# Community Detection in Networks using PageRank Contributions

by

Liyue Wang

B.Eng., Zhejiang University, 2012

Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences


Spring 2016

## APPROVAL

**Name:** Liyue Wang

**Degree:** Master of Science

**Title:** Community Detection in Networks using PageRank Contributions

**Examining Committee:** **Chair**: Dr. James P. Delgrande
Professor

---

**Dr. Andrei Bulatov**
Senior Supervisor
Professor

---

**Dr. Jian Pei**
Supervisor
Professor

---

**Dr. Qianping Gu**
Internal Examiner
Professor

**Date Approved:** March 8th, 2016

# Abstract

The modern science of networks has made significant contributions to our understanding of complex real world systems. One of the most relevant features of graphs representing the real world networks is their community structure. Therefore, a large body of work in industry and academia has been devoted to identifying community structure in complex networks. In this thesis, we design PC-KM, a variation of $k$-means clustering using PageRank contributions to detect communities in networks. In order to scale to large size networks, we propose another method PPC-KM, which uses random projections to reduce dimensionality while preserving features required for community detection. We also present a fuzzy version of PPC-KM to consider the overlapping communities in networks. We evaluate our algorithms on several datasets. The results show that our methods detect communities with high performance on real world networks.

**Keywords**: Community structure; PageRank contributions; Overlapping community

# Acknowledgements

First of all, I would like to express my deepest appreciation to my senior supervisor, Dr. Andrei Bulatov, who supported and encouraged me throughout the whole process of my thesis. Whenever I ran into a trouble on my research or writing, he was always available for giving me valuable advice and guidance.

Furthermore, I would also like to express my enough gratitude to my supervisor Dr. Jian Pei and my thesis examiner Dr. Qianping Gu, for reviewing my thesis and providing comments. A special thanks to Dr. James O. Delgrande for being the chair of my thesis defence.

Last but not least, many thanks go to my parents, sister and friends for their love and continuous support throughout my years of study.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The huge interdisciplinary effort in the study of networks, has brought significant advances to our understanding of complex real systems. One of the most relevant features of graphs representing the real world networks is their community structure. Therefore, community detection, the division of a graph into well-connected groups of nodes with only sparser connections between groups, has become one of the most important research topics in biology, statistics, economics, applied mathematics, statistical physics, and computer science [23, 50, 64].

Real world networks display big inhomogeneities, revealing a high level of order and organization. Society offers a wide variety of possible group organizations: families, friends, working circles, villages, towns, nations. The diffusion of Internet has also led to the creation of online communities, like reading clubs, movie groups, sports or music teams, etc [25, 41]. Communities also occur in many networked systems from computer science, biology, engineering, economics, politics, etc. In the graph of the World Wide Web they may correspond to groups of pages dealing with similar topics [16, 21]. In protein interaction networks, communities are likely to group proteins serving the same specific collective function within the cell [13, 48, 55]. In the metabolic networks they may be related to functional modules such as cycles and pathways [28, 44]. In food webs they may identify compartments, the subgroups of taxa [33, 45], and so on.

A large body of work has been devoted to community detection in networks, with a wide variety of methods have been applied to this problem. These methods range from the standard graph theory methods such as graph partition algorithms, to PageRank based algorithms, to approach some theoretical results. A remarkable progress has been achieved

in these areas that has allowed us to solve the problem in many important cases. However, although they give reasonable results for community structure in some cases, in other cases they are less successful.

## 1.1 Problem Statement

A large number of real world networks can be modeled as a graph, which is a set of vertices together with a set of edges connecting some pairs of vertices. A community (also referred to as a module or a cluster), is a group of vertices which probably share common properties or play similar roles within the graph. The problem of community detection is to discover the communities in graphs by only using the structural properties of the graphs.

However, the problem is actually not well defined, as the concept of community is not rigorously defined. There is no universally accepted quantitative definition of community. As a matter of fact, the definition often depends on the specific system at hand. Intuitively, a community is a group of nodes with more interaction amongst its members than between its members and the rest of the network, which is the reference guideline at the basis of most community definitions. But many alternative definitions [31, 42, 46] are compatible with it. Therefore, it is not surprising that there are plenty of definitions in the literature and that people do not even try to ground the problem on shared definitions.

Communities in networks also may overlap as nodes belong to multiple clusters at once, in which case one speaks of overlapping communities. As in many real world systems nodes may belong to more than one group. For example in social networks, an individual usually belongs to different organizations at the same time, from work colleagues to sports clubs, etc. Most algorithms of community detection assign each node to a single cluster. For the reason that, overlapping communities introduces the membership of nodes in different communities, which enormously increases the complexity of the problem and there is no standard algorithm to solve it. Therefore, identifying overlapping communities is usually much more computationally demanding and complicated than detecting disjoint communities. And the issue of detecting overlapping communities has become quite popular in the last few years [2, 8, 17].

The existing methods have been applied to a wide range of networks. Nevertheless, identifying meaningful communities in large networks has proven to be a hard and not

yet satisfactorily solved task [24, 36, 63]. Most methods have troubles scaling to large networks, and the lack of reliable performance evaluation of detected communities increases the difficulty. Thus, while networks have been extensively studied, and the existence and properties of communities in small networks is well-understood, it is still not clear how to identify communities in very large networks, which are increasingly common in real word.

In addition, in many cases, the real networks have a precise direction, which needs to be taken into account to understand the organization as a whole. For instance, in the World Wide Web, there are hyperlinks between web pages so that users can move from one page to another. It means the edges between nodes have directions. However, developing algorithms of community detection for directed graphs is a hard task. Only a few techniques can be easily extended from the undirected to the directed case [23].

## 1.2 Applications

In this section, we discuss some applications of community detection. By detecting communities with a set of reliable techniques, one then could proceed with careful investigations of real-world networks. The two primary applications are social networks and biological networks. Community detection also has applications in many other areas like computer science.

In modern society, social networks play a critical role in the way people communicate with each other. Social networking sites, like Facebook (www.facebook.com), Twitter (www.twitter.com), Linkedin (www.linkedin.com), etc. are online platforms, where users as vertices are connected if they are friends. And users can communicate with friends, send e-mails, solicit opinions and spread ideas on the social networking sites. Identifying the community structure of an existing network, where intuitively individuals that belong to the same community, share some similarities and possibly have common interests, or are connected by a specific relationship in the real-world, enables users to make more friends and attend amusing activities to enrich their life, to build and engage with their professional networks to access more insights and opportunities. It also arises in many commercial applications. For example, marketing and competitive intelligence investigations, and recommendation systems. In fact, users belonging to the same community can share tastes or interests in similar products. Social networks sites could send ads more effectively, and

online retailers (such as Amazon, eBay) can set up effective recommendation systems to better guide customers through the list of products and enhance business opportunities by detecting clusters of customers. For some offline companies, such as telecommunications corporations, they also can take advantage of identifying groups of customers based on social networks to support better services. These motivations inspired a large amount of work on community detection of social networks, see e.g., [18, 19, 40].

In addition to social networks, biological networks are also a popular field to apply the community detection techniques. Biological networks are characterized by a remarkable modular organization, reflecting functional associations between their components. For instance, proteins tend to be associated in two types of cellular modules: protein complexes and functional modules. Identifying cellular modules is fundamental to uncover the organization and dynamics of cell functions. With the abundance of protein-protein interaction data produced by genome-scale efforts, it is possible to create a global representation of the protein-interaction network of the yeast cell, representing protein-interaction network as graphs of vertices (relevant proteins) and edges (protein-protein interactions among them). A set of community detection methods have been developed to identify modules in these biological networks [13, 48, 55].

Community detection is also a popular topic in computer science. In parallel computing, it is essential to know what is the best way to allocate tasks to processors. So that we can minimize the communications between them, also enable a rapid performance of the calculation. This can be accomplished by community detection techniques to split the computer cluster into groups [23]. In the World Wide Web, clustering web clients who have similar interests and are geographically close to each other could be served by a dedicated mirror server, which may improve the performance of services. Hence, being able to identify the groups of clients can be helpful to both the web site and the clients [34].

## 1.3 Related Work

In this section, we give a brief review of the previous related works. First we survey the general area of community detection. Then we specifically review some works on PageRank based algorithms used for community detection.

### 1.3.1 Community Detection Algorithms

The first analysis of community structure was carried out by Weiss and Jacobson [58] in 1955, who searched for work groups within a government agency. They separated the work groups by removing the members working with people of different groups, which act as connectors between them. This idea of cutting the bridges between groups is at the basis of several modern divisive algorithms of community detection.

The most popular divisive algorithm is that proposed by Girvan and Newman [26], which is to detect the edges that connect vertices of different communities and remove them, so that the clusters get disconnected from each other. The detected edges are selected according to the values of a measure called betweenness, which is the number of shortest paths between all vertex pairs that run along the edge. This method is very important, which marked the beginning of a new family of algorithms in the community detection problem. Nevertheless, this approach is slow even on the medium size data sets. Secondly it provides no guide to how many communities a network should be split into. Later, a number of authors have proposed modifications of the Girvan and Newman approach. Tyler et al. [56] show that improved speed can be obtained with reasonably small sample sizes of vertices to compute betweenness, which could potentially offer substantial speed improvements over the original algorithm, although it does so at the cost of a reduction in accuracy. Radicchi et al. [46] have proposed another algorithm, which uses a different measure that can be calculated locally instead of betweenness to identify the edges to be removed. It works well for some cases, while is less successful in other cases. To address the second disadvantage, Newman and Girvan [38] proposed that the divisions the algorithm generates be evaluated using a measure they call modularity, which is a numerical index of how good a particular division is. Modularity represents one of the first attempts to achieve a quality measure of the clustering problem and has become an important element of many clustering methods. However, modularity optimization has a resolution limit that may prevent it from detecting clusters which are comparatively small with respect to the graph as a whole, even when they are well defined communities like cliques [10].

Spectral properties of graph matrices are frequently used to find partitions in graphs. In 1973, Fiedler [20] realized that from the eigenvector of the second smallest eigenvalue of the Laplacian matrix it is possible to obtain a bipartition of the graph with very low cut size.

However, the principal disadvantage is that it only bisects graphs. Division into a larger number of communities is usually achieved by repeated bisection, but this does not always give satisfactory results. Early works have also shown that the eigenvectors of the random walk transition matrix of graphs can be used to extract useful information on community structure. Capocci et al. [12] used eigenvector components of the right stochastic matrix, that is the transposed random walk transition matrix. Eigenvectors of the adjacency matrix may be localized as well if the graph has a clear community structure [54]. Donath and Hoffmann [14] proposed an algorithm that uses the eigenvectors of the adjacency matrix for clustering graphs. However, one common disadvantage of these algorithms is that they are too slow, as one needs to compute the whole spectrum of the matrix, which requires a time $O(n^3)$.

The classical techniques to find communities in social networks are hierarchical clustering and partitional clustering, where in both vertices are joined into groups according to their mutual similarity [23, 47, 52]. The vertices are embedded in a metric space, so that each vertex is a point and a distance or similarity measure is defined between pairs of points in the space. In the case when the graph may have a hierarchical structure, i.e. may display several levels of grouping of the vertices, with small clusters included within large clusters, one may use hierarchical clustering algorithms [29]. Once a similarity measure is chosen, it starts from the vertices as separate clusters, and clusters are iteratively merged if their similarity is sufficiently high. It does not require knowing the number of clusters in advance, which is one of the advantages. The procedure also yields a hierarchical structure by construction, which is rather artificial in most cases, since the graph at hand may not have a hierarchical structure at all. Another problem is that vertices with just one neighbor are often classified as separated clusters, which in most cases does not make sense.

For the partitional clustering, one of the most popular partitional technique is $k$-means clustering [30,39,47,51]. Given a data set of $n$ data points, the $k$-means clustering constructs $k(n \geq k)$ groups of the data points, with each partition representing a clusters. The goal is to separate the points in $k$ groups to maximize or minimize a given cost function based on distance or similarity between each pair of data points. Given a data set of data points, the $k$-means clustering aims to separate the data points into $k$ partitions, with each partition representing a cluster, such to maximize or minimize a given cost function based on distance or similarity between each pair of data points. The idea behind this method is simple, easily

understandable and work quite effectively in practical applications. Meanwhile, there are two major weaknesses for using $k$-means clustering in networks. One of the limitation is that the embedding of data points in a metric space can be natural for some graphs, but rather artificial and difficult for others. Another limitation is the expensive computations when run on even medium-size data sets [23, 49].

### 1.3.2 PageRank based algorithms

Most of the methods that have been proposed to detect communities are specifically designed for undirected networks. However, in the real world many networks of interest are directed. As a result, PageRank based algorithms were recently introduced to the area of community detection in complex networks, since they have good performance on analysis link structural of directed graphs.

