



SCHOOL of
GRADUATE STUDIES
EAST TENNESSEE STATE UNIVERSITY

East Tennessee State University
**Digital Commons @ East
Tennessee State University**

Electronic Theses and Dissertations

Student Works


8-2017

Vertex Weighted Spectral Clustering

Mohammad Masum

East Tennessee State University

Follow this and additional works at: <https://dc.etsu.edu/etd>

 Part of the [Discrete Mathematics and Combinatorics Commons](#), [Other Mathematics Commons](#), and the [Theory and Algorithms Commons](#)

Recommended Citation

Masum, Mohammad, "Vertex Weighted Spectral Clustering" (2017). *Electronic Theses and Dissertations*. Paper 3266.
<https://dc.etsu.edu/etd/3266>

This Thesis - Open Access is brought to you for free and open access by the Student Works at Digital Commons @ East Tennessee State University. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ East Tennessee State University. For more information, please contact digilib@etsu.edu.

Vertex Weighted Spectral Clustering

A thesis

presented to

the faculty of the Department of Mathematics

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Mathematical Sciences

by

Mohammad Masum

August 2017

Jeff Knisley, Ph.D.

Michele Joyner, Ph.D.

Debra Knisley, Ph.D.

Keywords: Spectral Clustering, Fiedler Vector, Laplacian Matrix, Vertex Weights

ABSTRACT

Vertex Weighted Spectral Clustering

by

Mohammad Masum

Spectral clustering is often used to partition a data set into a specified number of clusters. Both the unweighted and the vertex-weighted approaches use eigenvectors of the Laplacian matrix of a graph. Our focus is on using vertex-weighted methods to refine clustering of observations. Coefficients of a Fiedler vector are used to partition vertices of a given graph into two clusters. A vertex is classified as unassociated if the Fiedler coefficient of the vertex is close to zero compared to the largest Fiedler coefficient of the graph. We propose a vertex-weighted spectral clustering algorithm which incorporates a vector of weights for each vertex of a given graph to form a vertex-weighted graph. The proposed algorithm predicts association of data points while the unweighted clustering does not provide association. Finally, we implemented both the algorithms on several data sets to show that the proposed algorithm works in general.

Copyright by Mohammad Masum 2017

CONTENTS

ABSTRACT	2
ACKNOWLEDGMENTS	4
LIST OF TABLES	6
LIST OF FIGURES	7
1 INTRODUCTION	8
1.1 Clustering	8
1.2 Spectral Clustering Method	8
1.3 Application of Spectral Clustering Method	9
1.4 Vertex Weighted Graphs	13
2 BACKGROUND AND METHODS	15
2.1 Definitions	15
2.2 Graph-Theoretic Measures	19
2.3 Vertex-Weighted Laplacian Matrix	22
2.4 Vector-Weighted Vertices	23
3 A VERTEX-WEIGHTED SPECTRAL CLUSTERING ALGORITHM	24
3.1 Description of Vertex-Weighted Spectral Clustering	24
3.2 Implementation of the Algorithm using Python	26
3.3 Implementation of both Unweighted and Vertex-Weighted Clus- tering on a Graph	27
3.4 Generalization of results of using Vertex-Weighted Spectral Clus- tering	33
4 CONCLUSION	37

BIBLIOGRAPHY 38
APPENDICES 42
VITA 51

LIST OF TABLES

1	Unweighted Fiedler vector of graph G	28
2	Weighted Fiedler vector of graph G	30
3	Cosine similarity matrix of Fiedler weighted vector.	31
4	Characteristics of the graphs.	34
5	Unweighted and vertex-weighted spectral clustering results.	35
6	Clustering changes of unassociated vertices.	36

LIST OF FIGURES

1	A graph G and its Adjacency, Degree and Laplacian matrix	15
2	Implementing of unweighted and vertex-weighted clustering	28
3	Fiedler vector of G	29
4	Vertex distribution of unweighted clustering.	32
5	Vertex distribution of vertex-weighted Clustering.	32
6	Unweighted spectral clustering.	32
7	Vertex-weighted spectral clustering.	32
8	Eight different graphs with different characteristics	34

1 INTRODUCTION

1.1 Clustering

Clustering analysis is a technique to group a set of data objects into clusters so that objects similar to each other in the same cluster and dissimilar to the objects in other clusters. Han and Kamber discussed clustering techniques and defined them as objects “clustered or grouped based on the principles and maximizing the inter-class similarity and minimizing the intra-class similarity” [11]. Clustering is one of the most widely used techniques for exploratory data analysis. Clustering analysis has a broad range of applications in statistics, machine learning, computer science, bioinformatics, biology, social sciences, and psychology. In virtually every scientific field that deals with empirical data, scientists attempt to get the first impression of their data by trying to identify groups of similar behavior [27]. Clustering analysis is not a specific algorithm itself, but there are various algorithms that are used efficiently for clustering of different data sets. Spectral clustering is one of these algorithms [9]. Spectral clustering refers to a class of techniques which use spectral properties (eigenvalue and associated eigenvectors) of a similarity matrix to cluster points into disjoint clusters.

1.2 Spectral Clustering Method

There are several spectral clustering algorithms based on normalized and unnormalized clustering. Shi and Malik discussed normalized spectral clustering in [25]. There is another popular paper by Ng, Jordan, and Weiss(2002); they also discussed

a normalized spectral clustering algorithm [20]. These normalized algorithms are similar, differing only in the definitions of Laplacian matrix of a graph. Shi and Malik use the graph Laplacian L_{rw} , while Ng, Jordan and Weiss use a different graph Laplacian L_{sym} . For our work, we use the unnormalized graph Laplacian in spectral clustering. The algorithm is in [27]. Initially, a similarity graph is constructed as described in [27]. Let, W be its weighted adjacency matrix. The unnormalized Laplacian matrix L , is computed from the constructed similarity graph. We then compute the first k eigenvectors u_1, u_2, \dots, u_k of L and $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors u_1, u_2, \dots, u_k as columns. For $i = 1, 2, \dots, n$ let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i th row of U . Now, cluster the points $(y_i)_{i=1, \dots, n} \in \mathbb{R}^k$ with the k -means algorithm into clusters C_1, C_2, \dots, C_k , which provides output of the algorithm as clusters A_1, \dots, A_k with, $A_i = \{j | y_j \in C_i\}$.

1.3 Application of Spectral Clustering Method

The spectral clustering method has comprehensive applications including machine learning, computer vision, speech separation and many others.

- **Speech Separation**

The process of using spectral clustering for speech separation is discussed in [1]. In this paper, Bach and Jordan integrate physical and psychophysical properties of speech with learning algorithms for the problem of one-microphone blind source separation of speech. The physical properties provide parameterized similarity matrices for spectral clustering, while psychophysical properties make use of segmented training data. Using this approach, they are able to

differentiate speech signals from two speakers.

- **Distributed Systems**

Parallel spectral clustering has been used in distributed systems. Spectral clustering suffers a scalability problem in terms of memory and computational time when the data size is large. Chen, Song and Jin did an empirical study on a large document and photo data set. The document data set and the photo data set are consist of 193,844 instances and 2,121,863 instances respectively. The parallel spectral algorithm was used successfully on large data set problems [4].

