential time. How to extract a subgraph with at most b nodes from a large graph with millions of nodes is also a very challenging problem, because there are so many subgraphs with at most b nodes taken from the large graph. In other words, how to eliminate unimportant nodes and thus unimportant paths leading to a graph maximizing the goodness function with affordable cost is the goal of this research.

1.3 Contributions

The goals of my study are to find a way to best capture the relationship between two vertices in networks or graphs using a limited set of nodes. This is called the *Connection Subgraph Problem* defined by Christos in 2004 [2].

In order to find the connection subgraph, people first need to find a goodness function which can be appropriate in measuring the relationship between vertices. In this thesis, we propose a measure called path betweenness of a path between two nodes. Path Betweenness is the average value of the probabilities that a random walker from a start point following a specific path to an end node and back to the start point following the same path. One path has high path betweenness meaning that there is high probability for a random walker walking from s to t or t to s. Graph betweenness is defined on top of path betweenness. It is the sum of all the path betweennesses between two query nodes in the graph. Experiments show that the graph betweenness has the property of (1) the shorter a path between two nodes is, the larger the graph betweenness is; and (2) the more paths that can be found between two query nodes, the larger the graph betweenness is.

In addition, a generic two-stage framework is proposed to solve the *Connection Subgraph Problem*. The first stage is a node elimination process. It will generate a much smaller intermediate subgraph by eliminating most of the unimportant nodes based on some threshold value. A post-processing technique will then be applied to the intermediate subgraph to remove those unimportant edges using the same threshold value. In the second stage, the intermediate subgraph will be used as the input for a candidate subgraph generation algorithm to generate all the possible subgraphs with at most b nodes from the intermediate subgraph. The goodness score of each candidate subgraph will then be calculated, and the subgraph with maximal goodness score will be outputted as the connection subgraph.

1.4 Thesis Organization

In chapter 1, we first present the motivations and goals of this research. And we also discuss the challenges of this research including finding a goodness function, and designing algorithms

to quickly find the connection subgraph from a large graph. In addition, we also briefly describe as contributions of this research, the definition of Path Betweenness, Graph Betweenness and the proposal of a two-stage framework to quickly find the connection subgraph.

More information about this research will be provided in the following chapters. The remainder of this dissertation is organized as follows.

• In chapter 2, preliminaries of this research and related work are discussed. The literature review includes studies from several different domains such as social network, link analysis, and communication network.

• In chapter 3, we formally define the path betweenness. In addition, how to compute a path betweenness is also demonstrated. Then, the graph betweenness is introduced as the goodness function. The relationship between path betweenness and the graph betweenness will also be discussed.

• In chapter 4, we present a two-step framework to find a connection subgraph from a large graph or network. The two-step framework includes a node elimination process and candidate subgraph generation and selection.

• In chapter 5, experiments are conducted on both node elimination process and the connection subgraph generation. The datasets includes the 9 11 terrorist network and computer scientist network, the DBLP database. Evaluation on both steps is discussed and results are presented.

• We conclude this thesis in chapter 6. We summarize the work done in this research along with the main contributions. Several interesting future research threads related to this work are provided and discussed.

CHAPTER 2. Review of Literature

In this chapter, a review of literature is presented. The general terms and concepts are introduced And the related work is discussed.

2.1 Random Walk on Graphs

2.1.1 Random Walk Model

In this section, the basics of Random Walks on graphs will be discussed. Given a weighted graph and a starting point (node), a random walker selects a neighbor of it at random, and moves to this neighbor. Random here means that the probability of a random walker moving to one of its neighbors is proportional to the weight of their edge over the total weights of the current node to its neighbors along the edges. The sequence of nodes selected in this way is a random walk on the graph. In this paper, we will focus on the random walk on undirected graph.

Formally, given a start point s and its neighbors $v_1, v_2, \ldots, v_m$, and the corresponding edge weights $e(s, v_1)$,$e(s, v_2)$,...,$e(s, v_m)$, with weights annotated on each edge, the probability $P(s, v_i)$ of s moving to v_i is:

$$P(s, v_i) = e(s, v_i) / \sum_{j=1}^{m} e(s, v_j) \tag{2.1}$$

The following is an example of random walk on a graph:

As shown in the picture above, a random walker starts from node a and moves to one of its neighbors in the next step. The random walker has a probability of 3/5 walking from node a to node b and the random walker has a probability of 2/5 walking from a to c.

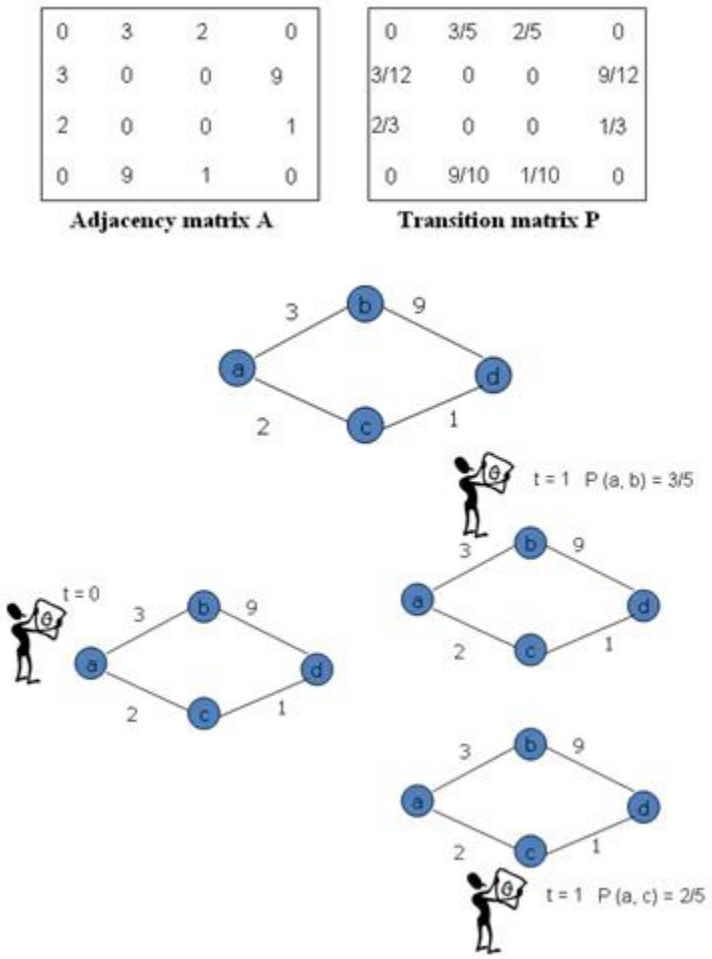

Figure 2.1 An Example of Random Walk Model

Hitting time is an important concept for random walk model. The hitting time from node i to j is the expected number of hops to hit node j starting at node i. The hitting time is not symmetric i.e. the hitting time from node i to node j is usually not equal to the hitting time from node j to node i.

Commute time is a related concept to the hitting time. Formally, commute time between nodes i and j is the expected time to hit node j from i and come back to node i from j. It is easy to observe that commute time is symmetric [8].

Some other related concepts, such as stationary distribution and converge rate, are also essential to the random walk model. Since those concepts are not directly related to this research, we do not discuss them in this paper. For references, one can find the materials in [7]

[8] [9].

Our goodness function as well as some related measures discussed in this chapter is based on the random walk theory.

2.1.2 Electric Current Based Measure

Electric current has been used in [2] as the goodness function. By assigning 1 voltage to one query node s and 0 voltage to the other query node t, the authors tried to find a connection subgraph with at most b other nodes which carries the most electric current from s to t.

Electric current is a random walk based measure. It has been proven that given an undirected graph and the vertices u and v, considering the electrical network where each node of the graph is replaced by a one ohm resistor, the commute time commute (u, v) equals $2mr_{uv}$ where r_{uv} is the effective resistance from u to v and m is the number of edges in the graph [8].

2.2 Measures of Importance or Relative Importance on the Graphs

2.2.1 Social Network Analysis

In the sociology area, there have been many previous studies on how to measure the proximity of the nodes in social networks. A lot of algorithms have been designed to identify the importance of individuals in a social network, spreading of the diseases, or their sources.

Freeman et al. (1979) [1] first developed several measures of the centrality of a vertex to determine its relative importance within the graph, such as degree centrality and closeness centrality. Another measure that Freeman et al. (1991) [15] suggested is called the flow betweeness, which utilizes the maximum flow to measure the relationship between different nodes. According to Freeman, the more flow can be carried from one node to another, the more important their relationship is. Newman (2003) [6] proposed a new measure of betweenness based on random walk. Instead of using fraction of shortest paths between two nodes as the betweenness centrality, he suggests to count how often a node is traveled by a random walk between every pair of nodes.

Overall, most of the measures focus on the relationship of nodes in the whole network.

2.2.2 Link Analysis

The analysis of hyperlinks and the graph structure of the Web have been instrumental in the development of web search. A lot of algorithms have been developed on the use of hyperlinks to rank web search results since 1990s, many of which have been proved to be successful in practice. In those algorithms, measuring the importance or relative importance of hyperlinks on the web graphs is essential for the web pages ranking.

Hyperlink-Induced Topic Search (HITS) (also known as Hubs and Authorities) is a link analysis algorithm for ranking web pages developed by Jon Kleinberg [13] [14]. The HITS assigns each page by two scores: its authority, by which the value of the page content is estimated, and its hub value which estimates the value of its links to other pages.

Named after Larry Page [11], PageRank has become one of the most famous link analysis algorithms. It has been used by the Google Internet search engine to assign a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set.

PageRank is a probability distribution used to express the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be computed for collections of documents of any size. Several research papers have considered that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value. The passes are called "iterations". By the definition of probability, a 0.6 probability is considered as a "60something happening. Thus, a PageRank of 0.6 means there is a 60a person clicking on a random link will be directed to the document with the 0.6 PageRank.

Topic-Sensitive PageRank algorithm [4] [12] is based on the PageRank algorithm. Unlike PageRank algorithm which tries to measure the importance of a web document within a set of documents (the whole graph), Topic-Sensitive PageRank tries to measure the relative importance of web documents to a certain preselected topic. Therefore, Topic-Sensitive PageRank provides a scalable way for personalizing search ranking using link analysis. Topic-Sensitive

PageRank algorithm can also be used to measure the relative importance of a node to other nodes in the network.

2.3 Connection Subgraph and its Recent Study

In 2.2, we discussed relevant work on the relationship of different nodes in the graphs. Yet, not much work has been done in representing the relationship on the graphs. On one hand, it is unrealistic to use the whole graph to represent how two nodes are related. On the other hand, it is also insufficient in many cases to use only a single path to represent a relationship between two nodes. Connection subgraph is known to utilize a small subgraph to capture the most important relationship between two nodes. We will introduce some recent work about the Connection Subgraph Problem in the following section.

2.3.1 Candidate Subgraph Generation

Faloutsos et al. first defined the Connection Subgraph Problem in 2004 [2]. In the same paper, he proposed the concept of candidate subgraph generation which aims to generate a much smaller intermediate subgraph which contains the most important paths from the original graph. Theoretically, candidate subgraph generation is a series of expansions beginning at the query node s and t. The algorithm repeatedly expands the subgraph and carefully selects other nodes by using some heuristic rules. The heuristic rules determine which node to expand next, and when to terminate the expansion. Since candidate subgraph generation uses heuristic approach, it cannot guarantee that all the important nodes and paths will be captured in the generated candidate subgraph. In addition, it is also hard to evaluate how good the candidate subgraph is.