Jeh and Widom [32] proposed an algorithm for computing personalized PageRank vectors, and can be used to calculate the PageRank contribution that node $u$ makes to other nodes in the graph. Later, Andersen et al. [3] introduced a local partitioning algorithm, for undirected graphs, to find a community around a given seed node by performing a sweep over a personalized PageRank vector, selecting a set that minimizes (or maximizes) some scoring function. One widely used scoring function is conductance. Selecting sets with low conductance tends to produce high quality clusters [63]. Then Andersen et al. generalized the basic local partitioning results from [3] to strongly connected directed graphs in [4]. Avron et al. [7] considered a special case of the time-dependent PageRank vector instead of the personalized PageRank vector in [1]. This algorithm showed that the time-dependent PageRank vector tends to produce smaller and more realistic communities than the ones produced using the personalized PageRank vector, with roughly the same conductance. These local partitioning algorithms could only find a set near a specified given seed vertex. However, in most cases, for the graph at hand we don't know the set of seed vertices for communities at all.

On the other hand, Andersen et al. [4] proposed an algorithm in reverse direction of [32] to calculate contributions received by a target node $v$ from other node of the graph. And the authors presented a local algorithm to compute a PageRank contribution vector representing contributions made to a target node $v$, which enables us to bound the running time required to obtain a fixed level of error. We will refer this type of PageRank contribution as path

contribution. Zhou and Pei [65] consider the concept of page contribution as the difference that the existence or non-existence of node $u$ makes in the PageRank score of node $v$, which is different from the path contribution. Later Shariaty [53] presented a systematic method for calculating page contribution values, and also includes a local algorithm with guaranteed error bounds based on [4]. All those algorithms significantly reduce the computational cost with an acceptable error, to compute PageRank Contributions. Nevertheless, they are used for finding supporting sets (also known as page farms in some works) for the purpose of detecting link spamming. Our work consider using the page contribution vector to represent a vertex in a graph, such that embedding each vertex in a metric space with rich structural information.

## 1.4 Our Algorithm and Contributions

In this thesis, we propose a variation of the $k$-means using PageRank contributions to detect communities in networks, which can apply on variety of networks, direct and undirect networks, small and large size networks, and also consider the overlapping communities.

In the case of networks, clustering (or community detection) problem refers to grouping nodes into clusters according to the structural features of graphs. It is well known that PageRank score is one of the most popular and effective measures to analyse the importance of nodes based on the structural properties of the graphs. Hence, we consider using a vector derived from PageRank score to represent each node in the graph. The vector, which we refer to page contribution vector in this thesis, is the vector whose entries measure the contributions of every node to the PageRank score of a target node. Therefore, it represents informative relations of nodes based on the graph structure. To compute page contribution vectors for each node, we follow the local algorithm approach that has been proposed by Shariaty in [53] which enables us to bound the running time required to obtain a fixed level of error. As the page contribution vector is derived from PageRank, our method can easily apply to both directed and undirected graphs.

Since we use page contribution vectors to represent all nodes of the graph, which embed each node in a metric space. In this way we can easily define a distance or similarity measure between pairs of nodes in the space. Then we can apply various methods to cluster the nodes in the graph. In the first stage, we have tried the spectral clustering, which is one of the most

popular modern clustering algorithms and makes use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. We computed the similarity matrix by cosine similarity on page contribution vectors and tested the spectral clustering on this similarity matrix with several standard benchmark networks. It works well with high accuracy. However, when we want to scale to large networks with millions of nodes, it is slow and does not work sometimes. The likely reason is that, it is difficult and cost expensive to compute the eigenvalues of a large matrix.

As a result, we returned back to consider the classic clustering algorithm, $k$-means. It is one of the most widely used clustering algorithm in scientific and industrial applications. It is simple, easily understandable and reasonably scalable, and can be easily modified to deal with different scenarios. It also has good accuracy on small networks. However, applying the standard $k$-means with PageRank Contributions has the same problem of expensive computation to process large size networks. The mainly computation process of $k$-means is to iteratively calculate the distances between pairs of nodes. For large networks, the number of pairs of nodes is large and each calculation of distance is expensive as the nodes are embedded in an $n$ dimensional vector space, where $n$ is the number of nodes. To deal with this problem, we apply the random projections method described by P. Li in [37] to reduce the dimensionality of the vector for each node, while maintain the structural features for calculating Euclidean distance measures. Therefore we reduce the computational complexity of $k$-means drastically and preserve the quality of page contribution vectors. As expected, this increase in scalability does not hinder accuracy.

Meanwhile, we also consider overlapping communities as it is quite common that a node can belong to more than one community in the real world. In [35], the author showed that the overlap is indeed a significant feature of many real-world social networks. However, only a few of existing methods can detect overlapping communities, which is much more complicated than identifying disjoint communities. In our work, we use fuzzy $k$-means clustering, which makes it possible to assign a node to two or more communities at the same time. So that we can get the probability (degree) of each node belongs to each community, and assign a node belong to the communities with high probability.

Finally, we note that as discussed in Section 1.1, the concept of community is not well defined. There are many different definitions in the literature and accordingly coming up with a set of performance evaluation measures for community detection methods. Applying

any kind of definition or evaluation measure among these is unreliable to detect communities in real world networks. Therefore, instead of using some unreliable performance evaluation measures, we evaluate our method on a set of real world networks with ground truth. Among them are the three standard benchmark networks of small size, whose community structure is already known, and five large social, collaboration, and information networks, whose communities are labelled with ground truth. Through the experiments, we demonstrate that our method detects communities with high quality, as well as in a reasonable running time for both small and large networks.

## 1.5 Thesis Organization

This thesis is organized as follows. In Chapter 2, we introduce necessary preliminaries. Chapter 3 provides a local algorithm for computing the page contribution vectors, followed in Chapter 4 by a variation of $k$-means that use the obtained page contribution vectors. This approach is evaluated on both classical Girvan and Newman's networks and large networks with ground truth in Chapter 5. Finally, a brief conclusion and discussion is given in Chapter 6.

# Chapter 2

# Preliminaries

In this chapter, we introduce some preliminaries that we will use in the rest of this thesis. In Section 2.1, we remind some basic concepts and notation used in this thesis. Section 2.2 presents graph model for networks, and the page rank algorithm is provided in Section 2.3. Finally, the concepts of path contribution and page contribution employed in this work, are briefly introduced in Section 2.4.

## 2.1 Notation

All vectors in this thesis are assumed to be row vectors (e.g. $\boldsymbol{u}$ is a row vector, which makes $\boldsymbol{u^T}$ a column vector). The $i$-th component of a vector is denoted by $\boldsymbol{u}(i)$. We denote $\boldsymbol{e_v}$ as a row vector whose $v$-th entry is equal to 1, and all other entries are 0, and $\mathbf{1}$ as a row vector of all 1's. For each vector, we have its 1-norm and Euclidean norm defined as follows:

**Definition 2.1.1** *(1-Norm [60]). For a vector $\mathbf{u}$, the 1-norm of $\mathbf{u}$ is the sum of the absolute values of all the entries:*

$$||\boldsymbol{u}||_1 = \sum_{i=1}^{n} |\boldsymbol{u}(i)| \tag{2.1}$$

**Definition 2.1.2** *(Euclidean-Norm [60]). For a vector $\mathbf{u}$, the Euclidean-norm of $\mathbf{u}$ is captured by the formula:*

$$||\boldsymbol{u}||_2 = \sqrt{\sum_{i=1}^{n} \boldsymbol{u}(i)^2} \tag{2.2}$$

In the rest of the thesis, we will omit the subscript of 2 for Euclidean-Norm and keep the subscript of 1 for 1-Norm to avoid confusion.

Matrices are denoted in upper case e.g. $A$. The element in the $i$-th row and $j$-th column is denoted by $A_{ij}$. Identity matrix $I$ of size $n$ is the $n \times n$ square matrix with ones on the main diagonal and zeros elsewhere.

**Definition 2.1.3** *(Expectation [59]). Let $X$ be a discrete random variable taking values $x_1, x_2, ..., x_k$ with probabilities $p_1, p_2, ..., p_k$ respectively. Then the expected value of this random variable is:*

$$E(X) = \sum_{i=1}^{k} x_i p_i \tag{2.3}$$

## 2.2 Network as a Graph

Many real-world networks can be modeled as a graph $G = (V, E)$ where $V$ is a set of vertices or nodes and $E$ is a subset of $V^2$, the set of unordered pairs of elements of $V$. The elements of $E$ are called edges or links. If each edge is an ordered pair of nodes then it is a directed graph. In this case an ordered pair $(v, w)$ is an edge directed from $v$ to $w$. In many real examples, graphs are weighted meaning that a real number is associated to each of the edges.

We denote the number of nodes and edges of the graph with $n$ and $m$, namely $|V| = n$ and $|E| = m$. Let $O(v)$ denote the set of out neighbors of node $v$, and $d_{out}(v) = |O(v)|$ be the number of its outgoing edges. We call nodes without outgoing edges: $\{v : d_{out}(v) = 0\} \in V$ as sinks.

We use the definition of induced subgraph suggested in [65], which is different from the conventional definition of induced subgraph in graph theory.

**Definition 2.2.1** *(Induced Subgraph). For a set of vertices $S$, the induced subgraph of $S$ is given by $G(S) = (V, E')$, where $E' = \{v \to u | (v \to u \in E) \wedge (v \in S)\}$. Hence, for the graph $G(V - S)$, is obtained by removing all the edges in $E'$ .*

## 2.3 The PageRank Algorithm

The PageRank Algorithm was firstly used on the web graph $G = (V, E)$, where $V$ are pages and a directed edge $e(i, j) \in E$ represents a link in $i$ that references $j$. It can also apply to other graphs as well.

Let $A$ denote the adjacency matrix of graph $G$, where

$$A_{ij} = \begin{cases} 1 & \text{if } e(i,j) \in E \ , \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

Let $D$ be the diagonal matrix of out-degrees, where

$$D_{ij} = \begin{cases} d_{out}(i) & i = j, \\ 0 & i \neq j. \end{cases} \tag{2.5}$$

Let $W$ denote the random walk transition matrix of $G$, which is given by $W = D^{-1}A$, and can be represented as:

$$W_{ij} = \begin{cases} \frac{1}{d_{out}(i)} & \text{if } e(i,j) \in E \ , \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

To deal with the problem of sinks which will make all PageRank score 0, we assume an artificial node with a single self-loop has been added to the graph, and an edge has been added from each sink to this artificial node.

**Definition 2.3.1** *(PageRank [43]). For a given graph $G = (V, E)$ and a teleportation constant $\alpha$, the PageRank vector of graph $G$ introduced by Page and Brin [43] is defined to be the vector satisfying the following equation:*

$$\boldsymbol{pr}(G) = \alpha \cdot \mathbf{1} + (1 - \alpha) \cdot \boldsymbol{pr}(G) \cdot W, \tag{2.7}$$

*where $\mathbf{1}$ is the row vector of all 1's.*

To calculate the PageRank vector, the PageRank scores for all nodes in a graph, one can assign a random PageRank vector value $\boldsymbol{pr}(G)^{(0)}$ as the initial value, and then apply Equation (2.7) iteratively until the PageRank vector converges as follows:

$$\boldsymbol{pr}(G)^{(t+1)} = \alpha \cdot \mathbf{1} + (1 - \alpha) \cdot \boldsymbol{pr}(G)^{(t)} \cdot W, \tag{2.8}$$

The PageRank score of page $u$ well be denoted $\boldsymbol{pr}(u, G)$. We may remove the graph notation $G$ when there is no confusion between different graphs. Note that the above definition corresponds to the normalization $\sum_u \boldsymbol{pr}(u) = |V|$.

## 2.4 The Path Contribution and Page Contribution

In this subsection, we introduce a definition of page contribution and path contribution, and also the relationship between them.

An intuitive method for computing the contribution of a node $u$ to the PageRank score of a target node $v$, is to neutralize the impact of the contributor node $u$ by removing its out-links from the graph. Then, we can measure the difference of the PageRank score of the target node $v$ with and without the contributor node $u$'s out-links. The formal definition of page contribution based on this idea is as follows.

**Definition 2.4.1** *(Page Contribution [65]). Consider graph $G = (V, E)$, for a target page $v \in V$, the page contribution of page $u \in V$ to the PageRank score of $v$ is*

$$PageCont(u, v) = \begin{cases} \boldsymbol{pr}(v, G) - \boldsymbol{pr}(v, G(V - \{u\})) & u \neq v, \\ \alpha & u = v, \end{cases} \tag{2.9}$$

*where $\alpha$ is the teleportation constant.*

This definition is intuitive, if $u \neq v$, the page contribution of page $u$ to $v$ is the decrease of the PageRank score of page $v$ after we removing page $u$. If $u = v$, the page contribution of page $u$ to $v$ is $\alpha$, which is the minimal value of PageRank score for every page by default. According to this definition, we can compute page contribution by calculating PageRank score for every node we need. However, to calculate PageRank score for each node in graph $G$ and graph $G(V - \{u\})$ is inefficient, especially for large graphs. An alternative way suggested in [65] is to calculate the page contribution through path contribution.