- **Monitoring of Evolving Blog Communities**

Incremental spectral clustering has been applied to the monitoring of evolving blog communities. An existing spectral clustering algorithm cannot incrementally update the clustering result when the data set is given with a small change and imported in the real time monitoring of evolving communities. A standard clustering algorithm can be extended to an incremental algorithm by introducing an incidence matrix of the evolving data set. This incremental algorithm continuously updates the eigenvalue system used in the spectral method efficiently. This extended method also can label the evolving data set into different clusters instantly, which brings the advantage of lowering the computational cost. This extended spectral clustering algorithm as well as its applications in evolution of individual multi-topic blogs is discussed [21].

- **Topological Mappings for Mobile Robots**

An incremental spectral clustering method can be effectively used in appearance-based, on-line topological mapping for mobile robots. To label new data in an existing system, a spectral clustering algorithm extended with updating affinity matrix can be applied. This process has been experimented with large outdoor and indoor environments, which shows that we can close loops correctly by computing a fraction of the entries in the affinity matrix [26].

- **Protein Sequences**

Many local methods based on analyzing the distribution of distances between proteins sequences have been used to cluster homologous proteins. The performance of these local methods is average, but using a spectral clustering algorithm on the same proteins sequence data produces drastically improved accuracy rates [22].

- **High Dimensional Natural Language Data**

The spectral clustering algorithm can be applied to a high-dimensional natural language data set. Standard multivariate methods like k -means can be applied on natural language data but when it comes to a high performance data set, these multivariate methods fails to produce rewarding results. Since the search space in high dimensional data is too large, multivariate techniques do not satisfy global optimality. At this point, spectral clustering techniques works as a rescue algorithm. The spectral clustering method has been successfully applied to a high-dimensional German verbs data set [3].

- **Co-clustering Genes and Conditions**

In genomic research, microarray experiments are becoming widely used to concurrently measure RNA expression of genes. Microarray experiments have been used in uncovering the function of genes in various cell populations, tumor classification, drug target identification, understanding cellular pathways, and prediction of outcome to therapy [16]. Microarray technology can be applied to gene expression profiling to predict outcomes in multiple tumor types [10]. Spectral clustering techniques can be applied to identify class distinction of genes and to classify tumors. Spectral biclustering of micro-array data is used to co-cluster genes and conditions [14]. In this paper, the researchers propose a spectral biclustering method that uses the information gained by clustering the conditions to facilitate the clustering of genes, and vice versa. They assume that each tumor type has a subset of marker genes that exhibit over expression and that typically are not over expressed in other tumors. Then they use the method to cluster observations of different tumors,

- **Detecting Protein complexes in Protein-Protein Networks**

Proteins and the study of protein-protein interactions (PPI) are becoming increasingly important in our effort to understand human diseases [23]. Protein-protein interactions play a very crucial role in biological processes. Different areas like biochemistry, quantum chemistry, molecular dynamics and signal transduction have been using the study of protein-protein interactions [12]. To understand PPI, we have to understand protein complexes. Protein complexes are a very important quality of biological processes. Protein complexes form

different kinds of molecular nano-technology, which perform an immense number of biological functions. Qin and Gao used spectral clustering for detecting protein complexes in PPI networks decomposition [24]. Before implementing spectral clustering algorithm, they do data preprocessing, since there is noise in PPI data. In the first step, they produce four different similar graphs for PPI networks based on different characteristics: adjacency matrix, common neighbor similarity, transmute similarity and commute similarity. Then they find a graph Laplacian based on these four similarity graphs. Qin and Gao then used a normalized spectral graph algorithm in [25] to detect the protein complexes in PPI networks.

1.4 Vertex Weighted Graphs

A significant number of papers and articles can be found on edge-weighted graphs and their applications. In a weighted graph, there is a value assigned to each edge of a graph, and this weight can be any real number. Edge weights can be measured based on distance from one node to another, or alternatively on costs or similarity. Weighted networks based on edge weights have been extensively used in genomics and systems biology [13]. Compared to edge weighted network analysis, a few works can be found on vertex-weighted graphs, and almost nothing about vectors as vertex weights can be found. Similar to an edge-weighted graph, in a vertex-weighted graph, a weight can be assigned to each vertex of a graph. J. Knisley and D. Knisley, in their recent paper [15], extensively discussed vertex-weighted graphs and graph-theoretic measures that can be incorporated as vertex weights.

The spectral clustering algorithm allows clustering a given data set into more than two clusters. But for our thesis, we only consider a partition of a data set into two clusters. In this thesis, we use a vertex-weighted spectral clustering method to cluster sample data sets or graphs. Both the unweighted and the vertex-weighted clustering can be used to partition data into different clusters. We seek to determine whether there is a change in clustering due to implementing vertex-weighted graph. Our primary focus is to observe the vertices which are classified unassociated after using unweighted spectral clustering.

Our question is whether the vertex weights of these unassociated vertices can be used to predict an association to one cluster or the other. We also want to see the results by implementing the proposed algorithm on different of graphs having different characteristics. The graphs have identical component sizes, various component sizes, more equidistant points from both of the components as well as containing a different number of edges from the nearly equidistant points to one or both components.

2 BACKGROUND AND METHODS

2.1 Definitions

Adjacency matrix

The adjacency matrix of a graph G having n number of vertices is the $n \times n$ matrix whose entries a_{ij} are given by

$$a_{i,j} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are adjacent} \\ 0, & \text{otherwise.} \end{cases}$$

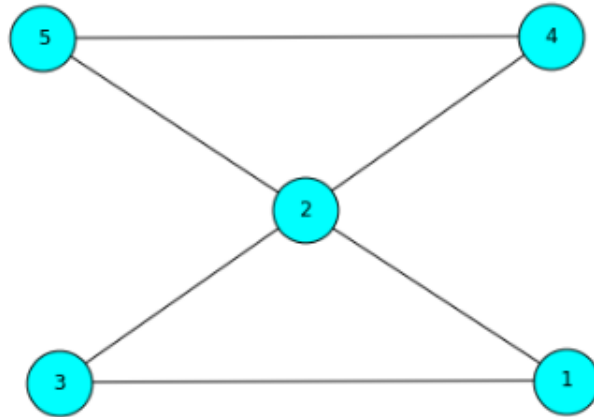


Figure 1: A graph G and its Adjacency, Degree and Laplacian matrix

For example, if $G = (V, E)$ is the graph in Figure 1, then the adjacency matrix of G is

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

From the above definition of the adjacency matrix, A is a real symmetric square matrix, and the trace of A is 0. To understand a graph, it is important to analyze and understand its adjacency matrix, since the rows and columns of A represent an arbitrary labeling of the vertices of the graph. The spectral properties (eigenvalues and associated eigenvectors) of an adjacency matrix is one of the most important features. The spectral properties of the adjacency matrix were much more investigated in the past than the Laplacian matrix [8]. But in [8, 19], the researchers gave the opinion that the eigenvalues of the Laplacian matrix are much more significant and natural than the eigenvalues of the adjacency matrix.

