Spectral Graph Theory: Help in Finding Communities in Social Networks

David Bakti Lodianto 13523083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523083@mahasiswa.itb.ac.id, [2]david.lodianto@gmail.com

***Abstract*— Spectral Graph Theory provides a helpful mathematical framework for analyzing networks through their graph and matrix representations. This paper explores its application in community detection within social networks, by utilizing spectral clustering methods to partition users based on shared interests and relationships with others. By using the properties of eigenvalues and eigenvectors of the Laplacian matrix of the graph, the natural divisions in the network can be identified. A Python implementation demonstrates the effectiveness of this approach, visualizing clusters and offering insights into the graph's structure. The results reveal that spectral clustering is a robust method for identifying communities, with potential applications in social media platforms and beyond.**

***Keywords*—clustering, graph, eigenvectors, partitions.**

I. Introduction

In the current era, where most is happening online, online social networks play a key role in communication and collaboration, whether it be personal or professional uses. Social media apps such as Instagram, X (Twitter), or LinkedIn have been the main platform for connecting millions of users worldwide through their massive social network. This network tells many interpersonal connections, and how connected those connections are.

These connections can be based on factors, such as the number of interactions the two individuals have, or even the interests these people share in common. By interacting with another person, an individual can get to know more of said person. Therefore, building relationships can be said to have many quality interactions with said person. Another factor is having things in common. People find it easier to connect and communicate with others who share things in common, it could be a phenomenon, an interest, or even common connections to other people. By these factors, understanding social networks can help us in building easier connections with other people if we know which people to connect to.

Analyzing social networks can help reveal many valuable insights to solve important problems, such as understanding how information is spread, or identifying the key individuals who wield the biggest influence in a social setting. These problems are hugely relevant in digital marketing, user management, or in big data analysis.

Analysis is especially helpful to identify communities, groups of people which have a strong relationship with one another. As more and more people join social media platforms, it may be difficult for a new user to hop in and find others to connect with. By grouping said users, they can choose a group of similar people with similar interests or backgrounds, instead of finding one specific user one by one to connect with.

As graphs are a representation of networks, Spectral Graph Theory is a perfect fit for analyzing social networks, as it is a branch of graph theory, which provides more powerful mathematical analysis of graphs with the help of concepts from linear algebra, especially eigenvalues and eigenvectors. Applications of this theory can be used in many fields, from social sciences to computer science. This paper will focus on how Spectral Graph Theory can be utilized to get clusters or partitions in a social network, so that it provides an effective method to group users of a social media app.

II. Theoretical Foundations

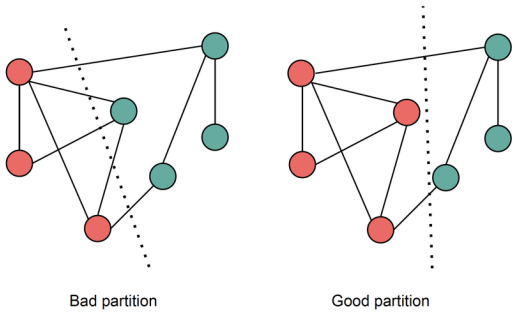
A. *Graph Theory*

Graphs are mathematical representations of a network, which consists of nodes and edges. Nodes represent a single entity, and edges are the connections that nodes have to one another. In the context of social networks, a node can represent an individual user, and an edge represents the relationship between users. A number can be assigned to an edge, showing how close the relationship is, making it a weighted graph.

1. Partition

In graph theory, a **partition** is a group of nodes which have a stronger interconnectivity with one another as composed to the other nodes outside the group. In the context of social networks, this mirrors how communities are. The process of dividing graphs into clusters is called **partitioning**. With partitioning, the clusters will be mutually exclusive, which means every node will belong to one and only one cluster.

This raises a question as to how the graph can be partitioned. A partition must contain a strong interconnectivity inside, and not as strong for the outside.



Picture 2.1. Graph Partitions
Source: SDP: Scalable Real-time Dynamic Graph Partitioner[4]

In partitioning the graph to two partitions, the problem effectively becomes how to **cut** the graph to two disjoint subsets. Fortunately, there is a topic which discusses this.

2. Cut

A **cut** is a way of dividing the nodes (vertices) of a graph into two disjoint subsets, and the **cut size** is the number (or total weight) of edges that have one endpoint in each subset. [5]

Formally, it is defined as the division of nodes V into two subsets S and T such that:

$$S \cup T = V \qquad S \cap T = \emptyset$$

The set of edges that connect nodes in S to nodes in T is called the **cut-set**, and the **cut size** is the number of edges in the cut-set. If the edges have weights, the cut size is the sum of the weights of the edges.

$$\text{cut}(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

Picture 2.2. Cut Size
Source: Spectral Clustering[5]

3. Normalized Cut

A cut(A, B) alone can lead to unbalanced partitions. For instance, isolating a single node as one cluster does minimize the cut, but would result in poor partitioning. To avoid this, the normalized cut balances the partition by incorporating the size (or "volume") of each subset, so that the partitions stay reasonably big, considering the edges with other nodes of its group.

$$\text{Ncut}(A, B) := \text{cut}(A, B)\left(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)}\right)$$

$$\text{vol}(A) = \sum_{i \in A} d_i$$

Picture 2.3. Normalized Cut (Ncut)
Source: Spectral Clustering[5]

B. *Eigenvalues and Eigenvectors*

In linear algebra, eigenvalues and eigenvectors are characteristic components of a square matrix. These play an important role in understanding its linear transformations properties. They can be obtained through this characteristic equation below.