**Definition 2.4.2** *(Path Contribution [65]). Consider graph $G = (V, E)$ and a target page $v \in V$. Let $P = p_0 \to p_1 \to \cdots \to p_n \to v$ be a directed path from $p_0$ to $v$ in the graph. The path contribution to the PageRank of $v$ from $P$ is defined as*

$$PathCont(P, v) = \alpha \prod_{i=0}^{n} \frac{1 - \alpha}{d_{out}(p_i)}, \tag{2.10}$$

*where $d_{out}(p_i)$ is the out-degree of page $p_i$, $\alpha$ is the teleportation constant.*

In this definition, the contribution of each link $p_i \to p_{i+1}$ makes to the PageRank score of node $p_{i+1}$ is $\alpha\frac{1-\alpha}{d_{out}(p_i)}$. As a result the whole contribution for this path $P$ makes to the end node $v$ is $\alpha\prod_{i=0}^{n}\frac{1-\alpha}{d_{out}(p_i)}$.

**Lemma 2.4.3** *( [11]). The PageRank score of $v$ can be calculated using the path contributions as follows:*

$$pr(v, G) = \alpha + \sum_{x \in V(v)} \left( \sum_{P \in \boldsymbol{S_{x \to v}}} PathCont(P, v) \right), \tag{2.11}$$

*where $V(v) = \{x | \text{there is a path from } x \text{ to } v\}$, $V(v) \subseteq V$.*

In the rest of this thesis, we will use the notation $\boldsymbol{S_{x \to v}}$ to denote the set of all paths from node $x$ to node $v$, and $\boldsymbol{S_{x \to u \to v}}$ for the set of all paths from node $x$ to node $v$ that go through node $u$.

In Lemma 2.4.3, the PageRank score of $v$ can be calculated using the path contribution, which we can use for calculating the page contribution in Lemma 2.4.4.

**Lemma 2.4.4** *( [65]) The page contribution of a node $u$ to node $v$ can be calculated through path contributions as follows:*

$$PageCont(u, v) = \sum_{P \in \boldsymbol{S_{u \to v}}} PathCont(P, v) + \sum_{x \in V_u(v)} \sum_{P \in \boldsymbol{S_{x \to u \to v}}} PathCont(P, v). \tag{2.12}$$

*For a specific case that the indegree of $u$ is 0, then $V_u(v) = \emptyset$ and*

$$PageCont(u, v) = \sum_{P \in \boldsymbol{S_{u \to v}}} PathCont(P, v), \tag{2.13}$$

*where $V_u(v) = \{x | \text{there is a path from } x \text{ to } v \text{ through } u\}$, $V_u(v) \subseteq V$.*

Lemma 2.4.4 easily follows Definition 2.4.1 and Lemma 2.4.3. Also, it give a method to use the path contribution to compute the page contribution we want.

# Chapter 3

# Local Computation of Page Contributions

In this chapter, we present a method to calculate the page contribution of nodes to each other. Firstly, we provide a matrix method to calculate page contribution as a series of matrix equations. Then we present an efficient local algorithm proposed by Shariaty [53] to calculate the page contributions, which forms the basis of our clustering algorithm in the next chapter.

## 3.1 A Matrix Method For Calculating Page Contributions

A simple way of estimating the page contribution values defined in the previous section (Equation (2.12)) is to exhaustively enumerate all paths involved in $PageCont(u, v)$ and sum up their contributions. As in [53] Shariaty already proposes an alternative method to calculate it using matrix equations. We restate the result of [53], but represent it in a more clear notation without ambiguity and give shorter and clearer proofs to achieve the same result.

**Lemma 3.1.1** *Consider graph $G = (V, E)$, for $u, v, x \in V$ $(u \neq v)$, let $\boldsymbol{S_{x \to u^1}}$ denote the set of all paths from $x$ to $u$ that visit $u$ exactly once, the sum of the contribution of all paths from all $x$ to $v$ that go through $u$ can be found as:*

$$\sum_{x \in V_u(v)} \sum_{P \in \boldsymbol{S_{x \to u \to v}}} PathCont(P, v) = \sum_{x \in V(u) - \{u\}} \sum_{P_1 \in \boldsymbol{S_{x \to u^1}}} PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot \sum_{P_2 \in \boldsymbol{S_{u \to v}}} PathCont(P_2, v), \quad (3.1)$$

*where $V(u) - \{u\} = \{x | \text{there is a path from } x \text{ to } u, \text{ and } x \neq u\}$, $V(u) - \{u\} \subseteq V$.*

**Proof:** Let $P_1$ be one of the paths $\boldsymbol{S_{x \to u^1}}$ ending at node $u$, $P_2$ be one of the paths $\boldsymbol{S_{u \to v}}$ starting at node $u$. Let $P_1 \cdot P_2$ denotes concatenation of path $P_1$ and $P_2$, which always

concatenates at node $u$. Specifically, let $P_1 = p_{10} \to p_{11} \to \cdots \to p_{1n} \to u$, where $p_{1i} \neq u$, and let $P_2 = u \to p_{20} \to p_{21} \to \cdots \to p_{2m} \to v$. According to Definition 2.4.2, we have:

$$\begin{aligned} PathCont(P_1 \cdot P_2, v) &= \alpha \prod_{i=0}^{n} \frac{1-\alpha}{d_{out}(p_{1i})} \cdot \frac{1-\alpha}{d_{out}(u)} \cdot \prod_{j=0}^{m} \frac{1-\alpha}{d_{out}(p_{2j})} \\ &= \left( \alpha \prod_{i=0}^{n} \frac{1-\alpha}{d_{out}(p_{1i})} \right) \cdot \frac{1}{\alpha} \cdot \left( \alpha \cdot \frac{1-\alpha}{d_{out}(u)} \cdot \prod_{j=0}^{m} \frac{1-\alpha}{d_{out}(p_{2j})} \right) \\ &= PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot PathCont(P_2, v) \end{aligned}$$

Applying this to all paths in $\boldsymbol{S_{x \to u^1}}$ and $\boldsymbol{S_{u \to v}}$, the lemma follows. □

**Corollary 3.1.2** *Consider graph $G = (V, E)$, for $u$, $x \in V$, the sum of the contributions of all paths from all $x$ to $u$ can be found as:*

$$\begin{aligned} &\sum_{x \in V(u) - \{u\}} \sum_{P \in \boldsymbol{S_{x \to u}}} PathCont(P, u) \\ &= \sum_{x \in V(u) - \{u\}} \sum_{P_1 \in \boldsymbol{S_{x \to u^1}}} PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot \left( \sum_{P_2 \in \boldsymbol{S_{u \to u}}} PathCont(P_2, u) + \alpha \right). \end{aligned} \tag{3.2}$$

**Proof:** In Lemma 3.1.1 we replace the target node $v$ with node $u$, then:

$$\begin{aligned} &\sum_{x \in V(u) - \{u\}} \sum_{P \in \boldsymbol{S_{x \to u}}} PathCont(P, u) \\ &= \begin{cases} \sum_{x \in V(u) - \{u\}} \sum_{P_1 \in \boldsymbol{S_{x \to u^1}}} PathCont(P_1, u) & \text{if } u \text{ has no self-loop,} \\ \sum_{x \in V(u) - \{u\}} \sum_{P_1 \in \boldsymbol{S_{x \to u^1}}} PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot \sum_{P_2 \in \boldsymbol{S_{u \to u}}} PathCont(P_2, u) & o.w. \end{cases} \end{aligned}$$

By combining the two cases, this corollary follows. □

**Theorem 3.1.3** *The sum of the contributions of all paths from $x$ to $v$ that go through $u$, can be calculated as:*

$$PageCont(u, v) = \frac{\sum_{P \in \boldsymbol{S_{u \to v}}} PathCont(P, v)}{\sum_{P \in \boldsymbol{S_{u \to u}}} PathCont(P, u) + \alpha} \cdot pr(u, G). \tag{3.3}$$

**Proof:** Following the lemmas above, we can easily prove the theorem as follows: From Lemma 2.4.4, we have:

$$PageCont(u,v) = \sum_{P \in S_{u \to v}} PathCont(P,v) + \sum_{x \in V_u(v)} \sum_{P \in S_{x \to u \to v}} PathCont(P,v)$$

Applying Lemma 3.1.1 and Corollary 3.1.2 to the second term we obtain

$$\sum_{x \in V_u(v)} \sum_{P \in S_{x \to u \to v}} PathCont(P,v) = \sum_{P \in S_{u \to v}} PathCont(P,v) \frac{\sum_{x \in V(u)-\{u\}} \sum_{P \in S_{x \to u}} PathCont(P,u)}{\sum_{P \in S_{u \to u}} PathCont(P,u) + \alpha}.$$

Then

$$\begin{aligned} PageCont(u,v) &= \sum_{P \in S_{u \to v}} PathCont(P,v) \cdot \left( 1 + \frac{\sum_{x \in V(u)-\{u\}} \sum_{P \in S_{x \to u}} PathCont(P,u)}{\sum_{P \in S_{u \to u}} PathCont(P,u) + \alpha} \right) \\ &= \sum_{P \in S_{u \to v}} PathCont(P,v) \cdot \frac{\sum_{x \in V(u)} \sum_{P \in S_{x \to u}} PathCont(P,u) + \alpha}{\sum_{P \in S_{u \to u}} PathCont(P,u) + \alpha} \\ &= \frac{\sum_{P \in S_{u \to v}} PathCont(P,v)}{\sum_{P \in S_{u \to u}} PathCont(P,u) + \alpha} \cdot pr(u,G). \end{aligned}$$

□

In the rest of this chapter, we denote $\boldsymbol{cpr}_v$ as the path contribution vector of node $v$, whose $u$-th entry is equal to the path contribution of node $u$ to a the target node $v$. $\boldsymbol{cpr}_{\boldsymbol{r}}$ as the path contribution vector of vector $\boldsymbol{r}$.

**Theorem 3.1.4** *The path contribution vector $\boldsymbol{cpr}_v$ can be calculated as follows:*

$$\boldsymbol{cpr}_v = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot \boldsymbol{e}_v \cdot (W^T)^t, \tag{3.4}$$

*where*

$$\boldsymbol{cpr}_v(u) = \begin{cases} \sum_{P \in S_{u \to v}} PathCont(P,v) & u \neq v, \\ \sum_{P \in S_{u \to v}} PathCont(P,v) + \alpha & u = v, \end{cases} \tag{3.5}$$

*$W$ is the random walk transition matrix of the graph $G$, and $\boldsymbol{e}_v$ is the row unit vector whose $v$-th entry is equal to 1.*

**Proof:** According to Definition 2.4.2, we have

$$\boldsymbol{cpr}_v(u) = \begin{cases} \sum_{x \in O(u)} \boldsymbol{cpr}_v(x) \frac{1-\alpha}{d_{out}(u)} & u \neq v, \\ \sum_{x \in O(u)} \boldsymbol{cpr}_v(x) \frac{1-\alpha}{d_{out}(u)} + \alpha & u = v. \end{cases} \tag{3.6}$$

Equation (3.6) can be summarized as:

$$\boldsymbol{cpr}_v = \alpha \cdot \boldsymbol{e}_v + (1-\alpha) \cdot \boldsymbol{cpr}_v \cdot W^T \tag{3.7}$$

Hence,

$$\boldsymbol{cpr}_v (I - (1-\alpha) W^T) = \alpha \cdot \boldsymbol{e}_v$$

where $I$ is the identity matrix.

So we have

$$\begin{aligned} \boldsymbol{cpr}_v &= \alpha \cdot \boldsymbol{e}_v \cdot (I - (1-\alpha) W^T)^{-1} \\ &= \alpha \cdot \boldsymbol{e}_v \cdot \sum_{t=0}^{\infty} (1-\alpha)^t \cdot (W^T)^t \\ &= \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot \boldsymbol{e}_v \cdot (W^T)^t. \end{aligned}$$

□

**Corollary 3.1.5** *( [53]). Let $\boldsymbol{gcpr_v}$ be the page contribution vector of node $v$, where $\boldsymbol{gcpr_v}(u) = PageCont(u, v)$. We can obtain $\boldsymbol{gcpr_v}(u)$ using Theorem 3.1.3 as follows:*

$$\boldsymbol{gcpr_v}(u) = \begin{cases} \frac{\boldsymbol{cpr_v}(u)}{\boldsymbol{cpr_u}(u)} \cdot \boldsymbol{pr}(u, G) & u \neq v, \\ \alpha & u = v. \end{cases} \tag{3.8}$$

*where vector $\boldsymbol{cpr}_v$ is defined in Theorem 3.1.4.*

This corollary obviously follows from Theorem 3.1.3 and Theorem 3.1.4.

The complexity of calculating path contribution vectors for each node using matrix multiplications is of $O(T * n^3)$, where $T$ is the number of iterations until convergence, $n$ is the number of nodes. Thus, to compute the page contribution vectors for all nodes we need cost $O(T * n^3)$ which is too expensive for large real networks. The next subsection will introduce a local algorithm to compute it, which enables us to bound the running time required to obtain a fixed level of error.

## 3.2 A Local Algorithm For Calculating Page Contributions

In this part, we present a local algorithm to calculate page contributions, which was proposed by Shariaty [53]

This algorithm consists of two steps. In the first step, it follows the same idea that has been proposed by Anderson et al. [4] to compute the path contribution vector. In the second step, it calculates the page contribution vector from the obtained path contribution vector as in Equation (3.8).