Degree Matrix

A degree matrix of a graph $G(V, E)$ with $|V| = n$, is an $n \times n$ diagonal matrix where $d_{i,j}$ of D are defined

$$d_{i,j} = \begin{cases} \deg(v_i), & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases}$$

where $\deg(v_i)$ is number of the incident at $v_i \in V(G)$ [5]. Both the degree matrix D and adjacency matrix A are used to construct the Laplacian matrix of a graph [2].

For example, the degree matrix of the graph in Figure 1 is

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}.$$

Laplacian Matrix

The Laplacian matrix of a graph G with order n is a $n \times n$ square matrix defined as

$$L(G) = D(G) - A(G)$$

where $A(G)$ is the familiar $(0, 1)$ adjacency matrix and $D(G)$ is the diagonal matrix of vertex degrees.[18]. For example, the Laplacian matrix of the graph G in Figure 1 is

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & -1 & 0 & -1 & 2 \end{bmatrix}.$$

Before the Laplacian matrix and its spectral properties became widely used, matrix theory and linear algebra were used to analyze adjacency matrices, and sometimes some eigenvalues of the adjacency matrix were regarded as the algebraic connectivity of graph [6]. A new powerful technique of spectral graph theory using a Laplacian matrix was proposed for analyzing graphs. Along with other researchers like Margulis, the study of spectral graph theory entered in a new generation of analyzing graphs [17]. The eigenvalues and corresponding eigenvectors are the most important features of the Laplacian matrix in spectral graph theory. The Laplacian matrix L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ [27]. The Laplacian matrix L is the basis for the spectral clustering of a graph. There are some important properties of L . To begin with, the rows and the columns of a Laplacian matrix L sum to 0. Second, the diagonal entries of L represent the number of adjacent vertices (i.e., the degree of a vertex) of a vertex in a graph. Finally, the singular value decomposition of L is the same as its diagonalization since L is positive-semidefinite. The

SVD of L provides eigenvalues of L in descending order. The eigenvalues of L are also singular values for L . We use the singular value decomposition to find eigenvalues and associated eigenvectors for L because the order of the eigenvalues is important for us. The smallest eigenvalue of L is zero with the corresponding eigenvector as the constant one vector. Thus, for the above graph:

$$L1 = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

For every vector $f \in R^n$ we have,

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2$$

Since, L is symmetric and positive semidefinite. $L = U\Sigma V^T$ implies, $U = V$. Therefore, $L = U\Sigma U^T$, where Σ is the singular values which is eigenvalues of the Laplacian matrix. Also for any eigenvector v , $v^T L v \geq 0$ [27].

Fiedler Vector

The Laplacian matrix L of a graph of order n can be expanded using singular value decomposition method. If $L = [v_1|v_2|\dots|v_n]$, then the SVD of L , which is also its diagonalization, is $L = U\Sigma U^T$, where $U = [u_1, u_2, \dots, u_{n-1}, u_n]$. The u_1, u_2, \dots, u_n are eigenvectors of L and

$$\Sigma = \begin{bmatrix} s_1 & & & & \\ & s_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & s_{n-1} \\ & & & & & 0 \end{bmatrix}.$$

Here, u_{n-1} is the associated eigenvector of second smallest eigenvalue s_{n-1} . This u_{n-1} is called the Fiedler vector. The eigenvalue s_{n-1} is also known as the algebraic

connectivity of a graph. We use the SVD because Σ contains the singular values of L in non-ascending order and the singular values of a positive semi-definite matrix are the same as its eigenvalues.

Suppose a graph has more than one component. Then the Fiedler vector can be expressed as a linear combination of the eigenvectors u_{n-1}, u_n corresponding to the two smallest eigenvalues, since the last two columns of U are orthogonal to $\mathbf{1}$:

$$FiedlerVector = (u_{n-1}\mathbf{1})u_n - (u_n\mathbf{1})u_{n-1}$$

Thus $FiedlerVector \in span(u_{n-1}, u_n)$. The Fiedler vector is orthogonal to the one vector. That is,

$$FiedlerVector \cdot \mathbf{1} = (u_{n-1} \cdot \mathbf{1})(u_n \cdot \mathbf{1}) - (u_n \cdot \mathbf{1})(u_{n-1} \cdot \mathbf{1}) = 0$$

The second smallest eigenvalue and associated Fiedler vector play a major role in different applications of graph theory, such as expander graphs, the isoperimetric problem and the maximum cut problem. The Fiedler vector answers many questions regarding a graph, most particularly about the connectedness of a graph.

2.2 Graph-Theoretic Measures

In a vertex-weighted graph, a vector of vertex-weights is assigned to each vertex. There are many graph theoretic measures that can be incorporated into a vector of weights and that can be applied to vertex weighted spectral clustering. Three graph-theoretic measures: weighted domination number, weighted periphery, and weighted

diameter are discussed in [15]. D. Knisley defines the weighted domination number as follows:

- Weighted Domination Number

Let D be a set of vertices of a graph G . If every vertex of G not in vertex set D is adjacent to at least one vertex of D , then D is a dominating set of G . For a vertex-weighted graph, the minimum dominating set is the set whose vertex sum is a minimum.

In general, all graphical invariants can be used as vertex weights. Here are some graphical invariants that are generalized into weighted graph-theoretic measures:

- Degree

The degree of a vertex v in a graph G is the number of direct neighbors of the vertex v . That degree is the number of edges incident to the vertex.

- Closeness Centrality

The closeness of a vertex v of a graph G is the inverse of the shortest path distance of the vertex to all other vertices in the graph. Closeness of a vertex measures centrality of the vertex in the graph. Let A be the adjacency matrix of G and let $n = |V(G)|$. If $u \in V(G)$, then closeness centrality of u is

$$C_{cl}(u) = \frac{(n-1)}{\sum_{v \in V(G)} dist(u, v)}$$

- Average Nearest Neighbor Degree

The average nearest neighbor degree of a vertex u in graph G is the average of the degree of its immediate neighbors. Let $N(u)$ is the neighbors of u , then the average nearest neighbor degree of u is

$$C_{an} = \frac{\sum C_d(u)}{N(u)}$$

where $C_d(u)$ is the number of degree of u .

- Clustering Coefficient

The clustering Coefficient of a vertex u of a graph G is the ratio of the number of connected neighbours of the vertex u to the number of all possible connected neighbors of u .

- Strength

The strength of a vertex u of a edge weighted network is the sum of weights of edges connected to vertex u . Then the strength of u is

$$C_s(u) = \sum_{v \in N(u)} w_{uv}$$

where w_{uv} is the weights of uv edges of the network.

There are other graphical invariants which can be generalized to weighted graphical invariants and also we can apply them to vertex-weighted graph.