2.3.2 Recent Studies Related to Connection Subgraph

Faloutsos et al. (2004) [16] applied his approach proposed in [2] to find the connections subgraphs in social networks. Ramakrishnan et al. in [17] used heuristics to discover the informative connection subgraphs within RDF graphs. Their heuristics are based on weighting mechanisms derived from the edge semantics theory suggested by the RDF schema. The

research shows that the connection subgraph can be used to find relationships in multi-relational graphs.

Also, Tong and Faloutsos (2006)[18] defined the Centerpiece Subgraph Problem (CEPS), which can be considered as a more general version of Connection Subgraph Problem. Centerpiece Subgraph Problem tries to find a connected subgraph which maximizes the goodness function between a set of nodes. They first applied Random Walk with Restart (RWR) approach proposed in [5] to measure the importance of the nodes on the graph. Further, they designed a dynamic programming algorithm to find the connection subgraph. Tong et al (2007) [20] generalized the CPES problem to a directed version of CEPS(dir-CEPS), in order to work for directed graphs. Nevertheless, their proposed method can not guarantee an optimal solution for the CenterPiece Subgraph Problem.

Our approach differs from the above in that we propose a two-stage strategy which will guarantee an optimal solution based on our goodness function.

CHAPTER 3. Path Betweenness and Graph Betweenness

This chapter provides the definitions of some important concepts. Two closely related concepts the path betweenness and graph betwee are introduced. And the graph betweenness is the goodness function for the *connection subgraph problem* in our research.

3.1 Goodness Function Problem

In Connection Subgraph Problem, goodness function is critical in measure the relationship between two nodes. A good goodness function will help us to capture the desired connection subgraph. There are several measures people may consider as goodness functions to solve the connection subgraph problem.

Finding the shortest path might be the simplest way to measure the relationship between two vertices. However, the shortest path solution may not be a good measure, because it ignores the fact that other paths can be also important in the relationship.

Flow may be another goodness function that is often proposed. It is noted that [21] the more flow a subgraph can carry, the more important the subgraph is in capturing the relationship between the two query nodes. But using flow as a measure can also be problematic for the reason that flow appears to only emphasize the capacity of each path while ignoring the length of the path. However, the length of the path should not be ignored as it is known that the longer a path is, the less important this path will be in connecting the relationship between two query nodes.

Electric current was proposed by C. faloutsos [2] as a goodness function for finding the connection subgraph between two nodes. This method considers both the number of paths and the length of paths. However, electric current approach does not take into account all the

possible paths, thus it cannot be considered a good goodness function, either.

As shown in Figure 1.1, by using voltages and electric current, only 5 paths are found: s-a-b-t, s-a-c-t, s-a-b-c-t, s-b-t, s-b-c-t. Apparently, it does not include another two possible paths of s-b-a-c-t and s-a-c-b-t which also connect the nodes s and t and may play an important role in the relationship between s and t.

The analysis above gives us some insight what an appropriate goodness function should cover. It is really the paths that build the connection between the two query nodes, because if there is no path between the two, there is no relationship between them, and if two nodes have many very short paths (mutual friends, coauthored papers), they have close relationship. Therefore a good goodness function should not only take into consideration of all the possible paths connecting the two query nodes, but also capture the difference of importance among various paths.

In this thesis, we look at the paths between two query nodes to measure their relationship and propose our goodness function. Similar to other research [3][4][5] where people use the probability of a random walk walking from node to node to measure the importance of a node on the graph, we use the probability of a random walker walking along the path between two query nodes to measure the importance of such a path to connect with the query nodes. Many algorithms for page ranking are considered as random walk based algorithms, such as PageRank algorithm developed by Larry Page [3], Topic Sensitive PageRank algorithm by Taher Haveliwala [4], Random Walk with Restart algorithm by Hanghang Tong and Christos Faloutsos [5], and some betweenness centrality measure by MEJ Newman [6], etc.

However, the methods mentioned above, except [5](i.e. the electric current approach) either simply use the shortest paths of node pairs on the graph, or only measure the importance of individual nodes on the graph; therefore, it is neither proper nor sufficient to use them as a goodness function. Our proposed goodness function of graph betweenness, one of the random walk based approaches, can overcome the shortcomings discussed above. In the next section, we will introduce the concept of path betweenness.

3.2 Path Betweenness

3.2.1 Definition of Path Betweenness and its Properties

In this section, we present the important concept of Path Betweenness. On a graph, each path has a path betweenness which can be used to represent the importance of the path.

Before we define the path betweenness, we need to define some related concepts.

Definition 1: A non-loop path $v_1, v_2, \ldots, v_m$ on a graph is a path with no repeated nodes.

Definition 2: The probability of moving from a start point (node) s to another node t in the graph following a non-loop path $s,\ v_1, v_2, \ldots, v_m,\ t$ is defined as the Path Probability $PPath(s, v_1, v_2, \ldots, v_m, t)$

$$PPath(s, v_1, v_2, \ldots, v_m, t) = P(s, v_1)(\prod_{j=1}^{m-1} P(v_i, v_{i+1}))P(v_m, t) \tag{3.1}$$

$P(v_i, v_{i+1})$ equals to the edge weight of moving from v_i to v_{i+1} divided by the sum of edge weights from v_i to all its neighbors except v_{i-1}. And $P(s, v_1)$ is the edge weight of s and $v_1$divided by the sum of edge weights from s to all its neighbors. It is easy to prove that path probability is a value between 0 and 1 including both 0 and 1.

Definition 3: The Path Betweenness of a path $v_1, v_2, \ldots, v_m$ connecting two query vertices s and t in an undirected weighted graph is the mean value of the path probability of moving from s to t through a particular path and the path probability of moving from t to s through the same path. i.e.,

The additive form of path betweenness is the arithmetic mean of the path probabilities:

$$PB_{s,v_1,v_2,\ldots,v_m,t}(s,t) = ((PPath(s, v_1, v_2, \ldots, v_m, t) + PPath(t, v_m, \ldots, v_2, v_1, s))/2 \tag{3.2}$$

The multiplicative form of path betweenness is the geometric mean of the path probabilities:

$$PB_{s,v_1,v_2,\ldots,v_m,t}(s,t) = \sqrt{(PPath(s, v_1, v_2, \ldots, v_m, t)PPath(t, v_m, \ldots, v_2, v_1, s)} \tag{3.3}$$

Note that in this thesis, we focus on the additive form of path betweenness. Hence, the path betweenness discussed in this paper refers to the additive form (3.2) of path betweenness.

From the definition, it is self-explanatory that the Path Betweenness has a physical meaning, i.e., the average probability of a random walker moving from s through a particular path to t

and then moving from t back to s through the same path. Thus, path betweenness is a value between 0 and 1. If a path betweenness of two nodes s and t is close to 1, then the path is considered important in the relationship between s and t. In contrast, if a path has a lot of intermediate nodes with small edge weights in the path; it is likely that the path betweenness will be quite small.

Furthermore, from the definition of path betweenness, one can easily infer that Path Betweenness is bi-directional and it is symmetric. In other words, it does not matter which node (s or t) is the start point and which node is the end point. The reason that path betweenness must be computed from both directions is because a path that is important to a query point might appear to be unimportant to the other point.

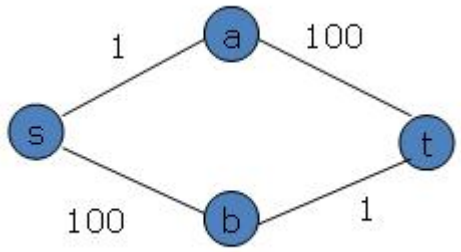

Figure 3.1 Example of Why Path Betweenness is Bi-Directional

In the above figure, the path betweenness of the path s-a-t is computed as $(P(s,a)P(a,t) + P(t,a)P(a,s))/2 = (1/101 \times 1 + 100/101 \times 1)/2 = 1/2$. Similarly, we can also calculate the path betweenness of the path s-b-t and get $1/2$. From the figure above, it is reasonable to conclude that the path s-a-t is important to t and the path s-b-t is important to s, but they are equally important to the nodes s and t, given the condition that s and t are equally important to each other.

In the following section, the comparison of Path Betweenness and random walk is discussed.

3.2.2 The Relationship Between Path Betweenness and Other Random Walk Model

From the definition of path betweenness, one can easily discovery that the computation of the path betweenness is a series random walk along the path except that the probability at each step is slightly different. The probability of next step does not include the incoming edge in the denominator. We can say that the path betweenness is a random walk based measure. In addition, according to its definition, path betweenness itself is a probability, and it is a value between 0 and 1.

Nevertheless, it is noteworthy that path betweenness differs from random walk in several aspects. In random walk, the random walker can move back to the previous node with a certain probability, while in the path betweenness, the random walker can not move back to the previous node in order to avoid a loop. Accordingly, in the random walk model, the probability of a random walker walking from a node to one of its neighbors is the percentage of their edge weight over the total edge weights from this node to all its neighbors, while in path betweenness, the probability of a random walker walking from a node to one of its neighbors is the percentage of their edge weight over the total edge weights from this node to all its neighbors excluding the incoming edge to this node in the path.

3.2.3 The Computational Complexity of Path Betweenness

Given a graph and a path with nodes and edges, the path betweenness can be calculated as demonstrated above.

However, to maximize the path betweenness in a graph is computationally too expensive, because one has to compute path betweennesses for all the possible paths between the query nodes. For instance, if given a graph with N nodes and two query nodes, in the worst case, the graph is a clique where each node has an edge connecting to all the other nodes in the graph. The total number of the paths from one query node to the other query node would be calculated as:

$$O(N-1+(N-1)(N-2)+,\ldots,+(N-1)!) = O((N-1)!) \tag{3.4}$$

From the above formula, the time complexity for finding the maximal path betweenness is super-exponential. Therefore, for a large graph, it is almost impossible for one to find the maximum of path betweenness between two nodes.

In addition, it is also very expensive to compute the maximal value of path betweennesses through a particular path with a specific node. In the worst case, when the graph is a clique, the total number of paths between two nodes passing a specific node would be

$$O(\frac{N-1+(N-1)(N-2)+,\ldots,+(N-1)!)}{N} = O((N-2)!) \tag{3.5}$$

Furthermore, it is also unrealistic to find the maximal path betweenness passing a specific node for a large graph. After defining the path betweenness, another important concept called graph betweenness will be introduced in the next section.

3.3 Graph Betweenness

3.3.1 The Definition and The Physical Meaning of Graph Betweenness

Path betweenness is a measure that can be used evaluate how important a path is. However, it cannot be used directly as a goodness function for the Connection Subgraph Problem because path betweenness is designed for a path while the connection subgraph is intent to maximize the goodness function for a subgraph. Thus, Graph Betweenness is proposed as a goodness function for the Connection Subgraph Problem.

Definition 4: Given a weighted graph G and two query nodes s and t, Graph Betweenness is defined as the sum of all the path betweennesses between two query nodes in the graph, i.e.,

$$GB(G) = \sum_{i=1}^{k} PB_{Path_i}(s,t) \tag{3.6}$$

where $Path_i = (s, v_1, v_2, \ldots, v_m, t) \in G$is the ith non-loop path between s and t, and k is the total number of path in G.

From the definition, graph betweenness takes into account contributions from all the paths with different path betweennesses. A subgraph with high graph betweenness should contain some important paths. Similar to path betweenness, the physical meaning of the graph be-

tweenness is the average probability of the random walker walking from s to t and back from t to s through all the possible paths in the graph.