**Definition 3.2.1** *(Approximate Contribution [4]). A vector $\widetilde{\boldsymbol{c}}$ is an $\epsilon$-absolute-approximation of the contribution vector $\boldsymbol{c} = \boldsymbol{cpr_v}$ if $\widetilde{\boldsymbol{c}} \geq 0$ and, for every vertex $u$,*

$$\boldsymbol{c}(u) - \epsilon \leq \widetilde{\boldsymbol{c}}(u) \leq \boldsymbol{c}(u). \tag{3.9}$$

### 3.2.1 Step 1 - Local Algorithm for the Path Contribution Vector

**Algorithm 1** ApproxPathCont($v$,$\alpha$,$\epsilon$) [4]

**Input:** $v$: target vertex; $\alpha$: teleportation constant; $\epsilon$: constant
**Output:** $\widetilde{\boldsymbol{cpr_v}}$

1: Let $\boldsymbol{p} = 0$, and $\boldsymbol{r} = \boldsymbol{e_v}$
2: **while** $\boldsymbol{r}(u) > \epsilon$ for some vertex $u$: **do**
3: Pick any vertex $u$ where $\boldsymbol{r}(u) > \epsilon$
4: Apply **Pushback**($u$,$\boldsymbol{p}$,$\boldsymbol{r}$).
5: **end while**
6: **return** $\widetilde{\boldsymbol{cpr_v}} = \boldsymbol{p}$.

The algorithm **ApproxPathCont($v$,$\alpha$,$\epsilon$)** [4] maintains two nonnegative vectors $\boldsymbol{p}$ and $\boldsymbol{r}$, starting with $\boldsymbol{p} = 0$, and $\boldsymbol{r} = \boldsymbol{e_v}$, and applies a series of pushback operations that increase $||\boldsymbol{p}||_1$ while maintaining the invariant $\boldsymbol{p} + \boldsymbol{cpr_r} = \boldsymbol{cpr_v}$. Each pushback operation picks a single vertex $u$ where $\boldsymbol{r}(u) > \epsilon$, moves an $\alpha$ fraction of the mass at $\boldsymbol{r}(u)$ to $\boldsymbol{p}(u)$, and then modifies the vector $\boldsymbol{r}$ by replacing $\boldsymbol{r}(u)\boldsymbol{e_u}$ with $(1-\alpha)\boldsymbol{r}(u)\boldsymbol{e_u}W^T$.

**Lemma 3.2.2** *(Invariant [4]). Let $\boldsymbol{p'}$ and $\boldsymbol{r'}$ be the result of performing **Pushback**($u$,$\boldsymbol{p}$,$\boldsymbol{r}$) on $\boldsymbol{p}$ and $\boldsymbol{r}$. If $\boldsymbol{p}$ and $\boldsymbol{r}$ satisfy the invariant $\boldsymbol{p} + \boldsymbol{cpr_r} = \boldsymbol{cpr_v}$, then $\boldsymbol{p'}$ and $\boldsymbol{r'}$ satisfy the invariant $\boldsymbol{p'} + \boldsymbol{cpr_{r'}} = \boldsymbol{cpr_v}$.*

From Lemma 3.2.2, we know each pushback operation does indeed maintain the invariant. During each pushback operation, the quantity $||\boldsymbol{p}||_1$ increase by $\alpha\boldsymbol{r}(u)$ and can never

exceed $||\boldsymbol{cpr_v}||_1$, which is equal to $\boldsymbol{pr}(v)$. By performing pushback operations only on vertices where $\boldsymbol{r}(u) > \epsilon$, we can ensure that $||\boldsymbol{p}||_1$ increases by a significant amount at each step, which allows us to bound the number of pushes required to compute an $\epsilon$-absolute-approximation if the contribution vector.

**Algorithm 2** Pushback($u$,$\boldsymbol{p}$,$\boldsymbol{r}$) [4]

**Input:** selected vertex $u$; vector $\boldsymbol{p}$; vector $\boldsymbol{r}$;
**Output:** updated vector $\boldsymbol{p}$; updated vector $\boldsymbol{r}$;
1: $\boldsymbol{p'}(u) = \boldsymbol{p}(u) + \alpha \boldsymbol{r}(u)$
2: $\boldsymbol{r'}(u) = 0$
3: **for** each vertex $w$ such that $w$ links to $u$: **do**
4: $\quad \boldsymbol{r'}(w) = \boldsymbol{r}(w) + (1-\alpha)\boldsymbol{r}(u)/d_{out}(w)$
5: **end for**
6: **return** $\boldsymbol{p} = \boldsymbol{p'}, \boldsymbol{r} = \boldsymbol{r'}$.

**Theorem 3.2.3** *( [4]). The algorithm* ***ApproxPathCont(v,***$\alpha$***,***$\epsilon$***)*** *has the following properties:*

- *The input is a vertex $v$, two constants $\alpha$ and $\epsilon$ in the interval (0,1]. The algorithm computes a vector $\widetilde{\boldsymbol{cpr_v}}$ such that $0 \leq \widetilde{\boldsymbol{cpr_v}} \leq \boldsymbol{cpr_v}$, and $\widetilde{\boldsymbol{cpr_v}}$ is an $\epsilon$-absolute-approximation of $\boldsymbol{cpr_v}$.*

- *The number of pushback operations $P$ performed by the algorithm satisfies the following bound:*

$$P \leq \frac{\boldsymbol{pr}(v)}{\alpha\epsilon}. \tag{3.10}$$

Theorem 3.2.3 shows that the local algorithm can approximate the path contribution vector within a fixed level of error. It also gives an upper bound on the number of pushback operations. The total running time of the algorithm depends on the in-degrees of the sequence of nodes on which the pushback operations were performed. Thus we can have a very loose analysis and say that the average in-degree of the sequence of pushback nodes is $O(\frac{|E|}{|V|})$, then the local algorithm runs in $O(\frac{P|E|}{|V|})$ time.

### 3.2.2 Step 2 - Local Algorithm for Page Contribution Vector

Following Equation (3.8), we can obtain the approximation page contribution vector $\widetilde{\boldsymbol{gcpr}_{\boldsymbol{v}}}$ from the approximation path contribution vector $\widetilde{\boldsymbol{cpr}_{\boldsymbol{v}}}$ above, where:

$$\widetilde{\boldsymbol{gcpr}_{\boldsymbol{v}}}(u) = \frac{\widetilde{\boldsymbol{cpr}_{\boldsymbol{v}}}(u)}{\widetilde{\boldsymbol{cpr}_{\boldsymbol{u}}}(u)} \cdot \boldsymbol{pr}(u). \tag{3.11}$$

**Lemma 3.2.4** *( [53]). Let $v$ and $u$ be two nodes in graph $G = (V, E)$. Then:*

$$\begin{cases} 0 \leq \boldsymbol{cpr}_{\boldsymbol{v}}(u) \leq 1 - \alpha & (u \neq v), \\ \alpha \leq \boldsymbol{cpr}_{\boldsymbol{v}}(u) \leq 1 & (u = v). \end{cases} \tag{3.12}$$

**Theorem 3.2.5** *( [53]). Let $\epsilon$ be the absolute error of approximation $\boldsymbol{cpr}_{\boldsymbol{v}}$ using the algorithm **ApproxPathCont($v$,$\alpha$,$\epsilon$)**. Then the absolute approximation error of approximating $\boldsymbol{gcpr}_{\boldsymbol{v}}(u)$ is $max\{\frac{\epsilon}{\alpha}, \boldsymbol{pr}(u)\frac{\epsilon(1-\alpha)}{\alpha(\alpha-\epsilon)}\}$.*

As the path contribution vector, using the local algorithm to compute the page contribution vector, enables us to bound the required running time with any given error bound.

# Chapter 4

# PageRank Contributions Based Clustering

In this chapter, we firstly present a variation of $k$-means using PageRank contributions as our PC-KM method. It embeds nodes in vector space by representing each node as a page contribution vector. Secondly, in order to scale to large size networks, we propose another method called PPC-KM. In this method, we use random projections to reduce dimensionality while preserving Euclidean distance calculated in $k$-means. Finally, we consider overlapping communities by using a fuzzy version of PPC-KM, which allows nodes to belong to more than one cluster.

## 4.1 PageRank Contributions $k$-means Clustering

The $k$-means clustering algorithm is an iterative method to cluster $n$ data points into $k$ clusters in which each data point belongs to the closest cluster. This algorithm has been discovered by several researchers across different disciplines, most notably by Stuart Lloyd (1957) [38], E.W.Forgy (1965) [22], and McQueen (1967) [39].

Although the intuitive idea behind $k$-means clustering is simple, the successful completion of the tasks depend on a large number of correct decisions and choices from several alternatives. In this section, we discuss several important issues of $k$-means clustering as follows.

### 4.1.1 Data Presentation

Given a set of $d$-dimensional vectors, $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} \in \mathcal{R}^d$ denotes the $i$-th data point, $k$-means clustering algorithm aims to partition the $n$ data points into $k\,(\leq n)$ sets $S = \{S_1, S_2, ..., S_k\}$ with the collection of centers for each set $C = \{\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}\}$.

To apply $k$-means clustering, the data points should be represented as vectors by embedding them into a vector space, so that a distance or similarity measure can be easily defined between pairs of data points. However, the embedding of data points in a vector space can be natural for some datasets, but rather artificial and difficult for others.

In the case of networks, the nodes are the data points to cluster. And the clustering problem refers to grouping nodes into clusters according to the topological features of graphs. There is no natural presentations of nodes in a vector space and it is hard to do embedding based on their topological differences. In this thesis, we consider using the page contribution vectors obtained from the previous chapter, which contains rich structural information of the network, to represent each node in $\mathcal{R}^n$ space.

From the results of previous chapter, we can obtain page contribution vectors $\boldsymbol{x_i} = \boldsymbol{gcpr_i}$ for each node $i$. As a result we have a set of $n$-dimensional vectors, $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} \in \mathcal{R}^n$ is the $i$-th vector of $X$.

### 4.1.2 Objective Function

Basically $k$-means clustering is an iterative process to divide a given data set into $k$ disjoint groups in which each node belongs to the closest cluster. One issue to resolve is how to quantify "closest" in the iterations. The closeness measure normally defined as a distance or similarity between two nodes. And the most commonly used measure is the Euclidean distance. In that case, the $k$-means clustering problem is to partition the $n$ nodes into $k$ clusters to minimize the following objective function:

$$\phi = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij} ||\boldsymbol{x_i} - \boldsymbol{c_j}||^2, \tag{4.1}$$

where

$$r_{ij} = \begin{cases} 1 & \text{if } \boldsymbol{x_i} \in S_j \ , \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

### 4.1.3 The Two Iterative Steps

The $k$-means clustering algorithm is initialized by picking $k$ nodes in $\mathcal{R}^n$ as the initial $k$ cluster representatives or centers $C = \{\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}\}$. Then the algorithm iterates between two steps till convergence:

- **Assignment Step**. Each node is assigned to its closest center. By using Euclidean distance, the assignment process is as follows:

$$S_j^{(t)} = \{\boldsymbol{x_i} : ||\boldsymbol{x_i} - \boldsymbol{c_j^{(t)}}||^2 \leq ||\boldsymbol{x_i} - \boldsymbol{c_l^{(t)}}||^2, \forall l, 1 \leq l \leq k\}, \tag{4.3}$$

  where each $\boldsymbol{x_i}$ is assigned to exactly one $S$, even if it could be assigned to two or more of them.

- **Update Step**. Each cluster representative is relocated to the center of all nodes assigned to it.

$$\boldsymbol{c_j^{(t+1)}} = \frac{1}{|S_j^{(t)}|} \sum_{\boldsymbol{x_i} \in S_j^{(t)}} \boldsymbol{x_i} \tag{4.4}$$

Repeating the assignment and update steps, the algorithm converges when the assignments (and hence the $\boldsymbol{c_j}$ values, $\forall j, 1 \leq j \leq k$) no longer change. During the iterations, the objective function $\phi$ will decrease whenever there is a change in the assignment steps or the update steps, and hence convergence is guaranteed in a finite number of iterations to reach a specified level of error.

Note that each iteration needs $n \times k$ comparisons, which determines the time complexity of one iteration. The number of iterations required for convergence varies and may depend on $n$, but as a first cut, this algorithm can be considered linear in the data set size.

### 4.1.4 Initialization Methods

However, the greedy-descent nature of $k$-means clustering on a non-convex cost function also implies that the convergence is only to a local optimum. There is no guarantee that it will converge to the global optimum, and the result is quite sensitive to the initial centers.

In the standard $k$-means clustering by Lloyd [38], the algorithm selects these initial seeds by sampling at random from the data set. Nevertheless, there are many examples for which the algorithm generates arbitrarily bad clusters.

In this thesis, we apply a variant of $k$-means clustering called $k$-means++, proposed by Arthur and Vassilvitskii [6], that chooses centers at random from the data set, but weighs the nodes according to their Euclidean-norm distance from the closest center already chosen as follows:

Let $SD(x)$ denote the shortest distance from a node to the closest center we have already chosen.

