University of Cincinnati

Date: 3/11/2020

I, Siqi Chen, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:

Community Detection in Large Directed Graphs

Student's name: **Siqi Chen**

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.

Committee member: Gowtham Atluri, Ph.D.

Committee member: Tesfaye Mersha, Ph.D.

36683

Community Detection in Large Directed Graphs

A thesis submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

Master of Science

in the Department Electrical Engineering and Computer Science
of the College of Engineering and Applied Sciences
by

Siqi Chen

Thesis Advisor & Committee Chair: Raj Bhatnagar, Ph.D.

Abstract

Community detection is an important topic in graph mining and is of great interest in various fields and tremendous real-world applications. Finding and analyzing community structure can help us understand the network components, their relationship, and the functionality of the underlying complex networks. A large number of networks representing real systems are directed. The existing community detection methods for directed graphs suffer from one or more drawbacks when applied to real-world datasets. For example, many common methods ignore edge directions and apply methods that were developed for undirected ones. Moreover, most of the methods are not easily scalable to very large graph datasets.

In this research, we have developed a new scalable Map-Reduce algorithm to discover PageRank based hierarchical communities in directed graphs. One major difference of our approach is that we seek to find a community centered around the core node with an upper and a lower hierarchical structure along the core node's directional edges. Nodes within our generated hierarchical community display similar importance ratings compared to the nodes outside the community. We have demonstrated in this thesis that our algorithm can find hierarchical communities by applying it on several real-world datasets with different sizes. We have also evaluated the effect of the PageRank value of the core node and the PageRank

threshold on the obtained communities in terms of the community coefficient and community size. Our validation experiments have proven that the communities generated from our method show strong connections among the nodes inside the community. We believe that this method is particularly suitable for detecting communities in the directed graphs with the structure of the flow hierarchy such as citation networks, trust networks, and defeat networks.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Raj Bhatnagar for his mentoring throughout my graduate study. His immense knowledge in the area and patience towards the development of the thesis made this a thoughtful and rewarding journey.

I would like to thank my defense committee members, Dr. Gowtham Atluri and Dr. Tesfaye B. Mersha for their valuable time and effort that have contributed to this work. I wish to extend my special thanks to Dr. Mersha for providing me the opportunity to work as a research assistant at the Cincinnati Children's Hospital Medical Center.

Last but not the least, I would like to thank all my friends and family for their unconditional support and encouragement in life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

A graph is a structure formed by a set of nodes (or vertices) and a set of edges (or links) that are connections between pairs of nodes. It is an extremely useful tool that has been used to represent a wide variety of systems in different domains. Many complex systems, including social systems like friendship networks and collaboration networks, biological systems like protein-protein interaction, genetic interaction and metabolic networks, technological systems like hyperlink structures on the World Wide Web, and physical systems, can be modeled by graphs or networks. Note that through this thesis, the terms 'graph' and 'network' will be used interchangeably.

The majority of real-world networks are not randomly organized; they have been shown to display meaningful patterns or relations that reveal the underlying structural properties of the systems. For example, a graph representing Facebook friendship typically contains highly connected regions because the friends of an individual are also very likely to be friends and thus are connected as well. This kind of pattern or structure is often referred to as clustering or community. More formally, a cluster or community in a network is defined as a group of connected nodes sharing one or more common measurable properties. In the example of

social media network presented above, a cluster is a group of tightly connected individuals who interact more frequently with members within the group than those outside the group; in a protein-protein interaction network, a cluster can be a set of proteins that demonstrate greater similarities among proteins in the same cluster than in different clusters [1]; in a web networks, a cluster can be a collection of web pages that are highly topically related [2].

Community detection is a very important topic. Finding community structures in networks can help us understand and visualize the structure of the networks, network components, and their relationships in the systems, as it provides a simplified representation of the complex interactions. Therefore, developing more efficient and accurate clustering strategies will help researchers better analyze the network and reveal interesting patterns in many domains.

1.2 Community Detection in Directed Graphs

A large number of real-world graph data are intrinsically directed and the directedness of edges is an essential feature of the system. Here we list examples of directed graphs in several applications:

- Trust network: With the growth of online communication and services, the topic of trust has become increasingly important in the social media, network security community, and e-commerce industry [3]. Trust information plays an essential role in facilitating interactions, transactions and collaborations among the network members. It also helps users make decisions, filter information and develop communities with respect to whom to trust and why [4]. In a trust network of an online retail website, for example, a node represents a person who can be either a customer or a vendor, and an edge from u to v represents that the customer u trusts the vendor v.

- Citation network: Since the analysis of citation network was initially started by Garfield et al. in 1964 [5], it has become a useful tool in complement to the traditional citation analysis. Finding the citation patterns of authors can help researchers better understand the relationship between disciplines, the collaboration of scientists and create more accurate scientific impact measures. A typical citation network consists of documents that reference each other. Each node represents a scientific paper and a directed edge from u to v denotes that the paper u cites the paper v.

- Defeat network: As a type of interaction network that models the result of games or competitions, defeat network can be used to analyze the players' performance, cooperation level of the team and create new ranking methods. Network analysis has been conducted on different professional sports like basketball [6], football [7], baseball [8] and so on. For example, in a defeat network among chess players giving the chess match outcomes, each node is a chess player, and a directed edge from u to v represents in a match in which the player u beat the player v.

It can be clearly seen that finding clusters in directed networks is significantly important in many domains and it is meaningful to incorporate information on the edge directionality during the clustering process. Various approaches have been proposed to discover communities in graphs. The most commonly used community detection/clustering algorithms include hierarchical clustering [9, 10], minimum-cut method [11], spectral clustering [12] and so on. However, most of these approaches on clustering or community detection of graphs have been focused on undirected networks. Community detection in directed networks is considered to be a more challenging task as compared to undirected networks [13] due to the following challenges:

(1) Several graph concepts that are developed for undirected graphs cannot be adapted for directed graphs as edge directionality is considered as an inherent network characteristic. Figure 1.1 shows a simple example of directed graphs. If one applies the traditional density-based clustering definition to this graph, these 6 nodes form a cluster according to their similar edge densities. However, due to the existence of directed edges, this clustering method cannot capture more sophisticated similarity that is not reflected by the edge density. In this example, node 1 is connected to every node only in one direction. If the task is to construct a community around node 4, node 1 should be considered outside the community since it does not have any incoming links from the other 5 nodes. As we can see from this example, edge directionality introduces an extra complication when dealing with the problem of community discovery. Moreover, since the nature of relationships captured by the directed edges is fundamentally different from that for undirected graphs, a more precise definition of clustering needs to be defined in various domains and contexts.

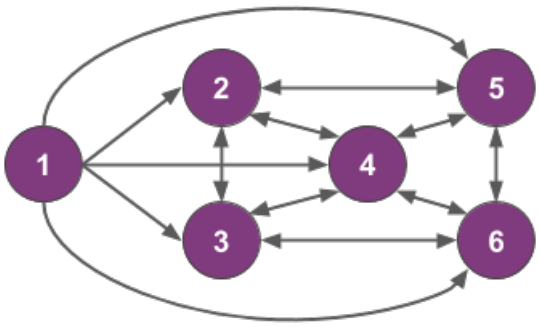


Figure 1.1: Community in directed graph

(2) Well-developed clustering methods for undirected networks cannot be easily extended to directed graphs. Since a large number of methods for detecting communities in undirected networks have been proposed, researchers

have started to explore the possibility of employing the clustering methods designed for undirected networks to find communities in directed networks. In the undirected setting, the symmetric relationship can be represented by symmetric matrices, such as the adjacency matrix and Laplacian matrix. In the directed relationship, these matrices are asymmetric, which makes it hard to generalize the clustering methods for directed graphs. For example, one of the most popular community detection algorithms is spectral clustering, in which a Laplacian matrix is used as the main tool. However, the non-symmetric matrices of directed graphs do not have a spectral decomposition, indicating that there does not necessarily exist an orthonormal basis of eigenvectors [14]. This establishes an obstacle for generalizations to directed graphs.

1.3 Motivation of This Work

Existing work related to clustering or community detection in directed graphs will be reviewed in Chapter 2. As we will see in the next chapter, most of them possess one or more of the following drawbacks:

(1) Most previous methods ignore edge directions and assume symmetric interactions. Using the World Wide Web (WWW) network as an example, even though hyperlinks are directed (i.e., a hyperlink from page A to page B does not guarantee a hyperlink from B back to A), one may assume that a hyperlink between two pages implies that they are content-related to the same topic. However, in many other real-world systems, the relationships between elements are not reciprocal. For example, suppose G is a directed graph that represents the outcome of a tournament between players. A directed edge (i, j) can be interpreted as a competition between player i and player j such that i defeats j. If we eliminate directionality from edges, we

automatically convert the edge's direction into a mutual relation. In this case, an additional edge (j, i) is introduced into the graph which does not exist in the graph originally. On the other hand, an edge from player i to player j may not necessarily represent the similarity between i and j when comes to the question of clustering. Player i may also defeat many other high-level players, while player j lose all the competitions in which player j is involved. In such a case, payer i and player j cannot be clustered into a group of players at a similar performance level.

(2) Another drawback of the existing methods is the poor scalability. Due to the growing amount of real-world network data produced with the help of new technologies, the scalability has been considered as an important factor in designing and evaluating graph clustering algorithms. Many social networks can be huge, with millions of users and hundreds of millions of connections. Most of the existing methods for finding the clusters in directed graphs are sequential algorithms, which are not easily scalable to very large graphs. In spectral clustering algorithms, for example, when the input graph is very large, the eigenvalue and eigenvector computation can be very time-consuming [15]. Thus there is a need for developing scalable graph algorithms to discover communities in very large networks.

(3) None of the existing approaches construct the strongly connected community for each individual node in the graph. In some directed systems, there exists the structure of flow hierarchy that can be defined as a layering of the nodes in which the direction of the edges follows a global flow. For example, in a defeat network, players can be grouped into different performance levels based on the outcome of the games they involved. Edges point from players at higher levels towards the players at lower levels, indicating the former group of players showed higher performance in the game. The

flow hierarchy is an importance characteristic in this type of graphs and should also be considered in the community detection process.

1.4 Proposed Approach

In this thesis, we have developed a new scalable Map-Reduce algorithm for discovering PageRank based hierarchical communities around core nodes in directed graphs, especially for these graphs with the structure of flow hierarchy, such as trust networks, citation networks and defeat networks. In our approach, the Map-Reduce paradigm has been employed for designing the much more scalable clustering method than the existing ones. The desired properties of clustering are to obtain the hierarchical structure around a core node such that

(1) The core node is located in the center of the hierarchical community where the nodes in upper levels can reach the core node, and nodes in lower levels can be reached by the core node.

(2) Nodes within the community have similar importance compared to the nodes outside the community.

For constructing hierarchical structures, we have extracted the upper-level and lower-level hierarchical subgraphs along directional edges of core nodes and then merged them into the final communities centered around the core nodes. To determine the importance of a node in a graph, we have selected PageRank as the ranking algorithm for two reasons. Firstly, it is a metric of global importance that computes a score for each node in the graph by utilizing the probability propagation of random surfers. Secondly, it is a computationally simple yet effective way that takes into account the edge direction. Nodes with low importance will correspondingly have low PageRank values.

Figure 1.2 shows an illustration of the community construction, where a nine-level hierarchical community around a core node is built by merging the forward

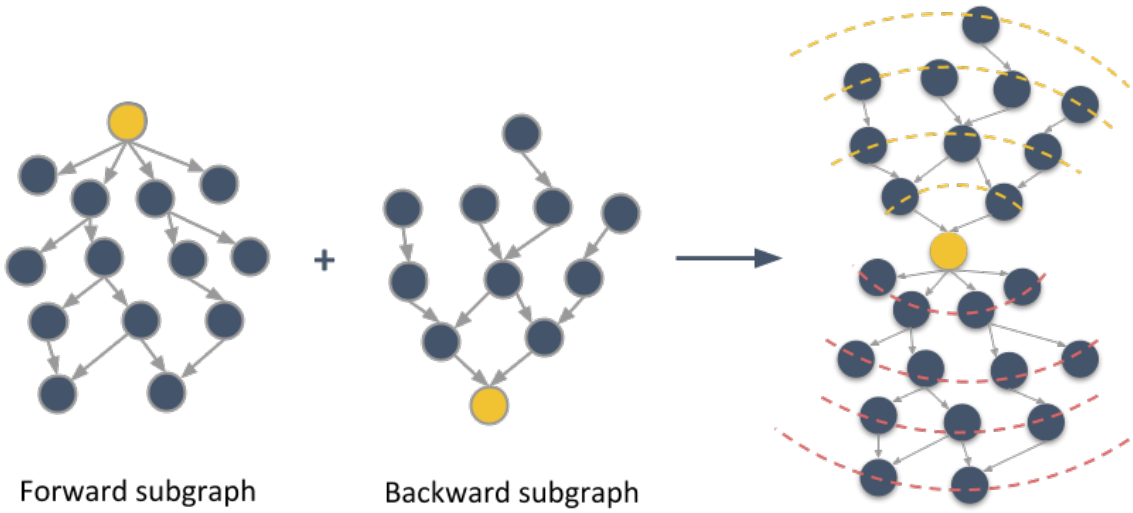


Figure 1.2: Generation of a nine-level community around a core node (in yellow) by combining a five-level forward subgraph with a five-level backward subgraph

and backward five-level subgraphs. The initial forward and backward subgraphs of a core node can be generated separately by growing the directed branches along the core node. In order to build forward high-level subgraphs, we perform a end-to-end concatenation of two short subgraphs iteratively. The backward subgraphs are generated in a similar way except that all edge directions of the input graph are reversed. As we repeatedly construct the forward and backward subgraphs, we further prune the subgraphs based on the node importance. If the PageRank value of a node is below a pre-specified threshold we discard the node from the community. Finally, the truncated forward and backward subgraphs are merged into one cluster around each core. As shown in Figure 1.2, the resulting community is a hierarchical subgraph around a core node with upper layers pointing to it and lower layers pointing from it.

1.5 Overview of Chapters

The rest of this thesis is structured as follows. In Chapter 2, the related research work on graph clustering and community detection in directed graphs along with their advantages and limitations has been reviewed. In Chapter 3, we have described our Map-Reduce based algorithm for discovering hierarchical community structures in directed graphs. The approach as well as pseudo-code have been explained with illustrative examples. In Chapter 4, experiments on real-world datasets and the result analysis have been provided to evaluate our developed approach. In the last chapters, we have concluded the thesis and discussed the possible future work.

Chapter 2

Related Work

2.1 Overview

In this chapter, we present various algorithms in discovering communities for. We first briefly go through some traditional approaches for undirected graphs and then discuss the common approaches for directed graphs. We also briefly discuss the advantages and disadvantages of these existing algorithms as well as the difference from our proposed approach for solving the problem.

2.2 Community Detection in Directed Graphs

2.2.1 Transforming Directed Graphs to Undirected Graphs

The simplest and commonest way to handle directed graphs is to ignore the edge directionality and treat them as undirected ones. After this simple transformation, a large number of clustering methods that have been proposed for undirected graphs can be applied to extract the community structure. However, as we have discussed in Section 1.2, most real-world networks have semantically meaningful directions that need to be taken into account to appropriately understand the system as a whole.

Another way to handle directed graph clustering is to convert directed graphs into undirected ones with the informative content of the direction being maintained, which is also known as graph symmetrization. Then the transformed undirected graph can be clustered using any existing state-of-the-art clustering methods without any modification. Lai et al. [16] proposed a transformation method that maps graphs into Euclidean space, as shown in Figure 2.1. This method uses PageRank induced random walk to obtain a new undirected representation of the original graph, which incorporates the information of edge directions into the weights of edges of the undirected network. The basic idea of this method is to define a directed version of the Laplacian matrix called directed PageRank combinatorial Laplacian (L^{PRd})

$$L^{PRd} = \Pi - W = \Pi - \frac{\Pi P + P^T \Pi}{2}, \tag{2.2.1}$$

where P is the PageRank random walk transition matrix and Π is a diagonal matrix of which diagonal is the probability of staying on each node in the stationary state. The symmetric matrix W can be considered as the adjacency matrix of a new undirected network and the induced network can be mapped into the Euclidean space. The authors also proved that information about edge directions is incorporated into weights of the new graph.

Similarly, Satuluri et al. [17] also discussed the random walk based symmetrization, where the directed normalized cut of the original networks will be equal to the normalized cut of the transformed undirected network [18]. In addition, the authors also explored several other ways of symmetrizing a directed graph into an undirected one, while information on edge directionality is incorporated into the edge weight of the transformed graph. Bibliometric symmetrization combines both the bibliographic coupling matrix (AA^T) and the co-citation strength matrix

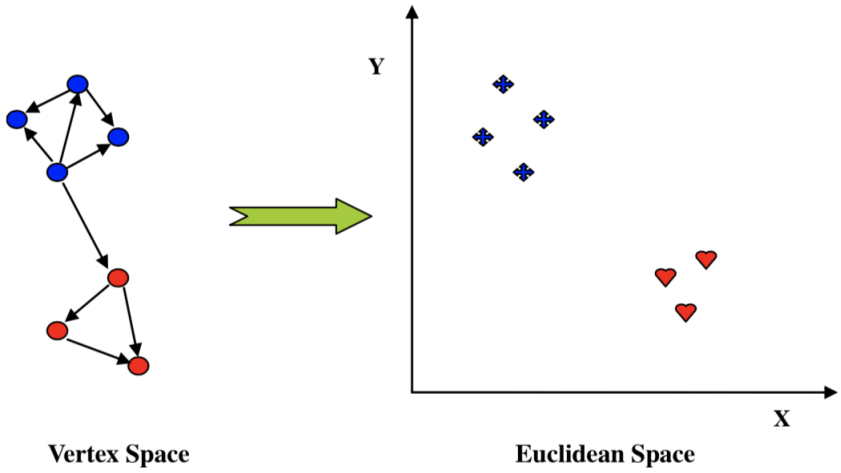


Figure 2.1: Illustration of the transformation of all the nodes in a directed network into the points in the Euclidean space, proposed by Lai et al. [16].