In Figure 3.3, the graph betweenness for s and t is the sum of path betweennesses of path s-a-t and path s-b-t which is (1/2+1/2) = 1. Using graph betweenness as a goodness function, the goal of the Connection Subgraph Problem then is to find a connected subgraph with at most b nodes to maximize the graph betweenness of the subgraphs.

3.3.2 Comparison Between Graph Betweenness and Other Measures

Graph betweenness is a random walk based approach because it uses path betweenness as its basis. However, unlike other random walk approaches, graph betweenness only considers the non-loop paths. Graph betweenness has several advantages compared to other goodness functions such as electric current or flow.

Also recognized as a random walk based measure for the connection subgraph problem, electric current measure tries to find a subgraph which carries the most current from one query node to the other. Compared to the graph betweenness, the electric current may ignore some paths because it requires to the electric current moving from the node with higher voltage to the node with lower voltage.

Flow is also utilized as the goodness function for the Connection Subgraph Problem to find a subgraph which carries the most current from one query node (source) to the other. However, compared to the graph betweenness, the flow measure cannot differentiate the contributions of two different paths with same capacity limit when there is a huge difference between them in terms of the length of path.

3.3.3 The Computational Issue of Graph Betweenness

Given an undirected weighted graph with N nodes and two query nodes of s and t, the graph betweenness can be computed by finding all the possible paths between the query nodes, and summing all the path betweennesses.

Note that each path in the graph contains at least two nodes (s and t) and at most N nodes. Therefore, the number of edges for each path is no more than N-1. The algorithm of computing

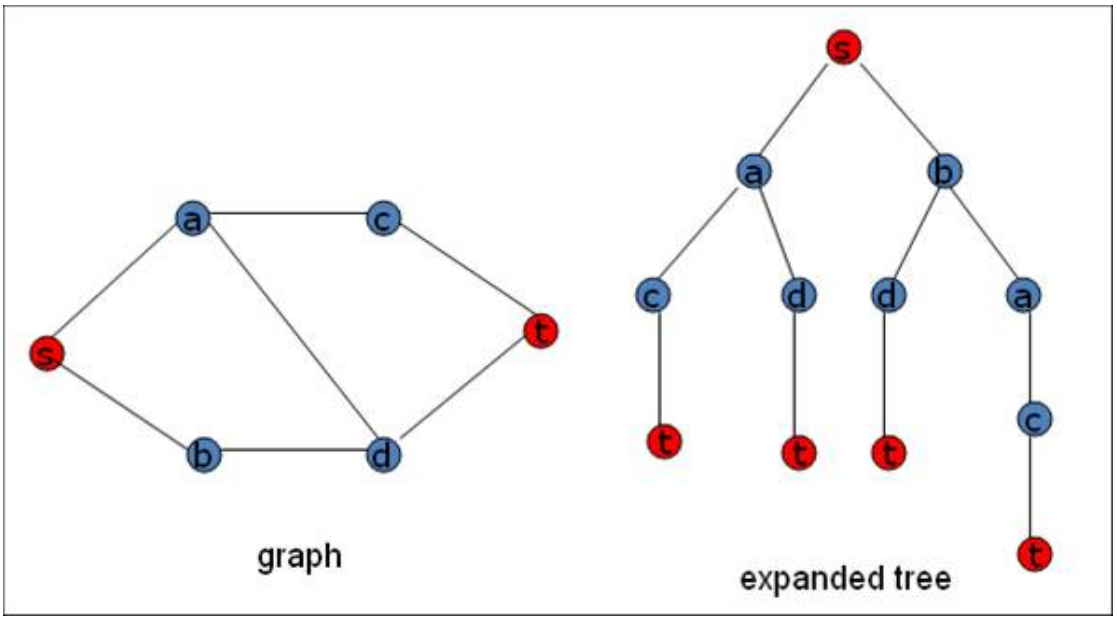

Figure 3.2 Expanded Tree to get Graph Betweenness

the graph between the nodes starting from s to s neighbors in the first step, and then expanding to s neighbors neighbors in the next step. This can be done in a recursive manner. And the process is to derive an expanded tree structure.

Algorithm to compute graph betweenness-Initial setting

1: Given an undirected weighted graph G, and two query nodes s and t in G

2: Add s to the queue visitedList

3: Return GetGraphBetweenness(visitedList, G, t)

Algorithm to compute graph betweenness

---

double GetGraphBetweenness(visitedList, G, t)

{

//visitedList is an array list

```
1:Let curVisit = visitedList.LastVisitedElement
2:GraphBetweenness = 0
3:For each neighboring node v of curVisit
4:      if (v == t) //find a path
5:          Add v to visitedList
6:          PB = Path Betweenness of visitedList
7:          GraphBetween = GraphBetween + PB
8:      else if (v is not visited)//v is not in visited
9:          Add v to visitedList
10:         GraphBetween = GraphBetween + GetGraphBetweenness(visitedList, G, t)
11:         end if
12:     end if
13:Return GraphBetween
}
```

In the worst case, when the graph is a clique, there are N-1 paths for moving from s to t in two steps, and (N-1)(N-2) paths for moving from s to t in three steps. Therefore, there are (N-1)! paths for moving from s to t in N-1 steps. The total paths from s to t are $O((N-1)!)$.

For small graphs, the computational cost of computing the graph betweenness might be much smaller. Yet for a large graph, it is unrealistic to compute the graph betweenness since the cost to accomplish it might be unaffordable.

3.4 Summary

This chapter discusses goodness functions for the connection subgraph. We first introduce the challenges of finding an appropriate goodness function. The properties of a good goodness function are discussed. The random walk model is presented to serve as foundation for Path Betweenness. Path Betweenness is the average probability that a random walker walks from a node s through a particular path to another node t and from t back to s following the same path. Graph betweenness of two query nodes is the sum of all the path betweennesses of

the query nodes in the graph. Graph betweenness is a goodness function because it accounts for all the paths with different path betweennesses, and the path betweenness of a path is a measure the importance of path. A subgraph with high graph betweenness should contain some important paths. Nevertheless, Calculation of the graph betweenness can be challenging for the computational complexity of computing a graph betweenness is super-exponential. In the next chapter, an approach will be proposed to solve this problem.

CHAPTER 4. Two-Stage Framework for Connection Subgraph Problem

In the previous chapter, we introduced the necessary goodness function to enable us to know the goals. In this chapter, we describe our work in detail. We start with an overview of our approach, and follow it up with an elaboration of each of the individual steps in the overall process.

4.1 A Two-Stage Framework for Solving Connection Subgraph Problem With Optimal Graph Betweenness

Using graph betweenness as the goodness function, the connection subgraph problem can be redefined as:

Given: an edge-weighted undirected graph G, and two vertices s and t in G, an integer budget b, and the graph betweenness function GB(H) as the goodness function.

Find: a connected subgraph H containing s, t and at most b other vertices that maximizes GB(H).

4.1.1 Computational Issue of Brute Force Method

Using the brute force method to solve the connection subgraph problem, one basically needs to enumerate all the possible subgraphs, compute their graph betweennesses, and output the subgraph with the maximal graph betweenness. Figure 4.1 shows how brute force method is utilized to solve the connection subgraph problem.

However, the computational issues would make the brute force approach unrealistic in the case of large graphs. In the worst case, when a graph G with N nodes is clique, the total number of subgraphs with b nodes produced by the graph G is $\binom{N}{b}$. And the number of paths

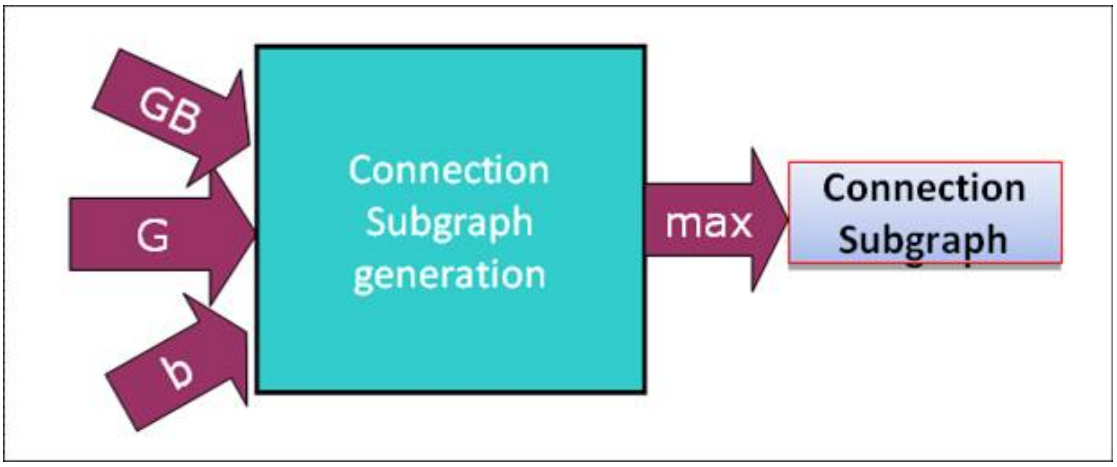

Figure 4.1 One-Step Approach to Compute Connection Subgraph

in a connected subgraph with b nodes is O((b-1)!). The time complexity of the brute force is:

$$O(\binom{N}{b} \times O((b-1)!)) = O(\frac{N!}{b}) \tag{4.1}$$

Note that b is usually a number smaller than 40. Yet, for a graph with 1000 nodes and b = 10, the number is almost (1000!).

In order to deal with the computational issue, we need to find a mechanism to greatly reduce the graph scale by eliminating unimportant nodes.

4.1.2 Using Graph Betweenness with Each Path Betweenness Greater than the Threshold Value

Recall that the physical meaning of path betweenness is the average probability that a random walk walks from a query point s to the other query point t and back from t to s following the same path. If the path betweenness of a path is quite small compared to other path betweennesses, the path is considered to be unimportant in the relationship between the query nodes. Thus, the path can be eliminated. In addition, usually b is quite small compared to the number of nodes, and the connection subgraph should capture the most important paths. Therefore, using graph betweenness with each path betweenness greater than the threshold value as a goodness function is reasonable for the Connection Subgraph Problem. In the rest of the thesis, we use graph betweenness with a threshold value to refer to the graph betweenness

with each path betweenness that is greater than the threshold value.

Furthermore, setting a threshold value for the path betweenness of all paths in the graph can greatly reduce the computation cost. Since path betweenness is a random walk based algorithm, usually the more hops the path has, the smaller the path betweenness is. In addition, as the hops of the paths increase, the number of paths usually increases sharply. For example, when a graph is a clique with N nodes, there will be (N-1)! paths with (N-1) hops.

Using graph betweenness with a threshold value not only provides a way to eliminate all the paths with path betweenness smaller than the threshold value, but also enable elimination of the unimportant nodes. If the maximal path betweenness of all paths passing a specific node between the query nodes is smaller than the threshold value, then this node should not be in the connection subgraph, which means it can be eliminated.

Definition 5: The Node Betweenness of a node v is defined as the maximal path betweenness of all paths passing a specific node between the query nodes. i.e.,

$$PB(s,t)_v = Max((PPath_i(s,t)_v + PPath_i(t,s)_v)/2 \tag{4.2}$$

The analysis above gives us some insights on how to reduce the size of the graph by eliminating those unimportant nodes.

4.1.3 The Two-Stage Framework

We will present a two-stage framework by using the threshold value. Since the connection subgraph has at most b nodes which is relatively quite small compared to the size of the graph, most of the nodes in the original graph G can be eliminated.

The idea of a two-stage framework is to first reduce the scale of the graph by eliminating most of all the unimportant nodes to generate a small enough intermediate subgraph, and next to find the connection from the much smaller intermediate subgraph.