- **Step I-a**. Take one center $\boldsymbol{c_1}$, chosen uniformly at random from the set of nodes $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$
- **Step I-b**. Take a new center $\boldsymbol{c_i}$, choosing $\boldsymbol{x} \in X$ with probability $\frac{SD(\boldsymbol{x})^2}{\sum_{\boldsymbol{x} \in X} SD(\boldsymbol{x})^2}$
- **Step I-c**. Repeat **Step I-b**, until we have taken $k$ centers $C = \{\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}\}$ altogether

**Theorem 4.1.1** *( [6]). Let $C_{OPT}$ denote the optimal clustering and $\phi_{OPT}$ the corresponding cost. If $C$ is constructed with k-means++, then the corresponding objective function $\phi$ satisfies, on the expectation $E[\phi] \leq 8(\ln k + 2)\phi_{OPT}$.*

Selecting centers in the way by $k$-means++ is simple and fast, and it achieves guarantees that the purely random method does not. In practice, using this technique to seed the initial centers always outperforms the random way in both accuracy and speed.

Finally, we obtain our first algorithm of PageRank contributions based $k$-means clustering as PC-KM Algorithm 3. The running time of PC-KM algorithm is $O(in^2k)$, where $i$ is the number of iterations needed until convergence, $k$ is the number of clusters and $n$ is the number of nodes. For each iteration, we have $nk$ euclidean distance computations, which costs $n^2k$ as each node is a $n$-dimensional vector. It is not a problem for medium size networks, but costs too expensive for large real networks. In the next section, we will introduce the PPC-KM algorithm to reduce the time complexity.

## 4.2 Projection PageRank Contributions $k$-means Clustering

The $k$-means clustering can work well on medium size data set, but is very slow when the size scales up to large millions of nodes. In this section, we apply random projections method described by Li in [37] to our PC-KM algorithm to reduce the dimensionality of

**Algorithm 3** PC-KM Algorithm

**Input:** a set of nodes represented as page contribution vectors: $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} = \boldsymbol{gcpr_i}$

**Output:** a set of clusters: $S = \{S_1, S_2, ..., S_k\}$

1: **Step 1-a**. Take one center $\boldsymbol{c_1}$, chosen uniformly at random from the set of nodes $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$

2: **Step 1-b**. Take a new center $\boldsymbol{c_i}$, choosing $\boldsymbol{x} \in X$ with probability $\frac{SD(\boldsymbol{x})^2}{\sum_{\boldsymbol{x} \in X} SD(\boldsymbol{x})^2}$

3: **Step 1-c**. Repeat **Step I-b**. until we have taken $k$ centers $C = \{\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}\}$ altogether

4: **Step 2 (Assignment)**. Each node is assigned to its closest center.
$S_j^{(t)} = \{\boldsymbol{x_i} : ||\boldsymbol{x_i} - \boldsymbol{c_j^{(t)}}||^2 \leq ||\boldsymbol{x_i} - \boldsymbol{c_l^{(t)}}||^2, \forall l, 1 \leq l \leq k\}$

5: **Step 3 (Update)**. Each cluster representative is relocated to the center of all nodes assigned to it.
$\boldsymbol{c_j^{(t+1)}} = \frac{1}{|S_j^{(t)}|} \sum_{\boldsymbol{x_i} \in S_j^{(t)}} \boldsymbol{x_i}$

6: Repeat **Step 2-3**, until it converges and hence the $C = \{\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}\}$ values no longer change

7: **return** a set of clusters: $S = \{S_1, S_2, ..., S_k\}$.

page contribution vectors and speed up the computation of distances between pairs of nodes in $k$-means clustering.

### 4.2.1 JL-embedding

In general, given a high-dimensional set of nodes, it is natural to ask if it could be embedded into a lower dimensional space without suffering great distortion. For this problem, there is a classic result from Johnson and Lindenstrauss [1] as follows:

**Lemma 4.2.1** *(Johnson and Lindenstrauss [1]). Given $\delta > 0$ and an interger $n$, let $k$ be a positive integer such that $k \geq k_0 = O(\delta^{-1} \log n)$. For every set $X$ of $n$ points in $\mathcal{R}^d$ there exists $f : \mathcal{R}^d \to \mathcal{R}^k$ such that for all $u, v \in X$*

$$(1 - \delta)||u - v||^2 \leq ||f(u) - f(v)||^2 \leq (1 + \delta)||u - v||^2. \tag{4.5}$$

This result asserts that any set of $n$ points in $d$-dimensional space can be embedded into $k$-dimensional space, so that all pairwise Euclidean distances are maintained within an arbitrarily small factor. We refer to embeddings providing a guarantee of Lemma 4.2.1 as JL-embeddings.

### 4.2.2 Random Projections

The constructions of JL-embeddings involve projecting the $n$ points onto a spherically random $k$-dimensional hyperplane through the origin. In the last few years, a set of constructions of JL-embeddings have been proposed. As we use Euclidean distances in our PC-KM, it is easy to apply JL-embeddings to reduce dimensionality without significantly affecting its quality. In this thesis, we apply the embedding presented by Li in [37].

In our problem, we have a data matrix $X$ of size $n \times n$ to be a collection of $n$ nodes $\{\boldsymbol{x_i} = \boldsymbol{gcpr_i}\}_{i=1}^{n} \in \mathcal{R}^n$. In each assignment step, we have $n \times k$ pairwise distances to be computed, at the cost of time $O(n^2k)$, which is often prohibitive for large $n$ in real world networks.

To speed up the computations, one can generate a random projection matrix $P \in \mathcal{R}^{n\times r}$ and multiply it with the original matrix $X \in \mathcal{R}^{n\times n}$ to obtain a projected data matrix

$$X' = \frac{1}{\sqrt{r}} XP \in \mathcal{R}^{n\times r}, \quad r \leq n. \tag{4.6}$$

The projected matrix $X'$ is much smaller than the original matrix $X$, and preserves all pairwise distances of $X$ in expectations, provided that $P$ consists of independent and identically distributed (i.i.d) entries with zero mean and constant variance. Thus, we can achieve a substantial cost reduction for computing pairwise distance in each assignment step, from $O(n^2k)$ to $O(r^2k)$.

Here we use projection matrix $P$ with i.i.d entries in

$$p_{ji} = \sqrt{s} \begin{cases} 1 & \text{with probability } \frac{1}{2s}, \\ 0 & \text{with probability } 1 - \frac{1}{s}, \\ -1 & \text{with probability } \frac{1}{2s}, \end{cases} \tag{4.7}$$

where we use $s = \sqrt{n}$ according to work proposed in [37], which achieves a significant $\sqrt{n}$-fold speedup due to only $\frac{1}{\sqrt{n}}$ of the data need to be processed.

**Lemma 4.2.2** *( [37]). Let $\{\boldsymbol{x_i}\}_{i=1}^{n} \in \mathcal{R}^n$ denote the rows in $X$ and $\{\boldsymbol{x'_i}\}_{i=1}^{n} \in \mathcal{R}^r$ the rows of the projected matrix $X'$. Then for arbitrary two nodes $\boldsymbol{x_i}, \boldsymbol{x_j}$ and the corresponding projected nodes $\boldsymbol{x'_i}, \boldsymbol{x'_j}$*

$$E(||\boldsymbol{x'_i} - \boldsymbol{x'_j}||^2) = ||\boldsymbol{x_i} - \boldsymbol{x_j}||^2. \tag{4.8}$$

**Corollary 4.2.3** *Let* $\{\boldsymbol{c_j}\}_{j=1}^{k} \in \mathcal{R}^n$ *denote the original centers and* $\{\boldsymbol{c'_j}\}_{j=1}^{k} \in \mathcal{R}^r$ *the centers after projection. Then:*

$$E(||\boldsymbol{x'_i} - \boldsymbol{c'_j}||^2) = ||\boldsymbol{x_i} - \boldsymbol{c_j}||^2. \tag{4.9}$$

**Theorem 4.2.4** *(Error Bounds [37]). Let* $\{\boldsymbol{x_i}\}_{i=1}^{n} \in \mathcal{R}^n$ *denote the rows in* $X$ *and* $\{\boldsymbol{x'_i}\}_{i=1}^{n} \in \mathcal{R}^r$ *the rows of the projected matrix* $X'$*.*

*If integer* $r \geq r_0 = \frac{4+2\beta}{\delta^2/2-\delta^3/3} \log n$*, then with probability* $1 - \frac{1}{s} \geq 1 - n^{-\beta}$*, for any two nodes* $\boldsymbol{x_i}, \boldsymbol{x_j}$*, we have*

$$(1-\delta)||\boldsymbol{x_i} - \boldsymbol{x_j}||^2 \leq ||\boldsymbol{x'_i} - \boldsymbol{x'_j}||^2 \leq (1+\delta)||\boldsymbol{x_i} - \boldsymbol{x_j}||^2. \tag{4.10}$$

This theorem is derived from Lemma 4.2.1. Here the parameter $\delta$ controls the desired accuracy in Euclidean distance preservation, and $\beta$ controls the projection's probability of success, which enables us to speed up the computations required to obtain a fixed level of error.

Hence, we get the random projection PC-KM algorithm as follows:

**Algorithm 4** PPC-KM Algorithm

**Input:** a set of nodes represented as page contribution vectors: $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} = \boldsymbol{gcpr_i}$

**Output:** a set of clusters: $S = \{S_1, S_2, ..., S_k\}$

1: $X' = \frac{1}{\sqrt{r}} XP \in \mathcal{R}^{n \times r}$

2: **Step 1-a**. Take one center $\boldsymbol{c_1}$, chosen uniformly at random from the set of nodes $X' = \{\boldsymbol{x'_1}, \boldsymbol{x'_2}, ..., \boldsymbol{x'_n}\}$

3: **Step 1-b**. Take a new center $\boldsymbol{c_i}$, choosing $\boldsymbol{x'} \in X'$ with probability $\frac{SD(\boldsymbol{x'})^2}{\sum_{\boldsymbol{x'} \in X} SD(\boldsymbol{x'})^2}$

4: **Step 1-c**. Repeat **Step I-b**. until we have taken $k$ centers $C' = \{\boldsymbol{c'_1}, \boldsymbol{c'_2}, ..., \boldsymbol{c'_k}\}$ altogether

5: **Step 2 (Assignment)**. Each node is assigned to its closest center.
$S_j^{(t)} = \{\boldsymbol{x'_i} : ||\boldsymbol{x'_i} - \boldsymbol{x'^{(t)}_j}||^2 \leq ||\boldsymbol{x'_i} - \boldsymbol{c_l}^{(t)}||^2, \forall l, 1 \leq l \leq k\}$

6: **Step 3 (Update)**. Each cluster representative is relocated to the center of all nodes assigned to it.
$\boldsymbol{c'_j}^{(t+1)} = \frac{1}{|S_j^{(t)}|} \sum_{\boldsymbol{x'_i} \in S_j^{(t)}} \boldsymbol{x'_i}$

7: Repeat **Step 2-3**, until it converges and hence the $C' = \{\boldsymbol{c'_1}, \boldsymbol{c'_2}, ..., \boldsymbol{c'_k}\}$ values no longer change

8: **return** a set of clusters: $S = \{S_1, S_2, ..., S_k\}$.

As we mentioned before, by the random projections, we reduce the time complexity from $O(n^2k)$ to $O(nrk)$ for each iteration, where $r$ is much smaller than $n$. The running time of the whole PPC-KM algorithm is $O(inrk)$. In practice, it works well for large size networks (see Chapter 5 for more details).

## 4.3 Fuzzy Clustering for Overlapping Community Detection

In many real world systems, overlap between groups is indeed a significant feature since it is well understood that each node may belong to more than one group, which results in overlapping communities.

In the case of overlapping community detection, the set of clusters found is called a cover $S = \{S_1, S_2, ..., S_k\}$, in which a node may belong to more than one cluster. Each node $i$ associates with a community according to a belonging factor $[u_{i1}, u_{i2}, ..., u_{ik}]$, in which $u_{ij}$ is a measure of the strength of association (we refer to membership degree in the rest of thesis), between node $i$ and cluster $j$. Without loss of generality, the following constraints are assumed to be satisfied [61]:

$$0 \leq u_{ij} \leq 1, \quad \forall i \in V, \forall j \in S, \tag{4.11a}$$

$$\sum_{j=1}^{k} u_{ij} = 1, \quad \forall i, \tag{4.11b}$$

$$0 < \sum_{j=1}^{n} u_{ij} < n, \quad \forall i. \tag{4.11c}$$

In this section, we modify our PPC-KM to a fuzzy clustering following the idea of fuzzy $c$-means algorithm proposed by Bezdek in [9], which is basically similar in structure to $k$-means clustering.

The fuzzy $c$-means clustering aims to cluster a finite collection of $n$ nodes $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$ into a collection of $k$ fuzzy clusters that can overlap with each other. The objective function that fuzzy $c$-means attempts to minimize is:

$$\phi_m = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^m ||\boldsymbol{x_i} - \boldsymbol{c_j}||^2, \tag{4.12}$$

where $m$ is a weighting exponent with $1 \leq m < \infty$, $C = \{\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}\}$ are viewed as cluster centers, $U = [u_{ij}]$, $u_{ij} \in [0, 1], i = 1, ..., n, j = 1, ..., k$ is the partition matrix where each $u_{ij}$ is a belonging factor subject to the constraints in Equations (4.11), which tells the membership degree of node $x_i$ belongs to cluster $c_j$.