$(A^T A)$ (L^{PRd})

$$U = AA^T + A^T A, \tag{2.2.2}$$

where AA^T captures the common out-links between each pair of nodes and $A^T A$ captures the common in-links between each pair of nodes. Therefore, bibliometric symmetrization ensures that edges will present between similar nodes even edges are absent in the original graph. Furthermore, considering the fact that in many real-world networks there exist a few nodes with very high degree compared to the majority of the nodes [19], the similarity score contributed by each node should be normalized according to its in- and out-degree. Therefore, the authors further proposed degree-discounted symmetrization (U_d)

$$U_d = B + C,$$

$$B = D_{out}^{-\alpha} A D_{in}^{-\beta} A^T D_{out}^{-\alpha} \text{ and } C = D_{in}^{-\alpha} A D_{out}^{-\beta} A^T D_{in}^{-\alpha},$$

where A is the adjacecy matrix, D_{in} and D_{out} are the diagonal matrices of in-degree and out-degree respectively, and α and β are the discounting parameters.

2.2.2 Extending Clustering Methods to Directed Graphs

The methods discussed in the previous section can largely benefit from the existing well-developed approaches for the undirected graph. However, one drawback of these methods is that since edge directionality is considered as an inherent property of the network, any transformation techniques may not be able to completely retain the information on edge directinoality. To address this problem, many studies have also tended to generalize or extend the clustering methods to directed graphs without changing the structure of the original graph.

PageRank Based Methods

PageRank and random walks have been employed in many clustering algorithms for undirected graphs [20–25]. Since they traditionally have been developed and studied in directed web graphs, it is natural to study if they can be generalized to detect clusters in a directed graph. Avrachenkov et al. [26] proposed a PageRank based clustering (PRC) algorithm for hypertext document collections that are represented by directed graphs. This method is consist of two steps. The first step is to determine a list of core nodes the nodes in the graph according to a node ranking measure such as PageRank. Then these selected nodes are assigned to different clusters using Personalized PageRank vector. In contrast to global clustering of the entire graph, Andersen et al. [14] proposed a local partitioning method that finds a set nodes near a specified seed node by examining only a small portion of the input directed graph, using personalized PageRank vectors. The authors proved that by sorting the nodes of the graph according to the ratio of the entries in the Personalized PageRank vector and the global PageRank vector, this method is able to detect a small set of local cluster efficiently.

Modularity Based Methods

Modularity is a objective metric that quantifies the quality of partitioning, introduced by Newman and Girvan [27] in 2004. Modularity Q is defined as $Q =$ (fraction of edges within communities) - (expected fraction of edges in a random graph), which is the fraction of edges that fall within the given communities minus the expected number of edges within the same groups for a random graph with the same node degree distribution as the given graph [27]. Mathematically, modularity Q is given by

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(i,j), \tag{2.2.3}$$

where A is the adjacency matrix, $m = \frac{1}{2} \sum_{ij} A_{ij}$ is the total number of edges in the graph, $k_i = \sum_j A_{ij}$ is the link degree of node i, and $\delta(i,j)$ is 1 if node i and j belong to the same community and 0 otherwise. Modularity optimization method has become an well-established clustering approach to extract the community structure in undirected graphs [27, 28]. Several studies proposed the extensions of modularity optimization to directed cases. Leicht and Newman [29] adapted for the notion of modularity for directed networks by considering the in-degree and out-degree of the nodes, and the directed modularity is defined as

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i^{in} k_j^{out}}{2m}] \delta(i,j), \tag{2.2.4}$$

where k_i^{in} and k_j^{out} represent the in-degree and out-degree of node i and j.

Chapter 3

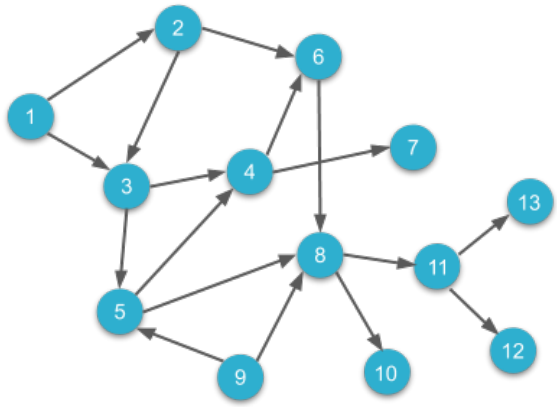
Community Detection in Directed Graphs

3.1 Introduction

In this chapter, we have proposed a Map-Reduce based method for detecting communities in directed graphs. This method uses the PageRank algorithm to rank nodes and extracts the hierarchical community for each node in the graph. An intuitive way to define a community around a node is to find a set of nodes that are reachable by the core node along directional edges and have a similar importance as that of the core node. To extract the hierarchical structures, our approach generates the forward and backward directed subgraphs for each node iteratively. To compute the importance ranking of nodes in the graph, we have employed the PageRank algorithm as a global criterion of importance to determine whether a node belongs to the same cluster of the core node. In the following sections, we have discussed our clustering method in detail.

3.2 Data Representation and Preparation

Since directed graphs represent asymmetric relationships, there is a directional flow in their edges. Therefore, the edge set of a directed graph is represented as ordered pairs. For example, a sample directed graph can be represented in tabular format as shown in Figure 3.1.



Source	Target
1	2
1	3
2	3
2	6
3	4
3	5
4	6
4	7
5	4
5	8
6	8
8	10
8	11
9	5
9	8
11	12
11	13

Figure 3.1: Directed graph and its tabular representation

Before constructing communities around core nodes, we need a computationally simple and effective metric to rank nodes in directed graphs. PageRank algorithm, as a global measure of importance, is used to identify the importance of nodes in the connected graph. For example, the PageRank algorithm is commonly used in trust networks and it computes a score for each user in the network that represents the average opinion of the whole community about that user. The PageRank of a node X from a directed graph is defined as

$$PR(X) = \frac{1-d}{N} + d\sum_{i=1}^{k}\frac{PR(X_i)}{C(X_i)}$$

, where N is the total number of nodes in the graph, X_i is a node that points to X, d is the damping factor, and $C(X_i)$ is the number of outgoing links of node X_i.

3.3 Our Approach

In our proposed clustering algorithm, the hierarchical community around a core node is constructed in two stages. The first stage is to grow the upper-level and lower-level directed subgraphs along the core node, and the second stage is to combine them into a community centered at the core node. The construction of directed subgraphs for both upper and lower levels can be achieved using our three-phase Map-Reduce algorithm. The entire workflow of the first stage is shown in Figure 3.2. Note that we reverse the edge direction in the input graph when constructing upper-level hierarchical subgraphs. This additional data pre-processing step is required to construct the backward subgraph where all the nodes can reach the core node directly or indirectly via one or more directed paths.

As shown in Figure 3.2, the first phase containing one Map-Reduce step is used to enumerate the incoming and outgoing nodes of each node in the graph. Next, another Map-Reduce step is used to generate three-level subgraphs from the lists of incoming and outgoing nodes. In the last phase, subgraphs with higher levels are constructed iteratively from the subgraphs with lower levels. For example, building five-level subgraphs requires two Map-Reduce steps in this phase. Ideally, one can easily construct longer subgraphs (e.g., nine-level, seventeen-level ...) by feeding the output from the third phase back to its own input iteratively.

In the following sections, we will use the example graph in Figure 3.1 as input data to illustrate the three-phase Map-Reduce approach for constructing five-level subgraphs. After obtaining the directed subgraph, the hierarchical community around the core node can be generated by simply combining the forward and backward subgraphs.

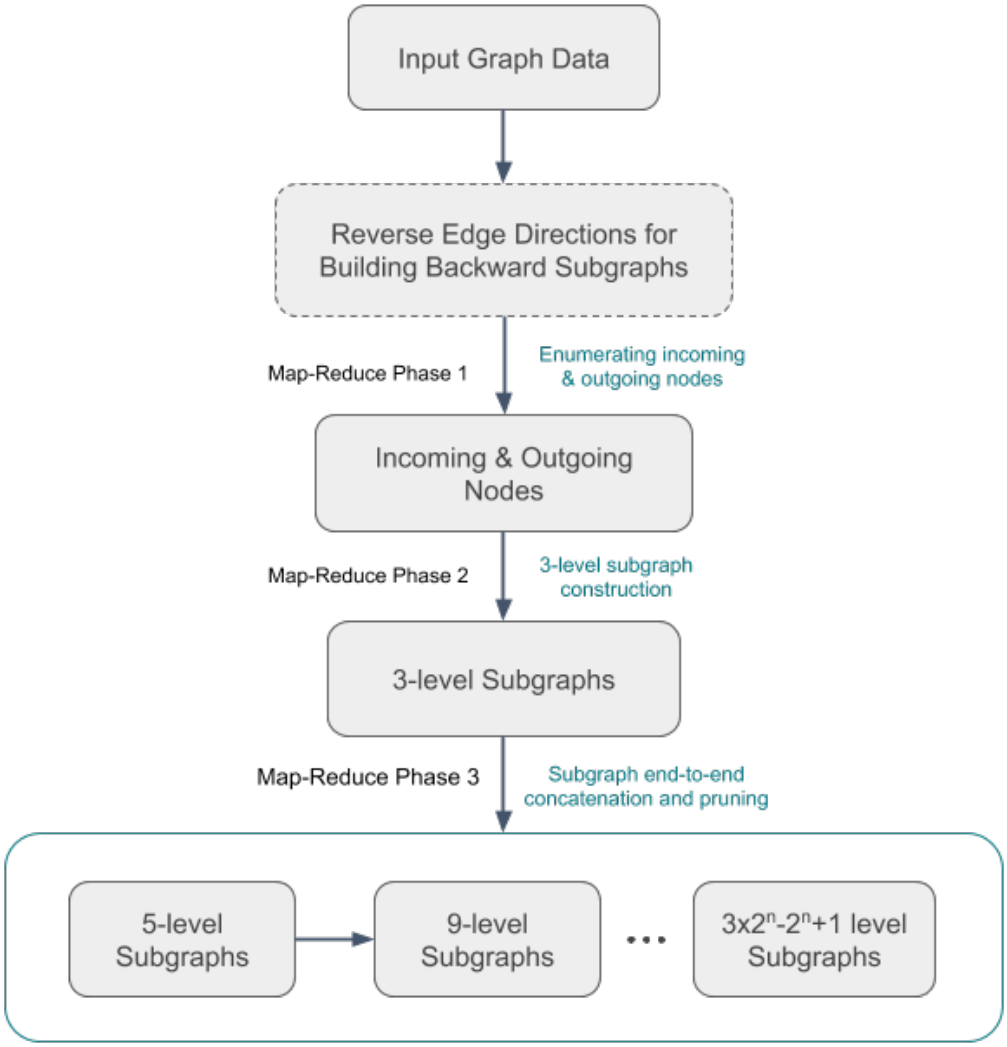


Figure 3.2: Flowchart of the three-phase Map-Reduce algorithm: construction of upper-level and lower-level directed subgraphs in directed graphs

3.3.1 Finding Incoming and Outgoing Nodes

In the first phase, a Map-Reduce job is required to find all outgoing and incoming nodes for each node in the graph. The outgoing nodes of a node u can be easily obtained by enumerating all the edges starting from node u. Similarly, the incoming nodes of node u can be obtained by enumerating all the edges ending at node u. The procedure performed by the Map-Reduce job is as follows:

- Mapper #1: For each node pair (u, v), emit two key/value pairs - the original node pair representing the out-link from u and the reversed node pair representing the in-link to u, in the form of (u, v) and (v, $\#u$), respectively.

- Reducer #1: For each key node, collect its incoming and outgoing nodes.

3.3.2 Building Three-Level Subgraphs

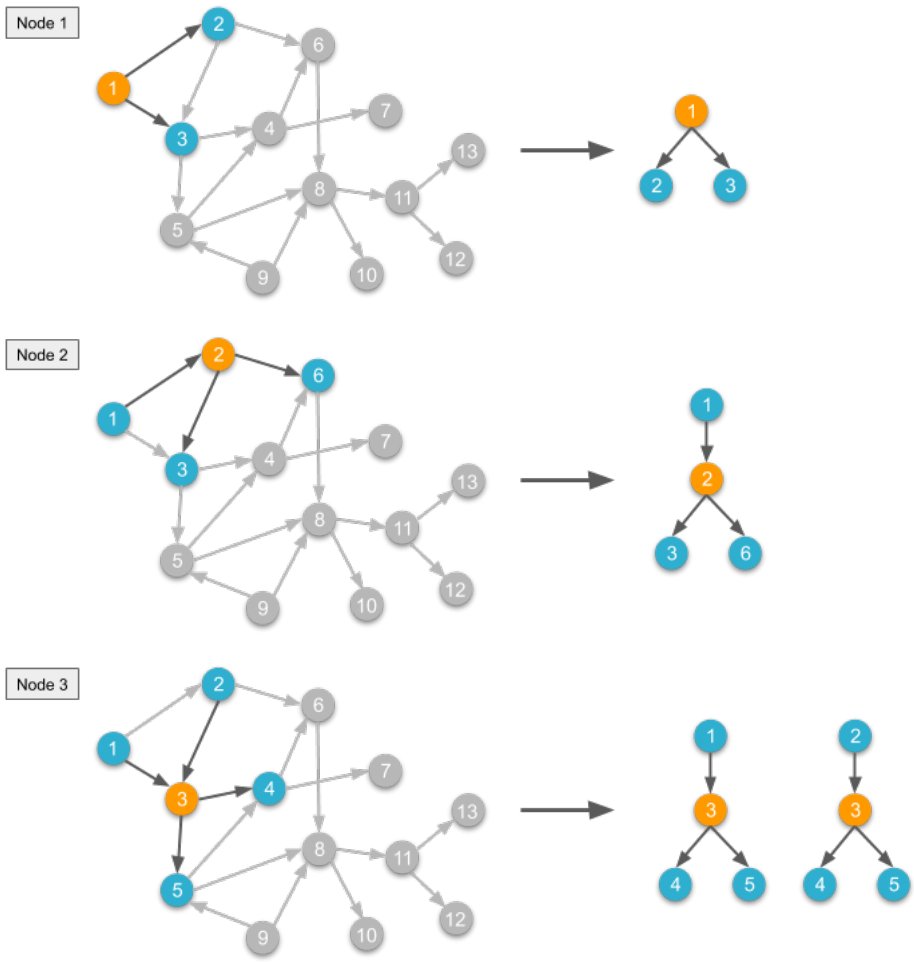


Figure 3.3: Mapper #2: generation of three-level branches from nodes 1, 2 and 3

From the list of outgoing and incoming nodes of a node, we can generate three-level subgraphs rooted at each incoming node in a Map-Reduce job as follows:

- Mapper #2: For each key/value pair (key-node, (outgoing-nodes, incoming-nodes)), generate new key/value pairs where key is a node from the incoming

node list, and value is a two-hop path originating from the key node to outgoing nodes.

- Reducer #2: For each key node, merge all the two-hop paths to construct the three-level subgraphs.

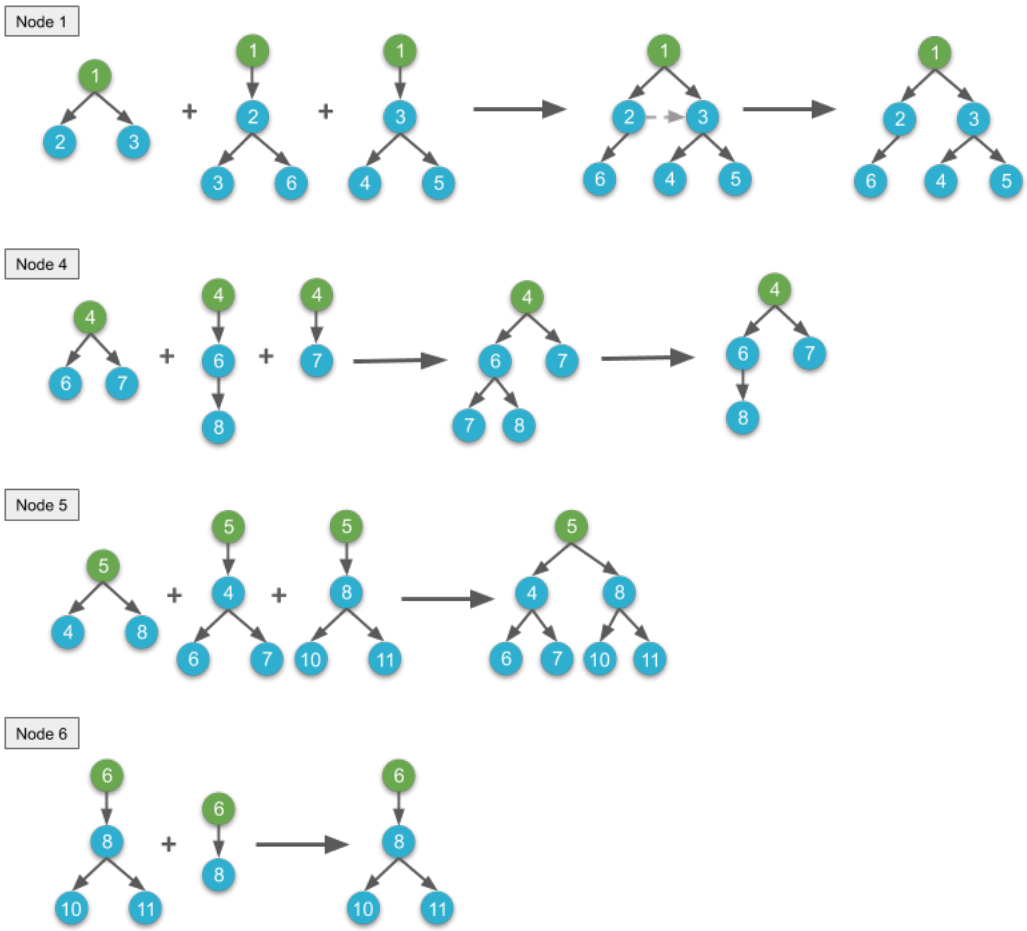


Figure 3.4: Reducer #2: generation of the two-hop paths from nodes 1, 4, 5 and 6

Given incoming and outgoing nodes of a pivot node, we generate all the two-hop paths starting from each incoming node. For example, Figure 3.3 highlights the incoming and outgoing nodes of nodes 1, 2 and 3. Since node 1 has no incoming nodes, the one-hop path starting from the core node 1 will be preserved. Node 2

has one incoming node and node 3 has two incoming nodes, therefore one two-hop path and two two-hop paths will be generated for nodes 2 and 3, respectively.