The Figure 4.2 illustrates the two-stage framework used to find the connection subgraph. In the first step, the node elimination algorithm takes the original graph G, the definition of the graph betweenness and the threshold value as inputs, and generates an intermediate subgraph. Those nodes eliminated should have a node betweenness smaller than the threshold

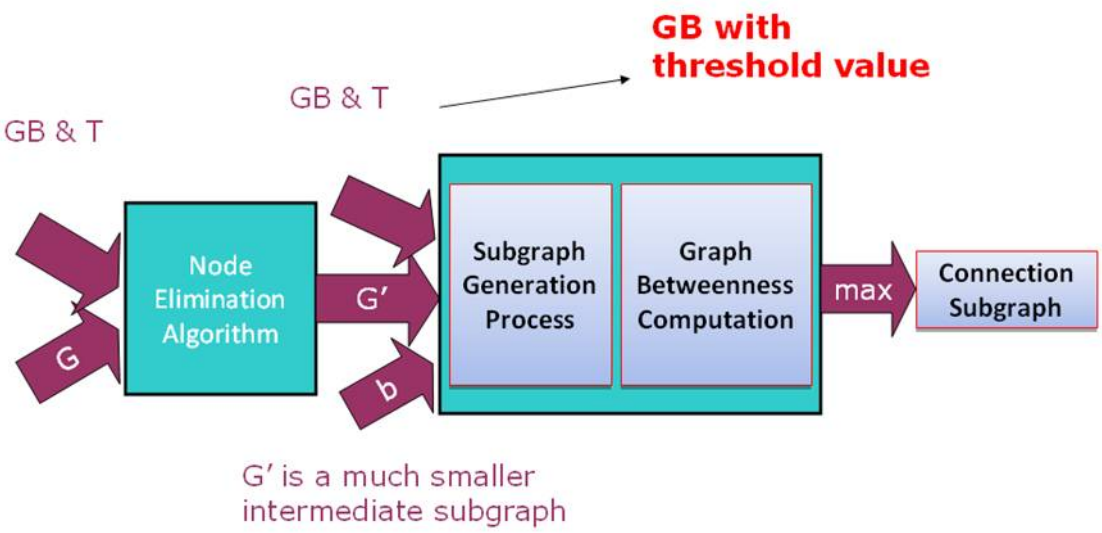

Figure 4.2 Two-Stage Framework for Connection Subgraph Problem

value. Since the node betweenness of a node is defined as the maximal path betweenness of the paths passing the node, if the node betweenness of a node is smaller than the threshold value, all the paths passing this node have the path betweennesses smaller than the threshold value, thus can be eliminated. In the second step, the intermediate subgraph is used as an input to identify the connection subgraph. This process includes candidate subgraph generation, and graph betweenness computation.

4.2 Node Elimination Process and Intermediate Subgraph Generation

4.2.1 Preprocessing of the Input Graph G

The preprocessing step is to eliminate all the nodes through which there is no non-loop path from s to t or t to s. We design an algorithm to remove all those nodes in the preprocessing step.

Preprocess the input graph G = (V, E)

Preprocessing (G =(V,E))

{

```
1: Queue Q is empty
2: for each vertex v in G // first step
3:    if v has only one neighbor u
4:       add u to Q;
5:       remove u from vs neighbor list
6:       remove v from us neighbor list
7:       end if
8:    end for
9:    While (Q is not empty) // the second step
10:         for each vertex v in Q
11:             if v has only one neighbor u
12:                 add u to Q;
13:                 remove u from vs neighbor list
14:                 remove v from us neighbor list
15:             end if
16:         end for
17:    end while
}
```

Since path betweenness is based on a path between the query nodes, if a node has only one neighbor, there is no non-loop path from s to t passing this node; therefore, this node can be eliminated.

During the preprocessing, in the first step, only the nodes with one degree are pruned. The total cost is O (N). In the second step, we try to remove those nodes which have only one degree after removing their connections to the pruned nodes. Each node in Q will be processed twice, and there are at most N nodes in Q. Therefore, the total operations are O (N). The time complexity for preprocessing is O (N). After the preprocessing, we obtain a graph in which each node has a degree of at least two.

4.2.2 Algorithms to Find an Upper Bound of the Node Betweenness

In order to eliminate a node, we need to compute the maximal path betweenness of all the paths passing the node. As it has been proved, the complexity of getting the maximal path betweenness passing a node is O ((N-2)!) which makes it inapplicable for large graphs.

One challenging task is to eliminate the nodes in polynomial time. Fortunately, we are able to find an upper bound for the node betweenness in polynomial time. From Formula 4.2, it is easy to get:

$$PB(s,t)_v \leq (PPath_{max}(s,t)_v + PPath_{max}(t,s)_v)/2 \tag{4.3}$$

Further, Formula 4.3 shows that the a node betweenness is smaller or equal to the sum of the maximal PPaths from s to t passing v and the maximal PPath from t to s. In addition, we can get:

$$Path_{max}(s,t)_v \leq \alpha PPath_{max}(s,v)_v PPath_{max}(v,t)_v \tag{4.4}$$

$$Path_{max}(t,s)_v \leq \alpha PPath_{max}(t,v)_v PPath_{max}(v,s)_v \tag{4.5}$$

where α is a value equal or greater than 1, and it is decided by the nodes v and its neighbors. Once a graph is given, α is fixed.

Also, Formula 4.4 presents a fact that the node betweenness of node v is the product of α, the maximal PPath from s to v and the maximal PPath from v to t. Note that the PPath from s to v does not include any path with t as an intermediate node in the path. Similarly, the PPath from v to t does not include any path with s as an intermediate node in the path. Formula 4.4 can be proved by contradiction. Suppose the right hand side of the formula above is smaller than its left hand side. It is easy to conclude that there must be a path from s to v with a PPath greater than $PPath_{max}(s,v)$, or there must be a path from v to t with a PPath greater than $PPath_{max}(v,t)$. In either way, we can come to the conclusion that neither $PPath_{max}(s,v)$ nor $PPath_{max}(v,t)$ is maximal, which contradictes with the fact that both of them are maximal. Similarly, we also prove that formula 4.5 is correct.

We are going to present an approach to find the upper bound for the node betweenness. The polynomial time algorithm to compute the maximal P-Path from s to v is described as follows:

Algorithm to compute the $PPath_{max}(s, v)$ or $PPath_{max}(t, v)$

ComputeMaxPPath(G, x,v)

{

1: Random walker starts from x to visit its neighbors

2: Queue Q is empty

3: Add x to Q, and PPath(x,x) = 1

4: Initialize PPath(x,i) = 0 for all the nodes except x

5: while (Q is not empty)

6: for each vertex u in Q

7: for each neighbor j of u

//w is us neighbor, and e(u,w) is weight of edge

8: $e(u, w) = max\{e(u, i) \mid e(u, i) > 0 \ \& \ i \neq j\}$

//if from x to u and u to j, there is a higher PPath(x,j), then update it

9: if $(PPath(x, u) \times e(u, j)/(\sum_{i=1}^{degree(u)} e(u, i) - e(u, w)))$

10: $PPath_{max}(x, j) = PPath(x, u) \times e(u, j)/(\sum_{i=1}^{degree(u)} e(u, i) - e(u, w))$

11: Add j to Q

12: end if;

13: end for;

14: end for

15: end while

}

In the algorithm, the while loop has at most N steps because the maximal hops of a path is (N-1), and the first for loop has at most (N-1) passes because in each pass there are at most N nodes added in Q. The second for loop has at most K passes where K is the maximal degree of a node. Therefore the time complexity to compute $PPath_{max}(s, v)$ is $O(KN^2)$ which is in polynomial time.

Likewise, we can compute $PPath_{max}(t, v)$ in $O(KN^2)$ time. To compute $PPath_{max}(v, t)$ or $PPath_{max}(v, s)$, we can use the algorithm shown above. However, we need to compute the value for (N-2) nodes. Then the computational complexity will be $O(KN^3)$.

Instead of computing $PPath_{max}(v, t)$ or $PPath_{max}(v, s)$ directly, we present an algorithm to compute an upper bound for $PPath_{max}(v, t)$ or $PPath_{max}(v, s)$ in $O(KN^2)$. The algorithm starts from s, and computes the maximal possible probability from s neighbors to s and the maximal possible probability from s neighbors neighbor to s until all the nodes in graph get their maximal possible probability to s.

Algorithm to compute a upper bound for $PPath_{max}(v, s)$ or $PPath_{max}(v, t)$

ComputeMaxPPath(G, x,v)

{

1: Random walker starts from x to visit its neighbors

2: Queue Q is empty

3: Initialize PPath(i,x) = 0 for all the nodes except x

4: Add x neighbors to Q, compute the probability from x neighbors to x

5: While (Q is not empty)

6: for each vertex u in Q

7: for each neighbor j of u

//w is js neighbor, e(j,w) is weight of edge

8: $e(j, w) = max\{e(j, i) \mid e(j, i) > 0 \ \& \ u \neq j\}$

//if from j to u and u to x has higher PPath(j,x), then update it

9: if $(PPath(x, u) \times e(u, j)/(\sum_{i=1}^{degree(u)} e(j, i) - e(j, w)))$

10: $PPath_{max}(x, j) = PPath(x, u) \times e(u, j)/(\sum_{i=1}^{degree(u)} e(j, i) - e(j, w))$

11: Add j to Q

12: end if;

13: end for;

14: end for

15: end while

}

In the algorithm, the while loop has at most N steps because the maximal hops of a path is (N-1), and the first for loop has at most (N-1) passes because in each pass there are at most N nodes added in Q. The second for loop has at most K passes where K is the maximal degree of a node. Therefore the time complexity to compute upper bound of $PPath_{max}(v,s)$ is $O(KN^2)$ which is in polynomial time.

Similarly, we can compute the upper bound of $PPath_{max}(v,s)$ in $O(KN^2)$ time. Therefore, the overall time complexity of computing an upper bound for $PPath_{max}(v,s)$ and $PPath_{max}(v,s)$ is $O(KN^2)$.

4.2.3 Node Elimination Process and Intermediate Subgraph Generation

Now we have calculated $PPath_{max}(s,v)$, $PPath_{max}(t,v)$, and the upper bound of $PPath_{max}(v,s)$ and $PPath_{max}(v,t)$. We refer to the upper bounds of $PPath_{max}(v,s)$ and $PPath_{max}(v,t)$ as $upper_bound(PPath_{max}(v,s))$ and $upper_bound((PPath_{max}(v,t))$. From formula 4.3, 4.4 and 4.5, we can get:

$$PB(s,t)_v \leq (PPath_{max}(s,v) \times \alpha PPath_{max}(v,t) + PPath_{max}(t,v) \times \alpha PPath_{max}(v,s))/2 \quad (4.6)$$

$$PB(s,t)_v \leq (PPath_{max}(s,v) \times upper\_bound(PPath_{max}(v,t)) + PPath_{max}(t,v) \times upper\_bound(PPath_{max}(v,s)))/ \quad (4.7)$$

where α is a value equal or greater than 1, and it is decided by the nodes v and its neighbors. Once a graph is given, α is fixed.

We eliminate the nodes from the graph G if

$$(PPath_{max}(s,v) \times upper\_bound(PPath_{max}(v,t)) + PPath_{max}(t,v) \times upper\_bound(PPath_{max}(v,s)))/2 < Threshold \quad (4.8)$$

After node elimination, we will obtain a smaller intermediate subgraph. The size of subgraph depends on the threshold value. Next, how to select the threshold value will be discussed.