The minimization of the objective function (4.12) represents a nonlinear optimization problem that can be solved by using iterative minimization. The most popular soluction is a Picard iteration through the first order conditions for stationary points of (4.12). If

$m > 1$, $\boldsymbol{x_i} \neq \boldsymbol{c_j}$ for all $i$ and $j$, then $U$ and $C$ may minimize $\phi_m$ only if [9]

$$\boldsymbol{c_j} = \frac{\sum_{i=1}^{n} {u_{ij}}^m \cdot \boldsymbol{x_i}}{\sum_{i=1}^{n} {u_{ij}}^m}, \quad 1 \leq j \leq k, \tag{4.13}$$

and

$$u_{ij} = \frac{1}{\sum_{z=1}^{k} (\frac{||\boldsymbol{x_i} - \boldsymbol{c_j}||}{||\boldsymbol{x_i} - \boldsymbol{c_z}||})^{\frac{2}{m-1}}}, \quad 1 \leq i \leq n, 1 \leq j \leq k. \tag{4.14}$$

The fuzzy $c$-means clustering iterates through Equation (4.13) and (4.14).

The weighting exponent $m$ controls the relative weights placed on each of the Euclidean distance between $\boldsymbol{x_i}$ and $\boldsymbol{c_j}$. As $m \to 1$, the partition becomes increasingly hard ($u_{ij} \in \{0, 1\}$) and $C$ are ordinary means of the clusters. Conversely, partition becomes completely fuzzy ($u_{ij} = \frac{1}{k}$) and the cluster means are all equal to the mean of $X$ as $m \to \infty$. Increasing $m$ tends to degrade membership degree towards the fuzziest state. No theoretical or computational evidence distinguishes an optimal $m$. From observation for most data, $1.5 \leq m \leq 3.0$ gives good results [9]. Usually, $m = 2$ is initially chosen.

Finally, we get the fuzzy PPC-KM Algorithm as follows:

**Algorithm 5** Fuzzy PPC-KM Algorithm

**Input:** a set of nodes represented as page contribution vectors: $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} = \boldsymbol{gcpr_i}$

**Output:** a partition matrix $U = [u_{ij}]$

1: $X' = \frac{1}{\sqrt{r}} XP \in \mathcal{R}^{n \times r}$

2: **Step 1**. Randomly initialize $U^0 = [u_{ij}]$ matrix with constraints in Equations (4.11)

3: **Step 2**. Calculate the centers vectors $C^{(t)} = [c_j]$ with $U^{(t)}$
$\boldsymbol{c_j^{(t)}} = \frac{\sum_{i=1}^{n} {u_{ij}^{(t)}}^m \cdot \boldsymbol{x_i'}}{\sum_{i=1}^{n} {u_{ij}^{(t)}}^m}$

4: **Step 3**. Update $U^{(t+1)}$
$u_{ij}^{(t+1)} = \frac{1}{\sum_{z=1}^{k} (\frac{||\boldsymbol{x_i'} - \boldsymbol{c_j^{(t)}}||}{||\boldsymbol{x_i'} - \boldsymbol{c_z^{(t)}}||})^{\frac{2}{m-1}}}$

5: Repeat **Step 2-3**, until $\max_{ij}\{|u_{ij}^{(t+1)} - u_{ij}^{(t)}|\} \leq \gamma$ , where $\gamma$ is a termination criterion between $[0, 1]$.

6: **return** a partition matrix $U = [u_{ij}]$.

For each iteration, it costs $O(nrk)$ to update the partition matrix $U$. Therefore, the running time of the Fuzzy PPC-KM algorithm is $O(inrk)$, the same as the PPC-KM algorithm. The Fuzzy PPC-KM algorithm works by assigning membership degree to each node corresponding to each cluster center on the basis of distance between the cluster center and the node. The closer a node is to the cluster center, the higher its membership degree

towards this cluster center. When we obtain the final partition matrix $U = [u_{ij}]$, we select a threshold to set nodes to multiple clusters, so that we can evaluate the algorithm based on the ground truth.

# Chapter 5

# Evaluation

In this chapter, we present an evaluation of our algorithms. In Section 5.1, we describe the setup for our experiments including the datasets that we use. Then in Section 5.2, we evaluate the PC-KM method on three small networks. Next, we evaluate the PPC-KM method on five large networks in Section 5.3. Finally in Section 5.4, we give the results of the Fuzzy PPC-KM method.

## 5.1 Setup

The three methods developed in this thesis are applicable for different scenarios. We select two groups of networks of different size to conduct experiments on: a group of three small networks for evaluation of the PC-KM method, and a group of five large networks with ground-truth defined in [63] for evaluation of the PPC-KM and Fuzzy PPC-KM method.

We evaluate the PC-KM method on the three commonly used datasets: the American Football schedule of Division I games for 2000 season [26], the social network of a community of 62 bottlenose dolphins by Lusseau [5], and also the social network of members of the karate club by Zachary [15]. They are standard benchmark in community detection [23]. The karate club dataset consists of 34 nodes, the members of a karate club in the United States. Edges connect individuals who were observed to interact outside the activities of the club. The club split in two, forming two smaller clubs, centered around the administrator and the instructor. The dolphins dataset has 62 dolphins as nodes and edges are set between dolphins that were seen together more often than expected by chance. The network splits naturally into two large groups. The larger subgroup consists almost entirely of females and

the other one almost entirely of males. The American football game dataset represents 115 teams as nodes, and two teams are connected if there was a regular season game between them. The teams are divided into conferences containing around 8 to 12 teams each and teams within the same conference tend to have more games between them.

As examples of large networks, we use five big datasets with ground-truth communities (see Table 5.1) from [63]. In the collaboration network of DBLP, nodes represent authors and edges connect nodes that have co-authored a paper. Research communities stem around conferences or journals. The Amazon product co-purchasing network represent products as nodes and edges connect commonly co-purchased products. Each product belongs to one or more product categories. The Youtube social network, the LiveJournal blogging community, and the Orkut social network are online social networks. Users in these three networks join groups with other users having similar interests, hobbies, affiliations, and geographical regions. In all these datasets, nodes are labeled with their ground-truth community memberships which allow us to quantify the accuracy of community detection methods, instead of using some unreliable performance evaluation measures. Besides, the ground truth communities overlap, since a node can belong to multiple groups at once. As a result, we can apply both the PPC-KM method and the fuzzy PPC-KM method on these datasets. For the PPC-KM, although it only identifies disjoint communities, it can detect the primary community structure in the networks.

| **Dateset** | **N** | **E** | **C** |
|---|---|---|---|
| DBLP | 317,080 | 1,049,866 | 13,477 |
| AMAZON | 334,863 | 925,872 | 151,037 |
| Youtube | 1,134,890 | 2,987,624 | 8,385 |
| LiveJournal | 3,997,962 | 34,681,189 | 287,512 |
| Orkut | 3,072,441 | 117,185,083 | 6,288,363 |

Table 5.1: Datasets statistics. N: number of Nodes, E: number of edges, C: number of communities.

On the five large datasets, we compare the results of PPC-KM and fuzzy PPC-KM with the BIGCLAM (Cluster Affiliation Model for Big Networks) algorithm [62]. The BIGCLAM algorithm is a probabilistic generative model for graphs that captures the organization of networks based on community affiliations. It explicitly models the affiliation strength of each node to each community, which allows for detecting overlapping communities. To compare with the BIGCLAM, we divide the experiments into two parts. The first part is to

evaluate the methods on the subnetworks of each of the five datasets. To obtain one such subnetwork, we firstly follow the way in [62]: pick a random node $v$ in a given graph $G$ that belongs to at least two communities and then select all the nodes that share at least one ground truth community membership with node $v$. The size of subnetwork selected in this way is small, to increase the size, we pick 10 random nodes and add all the nodes that share at least one ground truth community membership with these 10 nodes into a subnetwork. For each of the five large datasets, we created 50 different subnetworks in our experiments. Table 5.2 show the statistics on the average size of the subnetworks for each dataset. The second part is to evaluate the methods on two of the five datasets, Youtube and DBLP in full size. For all experiments in full size networks, we did clean the networks firstly to remove all isolated nodes that do not belong to any communities.

| **Dateset** | **N** | **E** | **C** |
|---|---|---|---|
| DBLP | 4,291 | 12,408 | 280 |
| AMAZON | 2,798 | 8,547 | 176 |
| Youtube | 3,744 | 12,672 | 301 |
| LiveJournal | 6,025 | 127,440 | 235 |
| Orkut | 7,324 | 116,633 | 35 |

Table 5.2: Subnetworks statistics on average. N: number of Nodes, E: number of edges, C: number of communities.

We run our experiments on several machines equipped with Intel CPUs of frequency ranging from 2.40 to 2.53 GHz. Each of our experiments executed single-threaded on a single core of one machine. The machines have up to 16 GB of memory. Moreover, all the methods are implemented in Java and Python. We use Java to implement the computation of page contribution vectors, and use Python to implement the $k$-means and fuzzy $k$-means part.

In all of our experiments, we use a value of 0.15 for the teleportation constant $\alpha$ (see Chapter 2), which is the common value used in the PageRank algorithm. We use a value of $\epsilon = 0.01$ (see Chapter 3) for computation of page contribution vectors on large datasets as has been suggested in the literature [53]. It shows that the choice of $\epsilon = 0.01$ provides reasonable accuracy while also resulting in reasonable running times.

## 5.2 Evaluation of the PC-KM method

In this section, we describe the experimental results of thw PC-KM method on the three commonly used small size networks, and compare the results with Newman's algorithm [26, 42].

### 5.2.1 Karate Club Network

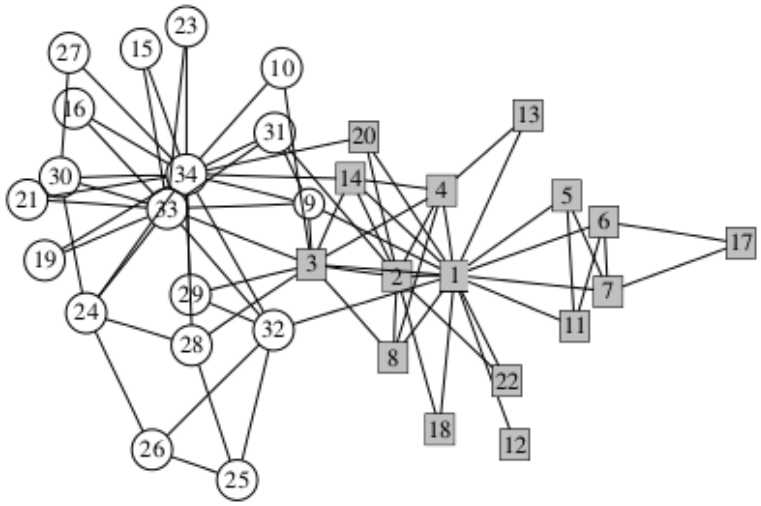



Figure 5.1: The network of friendship in the karate club of Zachary [15]. When this club splits, the right cluster unites with node 1 (the administrator) while the left cluster unites with node 33 (the instructor). The only node PC-KM misclassified is 3.

PC-KM does well in Karate club datasets as it manages to classify 33 nodes out of total 34 nodes. Newman used two community detection methods for this network [42]. His shortest-path version method has the same result with ours while his random-walk version has none misclassified nodes. The only misclassified node is node 3. Each cluster has equally 5 links connected to node 3. Therefore, it has similar connection to both clusters and is hard to classify.

### 5.2.2 Dolphins Network

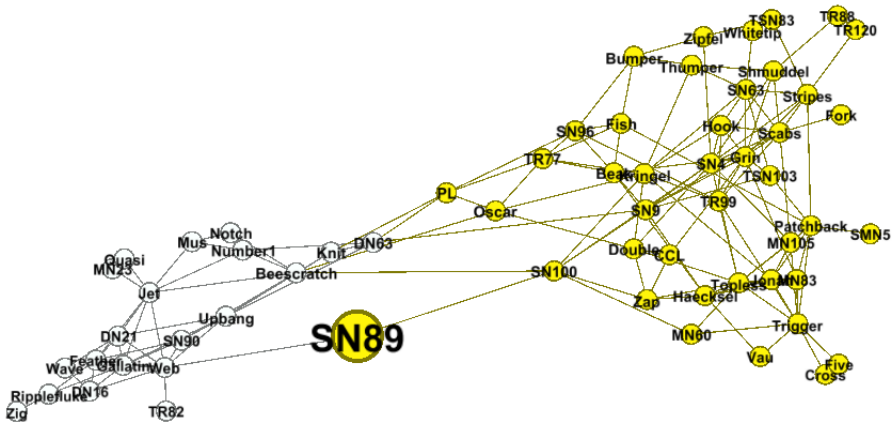



Figure 5.2: Community structure in bottlenose dolphin community. The large cluster stands for females and the small cluster stands for males. The only node misclassified by PC-KM is SN89.