2.3 Vertex-Weighted Laplacian Matrix

Let $G(V, E)$ be a graph with vertex set $V = \{v_1, \dots, v_n\}$ and edge set E . If d_v is the degree of a vertex $v \in V$, then the unweighted Laplacian L is defined

$$L(u, v) = \begin{cases} d_v, & \text{if } u = v \\ -1, & \text{if } u \sim v. \\ 0, & \text{otherwise} \end{cases}$$

Let α_v be the weight of vertex v ; then the unweighted Laplacian matrix can be extended to the weighted Laplacian by Chung and Langland [7],

$$L(u, v) = \begin{cases} \sum_{z \sim u} \alpha_z, & \text{if } u = v \\ -\alpha_v, & \text{if } u \sim v. \\ 0, & \text{otherwise} \end{cases}$$

Now for any function $f : V \rightarrow \mathbb{R}$, the Laplacian matrix satisfies:

$$Lf(v) = \sum_{u \sim v} \alpha_u (f(v) - f(u)).$$

Here L is not symmetric, but L can be expressed as a symmetric matrix L of G as follows:

$$L(u, v) = \begin{cases} \sum_{z \sim v} \alpha_v, & \text{if } u = v \\ -\sqrt{\alpha_u \alpha_v}, & \text{if } u \sim v. \\ 0, & \text{otherwise} \end{cases}$$

The vertex-weighted Laplacian is also defined by Lovasz:

$$L(u, v) = \begin{cases} \sum_{z \sim v} \alpha_v, & \text{if } u = v \\ -\alpha_u \alpha_v, & \text{if } u \sim v. \\ 0, & \text{otherwise} \end{cases}$$

2.4 Vector-Weighted Vertices

We can assign a vector of weights to each vertex of a graph to form vector valued graph. For any two vertices $u, v \in V(G)$, we denote the Laplacian of vector valued graph of G as $[L_{u,v}]_{u,v}$ where $L_{u,v}$ is an $m \times m$ matrix [15]. Then the coefficients of $[L_{u,v}]_{u,v}$ can be expressed by the following theorem.

Theorem 2.1 [15] *The coefficients $[L_{u,v}]_{u,v \in V}$ are of the form*

$$L(u, v) = \begin{cases} d_u I_m, & \text{if } u = v \\ -I_m, & \text{if } u \sim v \\ 0, & \text{otherwise} \end{cases}$$

where I_m is the $m \times m$ identity matrix.

Fiedler vector is the key component to cluster the data set and it is used in different clustering algorithm including spectral clustering algorithm. The following theorem shows that the Fiedler vectors of the vector-valued Laplacian matrix are related to the Fiedler vectors of unweighted Laplacian matrix.

Theorem 2.2 [15] *The Fiedler eigenvector e_1 satisfies*

$$\sum_{v \in V} e_1(v) = 0.$$

3 A VERTEX-WEIGHTED SPECTRAL CLUSTERING ALGORITHM

3.1 Description of Vertex-Weighted Spectral Clustering

The purpose of this thesis is to predict the association of equidistant or nearly equidistant data points from both clusters using vertex-weighted spectral clustering. These are the vertices that unweighted clustering does not associate to either cluster. In unweighted clustering the unassociated vertices correspond to zero coefficients in the Fiedler vector. Vertex weights can be used to predict association of those unassociated vertices.

For any given graph G , we compute the unweighted Laplacian matrix L using the degree and adjacency matrices of G . After adding a vector of weights $\{\alpha_v\}_{v \in V(G)}$ where $\alpha_v \in \mathbb{R}^m$ to each vertex of G , we form the vector-valued graph of G . We consider $\alpha_v \in \mathbb{R}^3$ in the result section. Then we find vertex-weighted Laplacian matrix of G using

$$L_w(u, v) = \begin{cases} I_m \sum_{z \sim v} \alpha_z, & \text{if } u = v \\ -I_m(\alpha_u \alpha_v), & \text{if } u \sim v. \\ 0, & \text{otherwise} \end{cases}$$

The spectral properties of the Laplacian matrix can be computed by using the singular value decomposition method, $L = U\Sigma U^T$. The reason for using the SVD of L is discussed in Chapter 2. To compute weighted Fiedler vectors, vectors of weights $\{\alpha_v\}$ are multiplied with the unweighted Fiedler vector. The unweighted Fiedler vector is calculated using the unweighted Laplacian matrix.

A vertex weighted graph must be constructed from the weights for the vertices and

the unweighted graph. To construct the graph, a cosine similarity matrix is formed using the weighted Fiedler vector. It computes the correlation between every pair of vertices in the graph G . A cosine similarity matrix is calculated using

$$Correlation = \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|}$$

where v_1, v_2 are any two vertices in the graph G .

- Unweighted Clustering

We use the unweighted Laplacian matrix and the unweighted Fiedler vector in the unweighted clustering technique. To partition the data set into clusters, we set a threshold so that the algorithm maximally provides two clusters and unassociated vertices. The Fiedler coefficient of an unassociated vertex corresponds to zero or closer to zero compared with the largest Fiedler coefficient of a graph. If a vertex Fiedler coefficient is greater than the threshold, then that vertex be in cluster 1 and if the coefficient is less than the negative of the threshold, then that vertex be in cluster 2. A vertex is classified as unassociated if the absolute value of its Fiedler coefficient is less than the threshold.

- Vertex-Weighted Clustering

In vertex-weighted clustering, to partition the given data set into clusters the weighted Laplacian matrix and the cosine similarity matrix of the weighted Fiedler vector are used. Two vertices will be in the same cluster if their correlation is greater than a given threshold.

3.2 Implementation of the Algorithm using Python

We used Python 3 to implement our proposed algorithm. There are several packages of Python 3 are used in our code. We used Numpy for fast operation in arrays, Scipy to do the mathematical operation of linear algebra, Networkx for graph theory, Pandas for data visualization. We used cosine similarity function which is imported from Scikit Learn, a Python library.

At first, we define a function called “SetPosAlpha” which takes a graph as input and in return sets vertex attributes: vertex position and vertex weights. We create a graph using Networkx and set a position of the graph components using the function “SetPosAlpha”. The unweighted Laplacian matrix and the SVD of Laplacian are calculated using the built-in function of Numpy. The Fiedler vector is acquired using DataFrame. A *threshold* = 0.01 is set up to identify the clusters. If *FiedlerVector* \geq 0.01 the vertices are in cluster 1 and if *FiedlerVector* \leq -0.01 implies the vertices are in cluster 2, and if $|FiedlerVector| < 0.01$ the vertex is labeled as unassociated. Thus, in this way, we get the unweighted clustering of the vertices.

For vertex-weighted clustering, at first, we create a class called coefficient matrix and initialize a method under this class which allows us to put attributes coeff, value and names for our class. The default value is a 2×2 identity matrix. The attribute for the class we created returns the product of coefficient and value. A special python method ‘mul’ is set up which takes a vector as input and returns the dot product of the coefficient and the vector.

Now we want to replace the coefficients by matrices, and for that, we define a function VertexWeighting which contains three arguments. This function takes as

input the unweighted Laplacian matrix as a data type numpy array and produces an output of weighted vector-valued Laplacian matrix. We also use this function to create the weighted Fiedler vector. To create the weighted Fiedler vector, an unweighted Fiedler coefficient of each vertex is multiplied with the vertex's vector of weights.