4.3 How to Decide the Threshold Value

4.3.1 The Importance of Threshold Value

In the node elimination process, we use threshold value to eliminate nodes. The nodes eliminated have node betweennesses smaller than the threshold value, meaning that the path betweenness of any path passing this node is smaller than the threshold value. Since the upper bound of each nodes node betweenness is fixed, the selection of the threshold value directly determines the number of nodes to be eliminated. On the one hand, if the threshold value is too small, only a few nodes might be eliminated. The produced intermediate subgraph might be too large for connection subgraph. On the other hand, if the threshold value is too large, some important nodes might be mistakenly eliminated in the elimination process. As a result, the generated connection subgraph might not include some important paths.

The above analysis indicates the importance of choosing a threshold value. A good threshold value not only can efficiently produce the connection subgraph, but also ensure that the final connection subgraph retains all the important nodes and paths.

4.3.2 How to Choose Threshold Value

There are three ways to choose the threshold value:

The first one is using elimination rate to choose threshold value. We can set an elimination rate r, and eliminate rN nodes from the original graph. In order to remove rN nodes, the upper bound of each nodes node betweenness is first computed, and the node betweennesses are sorted in the increasing order. The upper bound of the (rN+1)th nodes node betweenness is set as the threshold value. Thus the rN nodes with low node betweenness are eliminated.

Another way to find the threshold value is using b as a base. We try to generate an intermediate subgraph with size Kb. Similar to using elimination rate, first the upper bound of node betweennesses is sorted, then the (Kb)th largest upper bound of node betweenness is set as the threshold value. Therefore, Kb nodes will be kept in the intermediate subgraph.

The third way is a quite different method compared to the first two. In the third way, the predefined threshold value is given as an input. After calculating the upper bound of node

betweenness, any node whose node betweenness is smaller than this threshold value will be removed. Nevertheless, this approach leaves some uncertainty. Since the threshold value is fixed and predefined, the generated intermediate subgraph might contain a large number of nodes. It is also possible that most of the nodes in the graph will be eliminated.

The advantage for the first two approaches is that you have the full control of the size of the intermediate subgraph; therefore you can control the speed of producing the connection subgraph. However, some important nodes or paths could be eliminated if the threshold value is too large. Using a small threshold value, the fixed threshold value approach can guarantee a connection with all important paths and nodes retained while it may take a long time to output the connection subgraph.

In reality, the upper bond of nodes betweenness usually decreases exponentially due to its random-walk property. That leaves us some space in choosing the threshold value.

4.4 Postprocessing of Node Elimination

After the intermediate subgraph is generated, the postprocessing is applied to speed up the candidate subgraph generation. The postprocessing includes the two steps.

In the first step, the upper bound of node betweenness is used to prune those edges whose maximal path betweenness of all paths between s and t is smaller than the threshold value.

Definition 6: Edge Betweenness of an edge is defined as the maximal path betweenness of all paths passing this edge between the query nodes s and t.

We can use the upper bound of node betweenness of the edges two end points to get the upper bound of edge betweenness. It is not difficult to observe that if the edge betweenness of an edge is smaller than the threshold value, then all the paths passing this edge will have path betweenness smaller than the threshold value. Therefore, all those paths can be eliminated i.e. we can remove the edge from the graph. The detailed algorithm is shown below.

Postprocess the input graph G: Edge elimination

EdgeElimination (G, threshold, s, t) //G is the intermediate subgraph

{

1: for each vertex v in G

2: for each neighbor u of v

3: $PPath_{max}(s,u)' = PPath_{max}(s,v)e(v,u)/(\sum_{i=1}^{degree(v)} e(v,i) - maxe(v,j) \mid e(v,j) > 0 \ \& \ j \notin Path_{max}(s,v))$

4: upper bound of upper bounds of

5: PB0 = (the upper bound of + the upper bound of)/2

6:

7: upper bound of upper bounds of

8: PB1 = (the upper bound of + the upper bound of)/2

9: if (PB1 ¡ threshold && PB0 ¡ threshold)

10: remove edge e (u, v)

11: end if

12: end for

13: end for

}

The time complexity for this algorithm is O (| V | + | E |) where | V | is the number of nodes in the intermediate subgraph and | E | is the number of edges in the intermediate subgraph because each edge in the intermediate subgraph is checked only once.

The second step is the same as the preprocessing step. All the nodes with no non-loop paths between s and t will be removed from the intermediate subgraph. The algorithm is the same as shown in Figure 4.3.

After the postprocessing, we obtain an intermediate subgraph with the upper bound of the node betweenness and the upper bound of the edge betweenness of each node that is greater than the threshold value.

4.5 Candidate Subgraph and Connection Subgraph Generation

The second stage of the two-stage framework includes the candidate subgraph generation and connection subgraph generation. Candidate subgraph generation will generate all the

possible subgraphs with at most b nodes. In order to make sure that all the subgraphs are connected and include s and t from the interim graph, nodes are grouped by the number of hops from them to the query node s. Starting from each node with one hop away from s, the algorithm iterates all the possible combinations of the nodes in the group. Once a combination is given, the neighbors of current combination from next group are selected as the expanding candidates, and each combination of the expanding candidates will be tried. This process is repeated until we find a connection subgraph with b nodes.

Group nodes by their smallest hops from s

```
GroupNodeByHops (G, s) //G is the generated intermediate subgraph
{
1:Boolean allVisited = false;
2:NodeGroups = ArrayList¡ArrayList¡Integer¿¿
3:NodeGroups[0].add(s); //Add s to the 0 hop group
4:int[] HopNum = new int [size of G] and initialize all elements with -1;
5:HopNum[s] = 0;
6:int i = 0; //Current group number
7:While(!allVisited)
8:   Let allVisited = true;
9:   CurGroup = the ith List of NodeGroups;
10:  i++; // process next group
11:  Create a new group called NextGroup
12:  for each element v of CurGroup
13:     for jth neighbor u of v
14:        if HopNum[j] ¡0
15:           Add u to NextGroup;
16:           HopNum[j] = i;
17:           allVisited = false;
```

```
18:         end if
19:      end for;
20:  end for;
21:  if (!allVisited)
22:      add NextGroup to NodeGroup;
23:  end if;
24:end while;
}
```

Connection Subgraph Generation 1: Initialization

```
CSGInitialization (G, b, s, t) //G is the generated intermediate subgraph
{
1:Group nodes by their smallest number of hop from s
2:HopNum = 1;
3:TSlcted = true; // Boolean variable to indicate t is visited
4:ToExpand = neighbors of s;
5:MaxGraphBetweenness = 0;
6:for each element curComb of ToExpands power set
7:   If (curComb contains T)
8:       TSlcted = true;
9:   Else
10:      TSlcted = false;
11:  If(curComb.size == b)
12:      GraphBetweenness = GetGraphBetweenness(curComb, s, t);
13:      If (max ¡ GraphBetweenness)
14:          Max = GraphBetweenness;
15:          Store curComb;
16:       else
```

17: CandidateGraphGeneration(G, curComb, curComb, b-curComb.size, 1, TSlcted)

}

Connection Subgraph Generation 2: Recursively generate candidate connection subgraphs

---

CandidateGraphGeneration(G, SelectedNodes, CurSelected, NodeNum, HopNum, TSlcted)

{

1:ToExpand = null;

2:for each neighbor i of CurSelected; // CurSelected is a set of nodes

3: if (the hop number of i is greater than HopNum)

4: Add i to ToExpand;

5: end if

6:end for;

7:HopNum++;

8:For each combination curComb of ToExpand;

9: If(TSlcted != True && CurComb contains T)

10: TSlcted = true;

11: If (NodeNum == curComb.size)

12: GraphBetweenness = GetGraphBetweenness (SelectedNodes + curComb,s,t);

13: If (max < GraphBetweenness)

14: max = GraphBetweenness;

15: Store SelectedNodes + curComb;

16: end if

17: Else if(curComb.size ¡ NodeNum)

18: CandidateGraphGeneration (G, SelectedNodes + curComb, curComb, NodeNum-curComb.size, HopNum, TSlcted)

19: end if;

20: end if;

21:end for

}

Given a candidate connection subgraph with at most b nodes, we can use the algorithm in Figure 3.5 to calculate the graph betweenness of this subgraph. And the graph betweenness will be compared with the current maximal graph betweenness stored. If graph betweenness is greater than the current maximal graph betweenness, the current maximal graph betweenness will be updated. After all the subgraphs have been compared, the subgraph with the maximal graph betweenness will be updated as the connection subgraph.

4.6 Summary

We present a two-stage framework for the Connection Subgraph Problem. The first stage is the node elimination process. It is dedicated to generating a small enough intermediate subgraph by eliminating the unimportant nodes whose node betweenness is smaller than the threshold value. After the node elimination, a postprocessing is operated in order to remove some unimportant edges from the intermediate subgraphs. The second stage takes the intermediate subgraph as the input, and output the connection subgraph. In order to achieve the goal, a candidate subgraph generation algorithm, the calculation of the graph betweenness, and the comparison of algorithms are proposed.

CHAPTER 5. Experiments and Evaluation

In this chapter, we design experiments to evaluate our proposed approaches. First, we set up the experiments, and next we present the results. Our experiments were designed to answer the following questions:

• How well does the node elimination process perform?

• How do we decide the threshold value? What relationship is between the threshold value and the elimination rate?

• How well does our connection subgraph generation algorithm capture the relationship between nodes?

5.1 Experiments

5.1.1 System Implementation and Run

We implemented a system called SocialMiner in Java. SocialMiner currently runs on Intel(R) Core(TM) 2 Duo CPU and P8600 2.40 GHz processor. User first inputs the adjacent list representation of the original graph in a file and a threshold value or an elimination parameter, followed by running the node elimination process to output an intermediate subgraph. The generated intermediate subgraph is sent to the connection subgraph generation module with other user input information such as query nodes, threshold value, and budget b. The connection subgraph is outputted and displayed on the GraphViz system [32]. Here initial concepts and conditions are explained and several hypothesis are mentioned in brief.

5.1.2 Data Sets

We used the DBLP dataset in our experiments.

The DBLP data set presents information on computer science publications listed in the DBLP Computer Science Bibliography [33]. The data in this dataset were derived from a snapshot of the bibliography which contains a sample data of authorship graph from the ACM SIGMOD conference [22]. The sample data set contains 3379 computer scientists (nodes) and 8430 co-authorships (edges).

5.2 Intermediate Subgraph Generation

In this section, we evaluate the performance of the node elimination process. On the first stage, we try to retain the most important nodes and paths. We measure the important node coverage and the important path coverage in the intermediate subgraph. We found that a small intermediate subgraph can still retain most important nodes and edges.

5.2.1 Node Coverage Based on Shortest Distance to s and t

The first measure is the node coverage based on the shortest distance to query nodes. We first compute the top K nodes which have the shortest distances to two query nodes s and t. And we generate an intermediate subgraph with size M separately using the node elimination process. The ratio of the number of the nodes in both groups to the size M of the intermediate subgraph is recorded as the *short-distance node coverage*. In order to avoid bias, we randomly selected ten different pairs. The results are shown as follows:

Table 5.1 Average short-distance Node Coverage Change for Ten Pairs of Query Nodes

M	K=6	K=10	K=15	K=20	K=30	K=40	K=50	Average
M=K	0.367	0.367	0.406	0.483	0.456	0.483	0.467	0.433
M=2K	0.533	0.533	0.594	0.667	0.656	0.642	0.660	0.612
M=3K	0.700	0.700	0.711	0.733	0.711	0.767	0.853	0.739
M=4K	0.767	0.767	0.756	0.767	0.844	0.867	0.907	0.810
M=5K	0.800	0.800	0.772	0.800	0.922	0.925	0.927	0.849
M=10k	0.867	0.867	0.978	0.950	0.989	0.992	0.993	0.948

In Table 5.1, the first row represents the number K of the nodes with shortest distance to s and t, and the first column shows the size M of the intermediate subgraph. From the table, it

is clear that as the M/K increases, the shortest nodes coverage increases. In addition, given M is fixed, as the K increases, the short-distance nodes coverage increases slowly. For example, when M = K, the short-distance nodes increases from 0.367 (when K = 6) to 0.467 (when K = 50).