In the reducer, two-hop paths with the same root node are merged into a three-level subgraph. Figure 3.4 shows the merging process for the root nodes 1, 4, 5, and 6. To maintain the hierarchical structure of the community, if node u appears in more than one level, e.g., levels i and j ($i < j$), node u at the deeper level j and its descendants should be removed. For example, after merging the three paths rooted at node 1, the inter-level edge ($2 \rightarrow 3$) in the resulting three-level subgraph will be removed.

3.3.3 Building Higher-level Subgraphs

By combining multiple three-level subgraphs through an end-to-end concatenation, a five-level subgraph can be constructed. Similarly, a nine-level subgraph can be created from multiple five-level subgraphs. Ideally, we can construct a subgraph with $(3 \times 2^n - 2^n + 1)$ levels through n iterations. To make this approach scalable, every iteration is consist of the following one Map-Reduce step and an additional Reduce step (using the construction of five-level communities as an example):

- Mapper #3: For each three-level subgraph, decompose it into smaller paths by their leaf nodes.
- Reducer #3: Generate five-level branches via the end-to-end concatenation of these three-level paths and prune them based on the PageRank threshold value.
- Reducer #4: Merging all the five-level branches with the original three-level subgraph to five-subgraphs.

In order to perform the end-to-end concatenation in key/value pair fashion, for each leaf node in the three-level subgraph, a key/value pair with a leaf node as

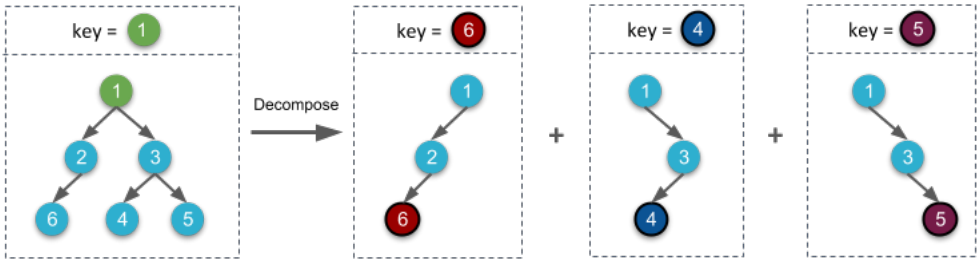


Figure 3.5: Mapper #3: decomposition of the three-level subgraph of node 1

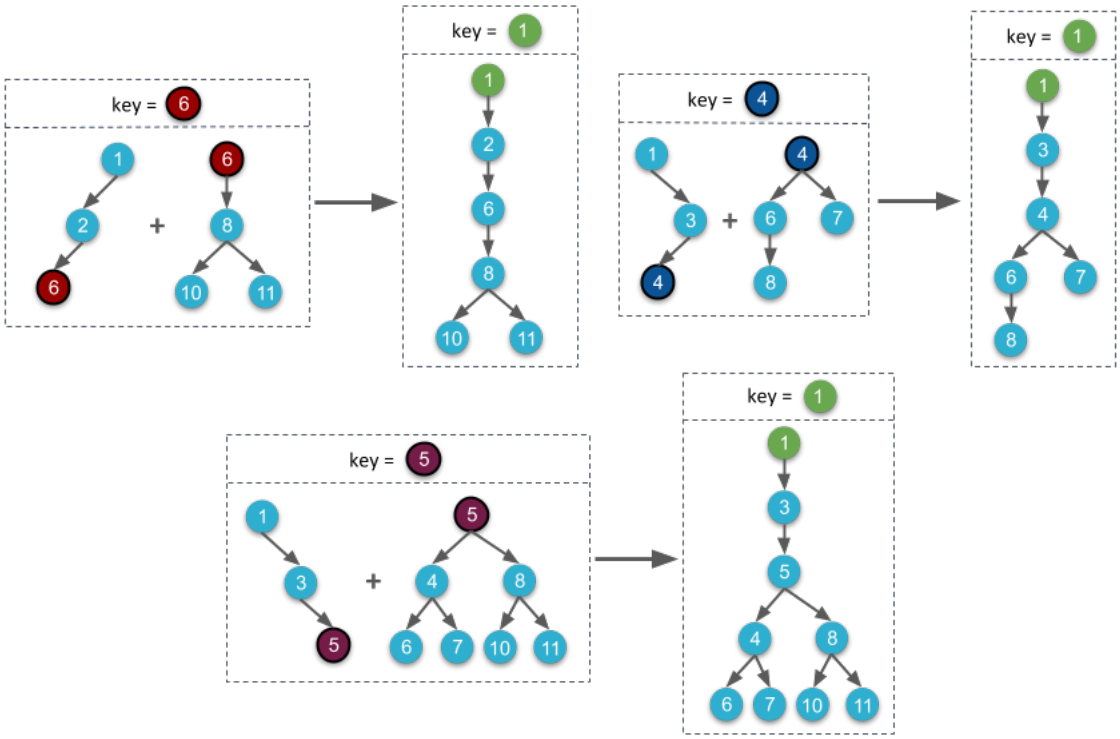


Figure 3.6: Reducer #3: generation of five-level branches of node 1

key and a three-level path from the root node down to the leaf node as value is generated. Figure 3.5 demonstrates the decomposition process of a three-level subgraph of node 1 generated in the previous phase. Since this three-level subgraph has three leaf nodes 6, 4 and 5, three root-to-leaf paths with the leaf node as key are produced.

Next, these root-to-leaf paths are combined with the original three-level subgraph that shares the same key node in Reducer #3. As shown in Figure 3.6,

the three root-to-leaf paths of node 1 are merged with the three-level subgraphs of nodes 6, 4 and 5, respectively, and eventually form three five-level branches rooted at node 1. Before the final merging step, these branches are further refined based on the PageRank value of the root node. In order to build a subgraph where all nodes have comparable PageRank values to that of the root node, we can remove these nodes that have PageRank values below a certain value by introducing a pre-specified PageRank threshold k (between 0 to 1). For example, with a threshold $k = 0.8$, we define that the PageRank value of any node in the subgraph should be equal to or larger than 80% of that of the root node.

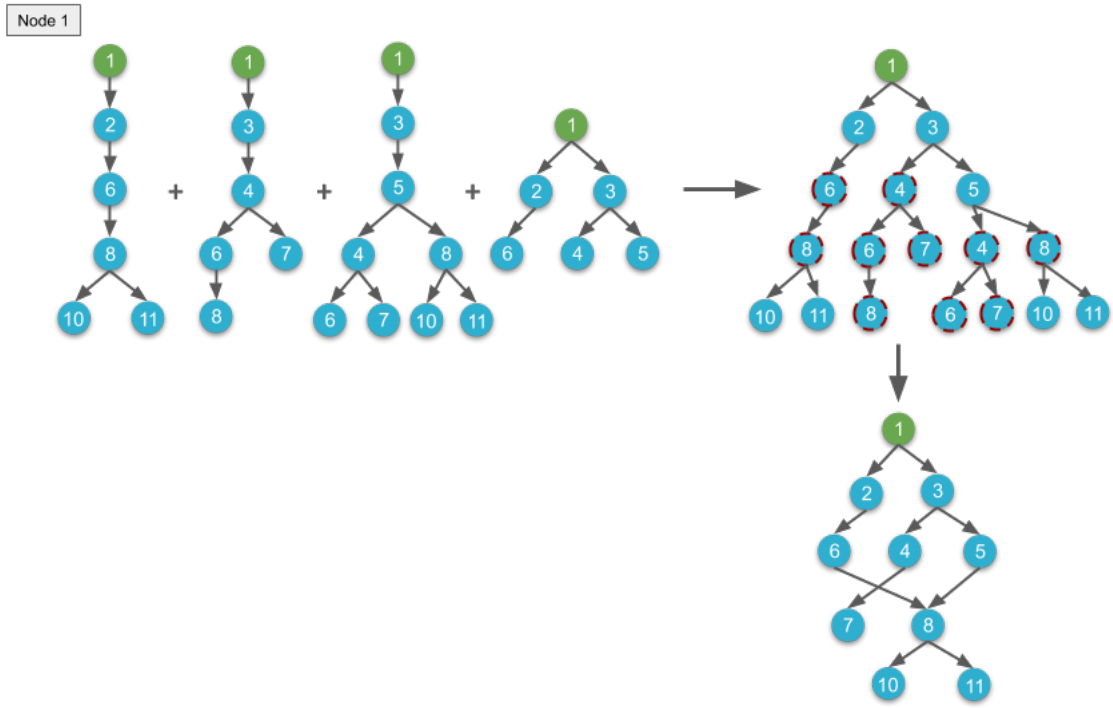


Figure 3.7: Reducer #4: generation of five-level subgraph of node 1 by combining all the five-level branches with the three-level subgraph

In Reducer #4, all of these five-level branches as well as the original three-level subgraph with the same key node are merged into a final five-level subgraph. An additional branch pruning is required to remove the duplicate nodes and inter-level edges in the merged subgraph. Figure 3.7 represents the final merging process of node 1. It can be seen that nodes 4, 6, 7 and 8 appear on multiple levels in the

merged subgraph. Using the same pruning approach as stated in the previous section, these duplicate nodes at the deeper level and their descendants are removed to retain the hierarchical structure.

3.4 Pseudo-Code

In this section, we will present the Map-Reduce pseudo-code for generating five-level forward subgraphs in directed graphs. Table 3.1, represents the directed graph in Figure 3.1, will be used as the input data of our algorithm. It includes node pairs and the corresponding PageRank values as rows, where each node pair (u, v) indicates that node u points to node v.

Source	**PR_Source**	**Target**	**PR_Target**
1	0.0322	2	0.0459
1	0.0322	3	0.0654
2	0.0459	3	0.0654
2	0.0459	6	0.0906
3	0.0654	4	0.0914
3	0.0654	5	0.0737
4	0.0914	6	0.0906
4	0.0914	7	0.0711
5	0.0737	4	0.0914
5	0.0737	8	0.1543
6	0.0906	8	0.1543
8	0.1543	10	0.0978
8	0.1543	11	0.0978
9	0.0322	5	0.0737
9	0.0322	8	0.1543
11	0.0978	12	0.0738
11	0.0978	13	0.0738

Table 3.1: Input table for the directed graph in Figure 3.1

3.4.1 Finding Incoming and Outgoing Nodes

Finding incoming and outgoing nodes for each node in the graph can be done in one Map-Reduce operation. In Mapper #1, the original node pair and the reversed

node pair are generated for each row as shown in line #1 and #2 of the code. For example, for the input node pair <1-0.0322, 2-0.0459>, two records (<1-0.0322, 2-0.0459> and <2-0.0459, #1-0.0322>) are written to the output. The key-value pairs from Mapper #1 are transferred to Reducer #1 by integration, sorting and further splitting among the Reducer servers. After collecting all the outgoing and incoming nodes for each *key* node, the output of Reducer #1 is generated as shown in Table 3.2. For instance, we can see that node 2 (PR=0.0459) has two outgoing nodes - node 3 (PR=0.0654) and node 6 (PR=0.0906), and one incoming node - node 1 (PR=0.0322).

Algorithm 1 Mapper #1

Input: Input table with PageRank values (*node1-PR1, node2-PR2*)
Output: Key-Value pair

1: **yield** (*node1-PR1, node2-PR2*)
2: **yield** (*node2-PR2, #node1-PR1*)

Algorithm 2 Reducer #1

Input: Key-Value pair from Mapper1
Output: Key-Value pair: each node as *key* and its incoming and outgoing nodes as *value*

1: Merge values for each key
2: Write Key-Value pair as output

3.4.2 Building Three-Level Subgraphs

The incoming and outgoing nodes of each key node are used in the second Map-Reduce operation to construct three-level subgraphs. In Mapper #2, for each node string *inNode-inPR* in the incoming node list of a key node *keyNode*, a two-hop directed subgraph that is rooted at *inNode*, connected through *keyNode* and pointed to the outgoing nodes of *outNode* is generated in line#3-15. To preserve all the linkage information, a one-hop subgraph from the key node *keyNode* to its outgoing nodes *outNode* is also produced in line#16-22. For example, since node 1 has a empty incoming node list, only a one-hop subgraph is generated:

Key	Value
1-0.0322	2-0.0459, 3-0.0654,
2-0.0459	3-0.0654, 6-0.0906, #1-0.0322
3-0.0654	4-0.0914, 5-0.0737, # 1-0.0322, #2-0.0459
4-0.0914	6-0.0906, 7-0.0711, #3-0.0654, #5-0.0737
5-0.0737	4-0.0914, 8-0.1543, #3-0.0654, #9-0.0322
6-0.0906	8-0.1543, #2-0.0459, #4-0.0914
7-0.0711	#4-0.0914
8-0.1543	10-0.0978, 11-0.0978, #5-0.0737, #6-0.0906, #9-0.0322
9-0.0322	5-0.0737, 8-0.1543
10-0.0978	#8-0.1543
11-0.0978	12-0.0738, 13-0.0738, #8-0.1543
12-0.0738	#11-0.0978
13-0.0738	#11-0.0978

Table 3.2: Output of Reducer #1: incoming and outgoing nodes of all the nodes in the graph

key=(1-0.0322) and value=(2-0.0459, 3-0.0654 &).

For node 2, a two-hop subgraph and a one-hop subgraph are generated:

key=(1-0.0322) and value=(2-0.0459 & 3-0.0654<1, 6-0,0906<1),

key=(2-0.0459) and value=(3-0.0654, 6-0.0906 &).

Here levels in the branch are separated by "&" and the parent node of each node in the third level is represented using "<" symbol before the node name.

In Reducer #2, to form three-level subgraphs, all the two-hop and one-hop directed branches generated from Mapper #2 are first merged level by level based on the key node, as shown in line#3-13 of the code. Then the three-level subgraph is pruned by removing the duplicate nodes among levels and inter-level links as shown in line # 14-18. For example, if a node appears in both the second level and third level, it will be deleted from the node list of the third level. In line # 19-22, the collected three-level list is reformatted by attaching the parent node to each node in the list accordingly. The final output pair is generated in line #23. The input key-value pair for node 1 in Reducer #2 is key=(1-0.0322) and value=(2-0.0459, 3-0.0654), (2-0.0459 & 3-0.0654<2, 6-0.0906<2), (3-0.0654 & 4-0.0914<3, 5-0.0737<3). After a level-by-level merging, the three subgraphs can

Algorithm 3 Mapper #2: generate three-level branches

Input: Key-Value pair from Reducer1, (key = *keyNode-keyNodePR*, value = *nodeList*)

Output: Key-Value pair: each node and its PageRank value as *key* and its three-level branch as *value*

```
keyNode, keyNodePR ← keyNode-keyNodePR
outList, inList ← nodeList
if inList is not empty then
    for inNode-inPR in inList do
        newOutList = []
        if outList is not empty then
            for outNode-outPR in outList do
                if outNode-outPR ≠ inNode-inPR then
                    newOutList.append(outNode-outPR + "<" + keyNode)
                end if
            end for
        end if
        yield (inNode-inPR, keyNode-keyNodePR +"&"+newOutList)
    end for
end if
if outList is not empty then
    newOutList = []
    for outNode-outPR in outList do
        newOutList.append(outNode-outPR)
    end for
    yield (keyNode-keyNodePR, newOutList +"&")
end if
```

be merged into value=(2-0.0459, 3-0.0654 & 3-0.0654<2, 6-0.0906<2, 4-0.0914<3, 5-0.0737<3). Node 3 (3-0.0654) appears in both second and third levels, so the inter-level link from node 2 to node3 should be removed and the final three-level subgraph for node 1 is key=(1-0.0322) and value=(2-0.0459, 3-0.0654 & 6-0.0906<2, 4-0.0914<3, 5-0.0737<3). The output data of three-level subgraphs for each node in our sample graph can be found in Table 3.3.