PC-KM successfully classifies 98.4% nodes into two clusters. The only misclassified node is node SN89. Both clusters connect SN89 using only one link, so there is some uncertainty to classify it. Also, our result is the same as the result from Newman using shortest-path version algorithm [42].

### 5.2.3 American Football Game Network

Figure 5.3: The left graph shows true community for each team and the right graph shows clustering result using PC-KM, which successfully detects most of the conference structure in the network.

Newman's edge betweenness community detection result [26] misclassified 10 nodes and PC-KM also misclassified 10 nodes with the same accuracy of 91.3%. The community with the worst result is cluster 5 (IA Independents Conference, the five red cycle nodes) as 80% teams in IA are classified into other clusters. Another bad result happens in cluster 11 (Sunbelt conference, the seven brown square nodes) as nearly half of its teams are grouped into cluster 5. The reason for this is that teams in both conferences have played almost as many games to teams in other conferences as games between themselves.

## 5.3 Evaluation of the PPC-KM method

In this section, we describe the experimental results of the PPC-KM method on five large networks. First, we report the running time by our method. Then as these five datasets have labelled ground truth, we evaluate the accuracy of the community detection methods

by four evaluation measures suggested in [62], which are all commonly used measures of accuracy.

As discussed in Section 5.1, we first apply the PPC-KM method on the subnetworks of the five datasets. Then run experiments on two of the five datasets in full size, which are Youtube and DBLP. Finally we compare all our results against the BIGCLAM method [62].

### 5.3.1 Evaluation Measures

For evaluation we use measures that quantify the level of correlation between the detected and the ground-truth communities [62].

Given a graph $G(V, E)$ modeling the network, we consider a set of ground truth communities $C$ and a set of detected communities $C^{'}$ where each ground truth community $C_i \in C$ and each detected community $C_i^{'} \in C^{'}$ is defined by a set of its member nodes. To quantify the level of correlation of $C^{'}$ to $C$ we consider:

1. **Average F1 Score [62]** is the average of two F1 score: the average F1 Score of the best mapping detected community to each ground truth community, and the average F1 Score of the best mapping ground-truth community to each detected community.

$$F1(C, C^{'}) = \frac{1}{2}(\frac{1}{|C|}\sum_{C_i \in C} F1(C_i, C^{'}_{g(i)}) + \frac{1}{|C'|}\sum_{C_i^{'} \in C^{'}} F1(C_i^{'}, C_{g'(i)})),$$

where the best mapping $g$ is defined by:

$$g(i) = \operatorname*{argmax}_j F1(C_i, C_j^{'}),$$

for each ground truth community $C_i$, we map it to the detected community $C_j^{'}$ that has the maximum F1 score with $C_i$. Likewise for $g'$:

$$g'(i) = \operatorname*{argmax}_j F1(C_i^{'}, C_j),$$

where each F1 score of ground truth community $C_i$ and detected community $C_j^{'}$, we compute as follows:

$$\begin{aligned} F1(C_i, C_j^{'}) &= 2 * \frac{Recall(C_i, C_j^{'}) * Precision(C_i, C_j^{'})}{Recall(C_i, C_j^{'}) + Precision(C_i, C_j^{'})} \\ &= 2 * \frac{\frac{a}{a+c} * \frac{a}{a+b}}{\frac{a}{a+c} + \frac{a}{a+b}} \\ &= \frac{2a}{2a+b+c}. \end{aligned}$$

where $a$ is the number of nodes that both in $C_i$ and $C_j^{'}$, $b$ is the number of nodes that only in $C_j^{'}$, $c$ is the number of nodes that only in $C_i$.

2. **Average Recall [62]** is the average Recall of the best mapping detected community to each ground truth community:

$$Recall(C, C^{'}) = \frac{1}{|C|} \sum_{C_i \in C} Recall(C_i, C^{'}_{g(i)})$$

where $Recall(C_i, C^{'}_{g(i)})$ is the recall of $C_j^{'}$ under the best mapping $g$.

3. **Omega Index [27]** is the accuracy on estimating the number of communities that each pair of nodes shares:

$$\Omega(C, C^{'}) = \frac{1}{|V|^2} \sum_{u,v \in V} 1\{|C_{uv}| = |C^{'}_{uv}|\}, \quad \Omega(C, C^{'}) \in [0, 0.5),$$

where $C_{uv}$ is the number of ground truth communities that $u$ and $v$ share, $C^{'}_{uv}$ is the number of detected communities that they share.

4. **Normalized Mutual Information [57]** adopts the criterion used in information theory to compare the ground truth communities and the detected communities:

$$NMI(C, C^{'}) = \frac{MI(C, C^{'})}{\sqrt{H(C)H(C^{'})}},$$

where the entropy of $C$ is defined by:

$$H(C) = \sum_{i=1}^{|C|} P(i) \log(P(i)),$$

where $P(i) = \frac{|C_i|}{n}$ is the probability that a node picked at random from $C$ falls into cluster $C_i$. Likewise for $C'$:

$$H(C') = \sum_{j=1}^{|C'|} P'(j) \log(P'(j)),$$

where $P'(j) = \frac{|C'_j|}{n}$ is the probability that a node picked at random from $C'$ falls into cluster $C'_j$. The mutual information (MI) between $C$ and $C'$ is calculated by:

$$MI(C, C') = \sum_{i=1}^{|C|} \sum_{j=1}^{|C'|} P(i,j) \log(\frac{P(i,j)}{P(i)P'(j)}),$$

where $P(i,j) = \frac{|C_i \cap C'_j|}{n}$ is the probability that a node picked at random falls into both classes $C_i$ and $C'_j$.

### 5.3.2 Running Time

We compare the running time on three graphs of different size, the Youtube graph of full size, a subgraph of DBLP, and a subgraph of Amazon. The graphs size are shown in Table 5.3. We plot in Figure 5.4 the time taken by PPC-KM for these three graphs. The figure shows that PPC-KM takes between 18.5 minutes to 30.9 minutes on the graph of size of 15,880 nodes. Besides, as expected, a longer time is taken for the graph of larger size.

We also compare the running time on different value of $\delta$. As shown in Figure 5.4, the PPC-KM method takes longer time with smaller $\delta$, which controls the desired accuracy in Euclidean distance preservation (see Chapter 4). This is also expected, because a smaller $\delta$ means a fewer dimensions reduced on the data and causes more computation during the iteration steps in the clustering process. The time for the median value $\delta = 0.6$, is about 0.48 hour on Amazon-sub graph, is 6.5 hours on DBLP graph, and is 24.25 hours on Youtube-full graph.

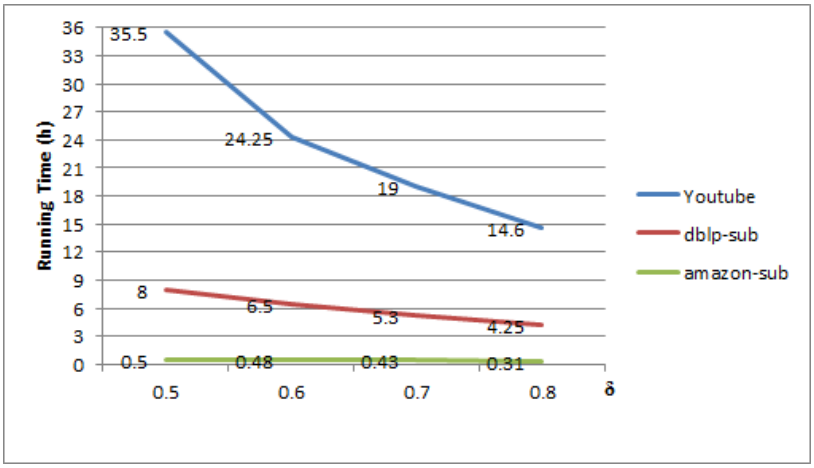



Figure 5.4: Runing Time on different size graphs

| **Dateset** | **N** | **E** | **C** |
|---|---|---|---|
| AMAZON-sub | 15,880 | 29,109 | 1,209 |
| DBLP-sub | 85,682 | 338,632 | 2,684 |
| Youtube-full | 1,134,890 | 2,987,624 | 8,385 |

Table 5.3: Three graphs statistics. N: number of Nodes, E: number of edges, C: number of communities.

As mentioned before, all our experiments executed single-threaded on a single core of one machine. On the one hand, we do not have powerful machine to process all the networks in full size. On the another hand, we didn't extend our methods to distribution algorithms to parallelize and speed up the process. However, we can do both to decrease the running time in the future. Comparably, the BIGLAM algorithm [62] can process the larger network in parallel. But it still took about one day with 20 threads on the LiveJournal network. It somehow shows that the running time of our method is reasonably acceptable with limited machine on a single core.

### 5.3.3 Accuracy on Subnetworks

We evaluate the PPC-KM method on 50 subnetworks of each of the five datesets and evaluate the accuracy using three measures we mentioned before: F1 score, Omega Index and Normalized Mutual Information (NMI for short). The size of each subnetwork is thousands of nodes, which is not too large, so we set $\delta = 0.5$ with more dimensions retained.

Table 5.4 shows the performance of PPC-KM over subnetworks of the five networks, which shows our method performs well in terms of the quality of detected communities. Each value of each measure for each dataset is averaged over all 50 subnetworks. For example, the average F1 score of 50 subnetworks of DBLP is 0.7540, the Omega Index is 0.4302 and the NMI is 0.5880.

| **Dataset** | **F1 Score** | **Omega Index** | **NMI** |
|---|---|---|---|
| DBLP | 0.7540 | 0.4302 | 0.5880 |
| AMAZON | 0.7205 | 0.4812 | 0.7595 |
| Youtube | 0.4683 | 0.4467 | 0.4364 |
| LiveJournal | 0.8892 | 0.4708 | 0.7799 |
| Orkut | 0.6198 | 0.4823 | 0.3996 |
| Average | 0.6904 | 0.4622 | 0.5926 |

Table 5.4: Averge Accuracy of thw PC-KM method over the subnetworks of five datasets

| **Methods** | **Average F1 Score** | **Average Omega Index** | **Average NMI** |
|---|---|---|---|
| BIGCLAM | 0.60 | 0.47 | 0.22 |
| PPC-KM | 0.69 | 0.46 | 0.59 |
| Improvement | 15.0 % | -2.1 % | 168% |

Table 5.5: Comparison Results between BIGCLAM and our PPC-KM method

Table 5.5 displays the comparison results between BIGCLAM [62] and the PPC-KM method. The Average F1 Score of our method is 0.69, which is 15.0% higher than that of BIGCLAM, the average Omega Index of our method is 0.46, which is 0.01 lower than that of BIGCLAM, the average NMI of our method is 0.59, which is 168% higher than that of BIGCLAM. The result show that our method achieves better accuracy on F1 Score and NMI, a little bit lower on Omega Index. Since the maximal value of Omega Index is 0.5, both 0.46 and 0.47 are high value.

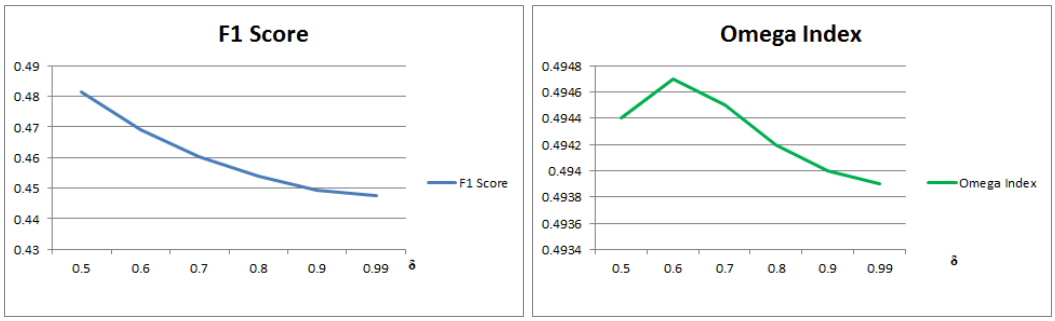
### 5.3.4 Accuracy on Full Networks

We evaluate the PPC-KM method on two of the five datasets with full size and quantify the accuracy by the three measures mentioned before: F1 Score, Omega Index and Recall.

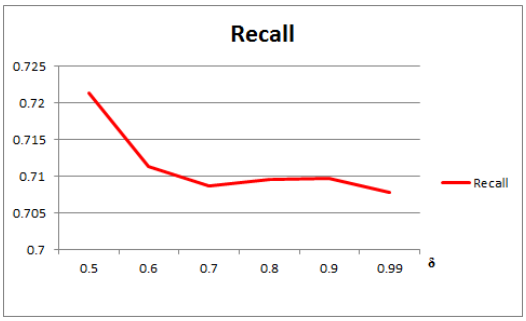
The size for these two datasets is millions of nodes, which is too large to run on a single core of a limited machine, so we first run experiments with different $\delta$ on Youtube dataset. Figure 5.5 shows that, each measure becomes lower with larger $\delta$, except the Omega Index with $\delta = 0.6$. However, it is only 0.002 difference, which is almost the same with different $\delta$. This is expected, because a larger $\delta$ means fewer dimensions retained on the data and causes less computation during the clustering process. Although the smaller $\delta$ the higher accuracy of our method. We have only limited machine to do the experiments. Doing trade-offs between speed and accuracy, we select $\delta$ with 0.8 for our following experiments.