Table 5.2 Node Coverage Change as M/K Changes

M	Pair(25,336)	Pair(78,224)	Average
M=K	62.3%	37.9%	44.2%
M=2K	84.2%	48.2%	61.5%
M=3K	91.9%	60.7%	72.6%
M=4K	96.0%	67.2%	78.9%
M=5K	99.3%	73.2%	83.8%
M=10K	100%	93.2%	95.1%

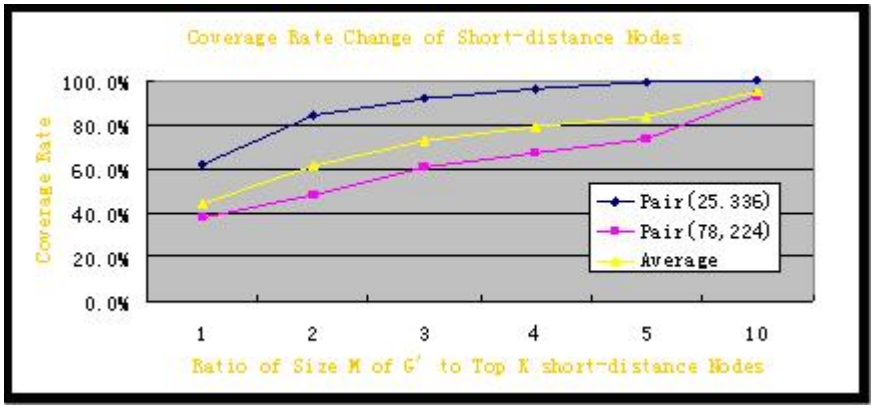

Figure 5.1 Coverage Rate Change as the Size of the Intermediate Subgraph Change

From Tables 5.1, 5.2 and Figure 5.1, as the ratio of the size M of the intermediate subgraph to the size K of the nodes with shortest distance to the query nodes increases, more nodes with shortest distance will be included in the intermediate subgraph. When the size of the intermediate subgraph is five times of the size of such nodes, the intermediate subgraph covers about 81.9% of K such nodes. When the ratio reaches 10, 95.4% of the nodes with shortest distance to the query nodes are in the intermediate subgraph.

In conclusion, the node elimination process does retain the nodes with shortest distance to the query nodes. M/K = 10 can achieve over 95% short-distance node coverage.

5.2.2 Node Coverage Based on Random Walk with Restart (RWR)

Random Walk with Restart (RWR) [5] [4] is an approach to measure the relative relevance between two nodes in a weighted graph. A random walker starting from the query nodes randomly walks to its neighbors, and each time the random walker has a small predefined probability of walking back to the starting point. To some point, the process will reach the stationary state. The number associated with each node in the stationary state is the probability of the random walker walking from the starting point to the node and finally stay at that node. RWR has been successfully used in numerous settings, such as automatic captioning of images, generalizations to the connection subgraphs, personalized PageRank, and many more. We compute the relevance score of every node in the graph to two query nodes, and therefore each node has two scores: one is to s and the other one is to t. We use the products of the two scores to represent the relative importance of nodes to the query nodes. We call it the RWR score. The physical meaning of the RWR score is the probability of two random walkers walking from s and t to a node, respectively, and finally staying at the same node.

Similarly, we first compute the top K nodes which have the highest RWR score to the two query nodes s and t. Then we generate an intermediate subgraph with size M separately using the node elimination process. The ratio of the number of the nodes in both groups to the size M of the intermediate subgraph is recorded as the RWR node coverage. In order to avoid bias, we randomly selected ten different pairs. The results of average value are shown as follows:

Table 5.3 Average RWR Node Coverage Change for Ten Pairs of Query Nodes

M	K=6	K=10	K=15	K=20	K=30	K=40	K=50	Average
M=K	44.4%	40.0%	37.8%	51.7%	53.3%	59.2%	62.0%	49.8%
M=2K	66.7%	60.0%	75.6%	78.3%	75.6%	81.7%	82.7%	74.3%
M=3K	77.8%	83.3%	82.2%	88.3%	80.0%	89.2%	90.7%	84.5%
M=4K	94.4%	90.0%	93.3%	93.3%	87.8%	90.0%	95.3%	92.0%
M=5K	100.0%	85.0%	100.0%	92.5%	88.9%	96.7%	95.3%	94.1%
M=10k	100.0%	100.0%	100.0%	100.0%	100.0%	99.2%	98.7%	99.7%

In Table 5.3, the first row represents the number K of the nodes with shortest distance to s and t, and the first column shows the size M of the intermediate subgraph. From the table,

it is clear that as the M/K increases, the short-distance nodes coverage increases. In addition, given M is fixed and M/K is small, as the K increases, the short-distance nodes coverage also increases slowly. For example, when M = K, the short-distance nodes increases from 44.4% (when K = 6) to 62.0% (when K = 50).

Table 5.4 RWR Node Coverage Change as M/K Changes

M	Pair(25,336)	Pair(226,931)	Average
M=K	59.9%	41.0%	50.5%
M=2K	85.1%	67.2%	74.6%
M=3K	88.5%	80.9%	84.6%
M=4K	91.8%	90.7%	92.3%
M=5K	93.0%	91.0%	94.9%
M=10K	98.8%	100.0%	99.6%

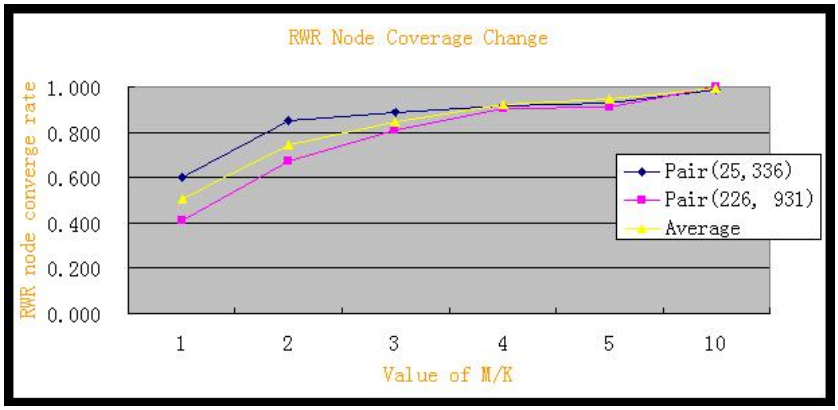

Figure 5.2 RWR Node Coverage Rate Change as M/K Changes

From Table 5.3, 5.4 and Figure 5.2, as the ratio of the size M of the intermediate subgraph to the size K of the nodes with shortest distance to the query nodes increases, more nodes with highest RWR scores will be included in the intermediate subgraph. When the size M of the intermediate subgraph is five times the size K of the nodes which have the highest RWR scores to the two query nodes, the intermediate subgraph covers about 94.9distance to the query nodes. When the ratio reaches 10, 99.6in the intermediate subgraph.

In Figure 5.3, we compare the coverage rates of nodes with shortest distances to query nodes and nodes with highest RWR scores to query nodes. It is easy to find that the RWR

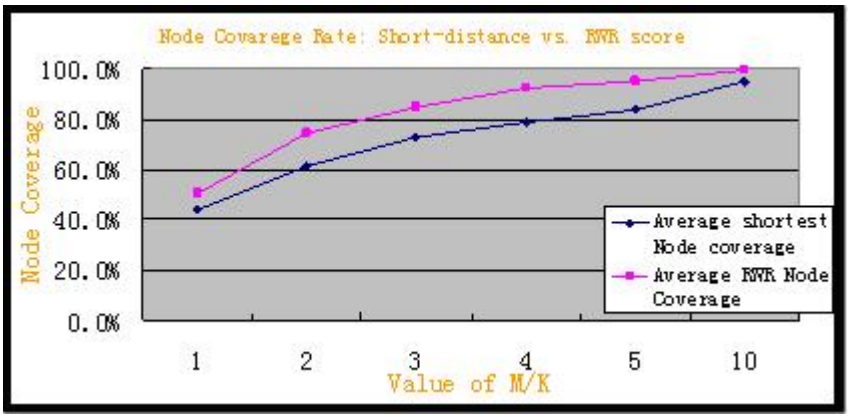

Figure 5.3 Comparison of Average short-distance Node Coverage and RWR Node Coverage Rate

node coverage rate is consistently higher than the short-distance node coverage rate. One possible explanation of the result is that the node elimination process uses the upper bound of the path betweenness which is closely related to the random walk approach.

5.2.1 and 5.2.2 show that we can generate an intermediate subgraph with only 200 nodes (less than 10% of the size of original graph), while retaining over 95% nodes with shortest distance to the query nodes and 95% nodes with highest RWR score to the query nodes. Therefore, from the perspectives of both short distance and random walk with restart, our proposed intermediate subgraph generation algorithm enables us to keep the most important nodes while eliminating most of the unimportant nodes.

5.2.3 Path Coverage Ratio

In this section, we try to evaluate the performance of our intermediate subgraph in keeping the shortest paths in the original graph. We first find the smallest number K of nodes that include the top T shortest paths. Then we try to generate the smallest the subgraph with size M to include all the nodes that cover that top T shortest paths. The node ratio K/M is reported.

Table 5.5 and Figure 5.4 illustrate the relationship between the top T shortest paths and the size of the intermediate subgraph G. Surprising to us, for the first top 15 nodes, we need to generate a subgraph of size of about 100 to 200. One possible reason is that we use the reciprocal

Table 5.5 Relationship between Number of Shortest Paths and the Size of G

Num of Shortest path	Pair(25,336)	Pair(268,1356)	Average
Top 5	4	199	71.6
Top 10	88	199	137.6
Top 15	93	229	160.4
Top 20	93	229	166.8
Top 30	163	229	188.4
Top 40	163	229	188.4
Top 50	163	229	198

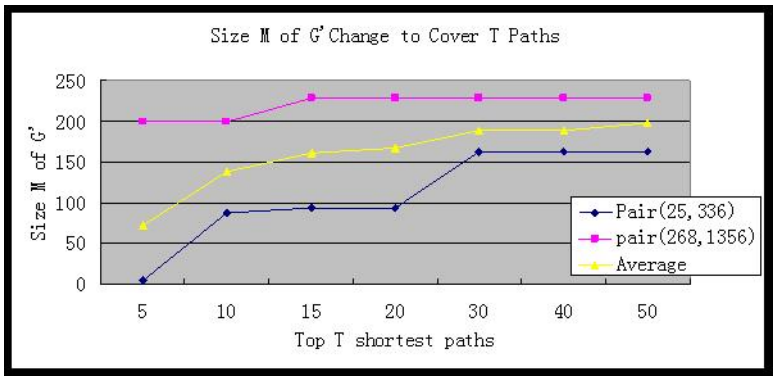

Figure 5.4 Relationship between Top T Shortest Paths and Size of G covers them

of the edge weight between two nodes to represent their distance, and this might cause some short paths containing nodes that have relative small node betweenness. The way chosen to represent the edge distance may have big influence on the size of the intermediate subgraph. However, as the number of shortest paths increase ($T > 30$), the size of the intermediate subgraph does not change much. Furthermore, the method of using only around 200 hundred (less than 10%) nodes from a large graph of about 4,000 nodes to keep most of the shortest paths is still attractive.