Algorithm 4 Reducer #2: merge three-level branches

Input: Key-Value pair from Mapper2, *keyNode-keyNodePR, nodeLists*

Output: Key-Value pair: each node as key and its three-level subgraph as value

```
secondLevel = []
thirdLevel = new dictionary {}
for nodeList in nodeLists do
    secondlevelStr, thirdlevelStr ← nodeList.split("&")
    for node-PR in secondlevelStr do
        secondLevel.append(node-PR)
    end for
    if thirdlevelStr is not empty then
        for node-PR<parentNode in thirdlevelStr do
            thirdLevel[node-PR].add(parentNode)
        end for
    end if
end for
for node-PR in thirdLevel.keys() do
    if node-PR in secondLevel then
        delete thirdLevel[outNode-outPR]
    end if
end for
newThirdLevel = []
for  node-PR, parentNodes in thirdLevel.items() do
    newThirdLevel.append(node-PR + "<" + parentNodes)
end for
yield (keyNode-keyNodePR, secondLevel +"&"+ newThirdLevel)
```

3.4.3 Building Five-Level Subgraphs

Constructing a five-level subgraph of a core node from multiple three-level subgraphs can be done in one complete Map-Reducer operation (Mapper #3 and Reducer #3) and one single Reducer operation (Reducer #4). In order to han-

Key	Value
1-0.0322	2-0.0459, 3-0.0654 & 6-0.0906<2, 4-0.0914<3, 5-0.0737<3
2-0.0459	6-0.0906, 3-0.0654 & 8-0.1543<6, 4-0.0914<3, 5-0.0737<3
3-0.0654	5-0.0737, 4-0.0914 & 8-0.1543<5, 6-0.0906<4, 7-0.0711<4
4-0.0914	6-0.0906, 7-0.0711 & 8-0.1543<6
5-0.0737	4-0.0914, 8-0.1543 & 10-0.0978<8, 11-0.0978<8, 6-0.0906<4, 7-0.0711<4
6-0.0906	8-0.1543 & 10-0.0978<8, 11-0.0978<8
8-0.1543	10-0.0978, 11-0.0978 & 12-0.0738<11, 13-0.0738<11
9-0.0322	5-0.0737, 8-0.1543 & 4-0.0914<5, 10-0.0978<8, 11-0.0978<8
11-0.0978	12-0.0738, 13-0.0738 &

Table 3.3: Output of Reducer #2: three-level subgraphs

dle the graph growth in a distributed way, we need to ensure that the three-level subgraphs that will be merged have the same key in the reducer. In Mapper #3, in line #1 we first emit the three-level subgraph in its original format to preserve the structure information. Next, we only process these three-level subgraphs with non-empty leaf/third levels (line #2). For each leaf node in the subgraph, we find all its parent nodes in the second level as shown in line #5-9 and then emit the output where the leaf node as key and the path from the root node to it as value (line #10). For example, for node 1, the following output will be generated:

key=(1-0.0322) and value=(2-0.0459, 3-0.0654 & 6-0.0906<2, 4-0.0914<3, 5-0.0737<3),

key=(6-0.0906) and value=(1-0.0322 > 2-0.0459),

key=(4-0.0914) and value=(1-0.0322 > 3-0.0654),

key=(5-0.0737) and value=(1-0.0322 > 3-0.0654).

The first record is the original output from Reducer #2 and the rest of them describe the path from node 1 to its three leaf nodes (nodes 4, 5 and 6), respectively. For any node that does not have a third level such as node 11, since it cannot further grow we simply emit its three-level subgraph. A complete output for Mapper #3 is shown in Table 3.4.

The root-to-leaf paths, as well as the original three-level subgraphs, are merged in Reducer #3 to grow five-level branches. First, the pruned three-level subgraph

Algorithm 5 Mapper #3: reverse three-level subgraphs

Input: Key-Value pair from Reducer2: (key = *keyNode-keyNodePR*, value = *secondLevel* & *thirdLevel*)

Output: Each node in the leaf level as key and the path from the key node to it as value

```
1: yield (keyNode-keyNodePR, secondLevel & thirdLevel)
2: if thirdLevel is not empty then
3:    pathToLeafNode = []
4:    for leafNode, leafNodePR, parentNodes in thirdLevel do
5:       for node, nodePR in secondLevel do
6:          if node in parentNodes then
7:             pathToLeafNode.append(node-nodePR)
8:          end if
9:       end for
10:      yield (leafNode-leafNodePR, keyNode-keyNodePR > pathToLeafNode)
11:   end for
12: end if
```

is preserved in line #1-9 of the code. In line #1, the PageRank limit is computed using the PageRank threshold k and the PageRank score of the key node. For each node in the three-level subgraph, if its PageRank score is less than the limit, we remove this node and its descendants in the subgraph. Then for each reversed three-level branches, a five-level branch is produced by combining with the original three-level subgraph (line #10-11). For instance, the input data for node 6 in Reducer #3 is key=6-0.0906, value=

8-0.1543 & 10-0.0978<8, 11-0.0978<8

1-0.0322 > 2-0.0459

3-0.0654 > 4-0.0914

5-0.0737 > 4-0.0914.

The first record is the three-level subgraph rooted at node 6, and the rest of three records are three-level branches that are rooted at nodes 1, 3 and 5 and have node 3 as the leaf node. Therefore, these five-level branches can be generated by attaching the three-level subgraph of node 3 to them:

1-0.0322 & 2-0.0459 & 6-0.0906<2 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8

3-0.0654 & 4-0.0914 & 6-0.0906<4 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8

Key	Value
1-0.0322	2-0.0459, 3-0.0654 & 6-0.0906<2, 4-0.0914<3, 5-0.0737<3
6-0.0906	1-0.0322 > 2-0.0459
4-0.0914	1-0.0322 > 3-0.0654
5-0.0737	1-0.0322 > 3-0.0654
2-0.0459	6-0.0906, 3-0.0654 & 8-0.1543<6, 4-0.0914<3, 5-0.0737<3
8-0.1543	2-0.0459 > 6-0.0906
4-0.0914	2-0.0459 > 3-0.0654
5-0.0737	2-0.0459 > 3-0.0654
3-0.0654	5-0.0737, 4-0.0914 & 8-0.1543<5, 6-0.0906<4, 7-0.0711<4
8-0.1543	3-0.0654 > 5-0.0737
6-0.0906	3-0.0654 > 4-0.0914
7-0.0711	3-0.0654 > 4-0.0914
4-0.0914	6-0.0906, 7-0.0711 & 8-0.1543<6
8-0.1543	4-0.0914 > 6-0.0906
5-0.0737	4-0.0914, 8-0.1543 & 10-0.0978<8, 11-0.0978<8, 6-0.0906<4, 7-0.0711<4
10-0.0978	5-0.0737 > 8-0.1543
11-0.0978	5-0.0737 > 8-0.1543
6-0.0906	5-0.0737 > 4-0.0914
7-0.0711	5-0.0737 > 4-0.0914
6-0.0906	8-0.1543 & 10-0.0978<8, 11-0.0978<8
10-0.0978	6-0.0906 > 8-0.1543
11-0.0978	6-0.0906 > 8-0.1543
8-0.1543	10-0.0978, 11-0.0978 & 12-0.0738<11, 13-0.0738<11
12-0.0738	8-0.1543 > 10-0.0978, 11-0.0978
13-0.0738	8-0.1543 > 10-0.0978, 11-0.0978
9-0.0322	5-0.0737, 8-0.1543 & 4-0.0914<5, 10-0.0978<8, 11-00978<8
4-0.0914	9-0.0322 > 5-0.0737
10-0.0978	9-0.0322 > 8-0.1543
11-00978	9-0.0322 > 8-0.1543
11-0.0978	12-0.0738, 13-0.0738 &

Table 3.4: Output of Mapper #3 for each key-value pair

5-0.0737 & 4-0.0914 & 6-0.0906<4 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8. The output of Reducer 3 is shown in Table 3.5.

Meanwhile, the PageRank values of all the nodes in the branch are compared with that of the root node (line #12-18). If the PageRank value of a node is samller than that of the root node, the node and its descendants are removed from the five-level branch. Suppose we choose a PageRank threshold $k = 0.8$, the five-level branch rooted at node 8 (key=(8-0.1543) and value=(10-0.0978, 11-0.0978 & 12-

0.0738<11, 13-0,0738<11)) will be discarded since the PageRank values of nodes 10 and 11 are much smaller than the PageRank threshold $0.1543 \times 0.8 = 0.1234$, which is computed from the PageRank value of the key node 8.

Algorithm 6 Rducer #3: generate five-level branches

Input: Key-Value pair from Mapper3, *keyNode-keyNodePR* as key and its three-level subgraph (*secondLevel* & *thirdLevel*) and a set of reversed two-level paths (*rootNode-rootNodePR* > *nextLevel*) as value;
PageRank threshold k

Output: Key-Value pair: each root node as key and its five-level branches as value

```
1:  PRthred1 = k × float(keyNodePR)
2:  newThreeLevel = secondLevel + thirdLevel
3:  for node-nodePR in newThreeLevel do
4:     if nodePR < PRthred1 then
5:        delete node-nodePR in newThreeLevel
6:        delete descendantsOf(node-nodePR) in newThreeLevel
7:     end if
8:  end for
9:  yield (keyNode-keyNodePR, newThreeLevel)
10: for rootNode-rootNodePR > nextLevel in all reversed two-level paths do
11:    newFiveLevelBranch = nextLevel & keyNode-keyNodePR & secondLevel &
       thirdLevel
12:    PRthred2 = k× float(rootNodePR)
13:    for node-nodePR in newFiveLevelBranch do
14:       if nodePR < PRthred2 then
15:          delete node-nodePR in newFiveLevelBranch
16:          delete descendantsOf(node-nodePR) in newFiveLevelBranch
17:       end if
18:    end for
19:    yield (rootNode-rootNodePR, newFiveLevelBranch )
20: end for
```

Finally, the five-level subgraphs that have the same root node are merged with its three-level subgraph in Reducer #4 as shown in line #1-9 of the code. Duplicated nodes that show in deeper levels are removed as well as their edges (line #5). The resulting five-level graphs for each node with a PageRank threshold $k = 0.8$ are shown in Table 3.6.

Key	Value
1-0.0322	2-0.0459, 3-0.0654 & 6-0.0906<2, 4-0.0914<3, 5-0.0737<3
2-0.0459	6-0.0906, 3-0.0654 & 8-0.1543<6, 4-0.0914<3, 5-0.0737<3
3-0.0654	5-0.0737, 4-0.0914 & 8-0.1543<5, 6-0.0906<4, 7-0.0711<4
4-0.0914	6-0.0906 & 8-0.1543<6
2-0.0459	3-0.0654 & 4-0.0914<3 & 6-0.0906<4, 7-0.0711<4 & 8-0.1543<6
1-0.0322	3-0.0654 & 4-0.0914<3 & 6-0.0906<4, 7-0.0711<4 & 8-0.1543<6
9-0.0322	5-0.0737 & 4-0.0914<5 & 6-0.0906<4, 7-0.0711<4 & 8-0.1543<6
5-0.0737	4-0.0914, 8-0.1543 & 10-0.0978<8, 11-0.0978<8, 6-0.0906<4, 7-0.0711<4
2-0.0459	3-0.0654 & 5-0.0737<3 & 4-0.0914<5, 8-0.1543<5 & 10-0.0978<8, 11-0.0978<8, 6-0.0906<4, 7-0.0711<4
1-0.0322	3-0.0654 & 5-0.0737<3 & 4-0.0914<5, 8-0.1543<5 & 10-0.0978<8, 11-0.0978<8, 6-0.0906<4, 7-0.0711<4
6-0.0906	8-0.1543 & 10-0.0978<8, 11-0.0978<8
1-0.0322	2-0.0459 & 6-0.0906<2 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8
3-0.0654	4-0.0914 & 6-0.0906<4 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8
5-0.0737	4-0.0914 & 6-0.0906<4 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8
2-0.0459	6-0.0906 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11
3-0.0654	5-0.0737 & 8-0.1543<5 & 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11
4-0.0914	6-0.0906 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11
9-0.0322	5-0.0737, 8-0.1543 & 4-0.0914<5, 10-0.0978<8, 11-00978<8
5-0.0737	8-0.1543 & 11-0.0978<8 & 12-0.0738<11, 13-0.0738>11 &
6-0.0906	8-0.1543 & 11-0.0978<8 & 12-0.0738>11, 13-0.0738>11 &
9-0.0322	8-0.1543 & 11-0.0978<8 & 12-0.0738>11, 13-0.0738>11 &

Table 3.5: Output of Reducer #3

3.5 Summary

In this chapter, we described our approach to construct PageRank based hierarchical structures in the directed graph along with the illustrative examples and details of the implementation. Based on the forward and backward hierarchical structure, a complete community around the core node can be generated. In the next chapter, we will present the experimental results of our algorithm using real-world datasets.

Algorithm 7 Rducer #4: merge five-level subgraphs

Input: Key-Value pair from Rducer #3,
keyNode-keyNodePR as key and its three-level graph (*secondLevel* & *thirdLevel*) and a set of five-level branches (*newSecondLevel* & *newThirddLevel* &; *newFourthLevel* & *newFifthLevel*) as value;

Output: Key-Value pair: each node as key and its five-level subgraph as value

1: *newSecondLevel* = merge(*secondLevel*, all *newSecondLevel*)
2: *newThirddLevel* = merge(*thirdLevel*, all *newThirddLevel*)
3: *newFourthLevel* = merge(all *newFourthLevel*)
4: *newFifthLevel* = merge(all *newFifthLevel*)
5: *result* = *newSecondLevel* & *newThirddLevel* & *newFourthLevel* & *newFifthLevel*
6: **yield** (*keyNode-keyNodePR*, *result*)

Key	**Value**
1-0.0322	2-0.0459, 3-0.0654 & 6-0.0906<2, 4-0.0914<3, 5-0.0737<3 & 7-0.0711<4, 8-0.1543<5/6 & 10-0.0978<8, 11-0.0978<8
2-0.0459	3-0.0654, 6-0.0906 & 8-0.1543<6, 4-0.0914<3, 5-0.0737<3 & 7-0.0711<4, 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11
3-0.0654	4-0.0914, 5-0.0737 & 8-0.1543<5, 6-0.0906<4, 7-0.0711<4 & 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11
4-0.0914	6-0.0906 & 8-0.1543<6 & 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11
5-0.0737	4-0.0914, 8-0.1543 & 10-0.0978<8, 11-0.0978<8, 6-0.0906<4, 7-0.0711<4 & 12-0.0738<11, 13-0.0738<11 &
6-0.0906	8-0.1543 & 10-0.0978<8, 11-0.0978<8 & 12-0.0738<11, 13-0.0738<11 &
9-0.0322	8-0.1543, 5-0.0737 & 4-0.0914<5, 10-0.0978<8, 11-0.0978<8 & 6-0.0906<4, 7-0.0711<4, 12-0.0738<11, 13-0.0738<11 &

Table 3.6: Output of Reducer #4: five-level subgraphs with PageRank threshold = 0.8

Chapter 4

Experiments and Results

4.1 Introduction

In the previous chapter, we have described our Map-Reduce based community detection algorithm for directed graphs and illustrated how to build the hierarchical community around a core node based on PageRank values. To evaluate and validate our approach, in this chapter we have applied it to several real-world directed networks from various domains. We will demonstrate how the algorithm works on these directed datasets, investigate the effect of the core node's PageRank score and the PageRank threshold in the clustering process, and examine the clustering results of our method using evaluation metrics.

4.2 Dataset Description

In our experiments, the following five directed datasets with different sizes (shown in Table 4.1) were utilized to evaluate our community detection algorithm.

Dataset	Type	# of Nodes	# of Edges
Highschool	Friendship Network	70	366
Physicians	Friendship/Trust Network	241	1,098
Bitcoin-Alpha	Trust Network	3,783	24,186
Bitcoin-OTC	Trust Network	5,881	35,592
Supreme-Court-citation	Citation Network	34,613	202,167

Table 4.1: Five datasets with different sizes

Highschool dataset

This dataset describes friendships between boys in a high school in Illinois [30]. A node represents a boy and an edge between two boys represents that the left boy considered the right boy as a friend. This dataset has 70 nodes and 366 edges.

Physicians dataset

This dataset was prepared from the data collected by Coleman, Katz and Menzel on medical innovation in 1966 [31]. It describes a social network among 245 physicians. A node represents a physician. An edge between two physicians represents that the left physician considered the right physician as a friend and turned to the right physician if he/she needs information or advice about questions of therapy.

Bitcoin Alpha dataset

This is a user-user trust network on a Bitcoin trading platform called Bitcoin Alpha where users can buy and sell products using Bitcoins [32, 33]. A node represents a user and a directed edge between two users represents that the left user trusts the right user.

Bitcoin OTC dataset

This is also a user-user trust network from another Bitcoin exchange platform called Bitcoin OTC [32, 33]. A node represents a user and a directed edge between two users represents that the left user trusts the right user.

Supreme Court citation dataset

This legal citation network was captured by J. H. Fowler et al.[34] from majority opinions written by the Supreme Court of the United States and cases that cite them from 1791 to 2005. It contains 202,167 citations of USSC majority opinions. A directed edge between two nodes represents that the left case cites the right case.

4.3 Experiments

We constructed nine-level communities around core nodes to evaluate our approach. A five-level upper-level subgraph and a five-level lower-level subgraph from each core node were generated and then merged to form a nine-level hierarchical community.

The importance scores of all the nodes in each dataset were computed before clustering. It should be pointed out that both the regular PageRank algorithm and the reversed PageRank algorithm (by reversing the directions of all edges) can be used to measure the node importance. The choice of the two algorithms is based on how influential nodes are defined in the graph. In some cases, the number of incoming links is related to the importance of a node. While in other cases, the number of outgoing links may contribute more to the importance of a node and thus the reversed PageRank should be used for ranking nodes. Large PageRank scores suggest nodes that can be reached by many nodes in the graph while large reverse PageRank scores suggest nodes that can reach many nodes in the graph.

For example, in a trust network, the person who is trusted by the most others has the highest PageRank score. On the other hand, in a defeat network, the player who defeated the most others has the highest reversed PageRank score. In our experiments, only the regular PageRank score was used due to the underlying semantics of our example datasets.

4.3.1 Experimental Setup

Our Map-Reduced based algorithm was implemented using Spark 2.3.2 in PySpark. The cluster configuration is shown in Table 4.2. The same setup was used for processing all the example datasets.