Table 5.6 displays the accuracy over Youtube dataset and DBLP dataset with $\delta = 0.8$. The average F1 score is 0.3957, the average Omega Index is 0.4797, and the average Recall is 0.6788. The value of the average score of BIGCLAM is 0.13 for average F1 Score, 0.11 for average Omega Index, and 0.32 for average Recall on four datasets (DBLP, Youtube, AMAZON, LiveJournal), which somehow shows that our PPC-KM method achieves high performance. To note that, the BIGCLAM is on four datasets, and our average accuracy is on two datasets, since we cannot process AMAZON and LiveJournal with $\delta = 0.8$ on a single core of a limited server. We only process the AMAZON dataset with a projected dimension $r = 50$, and the accuracy is 0.3312 (F1 Score), 0.4325 (Omega Index), and 0.6833 (Recall). The accuracy of our method maybe not much higher than BIGCLAM, however, we can get higher accuracy with smaller $\delta$ if we have powerful machine to run all possible experiments.

Although we choose $\delta = 0.8$ here to ensure a high accuracy. From Figure 5.5, we can see that even with $\delta = 0.99$ the accuracy is lower but not disastrous. By Theorem 4.2.4, we know that determining the projected dimension $r$ with the restriction ($\geq r_0$) ensures an error bound and obtains a specified level of accuracy controlled by $\delta$. However, in practice, for some specific applications, we can choose a smaller $r$ beyond the restriction, as long as the accuracy is acceptable for the real applications.




(a) F1 Score Results (b) Omega Index Results




(c) Recall Results

Figure 5.5: Three measures on Youtube dataset with different $\delta$

| Dataset | F1 Score | Omega Index | Recall |
|---|---|---|---|
| DBLP | 0.3377 | 0.4653 | 0.6482 |
| Youtube | 0.4538 | 0.4942 | 0.7095 |
| Average | 0.3957 | 0.4797 | 0.6788 |

Table 5.6: Accuracy of our PPC-KM method over the two datasets with full size. Here we select $\delta = 0.8$

## 5.4 Evaluation of the Fuzzy PPC-KM method

In this section, we evaluate the Fuzzy PPC-KM method to detect overlapping communities. For each dataset, our Fuzzy PPC-KM method returns a partition matrix $U = [u_{ij}]$, where each $u_{ij}$ tells to what membership degree the node $x_i$ bel ongs to cluster $c_j$ (see Section 4.3). In order to evaluate the accuracy, we assign each node to the clusters with membership degree larger than $\frac{1}{k}$, where $k$ is the number of clusters. We then compare the results against the PPC-KM method which only detect disjoint clusters, and the BIGCLAM method which also consider overlapping communities.

Figure 5.6 shows that the accuracy of the Fuzzy PPC-KM method is higher than that of the PPC-KM method on Omega Index and F1 Score as expected. This is because the PPC-KM assigns each node to only one cluster and does not consider overlapping communities. But they are very close, the likely reason is that although the number of overlapping communities is large, the overlap section of each community is small. It is also higher than the BIGCLAM method, which shows our method is powerful.

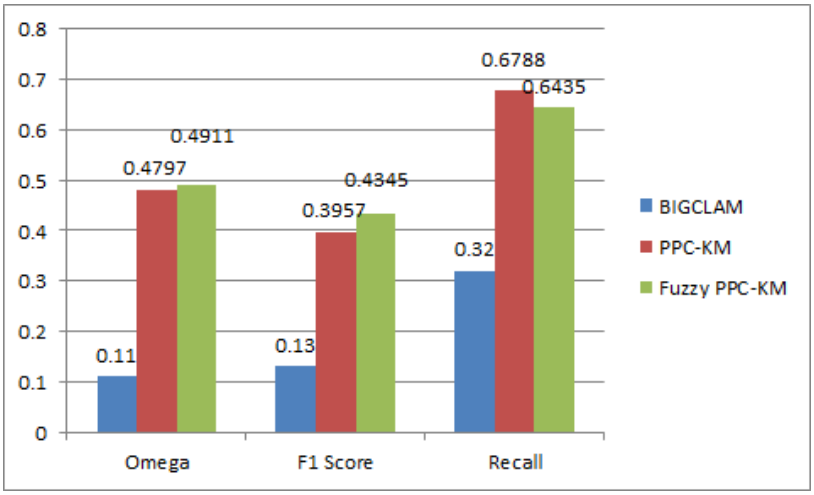



Figure 5.6: Comparison results bewteen BIGCLAM, PPC-KM and Fuzzy PPC-KM method

# Chapter 6

# Conclusion and Future Work

In this thesis, we considered the community detection problem and analyzed several difficulties one encounters when solving this problem. Firstly, we calculated page contribution vectors, which is derived from PageRank score of graphs, and used it to embed each node in a vector space. In this way we can easily define a distance or similarity measure between pairs of nodes in the vector space. Then we developed our first algorithm PC-KM, which is a variation of the standard $k$-means based on the page contribution vectors. The second algorithm PPC-KM, improve the performance of the PC-KM by applying random projections to reduce dimensions of data and save both memory and running time. It is therefore the recommended choice for large graphs. Finally, we also consider overlapping communities along with the third algorithm Fuzzy PPC-KM that assigns a node to more than one communities. We evaluate our algorithms on a set of real world networks: three small commonly used graphs and five large graphs with ground-truth. The experimental results demonstrate that our algorithms have high accuracy in detecting communities, as well as a reasonable running time to process large graphs.

Our work can be used as an efficient algorithm for the community detection problem. As one of the directions for future research, we would like to extend our algorithms to parallel versions so that we can process large graphs on regular PCs with limited performance.

Furthermore, as benchmark datasets with large size are in short supply, we want to explore more large graphs in real word including graphs without ground truth. To evaluate our work on graphs without ground truth, there exists a set of performance evaluation measures for community detection methods. However, there is no standard evaluation measure. We can also consider the evaluation measures problem by ourselves. Meanwhile,

without ground truth, the number of communities $k$ will be a problem in our algorithms. To determine $k$, we can increase the value of $k$ from a small value until the community structure does not improve as measured by the evaluation measures.

# Bibliography


[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003. 27

[2] Balázs Adamcsek, Gergely Palla, Illés J Farkas, Imre Derényi, and Tamás Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006. 2

[3] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 475–486. IEEE, 2006. 7

[4] Reid Andersen, Fan Chung, and Kevin Lang. Local partitioning for directed graphs using pagerank. In *Algorithms and Models for the Web-Graph*, pages 166–178. Springer, 2007. 7, 8, 20, 21

[5] Alex Arenas, Alberto Fernández, and Sergio Gómez. Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics*, 10(5):053039, 2008. 33

[6] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. 26

[7] Haim Avron and Lior Horesh. Community detection using time-dependent personalized pagerank. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1795–1803, 2015. 7

[8] Jeffrey Baumes, Mark Goldberg, and Malik Magdon-Ismail. Efficient identification of overlapping communities. In *Intelligence and Security Informatics*, pages 27–36. Springer, 2005. 2

[9] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2):191–203, 1984. 30, 31

[10] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):172–188, 2008. 5

[11] Michael Brinkmeier. Pagerank revisited. *ACM Transactions on Internet Technology (TOIT)*, 6(3):282–301, 2006. 15

[12] Andrea Capocci, Vito DP Servedio, Guido Caldarelli, and Francesca Colaiori. Detecting communities in large networks. *Physica A: Statistical Mechanics and its Applications*, 352(2):669–676, 2005. 6

[13] Jingchun Chen and Bo Yuan. Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics*, 22(18):2283–2290, 2006. 1, 4

[14] William E Donath and Alan J Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973. 6

[15] Luca Donetti and Miguel A Munoz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2004(10):P10012, 2004. viii, 33, 36

[16] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In *Proceedings of the 16th international conference on World Wide Web*, pages 461–470. ACM, 2007. 1

[17] Nan Du, Bai Wang, Bin Wu, and Yi Wang. Overlapping community detection in bipartite networks. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 176–179. IEEE Computer Society, 2008. 2

[18] Emilio Ferrara. Community structure discovery in facebook. *International Journal of Social Network Mining*, 1(1):67–90, 2012. 4

[19] Emilio Ferrara. A large-scale community structure analysis in facebook. *EPJ Data Science*, 1(1):1–30, 2012. 4

[20] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973. 5

[21] Gary William Flake, Steve Lawrence, C Lee Giles, and Frans M Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–70, 2002. 1

[22] E.W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768769, 1965. 23

[23] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010. 1, 3, 4, 6, 7, 33

[24] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007. 3

[25] Linton Freeman. The development of social network analysis. *A Study in the Sociology of Science*, 2004. 1

[26] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002. 5, 33, 36, 38

[27] Steve Gregory. Fuzzy overlapping communities in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(02):P02017, 2011. 40

[28] Roger Guimera and Luis A Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005. 1

[29] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005. 6

[30] Adel Hlaoui and Shengrui Wang. A direct approach to graph clustering. *Neural Networks and Computational Intelligence*, 4:158–163, 2004. 6

[31] Yanqing Hu, Hongbin Chen, Peng Zhang, Menghui Li, Zengru Di, and Ying Fan. Comparative definition of community and corresponding identifying algorithm. *Physical Review E*, 78(2):026121, 2008. 2

[32] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003. 7

[33] Ann E Krause, Kenneth A Frank, Doran M Mason, Robert E Ulanowicz, and William W Taylor. Compartments revealed in food-web structure. *Nature*, 426(6964):282–285, 2003. 1

[34] Balachander Krishnamurthy and Jia Wang. On network-aware clustering of web clients. *ACM SIGCOMM Computer Communication Review*, 30(4):97–110, 2000. 4

[35] Conrad Lee, Fergal Reid, Aaron McDaid, and Neil Hurley. Detecting highly overlapping community structure by greedy clique expansion. *arXiv preprint arXiv:1002.1827*, 2010. 9

[36] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009. 3

[37] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296. ACM, 2006. 9, 26, 28, 29

[38] Stuart P Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982. 23, 25

[39] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297. Oakland, CA, USA., 1967. 6, 23

[40] Mohamed Adnane Mellah, Abdelmalek Amine, Reda Mohamed Hamou, and AV Kumar. Link analysis for communities detection on facebook. *arXiv preprint arXiv:1406.6705*, 2014. 4

[41] James Moody and Douglas R White. Structural cohesion and embeddedness: A hierarchical concept of social groups. *American Sociological Review*, pages 103–127, 2003. 1

[42] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004. 2, 36, 37

[43] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999. 13

[44] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005. 1

[45] Stuart L Pimm. The structure of food webs. *Theoretical Population Biology*, 16(2):144–158, 1979. 1

[46] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004. 2, 5

[47] Matthew J Rattigan, Marc Maier, and David Jensen. Graph clustering with network structure indices. In *Proceedings of the 24th international conference on Machine learning*, pages 783–790. ACM, 2007. 6

[48] Alexander W Rives and Timothy Galitski. Modular organization of cellular networks. *Proceedings of the National Academy of Sciences*, 100(3):1128–1133, 2003. 1, 4

[49] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011. 7

[50] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. 1

[51] Adam Schenker, Mark Last, Horst Bunke, and Abraham Kandel. Graph representations for web document clustering. In *Pattern Recognition and Image Analysis*, pages 935–942. Springer, 2003. 6

[52] John Scott. *Social network analysis*. Sage, 2012. 6

[53] Shabnam Shariaty. *Local approximation of page contributions in the PageRank algorithm*. PhD thesis, Applied Science: School of Computing Science, 2011. 8, 16, 19, 20, 22, 35

[54] František Slanina and Y Zhang. Referee networks and their spectral properties. *ACTA PHYSICA POLONICA SERIES B*, 36(9):2797, 2005. 6

[55] Victor Spirin and Leonid A Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences*, 100(21):12123–12128, 2003. 1, 4

[56] Joshua R Tyler, Dennis M Wilkinson, and Bernardo A Huberman. E-mail as spectroscopy: Automated discovery of community structure within organizations. *The Information Society*, 21(2):143–153, 2005. 5

[57] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010. 40

[58] Robert S Weiss and Eugene Jacobson. A method for the analysis of the structure of complex organizations. *American Sociological Review*, pages 661–668, 1955. 5

[59] Wikipedia. Expected value — wikipedia, the free encyclopedia, 2015. [Online; accessed 23-December-2015]. 12

[60] Wikipedia. Norm (mathematics) — wikipedia, the free encyclopedia, 2015. [Online; accessed 31-October-2015]. 11

[61] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys (csur)*, 45(4):43, 2013. 30

[62] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a non-negative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013. 34, 35, 39, 40, 42, 43

[63] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015. 3, 7, 33, 34

[64] Pan Zhang, Cristopher Moore, and MEJ Newman. Community detection in networks with unequal groups. *arXiv preprint arXiv:1509.00107*, 2015. 1

[65] Bin Zhou and Jian Pei. Link spam target detection using page farms. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(3):13, 2009. 8, 12, 14, 15