From the above analysis, the experimental results show that it is possible to generate a much smaller intermediate subgraph while retaining the most important paths and nodes.

5.3 Connection Subgraph Generation

In order to evaluate the performance of connection subgraph generation, we also use the important node coverage and the important path coverage to evaluate the connection subgraph. Since the time complexity of the candidate subgraph generation and connection subgraph computation is quite high, we are only able to find the connection subgraph with no more than 20 nodes.

5.3.1 Node Coverage Based on Shortest Distance to s and t

The first measure is the node coverage based on the shortest distance to the query nodes. We first compute the top K nodes which have the shortest distances to the two query nodes s and t. And we generate our connection subgraphs with b nodes. Again, we use the short-distance node ratio to see how many nodes with shortest paths to the query nodes are covered by the connection subgraph. The experiment was conducted by using randomly selected 10 pairs of query nodes. Similar to 5.2.1, the data is shown as:

Table 5.6 The Node Ratio Covered by Connection Subgraph

b	K=6	MK=8	K=10	Average
b=K	50.0%	40.0%	46.7%	45.6%
b=2K	66.7%	55.0%	80.0%	67.2%
Average	58.3%	47.5%	63.3%	56.4%

Table 5.6 shows that when b = K, the connection subgraph can capture nearly 50% percent of the nodes with shortest paths to the query nodes. In addition, as b increases to 2K, the percentage jumps to nearly 70%. This indicates that connection subgraph can indeed capture the nodes with short paths to query nodes. We also compared the connection subgraph with the intermediate subgraph of the same size to see the difference in short-distance node coverage.

Figure 5.5 illustrates that compared to the intermediate subgraph, the connection subgraph with the same size can achieve higher short-distance node coverage. It is not surprising. At the point where b = 8, the two percentages are very close. One possible explanation is that the sample size is not big enough. As the sample size increases, the connection subgraph should

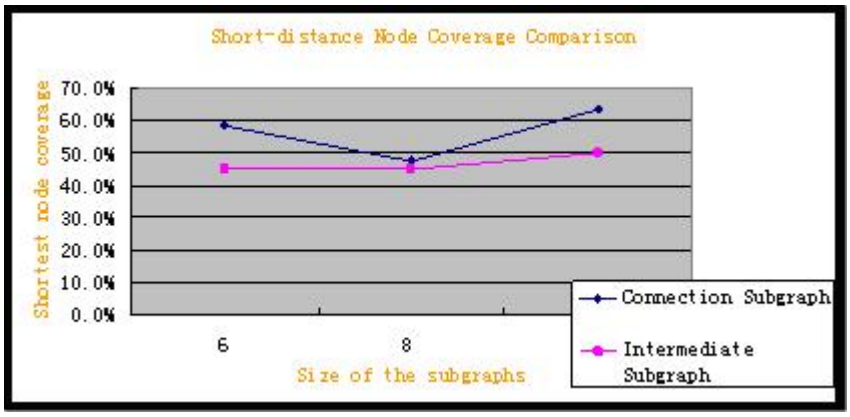

Figure 5.5 Connection Subgraph Achieves Higher short-distance Node Coverage

have stable higher short-distance node coverage.

5.3.2 Node Coverage Based on Random Walk with Restart (RWR)

Similar to 5.2.2, we first compute the top K nodes which have the highest RWR score to the two query nodes s and t. We then generate a connection subgraph with size b using connection subgraph generation algorithm. In order to generate an unbiased result, we randomly selected ten different pairs. The results of computing the average value are shown as follows:

Table 5.7 RWR Node Coverage for Connection Subgraph

b	K=6	MK=8	K=10	Average
b=K	66.7%	60.0%	60.0%	63.3%
b=2K	83.3%	80.0%	89.2%	86.3%
Average	75.0%	70.0%	74.6%	74.8%

From Table 5.7, we can see that when b = K, the connection subgraph captures 63.3% percent of the nodes with shortest paths to the query nodes. In addition, as b increases to 2K, the percentage jumps to 86.3%, indicating that connection subgraph can indeed capture the nodes with high RWR. This can probably attribute to the fact that graph betweenness is a random walk based approach. We also compared the connection subgraph with the intermediate subgraph with the same size to see the difference in RWR node coverage.

The results presented in Figure 5.6 shows that compared to the intermediate subgraph,

Table 5.8 Connection Subgraph Achieves Higher short-distance Node Coverage

Two subgraphs	Size=6	Size=8	MSize=10
Connection Subgraph	75.0%	70.0%	74.6%
Intermediate Subgraph	55.6%	50.0%	56.7%

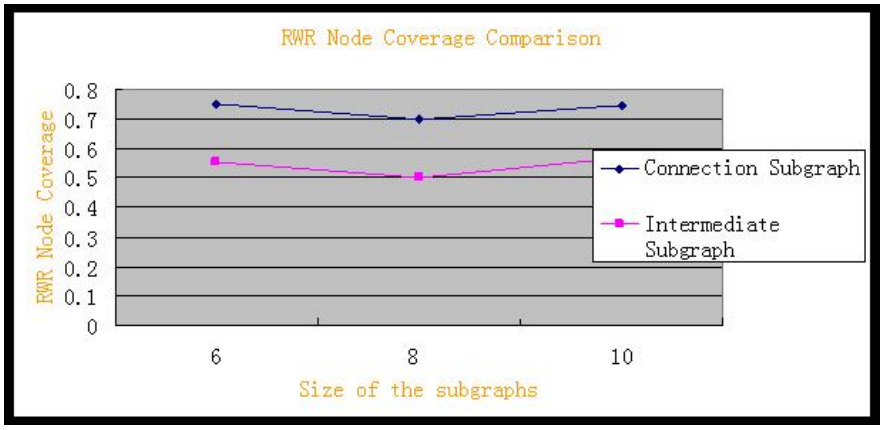

Figure 5.6 Connection Subgraph Achieves Higher RWR Node Coverage

the connection subgraph with the same size can achieve higher RWR node coverage. Again this result is possibly due to the fact that the graph betweenness is exactly computed in the connection subgraph generation process, while the intermediate subgraph is obtained through eliminating unimportant nodes thus it may still keep some unimportant nodes.

5.3.3 The Relationship between Threshold Value and the Size of the Original Graph

Lastly, we investigated the relationship between threshold value and the size of the intermediate subgraph. Since the size of the original graph is fixed, we want to set the size M of the intermediate subgraph to be a certain percentage of size N for Graph G. The data is shown below:

Table 5.9 and Figure 5.7 show that the threshold value changes dramatically as the size of M changes from 0.001N to 0.1N. For example, in Figure 5.9, the threshold value changes from 0.000925 to 1.14E-07 when the size of M increases by 100 times. Recall the physical meaning of the threshold value. When the threshold value is quite small, each node with a

Table 5.9 Threshold Value Change as M Changes

Size of G'	Max	Min	Average
M=0.001N	0.001849	4.32E-07	0.000925
M=0.005N	2.30E-04	1.29E-07	0.000115
M=0.01N	8.09E-05	5.82E-08	4.05E-05
M=0.05N	1.17E-06	1.89E-09	5.87E-07
M=0.1N	2.28E-07	3.28E-10	1.14E-07

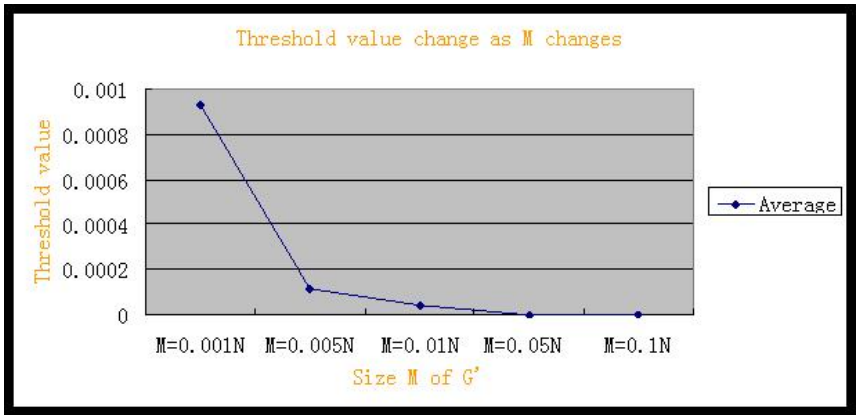

Figure 5.7 Threshold Value Change as M Changes

node betweenness smaller than the threshold value contributes very little to the relationship between the query nodes. This result validates our hypothesis.

5.4 Summary

In this chapter, we conduct experiments to evaluate our proposed approach. For the intermediate subgraph generation, we focus on the short-distance node coverage, RWR node coverage and the shortest path coverage. Our experiments show that the node elimination works quite well for the DBLP dataset, and a small intermediate subgraph with around 200 nodes can retain the most important nodes and edges. In addition, experiments on the connection subgraph show that connection subgraph can retain higher short-distance node coverage and RWR node coverage, compared to the intermediate subgraph with the same size. We also validated our hypothesis that a lot of nodes can be eliminated through the threshold value because the node betweenness for different nodes varies a lot.

CHAPTER 6. Summary and Discussion

6.1 Summary

Graphs have been playing a valuable role in numerous domains. Ever since 2000s, the social network has been developed rapidly and become indispensable in peoples life. Given two nodes A and B, in a social network learning how these two nodes relate with each other has a lot of applications.

The *Connection Subgraph Problem* [2] is intended to capture the most important relationship between the query nodes by using specified limited set of nodes. In order to meet this objective, one first needs to identify an appropriate goodness function. Unfortunately, very few discussions of such goodness functions have been found in the literature. Besides, due to the scalability issue, it is usually computationally too expensive to identify the connection subgraph for large graphs.

In this thesis, we have addressed this issue by finding a goodness function to measure the relationship between two query nodes in the graphs. Specially, we define two importance concepts of Path Betweenness and Graph Betweenness, we also designed algorithms to compute the graph betweenness for a given graph and query nodes.

6.2 Contributions

The main contributions of this thesis are as follows:

1. We define the path betweenness and graph betweenness. Path betweenness can be used to determine how important a path is between two query nodes. And the graph betweenness is the sum of the path betweenness of all the paths between two query nodes in the graph. Graph betweenness is defined on top of path betweenness, and has been proven to be a good

goodness function for the Connection Subgraph Problem.

2. Next, we proposed a two-stage framework for solving the Connection Subgraph Problem. The two-stage framework includes a node elimination process and an edge elimination process. The main ideas behind the framework is first to reduce a large graph to a much smaller intermediate one by removing those unimportant nodes, we then operate over the intermediate subgraph. Generally, this framework is not restricted to the Connection Subgraph Problem.

3. We have formulated the following set of algorithms:

(a) An algorithmic preprocess for the graph to make sure that each node in the graph has a non-loop path between the query nodes;

(b)The node elimination algorithm that computes the upper bound of the node betweenness for each node and use the threshold value to eliminate unimportant nodes.

(c) An algorithm which removes unimportant edges from the graph based on the threshold value.

(d) A candidate subgraph generation algorithm which enumerates all the possible connected subgraphs with a size of at most b from the intermediate subgraph.

(e) An algorithm that computes the graph betweenness given a graph and the query nodes.