Number of nodes	6
Number of cores per node	12
RAM per node	48.25 GB
Framework	Apache Spark 2.3.2
Development platform	PySpark 2.4.3

Table 4.2: Cluster configuration

4.3.2 Evaluation Metric

To measure the quality of clustering results, we have introduced an evaluation metric called community coefficient (CC). The community coefficient is the ratio of the actual number of edges in the community to the maximum possible number of edges in the community, defined as follows:

$$CC = \frac{\text{\# of actual edges in the community}}{\text{\# of all possible edges in the community}} = \frac{N_{edge}}{C(N_{node}, 2) \times 2}, \tag{4.3.1}$$

where N_{edge} is the total number of edges among all the nodes of the generated community in the graph and N_{node} is the total number of nodes in the commu-

nity. The community coefficient is in the range 0 - 1. The higher the community coefficient, the more densely connected the nodes in the community.

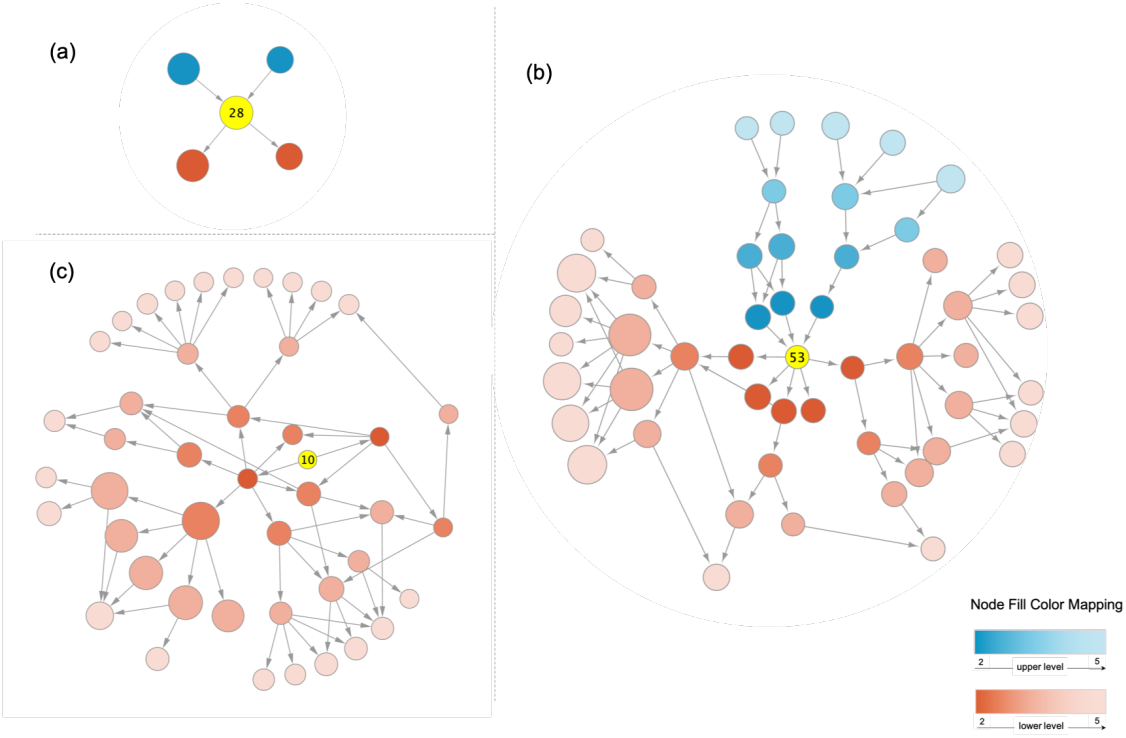
4.4 Results and Discussion

In this section, we have used the five example datasets for the evaluation of hierarchical communities around core nodes generated using our approach. We have discussed the effect of the PageRank score of the core node and the PageRank threshold on the size of the generated communities as well as their community coefficient values. We have also conducted the validation experiment to validate the obtained communities.

4.4.1 Nine-level Communities

Highschool dataset

The highschool dataset has 70 nodes which represent 70 high school students. In this dataset, nodes with large PageRank scores are the students who were chosen as friends by many other students. Three nodes with the highest, medium and lowest PageRank scores are selected as examples to demonstrate the generated 9-level communities, as shown in Figure 4.1. For the generated community of each core node (highlighted in yellow), blue nodes constitute the upper part of the community where nodes directly or indirectly point to the core node, and red nodes constitute the lower part of the community where the core node directly or indirectly points to these nodes. 4 shades in blue and red represent the level number of a node in the upper part and lower part of the community, respectively. Lighter shades indicate nodes with deeper levels. The dimension of the node represents its PageRank score. Node 28 has the highest PageRank score of 0.0662 and shows a small community of 4 nodes with only one upper level and one lower level. In contrast, node 10 has the smallest PageRank score of 0.0021 and shows a



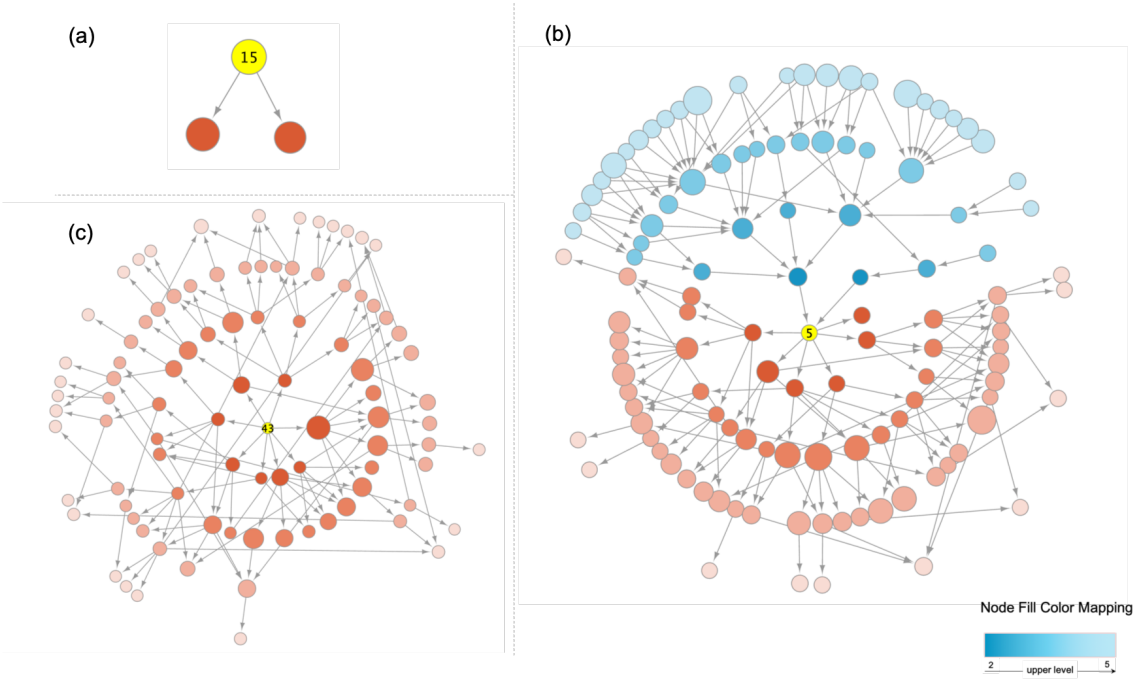
Core node	28	53	10
PageRank score	0.0662	0.0091	0.0021
# of nodes in upper levels	2	14	0
# of nodes in lower levels	2	37	45

Figure 4.1: Highschool dataset: nine-level communities of node (a) 28, (b) 53 and (c) 10 with PageRank threshold = 0.8

community with no upper levels. Node 53 has a medium PageRank score (0.0091) and displays a community with 4 levels in both the upper and lower part.

Physicians dataset

The physicians dataset has 245 nodes representing 245 physicians. Nodes with higher PageRank scores represent the physicians who were considered as friends for suggestions and advice by many other physicians. Three generated 9-level communities for nodes 15, 5 and 43 with the highest, medium and lowest PageRank scores of 0.0223, 0.0028 and 0.0009 are shown in Figure 4.2. Node 15 has the highest PageRank score and forms a small community with only a lower part of 2 nodes. Since a PageRank threshold of $k = 0.8$ was used, the generated community implies that those nodes that point to node 15 have much smaller PageRank scores.



Core node	15	5	43
PageRank score	0.0223	0.0028	0.0009
# of nodes in upper levels	0	48	0
# of nodes in lower levels	2	67	92

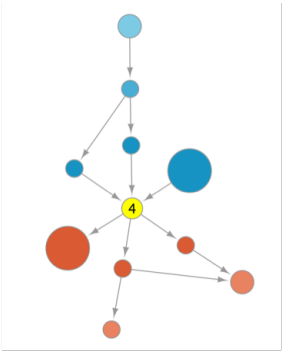
Figure 4.2: Physicians dataset: nine-level communities of nodes (a) 15, (b) 5 and (c) 43 with PageRank threshold = 0.8

Similar to the highschool dataset, the community around node 5 exhibits complete 9 levels and the community around node 43 only has lower levels.

Bitcoin Alpha dataset

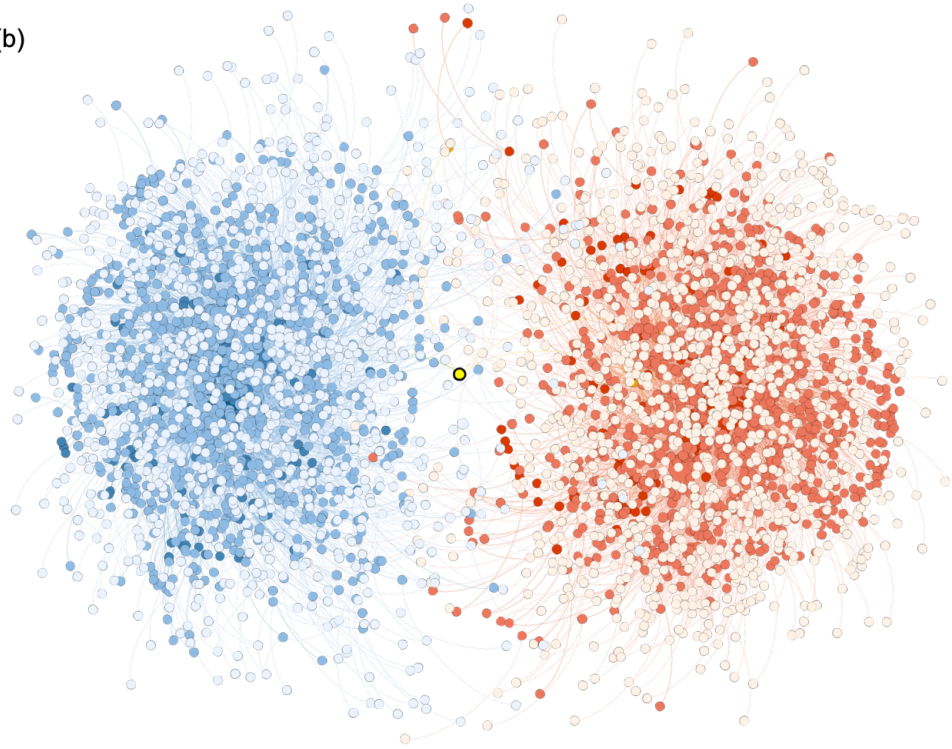
Bitcoin Alpha dataset represents a trust network where nodes with high PageRank scores are the users who are trusted by many other users in the Bitcoin transaction platform. Node 4 has the highest PageRank score and thus is the most trustworthy user in the graph, while node 6434 has the lowest PageRank score, indicating that either he/she is not trusted by most of the users or there is no enough transaction data available related to him/her. As we can see in Figure 4.3, the community around node 4 is consist of 10 other nodes; the community around node 2278 with a medium PageRank score has many more nodes and its upper part and lower part have comparable sizes. Node 6434 has the smallest PageRank score. It can

(a)



Core node	(a) 4	(b) 2278	(c) 6434
PageRank score	0.00803	0.00012	0.00005
# of nodes in upper levels	5	1,971	0
# of nodes in lower levels	5	2,108	3,060

(b)



(c)

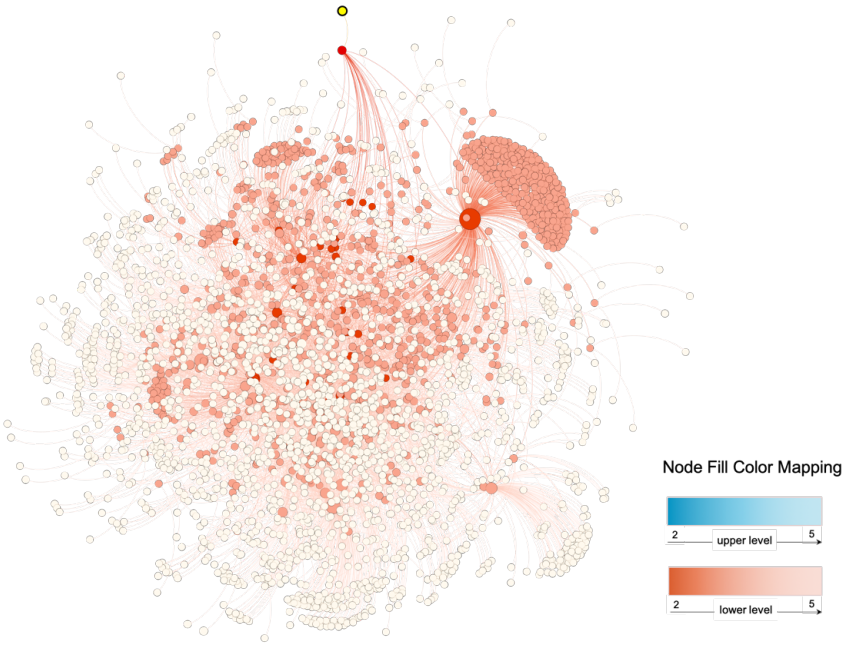


Figure 4.3: Bitcoin Alpha dataset: nine-level communities of node (a) 4, (b) 2278 and (c) 6434 with PageRank threshold = 0.8

be observed clearly that in the community of node 6434, several nodes with much higher PageRank scores than that of node 6434 appear on the second and third levels of the community. These high PageRank nodes attract a lot of deeper-level nodes, which largely increases the size of the community.

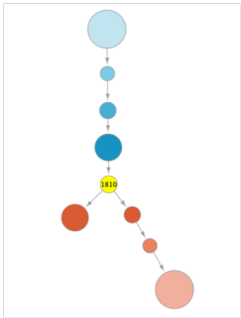
Bitcoin OTC dataset

Bitcoin OTC is also a trust network. Three 9-level communities for nodes 1810, 3831 and 3386 with the highest, medium and lowest PageRank scores of 0.00697, 0.00008 and 0.00004 are shown in Figure 4.4, respectively. These communities generated from the three example nodes show similar structures and behaviors as in the Bitcoin Alpha dataset. Node 1810 has the highest PageRank score and thus results in a small community with 8 nodes. Node 3831 has a medium PageRank score and displays a complete 9-level community with both the upper and lower parts. The community around node 3386 contains several high PageRank nodes that form small highly connected regions in the community.

Supreme Court citation dataset

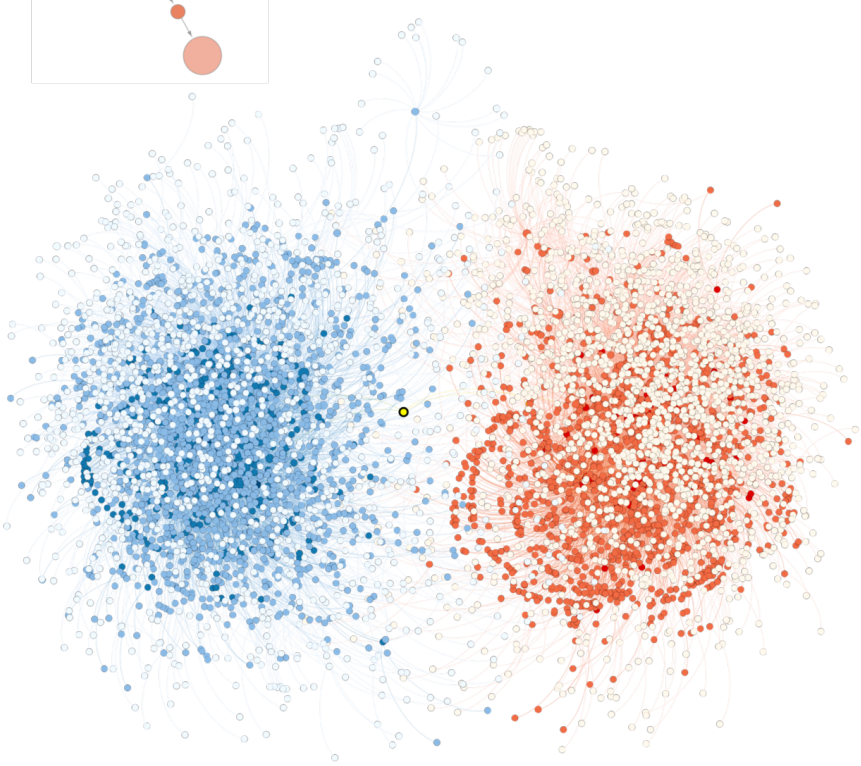
Supreme Court citation dataset is a legal citation network in which a node represents a majority opinion. Nodes with higher PageRank scores represent the opinions that were cited by many other opinions. Three 9-level communities of nodes 33549, 24769 and 7072 with the highest, medium and lowest PageRank scores are shown in Figure 4.5, respectively. As we can see, all the core node have smaller communities compared to that of the Bitcoin OTC dataset. Considering the size of the Supreme Court citation dataset is larger than that of the Bitcoin OTC dataset, we can infer that the Supreme Court citation dataset is more sparser.