We import cosine similarity function from Scikit Learn package to calculate the similarity between two non-zero vectors. We use this similarity in the PredictPartition function. We define a function PredictPartition for vertex-weighted clustering. The function works as a signal to label the vertices of a given graph. The function takes a list of vectors as input. We pass the cosine similarity matrix which we obtain using weighted Fiedler vector through the PredictPartition function. A parameter threshold = 0.02 is set up as a default argument for this function. Initially, it assumes all the vertices are unassociated. Then the function finds two vertices that achieve maximum correlation, and it assigns the vertices those correlation is greater than threshold to cluster 1. Then the function removes the vertices which already belong to cluster 1 and then follow the similar procedure with the rest of the vertices. In the process, it creates cluster 2. If there is still some vertices remain then they are labeled as unassociated. In this way, we achieve the clustering of vertices of a graph using vertex-weighted clustering.

3.3 Implementation of both Unweighted and Vertex-Weighted Clustering on a Graph

For instance, we consider the following simple graph G in Figure 2. We formed the unweighted Laplacian matrix from the graph G . The second smallest eigenvalue of L

and a corresponding eigenvector of Laplacian matrix is computed. This eigenvector is the required Fiedler vector which produces clustering of the data set. The Fiedler vector of the graph G in Figure 2 is in Table 1.

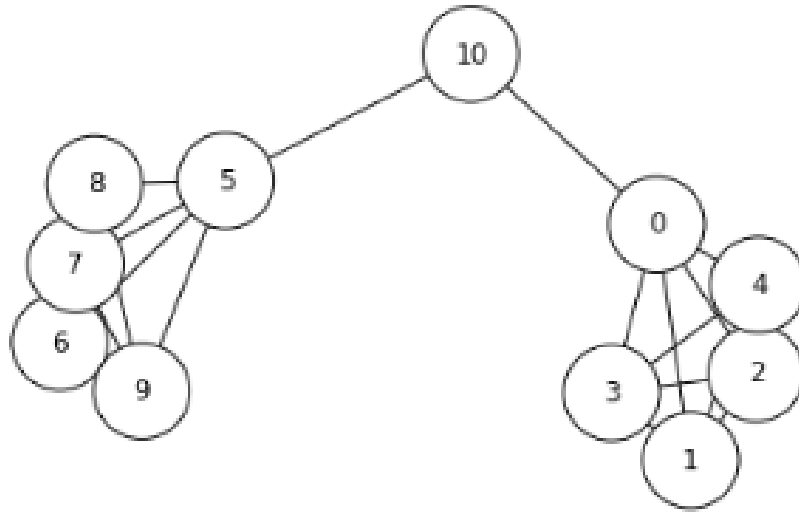


Figure 2: Implementing of unweighted and vertex-weighted clustering

Table 1: Unweighted Fiedler vector of graph G .

Vertex	Fiedler Coefficient
0	-0.27060
1	-0.32664
2	-0.32664
3	-0.32664
4	-0.32664
5	0.27060
6	0.32664
7	0.32664
8	0.32664
9	0.32664
10	0.00000

Entries of the Fiedler vector are called Fiedler coefficients. The coefficients imply that the graph naturally divides into two different clusters as identified by the different signs of the coefficients. We set the parameter threshold = 0.01 to label the vertices. The Fiedler coefficients of vertex set $\{0,1,2,3,4\}$ are greater than 0.01. Thus, these vertices form cluster 1. The Fiedler coefficients of vertex set $\{5,6,7,8,9\}$ are less than -0.01 . Thus the vertex set $\{5,6,7,8,9\}$ form cluster 2. The vertex (10) has Fiedler coefficient 0.00 which is less than 0.01. Thus, the vertex (10) is labeled as unassociated with any of the clusters. Figure 3 shows the coefficients of the Fiedler vector, illustrating that there are two clusters with one unassociated vertex.

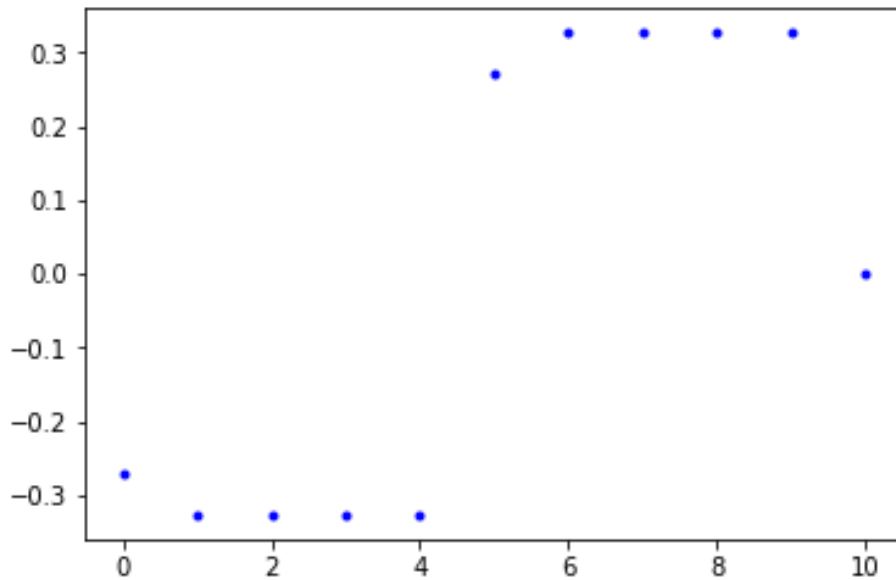


Figure 3: Fiedler vector of G

In an actual application, the vertex weight vectors represent additional information not used in the construction of the original network. Vertex weights and graph invariants are discussed in Chapter 2. For example, we incorporated the $[1, 0, 0]$ vector on vertices in cluster 1 and $[0, 0, 1]$ vector on vertices in cluster 2. Recall that the question of this thesis addresses if and how vertex weights can be used to associate vertices to a cluster that were unassociated in the unweighted case. To illustrate that it can, a vector of weights $[0.5, 0, 0.5]$ is assigned to vertex (10). We extended unweighted Laplacian matrix to vertex-weighted Laplacian of the vertex-weighted graph. Consequently, we computed a weighted Fiedler vector which is noted in Table 2.

Table 2: Weighted Fiedler vector of graph G .

Vertex	Fiedler Vector
0	$[-0.27060, 0, 0]$
1	$[-0.32664, 0, 0]$
2	$[-0.32664, 0, 0]$
3	$[-0.32664, 0, 0]$
4	$[-0.32664, 0, 0]$
5	$[0, 0, 0.27060]$
6	$[0, 0, 0.32664]$
7	$[0, 0, 0.32664]$
8	$[0, 0, 0.32664]$
9	$[0, 0, 0.32664]$
10	$[0, 0, 0]$

From the weighted Fiedler vector, we created a cosine similarity matrix in Table 3 to cluster the vertices. A parameter, $threshold = 0.02$ was set up to classify the vertices.