4. Based on the above analysis, we have implemented SocialMiner, a system which can capture and display the relationship between nodes. Given an adjacent list represented graph, a threshold value, and a subgraph with a size constraint of b, the system will output an intermediate subgraph and display the connection subgraphs.

6.3 Future work

Besides being a significant extension of the current state of the art for relationship extraction and display, the work presented in this thesis provides an extensible framework on top of which numerous research threads and applications can be based. We outline some of the future work here:

• **Identify Important Paths between Nodes**: The path betweenness measure discussed in this paper can be used to identify important paths for some applications.

• **Compare Importance of Graphs to Query Nodes**: So far, not much work has been done to compare relationship between graphs. Graph betweenness presented in previous chapters can be used to compare the importance of graphs to query nodes.

• **Solve the Centerpiece Subgraph Problem (CEPS)**: Centerpiece Subgraph problem is a more general version of the Connection Subgraph Problem [18]. The major difference is that CEPS may have more than two query nodes whereas the CSP only has two nodes. Our preliminary work has shown the proposed framework should work for the Centerpiece Subgraph Problem. The remaining question is, in order to find the centerpiece subgraph faster, how to design the set of algorithms based on the framework.

• **Extend Connection Subgraph Problem in Directed graphs**: Connection Subgraph Problem is initially defined only for the undirected subgraph, because the electric current measure does not apply to the directed graphs. However, graph betweenness can be used as a goodness function for the directed graphs as well. Also, the two-stage framework can also apply in direct graphs. Yet, more study needs to be conducted in order to eliminate unimportant nodes in the directed graph, and design new algorithms to identify the connection subgraph.

• **Capturing Node Relationship on More Complicated Network**: Currently, most researches focus on graphs with only one type of nodes and one type of edges. How to capture the relationship between nodes on a network with multiple types of edges and multiple types of nodes has a lot of applications [17], and yet is an open problem. Can we use the path betweenness and graph betweenness to identify important paths and subgraphs in such complicated networks? And how can we apply them? Can we use the two-stage framework to capture the nodes relationship over those complicated graphs? Those questions still remain unsolved.

APPENDIX A. Symbols and Definitions

Symbol	Definition
$G(V, E)$	an undirected graph
V	set of nodes
E	set of edges
G'	intermediate subgraph of graph G
$deg(u)$	degree of u
$neighbor(u)$	degree of u
$e(u, v)$	edge weight of edge (u, v)
$P(u, v)$	probability from u to v
$PPath(s, v)$	path probability from s to v
$PB(s, t)_v$	node betweenness of v between s and t
$PB_{s,v_1,v_2,...,v_m,t}(s, t)$	path betweenness of path $s, v_1, v_2, \ldots, v_m, t$ between s and t
$GB(G)$	graph betweenness of graph G
$threshold/threshold\_value$	the threshold value

Table A.1 Symbols and Definitions

APPENDIX B. Connection Subgraph: A example

This is an example of connection subgraph with a budget 6 from the DBLP data set. There are two query nodes in this connection subgraph, *Jiawei Hang* and *H. V. Jagadish* who both are computer scientists in the area of data mining and machine learning. The connection subgraph caputres some important nodes and paths through which their academic ideas may propagate from one to the other.

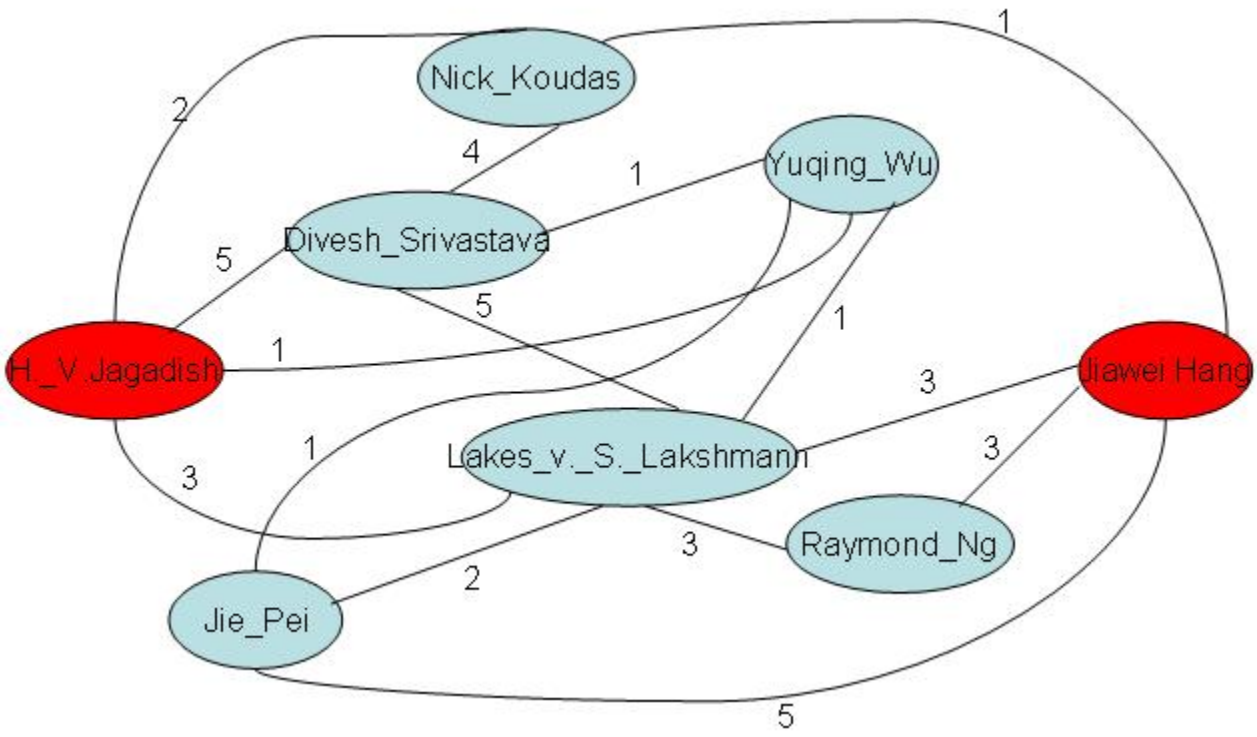

Figure B.1 A connection subgraph with 6 nodes from DBLP data set

BIBLIOGRAPHY

[1] L. C. Freeman. *Centrality in Social Networks: Conceptual Clarification*. Social Networks, 1(3): 215-239, 1979.

[2] C. Faloutsos, K.S. McCurley, and A. Tomkins. (2004). *Fast Discovery of Connection Subgraphs*. In KDD, 2004.

[3] L. Page. *PageRank: Bringing Order to the Web*. Stanford Digital Library Project, talk, August 18, 1997 (archived 2002).

[4] T. Haveliwala. *Topic-Sensitive PageRank*. Proceedings of the Eleventh International World Wide Web Conference (Honolulu, Hawaii), 2002.

[5] H. Tong, C. Faloutsos, and J.-Y. Pan. *Fast Random Walk with Restart and its Applications*. In ICDM: 613622, 2006.

[6] M.E.J. Newman. *A Measure of Betweenness Centrality Based on Random Walks*. Soc. Netw: 39-54, 27, 2005.

[7] P. Gerl. *Random Walks on Graphs*. Lecture Notes Math 1210, Springer: 285-303, 1986.

[8] P.G. Doyle and J.L. Snell. *Random Walks and Electric Networks*. Carus Mathematical Monographs 22, Mathematical Association of America, Washington, DC, 1984.

[9] L.Lovasz. *Random Walk on Graphs: A Survey*. Tech. Rep. Dept. Computer Science, Yale University, New Haven, Conn, 1991.

[10] J. Conrad. *Connections in Networks: Hardness of Feasibility versus Optimality.* Proceeding CPAIOR '07, Proceedings of the 4th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2007.

[11] B. Page and W. Motwani. *The PageRank Citation Ranking: Bringing Order to the Web.* Stanford University, Computer Science Department Technical Report, 1997.

[12] T. Haveliwala. *Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search.* IEEE Transactions on Knowledge and Data Engineering, 2003.

[13] J. Kleinberg. *Hubs, Authorities, and Communities.* Cornell University, 1999-12, Retrieved 2008-11-09.

[14] J. Kleinberg. *Authoritative Sources in A Hyperlinked Environment.* Journal of the ACM 46 (5): 604632, 1999.

[15] L. C. Freeman, S. P. Borgatti and D. R. White. *Centrality in Valued Graphs: A Measure of Betweenness Based on Network Flow.* Social Network Volume 13: 141-154, Issue 2, 1991.

[16] C. Faloutsos, K.S. McCurley and R. Tomkins. *Connection Subgraphs in Social Networks: Workshop on Link Analysis*, Counterterrorism, and Privacy. SIAM International Conference on Data Mining, 2004.

[17] C. Ramakrishnan, W. Milnor, M. Perry and A. Sheth. *Discovering Informative Connection Subgraphs in Multi-relational Graphs.* SIGKDD Explorations, 7(2), December 2005.

[18] H. Tong and C. Faloutsos. *Center-piece Subgraphs: Problem Definition and Fast Solutions*, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, 2006.

[19] H. Tong, C. Faloutsos and J-Y Pan. *Fast Random Walk with Restart and its Applications.* Proceedings of the Sixth International Conference on Data Mining: 613-622, December 18-22, 2006.

[20] H. Tong , C. Faloutsos , Y. Koren. *Fast Direction-Aware Proximity for Graph Mining*, Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August: 12-15, 2007.

[21] A. O. Acemoglu and A. ParandehGheibi. *Spread of (Mis) Information in Social Networks*, 2009.

[22] SIGMOD co-authorships dataset, http://www.informatik.uni-trier.de/ ley/db/. URL http://www.informatik.uni-trier.de/ ley/db/, 10 November, 2011.

[23] V. Krebs. *Uncloaking Terrorist Networks*. First Monday, Volume 7 Number 4 - 1, 2002

[24] V. Krebs. *Mapping Networks of Terrorist Cells*. VE Krebs - Connections, 2002.

[25] S. White and P. Smyth. *Algorithms for Estimating Relative Importance in Networks*. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C, August 24-27, 2003.

[26] L. Katz. *A New Status Index Derived from Sociometric Index*. Psychometrika, 39-43, 1953.

[27] K Stephenson. A. and Zelen, M., *Rethinking centrality: Methods and examples*. Social Networks 11, 137, 1989.

[28] R. Lempel , S. Moran *The stochastic approach for link-structure analysis (SALSA) and the TKC effect*. Computer Networks: The International Journal of Computer and Telecommunications Networking, v.33 n.1-6, p.387-401, 2000.

[29] A. Borodin, G. O. Roberts, J.S. Rosenthal, and P.Tsaparas. *Finding authorities and hubs from link structures on the world wide web*. In The Eleventh International World Wide Web Conference, pages 415429, 2001.

[30] G Jeh and J Widom. *Scaling Personalized Web Search* , In Proceedings of the Twelfth International World Wide Web Conference, 2003.

[31] H Chang, D. Cohn and A. McCallum *Creating Customized Authority Lists*, Proceedings of the Seventeenth International Conference on Machine Learning. Stanford, CA, 2000.

[32] E. Gansner and S. C. North. *An Open Graph Visualization System and its Applications to Software Engineering*. Software-Practice and Experience, 30:12031233, 1999.

[33] DBLP Computer Scientist database, http://www.informatik.uni-trier.de/ *ley*/db/. URL http://www.informatik.uni-trier.de/ ley/db/, 11 November, 2011.