(a)



Core node	(a) 1810	(b) 3831	(c) 3386
PageRank score	0.00697	0.00008	0.00004
# of nodes in upper levels	4	3,218	0
# of nodes in lower levels	4	3,195	5,487

(b)



(c)

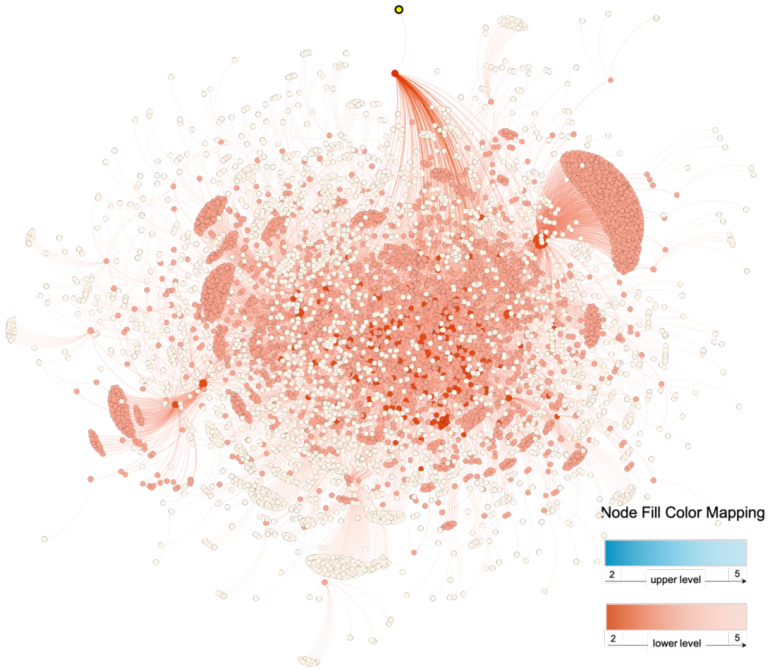
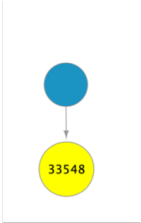


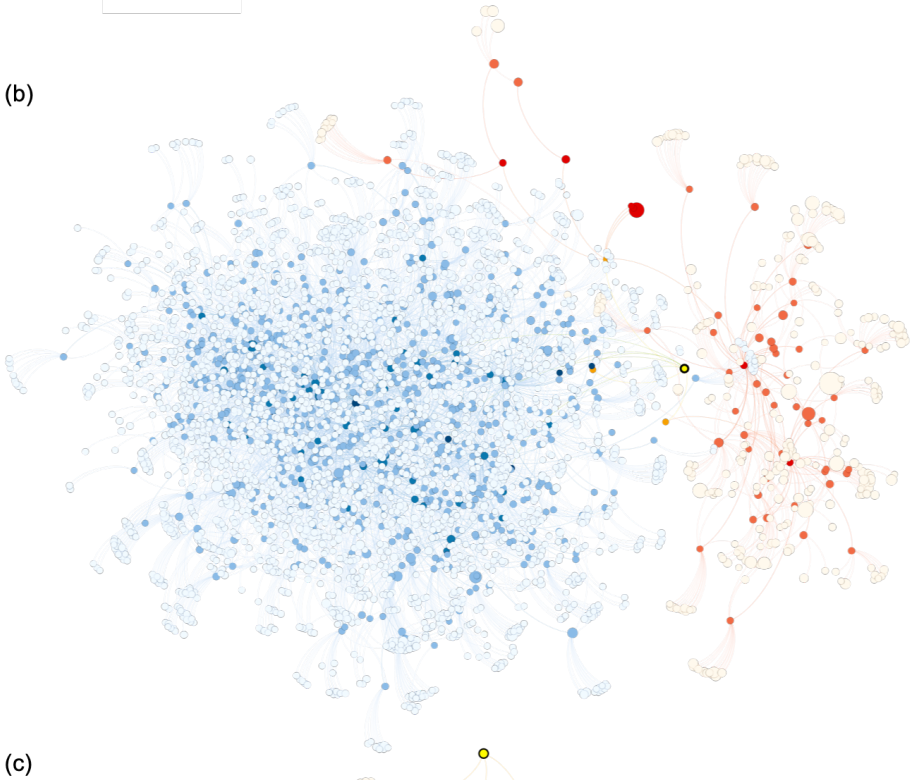
Figure 4.4: Bitcoin OTC dataset: nine-level communities of node (a) 1810, (b) 3831 and (c) 3386 with PageRank threshold = 0.8

(a)



Core node	(a) 33548	(b) 24769	(c) 7072
PageRank score	0.000795	0.000016	0.000010
# of nodes in upper levels	1	3,570	0
# of nodes in lower levels	0	357	1,186

(b)



(c)

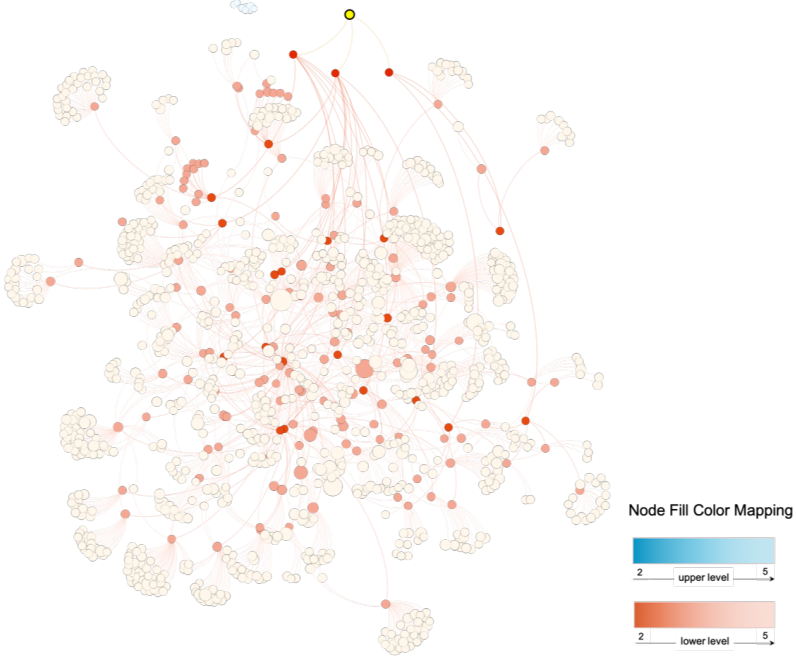


Figure 4.5: Supreme Court citation dataset: nine-level communities of node (a) 33548, (b) 24769 and (c) 7072 with PageRank threshold = 0.8

4.4.2 Community Coefficient and Community Size

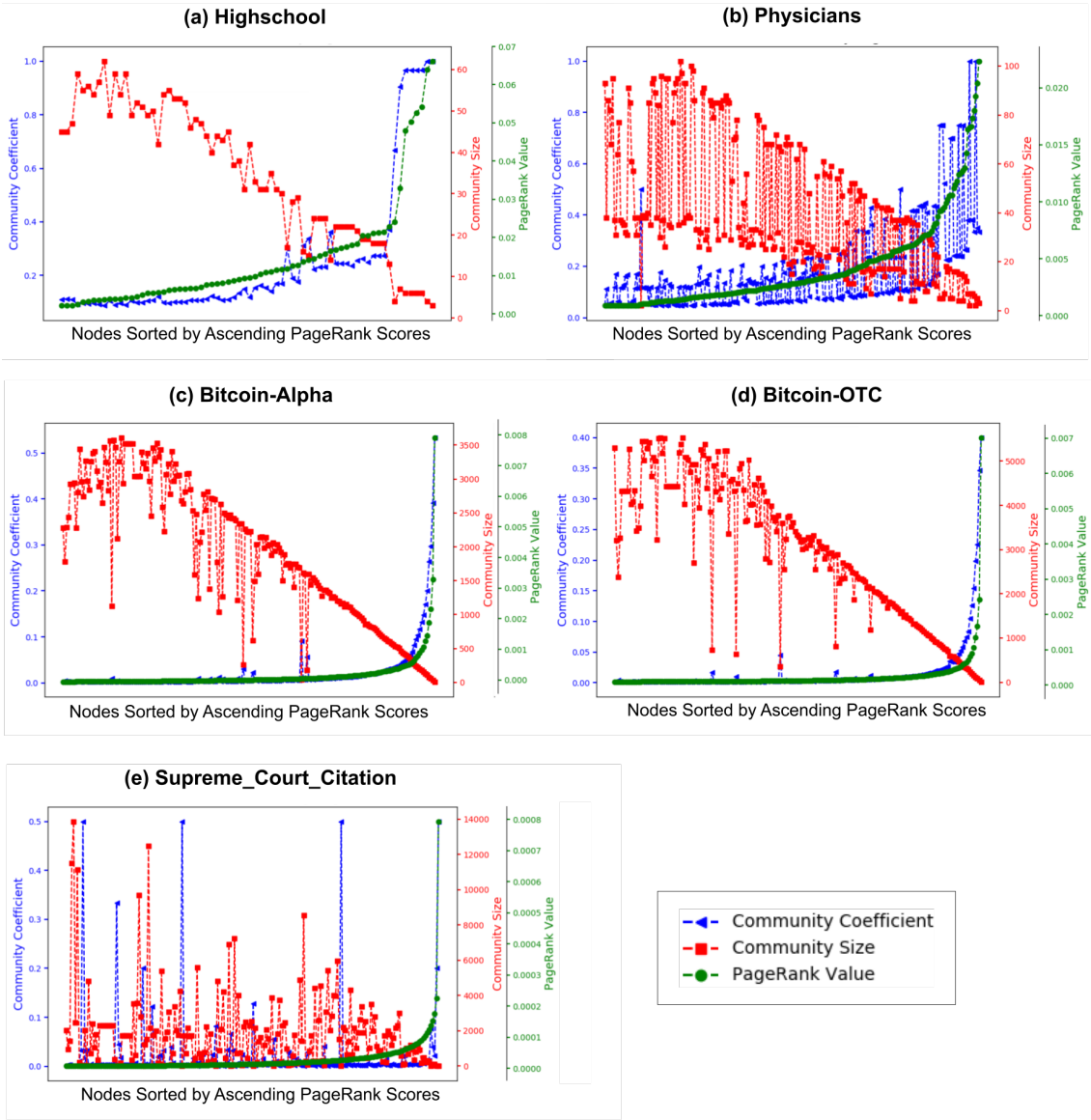


Figure 4.6: Change of community coefficient and community size of the generated communities for core nodes with different PageRank scores. Nodes are ordered by their PageRank scores. (a) Plot of the highschool dataset: 70 nodes in the graph; (b) plot of the physicians dataset: 241 nodes in the graph; (c) plot of the Bitcoin Alpha dataset: 200 sample core nodes are selected with equal width in terms of PageRank scores; (d) plot of the Bitcoin OTC dataset: 200 sample core nodes are selected with equal width in terms of PageRank scores; (e) plot of the Supreme Court citation dataset: 200 sample core nodes are selected with equal width in terms of PageRank scores.

Furthermore, we have also investigated the change of the community coefficient value and the community size when the PageRank scores of the core node increases. The resulting plots for all five example datasets are shown in Figure 4.6. Each marker represents a core node in the graph. The green line is the PageRank score of the core node, the blue line is the community coefficient of the generated community around the core node, and the red line is the community size. Our experiments show that in the case of HighSchool, Physicians, Bitcoin Alpha and Bitcoin OTC datasets, as the PageRack score of the core node increases, the community coefficient increases while the community size decreases in general. For Supreme Court Citation dataset, however, there is no clear relationship between the PageRank score of the core node and the community coefficient and community size.

4.4.3 Effect of PageRank Threshold k

In the clustering process, we need to identify the nodes that show comparable or higher importance than that of the core node and refine the cluster size. Therefore, a PageRank threshold k is applied to all the nodes in the forward and backward five-level subgraphs rooted at the core node. In this section, we will discuss the effect of the PageRank threshold on the obtained community structure.

When various PageRank threshold values k are used, the number of nodes in upper levels and lower levels of resulting communities for the Bitcoin Alpha dataset and Bitcoin OTC dataset are shown in Table 4.3 and 4.4, respectively. For each dataset, 6 sample core nodes and their communities under different k values were selected based on PageRank scores: we selected two nodes with highest PageRank scores, 2 nodes with medium PageRank scores and 2 nodes with lowest PageRank scores. It can be seen in both tables, for these nodes with high and medium PageRank scores, as we lower the PageRank threshold, the number of nodes in both upper and lower levels of the generated community increases. Using the

Node		PageRank Threshold									
		0.9		**0.85**		**0.8**		**0.75**		**0.7**	
		u	***l***	***u***	***l***	***u***	***l***	***u***	***l***	***u***	***l***
Highest	#4	1	1	1	1	5	5	6	6	6	6
	#2	6	6	6	6	7	7	8	8	10	10
Medium	#2278	1808	1913	1912	2032	1971	2108	2073	2213	2235	2405
	#1532	1808	1913	1912	2034	1971	2110	2073	2215	2235	2409
Lowest	#6434	0	3060	0	3060	0	3060	0	3060	0	3060
	#7063	0	3503	0	3503	0	3503	0	3503	0	3503

Table 4.3: Bitcoin Alpha dataset: number of nodes in communities around core nodes #4 and #2 (highest PageRank scores), #2278 and #1532 (medium PageRank scores), and #6434 and #7063 (lowest PageRank scores) when PageRank thresholds $k = 0.9$, 0.85, 0.8, 0.75 and 0.7 are used (u: upper level of the community, l: lower level of the community)

Node		PageRank Threshold									
		0.9		**0.85**		**0.8**		**0.75**		**0.7**	
		u	***l***	***u***	***l***	***u***	***l***	***u***	***l***	***u***	***l***
Highest	#1810	2	2	2	2	4	4	5	5	7	7
	#2028	2	2	4	4	4	4	6	6	8	8
Medium	#3831	2921	2964	3039	3076	3218	3195	3501	3478	3861	3858
	#4770	2872	3162	2984	3313	3147	3515	3429	3853	3784	4302
Lowest	#3386	0	5487	0	5487	0	5487	0	5487	0	5487
	#2218	0	5503	0	5503	0	5503	0	5503	0	5503

Table 4.4: Bitcoin OTC dataset: number of nodes in communities around core nodes #1810 and #2028 (highest PageRank scores), #3831 and #4770 (medium PageRank scores), and #3386 and #2218 (lowest PageRank scores) when PageRank thresholds $k = 0.9$, 0.85, 0.8, 0.75 and 0.7 are used (u: upper level of the community, l: lower level of the community)

Bitcoin OTC dataset as an example, Figure 4.7 and 4.8 visualize the nine-level communities of nodes 1810 and 3831 with threshold $k = 0.9$, 0.8 and 0.7. As

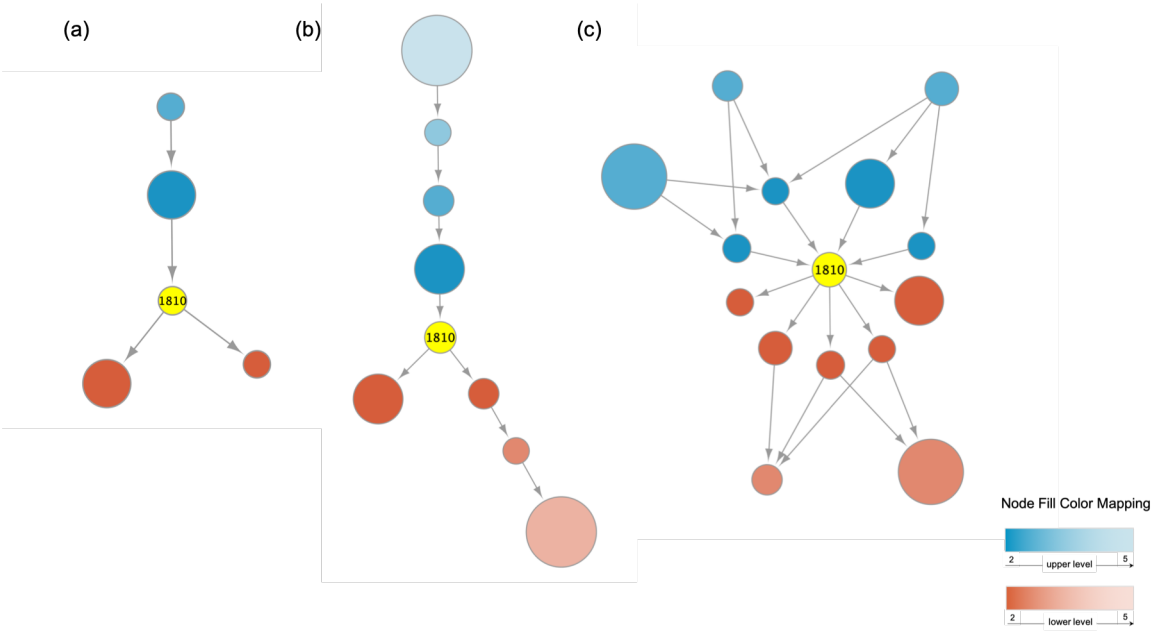


Figure 4.7: Nine-level communities of node 1810 with PageRank threhsold = (a) 0.9, (b) 0.8 and (c) 0.7 for the Bitcoin OTC dataset

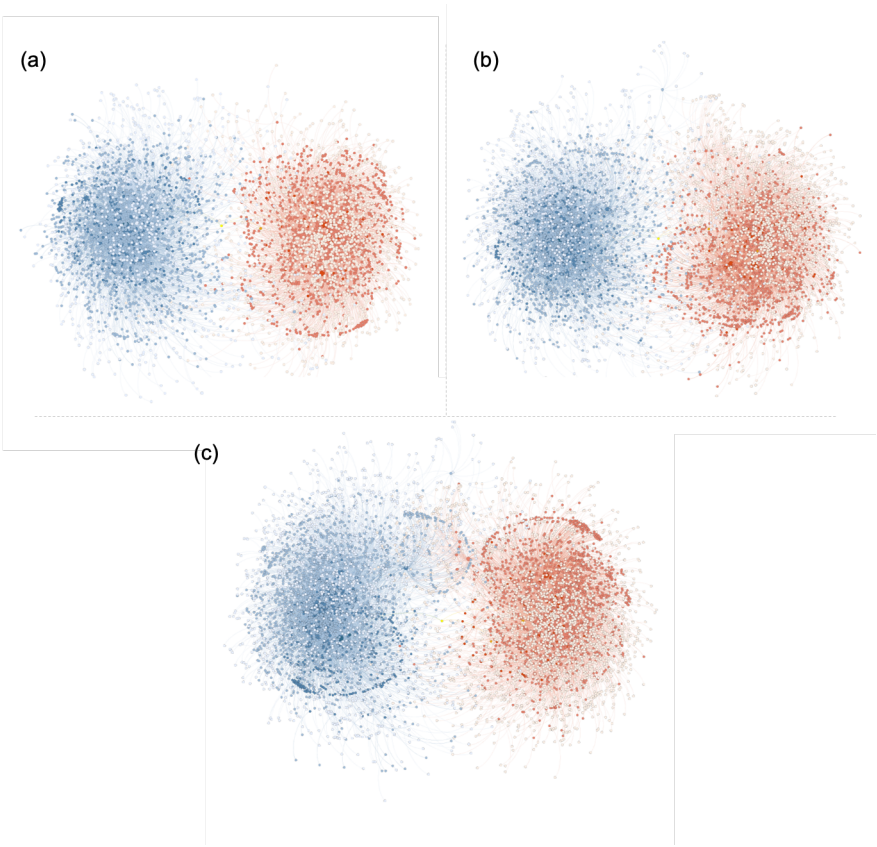


Figure 4.8: Nine-level communities of node 3831 with PageRank threhsold = (a) 0.9, (b) 0.8 and (c) 0.7 for the Bitcoin OTC dataset

can be seen clearly in Figure 4.7, when a high k value of 0.9 is used, since node 1810 has the highest PageRank score, a very small community with 4 nodes is generated. As we relax the threshold, more nodes are found in the community. However, as shown in Figure 4.8, nodes with smaller PageRank scores are not obviously affected by the PageRank threshold. This is because most of the nodes in the community have higher PageRank scores than that of the core node and thus the change of the PageRank threshold does not significantly alter the cluster structure. On the other hand, we also obverse that the community size of low PageRank nodes does not change with the PageRank threshold. Because those core nodes have smaller PageRank scores than any other nodes in the community, the PageRank threshold has no affect on the community structure.