The cosine similarity matrix is symmetric. Since we want to avoid double count,

Table 3: Cosine similarity matrix of Fiedler weighted vector.

	0	1	2	3	4	5	6	7	8	9	10
0	1.00	1.00	1.00	1.00	1.00	0	0	0	0	0	.707
1	1.00	1.00	1.00	1.00	1.00	0	0	0	0	0	.707
2	1.00	1.00	1.00	1.00	1.00	0	0	0	0	0	.707
3	1.00	1.00	1.00	1.00	1.00	0	0	0	0	0	.707
4	1.00	1.00	1.00	1.00	1.00	0	0	0	0	0	.707
5	0	0	0	0	0	1.00	1.00	1.00	1.00	1.00	-.707
6	0	0	0	0	0	1.00	1.00	1.00	1.00	1.00	-.707
7	0	0	0	0	0	1.00	1.00	1.00	1.00	1.00	-.707
8	0	0	0	0	0	1.00	1.00	1.00	1.00	1.00	-.707
9	0	0	0	0	0	1.00	1.00	1.00	1.00	1.00	-.707
10	.707	.707	.707	.707	.707	-.707	-.707	-.707	-.707	-.707	1.00

we set the diagonal and above equal to 0. We cluster the data set from the cosine similarity matrix using the idea: positives have a positive correlation with each other, negatives have a positive correlation with each other, but a positive and a negative are negatively correlated. Initially, we assume all the vertices are unassociated (label=0). We find two vertices v_1 and v_2 that achieve maximum correlation. Then we create a cluster of all vertices whose correlation with them is greater than a given threshold. We set the threshold = 0.02. We find that any two vertices in the vertex set $\{0, 1, 2, 3, 4\}$ have correlation 1 which is maximum. Thus these vertices form cluster 1. The vertex (10) has a correlation of 0.707 (which is greater than the threshold = 0.02) with any vertex in the cluster 1. For this, the vertex (10) moves to cluster 1. Since the vertices in $\{0, 1, 2, 3, 4, 10\}$ form cluster 1, we remove these vertices from the data set. Again we look for a maximum correlation between two vertices in the remaining vertices in $\{5, 6, 7, 8, 9\}$. We find that any two vertices in the set has correlation 1 between them. Thus, these vertices form cluster 2.

Figure 4 and Figure 5 illustrate the distribution of vertices due to using unweighted and vertex-weighted clustering respectively. Figure 6 and Figure 7 are graphical representation of clustering of vertices for using unweighted and vertex-weighted clustering respectively. In the Figures, the red, green and yellow color represent cluster 1, cluster 2 and unassociated vertices respectively. The Vertex (10) changes from yellow to red in Figure 7, which suggests that the Vertex (10) has become associated with cluster 1.

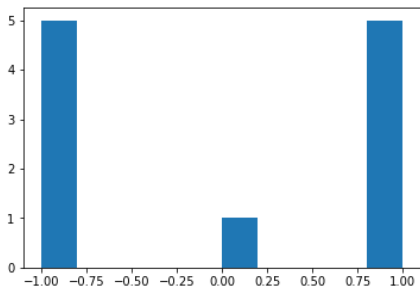


Figure 4: Vertex distribution of unweighted clustering.

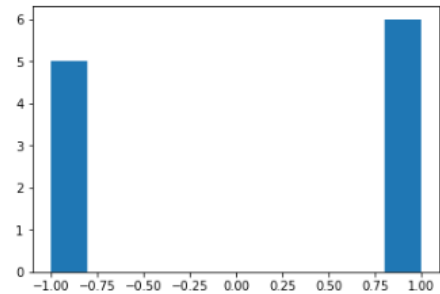


Figure 5: Vertex distribution of vertex-weighted Clustering.

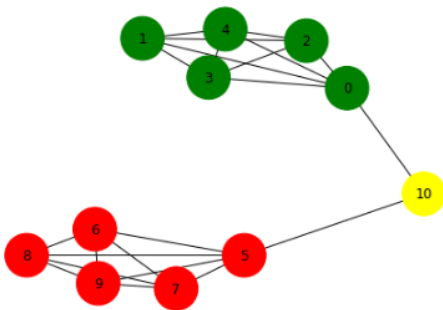


Figure 6: Unweighted spectral clustering.

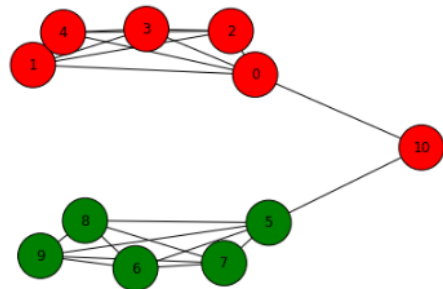


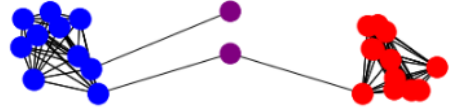
Figure 7: Vertex-weighted spectral clustering.

3.4 Generalization of results of using Vertex-Weighted Spectral Clustering

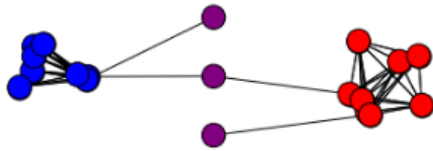
We applied our proposed vertex-weighted spectral clustering on eight different graphs in Figure 8 to show that the proposed algorithm can work in general. The graphs have the following variations: identical components sizes, a different component sizes, more equidistant points from both of the components as well as containing a different number of edges from the nearly equidistant points to one or both components. Characteristics of the graphs are recorded in Table 4.



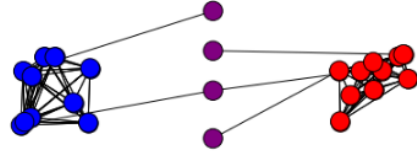
8(a)



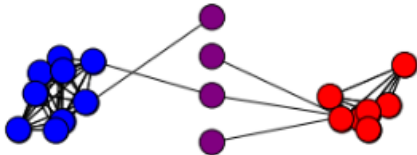
8(b)



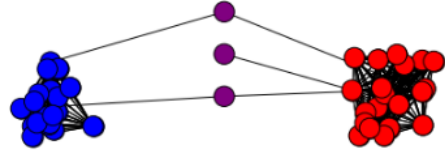
8(c)



8(d)



8(e)



8(f)

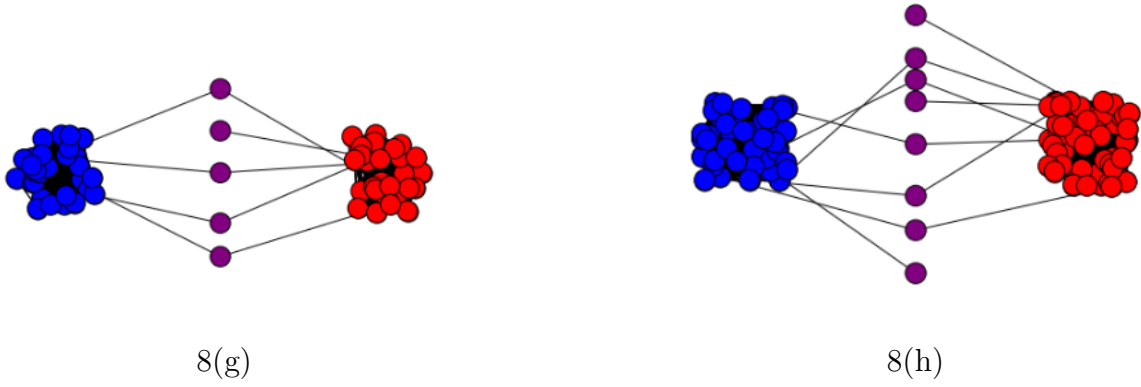


Figure 8: Eight different graphs with different characteristics

Table 4: Characteristics of the graphs.