In addition, we have further evaluated the effect of the PageRank threshold k on the community coefficient. Taking the Bitcoin Alpha and Bitcoin OTC datasets as examples, Figure 4.9 and 4.10 show the computed community coefficient value and the community size of the resulting communities for top 50 nodes with highest PageRank scores. The x-axis represents the nodes sorted by their PageRank scores in descending order. It can be clearly observed that as the PageRank score of the core node decreases, the value of community coefficient drops while the community size increases. When the PageRank score of the core node is high, the size of the generated community is generally small, resulting in a high value of the community coefficient. When the PageRank score of the core node decreases, more nodes with slightly smaller PageRank scores start to be included in the community. As the community size becomes larger, the cluster connectivity may reduce accordingly, leading to a smaller value of the community coefficient. Meanwhile, with the decrease of the PageRank threshold, the community coefficient decreases and the community size increases. When we lower the PageRank threshold, more nodes with relatively lower PageRank scores are added to the community, which may introduce weak connections to the original community.

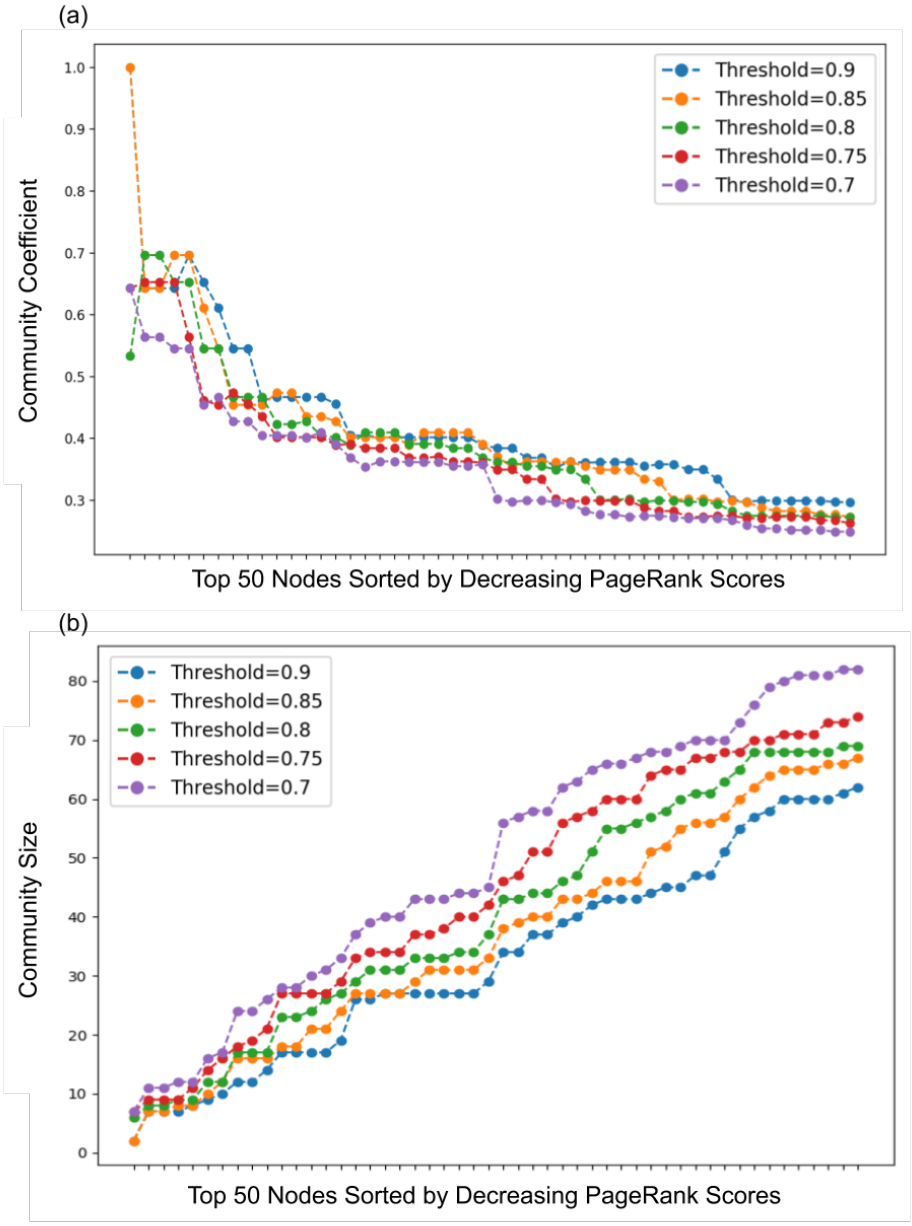


Figure 4.9: Bitcoin Alpha dataset: change of (a) community coefficient and (b) community size for top 50 nodes with highest PageRan scores when PageRank threhsold = 0.9, 0.85, 0.8, 0.75 and 0.7

In conclusion, when selecting the PageRank threshold for generating hierarchical communities, it is necessary to consider what the community size we would like to generate and how large the community coefficient we would like to achieve. Then we can tune the PageRank threshold to obtain the communities that meet our desired performance.

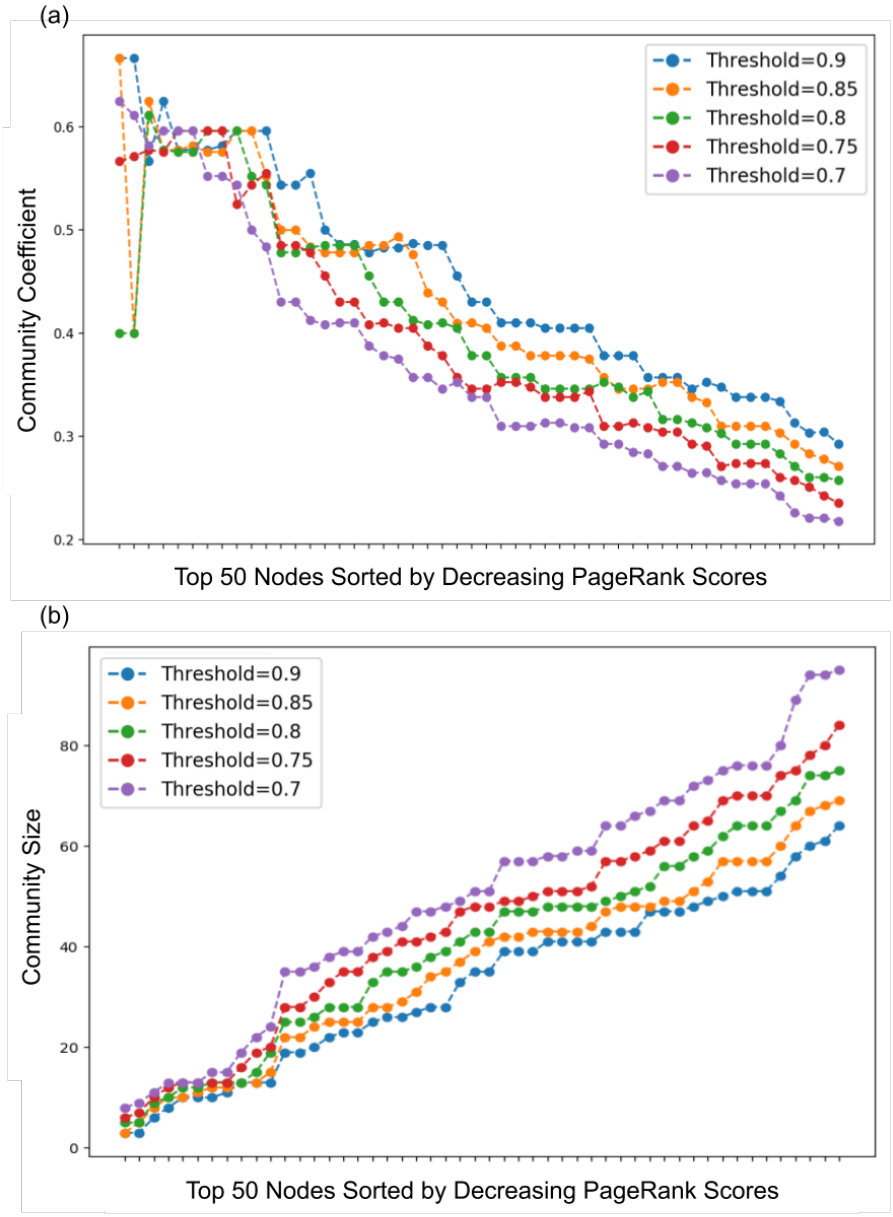


Figure 4.10: Bitcoin OTC dataset: change of (a) community coefficient and (b) community size for top 50 nodes with highest PageRan scores when PageRank threhsold = 0.9, 0.85, 0.8, 0.75 and 0.7

4.4.4 Validation of Hierarchical Communities

In this section, we have validated our approach for finding communities around core nodes and evaluated the goodness of the discovered communities. The basic idea of our validation experiment is to add a new node to an existing community

around a core node and examine if the newly added node improves or degrades the quality of the original community. To assess the connectivity of the nodes within the new community, the community coefficient is re-computed. The new community coefficient can be calculated using the following formula:

$$CC_{new} = \frac{N_{edge} + N'_{edge}}{C(N_{node} + 1, 2) \times 2}, \tag{4.4.1}$$

where N_{edge} is the number of edges between all the nodes of the original community in the graph, N'_{edge} is the number of additional edges introduced by the new node, and N_{node} is the total number of nodes in the original community. If the new community coefficient is increased compared to the old value, it indicates that this new node has strong connections with the nodes in this community; however, if the new community coefficient value is lower than the old one, it implies that the new node has a weak connection to the community and by introducing the new node to the community, the connectivity of the initial community decreases.

The validation experiment has been performed on the Bitcoin Alpha and Bitcoin OTC datasets, as shown in Figure 4.11 and 4.12. For each dataset, we have selected three sets of communities based on the PageRank score of the core node. The blue line is the new community coefficient value and the red line is the new average PageRank score of the community after adding new nodes to the community. The x-axis represents newly added nodes and is ordered by the values of the new computed community coefficient. The two dashed horizontal lines correspond to the community coefficient of the original community (red) and the average PageRank score of all nodes in the original community (blue).

As seen in Figure 4.11, for nodes 4 and 2 with high PageRank scores, their original communities have high values of community coefficient (0.53 for node 4 and 0.67 for node 2) and average PageRank score (0.0090 for node 4 and 0.0082 for node 2). For most of the new nodes outside the original community, they

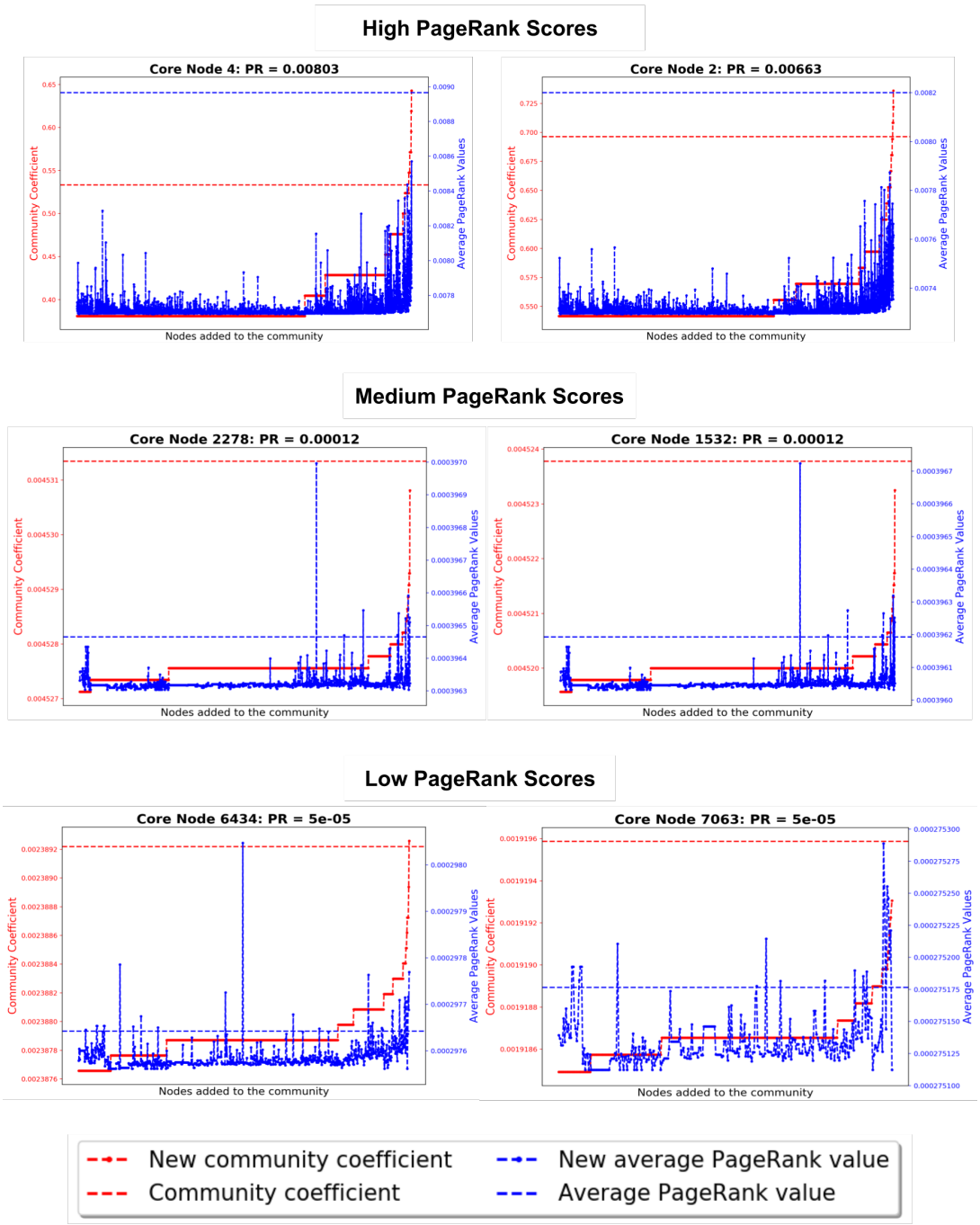


Figure 4.11: Bitcoin Alpha dataset: change of community coefficient and average PageRank score by adding new nodes to the community. The node labels are omitted here.

lower the community coefficient and the average PageRank score when we place them in the community. Although we observe that there exist several nodes that