Graph	1st Component	2nd Component	(Nearly) Equidistant Vertices
8(a)	K_6	K_6	1
8(b)	K_{12}	K_{10}	2
8(c)	K_8	K_8	3
8(d)	K_{10}	K_{10}	4
8(e)	K_7	K_9	4
8(f)	K_{20}	K_{20}	3
8(g)	K_{30}	K_{32}	5
8(h)	K_{50}	K_{60}	8

We applied both unweighted and vertex-weighted clustering algorithms on all the graphs. The results of applying algorithms are documented in Table 5. The number of vertices classified as cluster 1, cluster 2 and unassociated for using both the algorithms are recorded in the Table 5.

Table 5: Unweighted and vertex-weighted spectral clustering results.

Graph	Order	Unweighted Clustering			Vertex-Weighted Clustering		
		Cl 1	Cl 2	Unassociated	Cl 1	Cl 2	Unassociated
8(a)	13	6	6	1	6	7	0
8(b)	24	11	12	1	12	12	0
8(c)	19	9	9	1	10	9	0
8(d)	24	12	11	1	12	12	0
8(e)	20	9	10	1	10	10	0
8(f)	43	21	20	2	22	21	0
8(g)	67	31	32	4	35	32	0
8(h)	118	62	51	5	62	56	0

Clustering of the vertices is changed due to applying vertex-weighted clustering. Our primary focus is to observe the changes of clustering of the vertices that are unassociated to both clusters after using unweighted clustering techniques. We documented cluster changes of those vertices in Table 6. In all the graphs from 8(a) to 8(h), the unassociated vertices become associated. In some graphs, the unassociated vertices become related to cluster 1 while for other graphs, the vertices become associated to cluster 2.

Table 6: Clustering changes of unassociated vertices.

Graph	Vertex	Unweighted Clustering	Vertex-Weighted Clustering
8(a)	12	Unassociated	Cl-2
8(b)	22	Unassociated	Cl-1
8(c)	16	Unassociated	Cl-1
8(d)	20	Unassociated	Cl-2
8(e)	16	Unassociated	Cl-2
8(f)	40,42	Unassociated	Cl-2
8(g)	62-66	Unassociated	Cl-1
8(h)	110, 112-115	Unassociated	Cl-2

4 CONCLUSION

Applying the vertex-weighted spectral clustering technique can change the clustering of a graph. The unweighted technique does not provide an association of the vertices which are less connected compared with the other vertices of a graph. Implementing a vertex-weighted clustering technique can influence those vertices to be associated to any one of the clusters. We used the cosine similarity matrix to construct the graph. In the future, the nearest neighbor approach can be added to the vertex-weighted spectral clustering and compare the results.

In contrast with the past, there are many high dimensional data sets like microarrays, DNA data, time series data, EEG and MEG is available due to the advancement of technology. To embed these high dimensional data into a low dimensional space, we use the dimensionality reduction methods to avoid the curse of dimensionality. In the reduction process, we discard less important features. But using those features as vertex-weights we can keep those and for a clustering situation, we can use the vertex-weighted spectral clustering algorithm to cluster the data set.

BIBLIOGRAPHY

- [1] Francis R Bach and Michael I Jordan. Learning spectral clustering, with application to speech separation. *Journal of Machine Learning Research*, 7(Oct):1963–2001, 2006.
- [2] Lowell W Beineke and Robin J Wilson. *Topics in algebraic graph theory*, volume 102. Cambridge University Press, 2004.
- [3] Chris Brew and Sabine Schulte im Walde. Spectral clustering for german verbs. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing- Volume 10*, pages 117–124. Association for Computational Linguistics, 2002.
- [4] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y Chang. Parallel spectral clustering in distributed systems. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):568–586, 2011.
- [5] Fan Chung, Linyuan Lu, and Van Vu. Spectra of random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 100(11):6313–6318, 2003.
- [6] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.
- [7] Fan RK Chung and Robert P Langlands. A combinatorial laplacian with vertex weights. *journal of combinatorial theory, Series A*, 75(2):316–327, 1996.

- [8] Dragos M Cvetkovic, Michael Doob, Ivan Gutman, and Aleksandar Torgašev. *Recent results in the theory of graph spectra*, volume 36. Elsevier, 1988.
- [9] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD explorations newsletter*, 4(1):65–75, 2002.
- [10] Todd R Golub, Donna K Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P Mesirov, Hilary Coller, Mignon L Loh, James R Downing, Mark A Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, 286(5439):531–537, 1999.
- [11] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques (the morgan kaufmann series in data management systems)*. 2000.
- [12] Henry D Herce, Wen Deng, Jonas Helma, Heinrich Leonhardt, and M Cristina Cardoso. Visualization and targeted disruption of protein interactions in living cells. *Nature communications*, 4, 2013.
- [13] Steve Horvath. *Weighted network analysis: applications in genomics and systems biology*. Springer Science & Business Media, 2011.
- [14] Yuval Kluger, Ronen Basri, Joseph T Chang, and Mark Gerstein. Spectral biclustering of microarray data: coclustering genes and conditions. *Genome research*, 13(4):703–716, 2003.
- [15] Jeff R Knisley and Debra Knisley. *Vertex-weighted graphs and their applications*.
- [16] David J Lockhart and Elizabeth A Winzeler. Genomics, gene expression and dna arrays. *Nature*, 405(6788):827–836, 2000.

- [17] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Explicit expanders and the ramanujan conjectures. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 240–246. ACM, 1986.
- [18] Russell Merris. Laplacian matrices of graphs: a survey. *Linear algebra and its applications*, 197:143–176, 1994.
- [19] Bojan Mohar, Y Alavi, G Chartrand, and OR Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2(871-898):12, 1991.
- [20] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.
- [21] Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas Huang. Incremental spectral clustering with application to monitoring of evolving blog communities. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 261–272. SIAM, 2007.
- [22] Alberto Paccanaro, James A Casbon, and Mansoor AS Saqi. Spectral clustering of protein sequences. *Nucleic acids research*, 34(5):1571–1580, 2006.
- [23] Kristine A Pattin and Jason H Moore. Role for protein–protein interaction databases in human genetics. *Expert review of proteomics*, 6(6):647–659, 2009.
- [24] Guimin Qin and Lin Gao. Spectral clustering for detecting protein complexes in

protein–protein interaction (ppi) networks. *Mathematical and Computer Modelling*, 52(11):2066–2074, 2010.

- [25] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.
- [26] Christoffer Valgren, Tom Duckett, and Achim Lilienthal. Incremental spectral clustering and its application to topological mapping. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4283–4288. IEEE, 2007.
- [27] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

APPENDIX: Python Code

```
%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

import warnings

warnings.filterwarnings("ignore")

import networkx as nx

from scipy import linalg

from pandas import DataFrame, set_option

from numpy import array, matrix

def fmt(x):

    if( abs(x) < 1e-8):

        return '0'

    else:

        return '%.5f' % x

np.set_printoptions(precision = 5, suppress = True)

set_option('display.float_format',fmt)

def SetPosAndAlpha( G, clr , offset = np.array([0,0]) ):

    pos = nx.random_layout(G)

    for ky in pos.keys():
```

```

        pos[ky] += offset

nx.set_node_attributes( G, 'pos', pos )

Vweights = {}
for nd in G.nodes():
    Vweights[nd] = clr

nx.set_node_attributes( G, 'alpha', Vweights )

return pos

class CoeffMatrix(object):
    def __init__(self, coeff,
                 value = np.eye(2), name = "I"):
        self.coeff = coeff
        self.value = coeff*value
        self.name = name

    def __repr__(self):
        if( self.coeff == 1 ):
            return "%s" %self.name

```

```

elif( self.coeff == -1 ):
    return "-%s" % self.name
elif( self.coeff == 0 ):
    return "0"
else:
    return '%s*%s' % (self.coeff, self.name)

def __mul__(self, vector):
    return self.coeff.dot(vector)

```

Replaces coefficients by matrices

```

def VertexWeighting( NdArray, values = np.eye(2), names = 'I' ):
    M = NdArray
    Lw = np.empty( M.shape, dtype = object )
    if( hasattr( values, 'shape' )
        and len( values.shape ) > len( M.shape ) ):
        assert M.shape == values.shape[: len( M.shape )]
        assert M.shape == names.shape
        for index in np.ndindex(*M.shape):

```

```

        Lw[ index ] = CoeffMatrix(
            M[ index ], values[ index ], names[ index ] )
    return Lw
else:
    for index in np.ndindex(*M.shape):
        Lw[ index ] = CoeffMatrix( M[ index ], values, names)
    return Lw

import math

import itertools

from sklearn.metrics.pairwise import cosine_similarity

def PredictPartition( ListVec, threshold = 0.2, verbose = True ):
    assert threshold > 0, "Threshold cannot be exactly 0"
    Corrs = cosine_similarity( ListVec )

    Labels = [0]*len(ListVec)

    for i in range(len(ListVec)):
        Corrs[i, i:] = 0

```

```

mx = np.max(Corrs)

assert mx >= threshold

if( verbose ):
    print("Max for first cluster is {0}".format(mx))

v1,v2 = np.unravel_index(
    np.argmax(Corrs), Corrs.shape )

IndexList = list(range(len(ListVec)))

IndexList.remove(v1)

IndexList.remove(v2)

PartitionList = [v1,v2]

Corrs[v1,v2] = 0

Corrs[v2,v1] = 0

Labels[v1] = Labels[v2] = 1

AddingToCluster = True

while AddingToCluster:
    for i,j in itertools.product(IndexList, PartitionList):
        AddingToCluster = False
        if( Corrs[i,j] >= threshold ):

```

```

        Labels[i] = 1
        Corrs[i,j] = 0
        IndexList.remove(i)
        PartitionList.append(i)
        AddingToCluster = True
        break

for i,j in itertools.product(PartitionList, PartitionList):
    Corrs[i,j] = 0

mx = np.max(Corrs)
assert mx >= threshold

if( verbose ):
    print("Max for second cluster is {0}".format(mx))

v1,v2 = np.unravel_index( np.argmax(Corrs), Corrs.shape )

IndexList.remove(v1)
IndexList.remove(v2)

PartitionList = [v1,v2]

Corrs[v1,v2] = 0

Labels[v1] = Labels[v2] = -1

```



```

AddingToCluster = True

while AddingToCluster:

    for i,j in itertools.product(IndexList, PartitionList):

        AddingToCluster = False

        if( Corrs[i,j] >= threshold ):

            Labels[i] = -1

            Corrs[i,j] = 0

            IndexList.remove(i)

            PartitionList.append(i)

            AddingToCluster = True

            break

        else:

            break

return Labels

```

```
numA = 5
```

```
numB = 5
```

```
Cluster1 = nx.complete_graph(numA)
```

```
Cluster2 = nx.complete_graph(numB)
```

```

pos1 = SetPosAndAlpha( Cluster1 , np.array ([1,0,0]) ,
                        offset = np.array ([ 1.5, -0.5]) )
pos2 = SetPosAndAlpha( Cluster2 , np.array ([0,0,1]) ,
                        offset = np.array ([-2.5, -0.5]) )
nx.draw(Cluster1 , pos1 , node_color = np.array ([1,0,0]) )
nx.draw(Cluster2 , pos2 , node_color = np.array ([0,0,1]) )
G = nx.disjoint_union( Cluster1 , Cluster2)
G.add_node( numA+numB, { 'pos ': np.array ([0,0]) ,
                        'alpha ': np.array ([0.5,0,0.5]) } )
G.add_edge( 0 , numA+numB )
G.add_edge( numA, numA+numB )

nx.draw(G,nx.get_node_attributes(G,'pos') ,
        node_color = [ G.node[v][ 'alpha ' ] for v in G ])

plt.axis('equal');

L = DataFrame( nx.laplacian_matrix(G).todense() ,
               columns = G.nodes() , index = G.nodes() )
V,Sigma,Vt = linalg.svd(L)

```

```

FiedlerVector = Vt[-2]

threshold = 0.01

Component1 = L.columns[ FiedlerVector >= threshold ]
Component2 = L.columns[ FiedlerVector <= -threshold ]
Unassociated = L.columns[ np.abs(FiedlerVector) < threshold ]

Lw = DataFrame(VertexWeighting(L.as_matrix()),
               columns = L.columns, index = L.index)

Weights = np.array( [ G.node[i]['alpha']
                    for i in G.nodes()])

Names = np.array( ['alpha%s' % i for i in G.nodes()])

Fw = VertexWeighting( FiedlerVector, Weights, Names)

cos_sim_Fw = cosine_similarity( [ m.value for m in Fw ])

cos_sim_Fw_1 = [cos_sim_Fw[:, i]
                for i in range(cos_sim_Fw.shape[1])]

PredictPartition(cos_sim_Fw_1)

```

VITA

MOHAMMAD MASUM

Education: B.S. Mathematics, University of Dhaka,
Dhaka, Bangladesh 2012
M.S. Applied Mathematics , University of Dhaka
Dhaka, Bangladesh 2014
M.S. Mathematical Sciences,
East Tennessee State University
Tennessee, USA 2017

Professional Experience: Graduate Assistant,
East Tennessee State University
Tennessee, 2015-2017