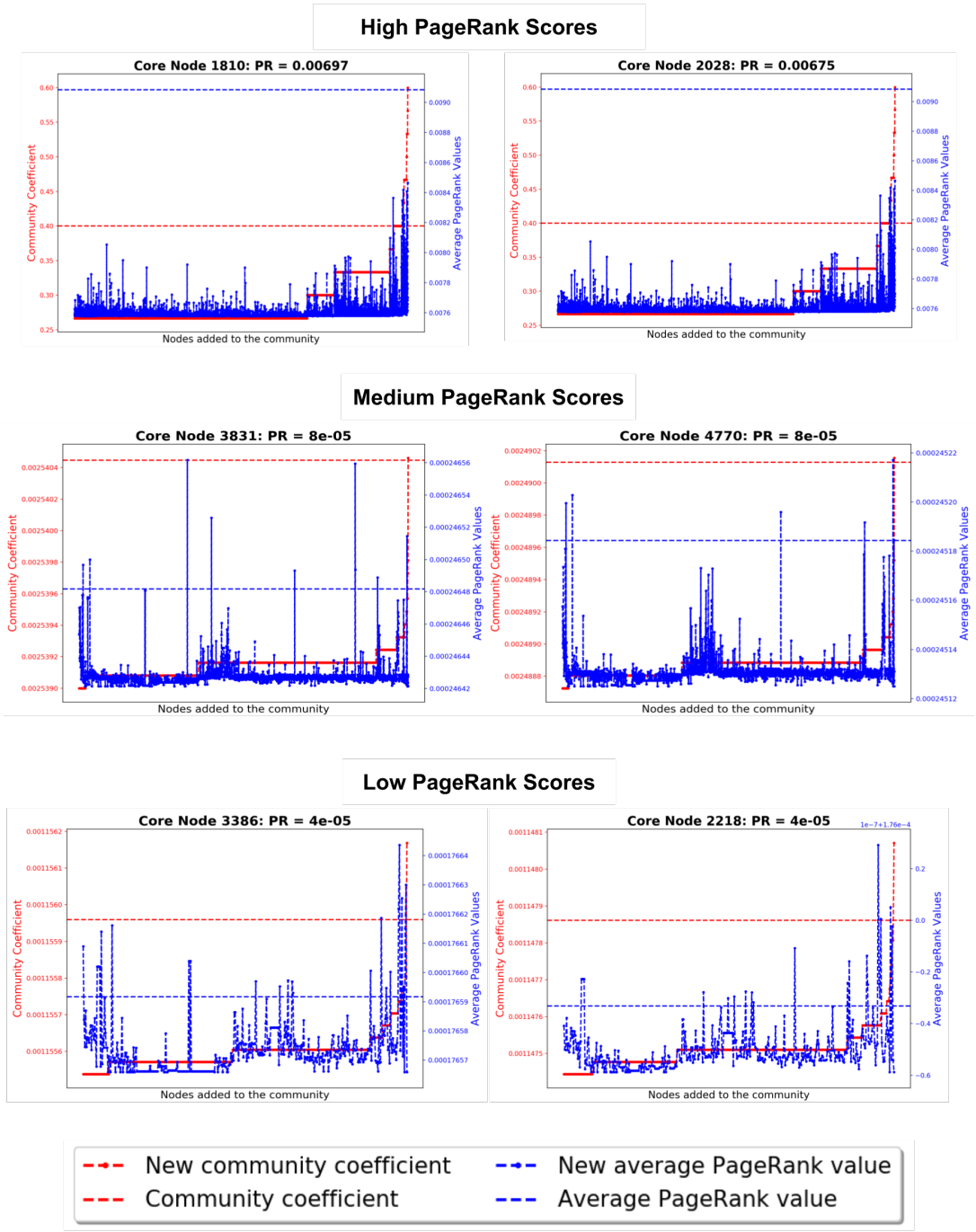


Figure 4.12: Bitcoin OTC dataset: change of community coefficient and average PageRank score by adding new nodes to the community. The node labels are omitted here.

do increase the community coefficient, these nodes have low PageRank scores and decrease the average PageRank score, which fails the PageRank threshold criterion.

For nodes 2278 and 1532 with medium PageRank scores, the addition of new nodes does not improve the community coefficient in both cases. For nodes 6434 and 7063, since they have the lowest PageRank score, the addition of some new nodes helps on the improvement of the average PageRank score of the community. However, almost all the new nodes do not improve the value of the community coefficient (except one node in the plot of node 6434). Similar results can be observed for Bitcoin OTC dataset shown in Figure 4.12. It should be pointed out that these communities are generated by utilizing a PageRank threshold of 0.8. Better quality of communities can always be obtained in terms of the community coefficient value when using higher PageRank thresholds. From the above results, we can conclude that the communities generated from our proposed algorithm have strong connections among the nodes with comparable PageRank scores and weak connections to the nodes outside the community.

4.4.5 Scalability Analysis

In order to evaluate scalability of our approach, we have compared the execution statistics of our algirithm on the five datasets for building nine-level communities when using PageRank threshold $k = 0.8$. To gain further insight of scalability, we have also created two smaller versions of the largest dataset, Supreme Court Citation dataset, by randomly sampling one third and two third of its edges. The Supreme Court Citation dataset originally contains 202,167 edges, thus the two datasets with 67,322 and 134,845 edges (labeled as Supreme-Court-citation_1 and Supreme-Court-citation_2) were created. A comparison of the execution statistics including input data size, the volume of the intermediate data, output data size, number of generated five-level communities, average community length, and total execution time are as shown in Figure 4.13. The intermediate data is the maximum amount of data generated from the Map-Reduce phases. The average community

length is the average number of levels in the generated communities. Note that all data processing was carried out using our cluster with 6 nodes.

	Highschool	Physicians	Bitcoin-Alpha	Bitcoin-OTC	Supreme-Court-citation_1	Supreme-Court-citation_2	Supreme-Court-citation
# of Nodes	70	241	3,783	5,881	26,720	31,557	34,613
# of Edges	366	1,098	24,186	35,592	67,322	134,845	202,167
Input Size	17 KB	54 KB	1.2 MB	1.9 MB	3.9 MB	7.8 MB	11.8 MB
Intermediate Data (upper / lower)	144 KB / 265 KB	574 KB / 1,082 KB	3.2 GB / 2.9 GB	8.7 GB / 7.5 GB	31.8 MB / 93.3 MB	233 MB / 790.3 MB	0.8 GB / 2.9 GB
Output Size (upper / lower)	37 KB / 62 KB	164 KB / 247 KB	229.1 MB / 205.9 MB	581.4 MB / 497.8 MB	29.7 MB / 98.7 MB	267.2 MB / 726.5 MB	792. 5 MB / 1,870 MB
# of Generated Communities	67 / 69	213 / 214	3,740 / 3,273	5,839 / 4,794	16,939 / 17,451	24,812 / 21,656	29,671 / 23,512
Average Community Length	4.37 / 4.52	4.57 / 4.64	4.95 / 4.92	4.97 / 4.91	4.22 / 4.40	4.64 / 4.73	4.79 / 4.83
Total Time (upper / lower)	18 s / 18 s	21 s / 19 s	305 s / 298 s	561 s / 526 s	37 s / 60 s	133 s / 394 s	212 s / 634 s

Figure 4.13: Execution time and data statistics for constructing nine-level communities using PageRank threshold = 0.8. Supreme-Court-citation_1 and Supreme-Court-citation_2 were created by sampling edges from the Supreme-Court-citation dataset. Intermediate data, output data size, number of generated communities, average community length and the total time are recorded for both the upper and lower five-level subgraphs.

From the table in Figure 4.13, we can see that as the size of the dataset increases, the execution time and the size of the intermediate data increase accordingly in most cases. For each dataset, the processing time for generating the upper and lower five-level subgraphs depends on the average community length. It can also be observed that the size of the intermediate data is about 1-15 times of the size of the output data. The size of generated intermediate data plays a vital role in program efficiency. The larger the intermediate data, the longer the execution time. We also notice that for all the Supreme Court Citation datasets, much less intermediate data was generated compared to that of other datasets. For example, although the number of nodes and edges in the Supreme Court Citation dataset are almost 6 times than those in the Bitcoin OTC dataset, the generated intermediate

data and the execution time of upper-level communities for the Supreme Court Citation are much less than those of Bitcoin OTC dataset. Furthermore, 99% and 82% of nodes in the Bitcoin OTC dataset have developed five-level subgraphs in upper and lower parts, respectively; while 29,671 upper-level and 23,512 lower-level subgraphs are generated in the Supreme Court Citation dataset, which count for 86% and 68% of the total number of nodes in the graph, respectively. We can infer from above observations that in the Supreme Court Citation dataset there exist more nodes with very sparse connectivity so that they cannot develop into communities, which results in less intermediate data and thus shorter execution time.

Bitcoin-Alpha dataset			
	5-level	**9-level**	**17-level**
Intermediate Data (upper / lower)	21.3 MB / 21.5 MB	3.2 GB / 2.9 GB	15.3 GB / 19.4 GB
Output Size (upper / lower)	15 MB / 13.7 MB	229.1 MB / 205.9 MB	274.4 MB / 243.8 MB
Average Community Length	2.99 / 2.99	4.95 / 4.92	7.66 / 7.24
Total Time (upper / lower)	25 s / 27 s	305 s / 298 s	2,539 s / 2,125 s

Figure 4.14: Bitcoin Alpha dataset: execution time and data statistics for constructing 5-level, 9-level and 17-level communities (with 3-level, 5-level and 9-level upper and lower subgraphs) when using PageRank threshold = 0.8

To investigate the impact of the community length on the performance of our algorithm, we have also compared the processing time, size of the output data and the intermediate data, and average community length when constructing 5-level, 9-level and 17-level communities in Bitcoin Alpha dataset. From Figure 4.14, we can see that the total execution time increases with increase in the length of

the community. It takes about 30 seconds to build all the 5-level communities, whereas more than 2,000 seconds are required when we build 17-level communities. This is because the construction of longer communities involves more iterations, resulting in more data shuffling and intermediate data. However, we notice that the size of the output data and the average community length do not increase much when building 17-level communities. It can be inferred from the table that the maximum average length of the community that can be generated from the Bitcoin Alpha dataset is around 7 levels. Most of the community will not grow further even when a higher community-length parameter is used.

4.5 Summary

In this chapter, we have described the experiments we utilized to evaluate and validate our algorithm for constructing hierarchical communities around core nodes. We have analyzed and discussed the results obtained from several real-world directed networks. In addition, we have studied the relationship between the PageRank threshold and the size and connectivity of the resulting communities. The validation experiment has demonstrated that our approach is an effective way to discover communities in directed graphs.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

In this thesis, we have proposed a new Map-Reduce based approach for discovering PageRank based hierarchical communities in directed graphs. Most of the previous community detection approaches for directed graphs have ignored the edge directionality and applied methods that were developed for undirected ones. Although some methods do incorporate information on edge directions during clustering, they suffer from poor scalability. To our knowledge, our method is the first scalable Map-Reduce algorithm for community detection in directed graphs that constructs hierarchical structures around core nodes with maintaining edge directions. These hierarchical communities are generated in a three-phase Map-Reduce process. By utilizing the PageRank ranking algorithm, nodes with a similar rank as the core node are considered as the candidate nodes. An upper-level and a lower-level hierarchical subgraphs are built level by level and then merged into a final community centered around the core node. This method is especially suitable for finding communities in the graphs with the structure of the flow hierarchy, such as trust networks, citation networks, and defeat networks. We have successfully applied our algorithm to several real-world networks from various domains including the friendship social network, trust network and legal citation network, and

demonstrated the effectiveness of our method. In addition, we have investigated the effect of the PageRank value of the core node and the PageRank threshold on the size and structure of the generated communities.

5.2 Future Work

Several potential directions for the future research are outlined as follows:

- **Using improved ranking algorithm.** In our study, we treat our sample graphs as unweighted graphs. However, in many real network systems, the linkages between nodes are typically not of the same strength. For example, in a directed trust network of a online transaction platform, an edge from i to j indicates that user i has some trust relationship with user j. A weight can be added to reflect the trust level of user i on user j. In such a case, strong links and weak links play different roles in community formation. In our work, we use the PageRank algorithm to measure the global importance / influence of a node in the graph. In order to incorporate the trustworthiness of a individual by other individuals, an extended PageRank algorithm that considers the edge weight can be used. The weighted PageRank is defined as

$$Weighted\_PR(i) = \frac{1-d}{N} + d\sum_{j=1}^{k} PR(j) \times w_{j,i},$$
$$w_{j,i} = \frac{W_{j,i}}{W_i},$$

  where $W_{j,i}$ is the edge weight of the link (j, i) and $w_{j,i}$ is the ration of edge weight of the link (j, i) to the total edge weight of all incoming links to i. The normalized edge weight $w_{j,i}$ indicates how strongly one node is related another node.

Bibliography

[1] Jianxin Wang et al. "Recent advances in clustering methods for protein interaction networks". In: *BMC genomics* 11.3 (2010), S10.

[2] Gary William Flake et al. "Self-organization and identification of web communities". In: *Computer* 3 (2002), pp. 66–71.

[3] Audun Jøsang, Ross Hayward, and Simon Pope. "Trust network analysis with subjective logic". In: *Proceedings of the 29th Australasian Computer Science Conference-Volume 48*. Australian Computer Society, Inc. 2006, pp. 85–94.

[4] Thomas DuBois, Jennifer Golbeck, and Aravind Srinivasan. "Predicting trust and distrust in social networks". In: *2011 IEEE third international conference on privacy, security, risk and trust and 2011 IEEE third international conference on social computing*. IEEE. 2011, pp. 418–424.

[5] Eugene Garfield, Irving H Sher, and Richard J Torpie. *The use of citation data in writing the history of science*. Tech. rep. INSTITUTE FOR SCIENTIFIC INFORMATION INC PHILADELPHIA PA, 1964.

[6] Filipe Manuel Clemente et al. "Network analysis in basketball: inspecting the prominent players using centrality metrics". In: *Journal of Physical Education and Sport* 15.2 (2015), p. 212.

[7] Raffaele Trequattrini, Rosa Lombardi, and Mirella Battista. "Network analysis and football team performance: a first application". In: *Team Performance Management* 21.1/2 (2015), pp. 85–110.

[8] Serguei Saavedra et al. "Mutually-antagonistic interactions in baseball networks". In: *Physica A: Statistical Mechanics and its Applications* 389.5 (2010), pp. 1131–1141.

[9] Stanley Wasserman, Katherine Faust, et al. *Social network analysis: Methods and applications*. Vol. 8. Cambridge university press, 1994.

[10] Harrison C White, Scott A Boorman, and Ronald L Breiger. "Social structure from multiple networks. I. Blockmodels of roles and positions". In: *American journal of sociology* 81.4 (1976), pp. 730–780.

[11] Gary William Flake, Robert E Tarjan, and Kostas Tsioutsiouliklis. "Graph clustering and minimum cut trees". In: *Internet Mathematics* 1.4 (2004), pp. 385–408.

[12] FR Chung. "Spectral Graph Theory (American Mathematical Society, Providence, RI)". In: (1997).

[13] Fragkiskos D Malliaros and Michalis Vazirgiannis. "Clustering and community detection in directed networks: A survey". In: *Physics Reports* 533.4 (2013), pp. 95–142.

[14] Reid Andersen, Fan Chung, and Kevin Lang. "Local partitioning for directed graphs using PageRank". In: *International Workshop on Algorithms and Models for the Web-Graph*. Springer. 2007, pp. 166–178.

[15] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. "Weighted graph cuts without eigenvectors a multilevel approach". In: *IEEE transactions on pattern analysis and machine intelligence* 29.11 (2007), pp. 1944–1957.

[16] Darong Lai, Hongtao Lu, and Christine Nardini. "Finding communities in directed networks by pagerank random walk induced network embedding". In: *Physica A: Statistical Mechanics and its Applications* 389.12 (2010), pp. 2443–2454.

[17] Venu Satuluri and Srinivasan Parthasarathy. "Symmetrizations for clustering directed graphs". In: *Proceedings of the 14th International Conference on Extending Database Technology*. ACM. 2011, pp. 343–354.

[18] David Gleich. "Hierarchical directed spectral graph partitioning". In: *Information Networks* (2006).

[19] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. "On power-law relationships of the internet topology". In: *ACM SIGCOMM computer communication review*. Vol. 29. 4. ACM. 1999, pp. 251–262.

[20] David Harel and Yehuda Koren. "On clustering using random walks". In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer. 2001, pp. 18–41.

[21] Luh Yen et al. "clustering using a random walk based distance measure." In: *ESANN*. 2005, pp. 317–324.

[22] Reid Andersen, Fan Chung, and Kevin Lang. "Local graph partitioning using pagerank vectors". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE. 2006, pp. 475–486.

[23] Martin Rosvall and Carl T Bergstrom. "Maps of random walks on complex networks reveal community structure". In: *Proceedings of the National Academy of Sciences* 105.4 (2008), pp. 1118–1123.

[24] Morteza Alamgir and Ulrike Von Luxburg. "Multi-agent random walks for local clustering on graphs". In: *2010 IEEE International Conference on Data Mining*. IEEE. 2010, pp. 18–27.

[25] Shayan A Tabrizi et al. "Personalized pagerank clustering: A graph clustering algorithm based on random walks". In: *Physica A: Statistical Mechanics and its Applications* 392.22 (2013), pp. 5772–5785.

[26] Konstantin Avrachenkov et al. "Pagerank based clustering of hypertext document collections". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2008, pp. 873–874.

[27] Mark EJ Newman and Michelle Girvan. "Finding and evaluating community structure in networks". In: *Physical review E* 69.2 (2004), p. 026113.

[28] Mark EJ Newman. "Modularity and community structure in networks". In: *Proceedings of the national academy of sciences* 103.23 (2006), pp. 8577–8582.

[29] Elizabeth A Leicht and Mark EJ Newman. "Community structure in directed networks". In: *Physical review letters* 100.11 (2008), p. 118703.

[30] James Samuel Coleman et al. "Introduction to mathematical sociology." In: *Introduction to mathematical sociology.* (1964).

[31] James Samuel Coleman, Elihu Katz, and Herbert Menzel. *Medical innovation: A diffusion study.* Bobbs-Merrill Co, 1966.

[32] Srijan Kumar et al. "Edge weight prediction in weighted signed networks". In: *Data Mining (ICDM), 2016 IEEE 16th International Conference on.* IEEE. 2016, pp. 221–230.

[33] Srijan Kumar et al. "Rev2: Fraudulent user prediction in rating platforms". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining.* ACM. 2018, pp. 333–341.

[34] James H Fowler et al. "Network analysis and the law: Measuring the legal importance of Supreme Court precedents". In: *Political Analysis* 15.3 (2007), pp. 324–346.