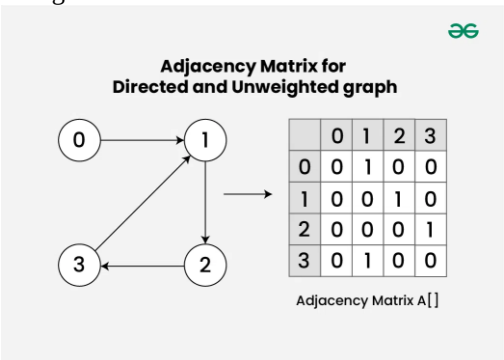
$$\det(\lambda I - A) = 0$$

where A is the square matrix, v is a non-zero eigenvector, and λ is an eigenvalue. An eigenvalue represents the scalar factor of an eigenvector when it is projected by the matrix. In the context of this paper, the eigenvalue and eigenvector are used on the matrix representation of a graph, which is the Laplacian Matrix, to help us identify the most balanced cuts to make.

C. *Matrix Representations of A Graph*

1. Degree Matrix

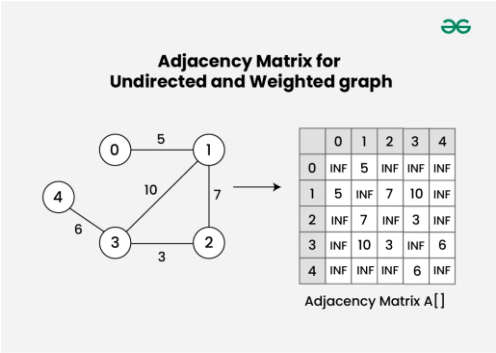
A degree matrix is a diagonal matrix, where the value of a_{ii} is the number of edges that the node has. If the graph is weighted, the value would be the sum of all the edges connected to the node.



Picture 2.4 Degree Matrix
Source: https://www.geeksforgeeks.org/

2. Adjacency Matrix

An adjacency matrix is a matrix representation of a graph where the element a_{ij} is 1 if there is an edge connecting node i and j. If there is no edge, a_{ij} will be 0. If the graph is weighted, the value would be the weight of the edge.



Picture 2.5 Adjacency Matrix
Source: https://www.geeksforgeeks.org/

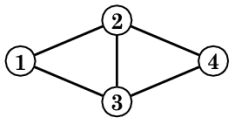
3. Laplacian Matrix

A Laplacian Matrix is one of the matrix representations of a graph, and the key tool to spectral graph analysis. This matrix represents how the nodes are connected, and its properties can provide valuable information on the graph structure. It can identify the number of connected components in the graph based on the number of zero eigenvalues of the matrix. For instance, a graph with one connected component will have a single zero eigenvalue on its Laplacian Matrix. It is defined by this equation below.

$$L = D - A$$

Where D is the degree matrix, and A is the adjacency matrix.

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$



Picture 2.6 Laplacian Matrix
Source:
https://www.researchgate.net/publication/305653264_Experimental_study_on_relationship_between_indices_of_network_structure_and_spectral_distribution_of_graphs

D. Spectral Clustering

Spectral clustering aims to partition a graph G into clusters such that the connections within clusters are strong, and the connections between clusters are as weak as possible. Mathematically, this is often formalized using the **cut** of the graph.

While minimizing this cut ensures weak inter-cluster connections, it can result in unbalanced clusters. To address this, the **normalized cut (Ncut)** is introduced. However, directly optimizing Ncut is computationally intractable (NP-hard). To simplify, the problem is made relaxed using linear algebra.

The problem can be relaxed by encoding cluster membership through a vector *f*, where nodes with similar values belong in a cluster, with positive values, and the other, with negative values. This relaxation transforms the Ncut optimization into minimizing the smoothness of *f* over the graph, represented by the **quadratic form** of the Laplacian matrix L. The quadratic form ensures that nodes with strong connections are assigned similar values in *f*. However, this only considers the cut term in the Ncut equation. The normalization term will then balance the cut.

$$\text{Let } f = [f_1\, f_2 \ldots f_n]^T \text{ with } f_i = \begin{cases} \frac{1}{\text{vol}(A)} & \text{if } i \in A \\ -\frac{1}{\text{vol}(B)} & \text{if } i \in B \end{cases}$$

$$f^T L f = \sum_{ij} w_{ij}(f_i - f_j)^2 = \sum_{i \in A, j \in B} w_{ij} \left(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)} \right)^2$$

$$f^T D f = \sum_j d_i f_i^2 = \sum_{i \in A} \frac{d_i}{\text{vol}(A)^2} + \sum_{j \in B} \frac{d_i}{\text{vol}(B)^2} = \frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)}$$

$$\text{Ncut}(A, B) = \frac{f^T L f}{f^T D f}$$

Picture 2.6. Ncut Relaxation
Source: Spectral Clustering[5]

The optimization problem then becomes finding the minimum Ncut. This is equivalent to solving the eigenvalue problem for the Laplacian matrix L.

$$\min \text{Ncut}(A, B) = \min \frac{f^T L f}{f^T D f}$$

$$\text{where } f = [f_1\, f_2 \ldots f_n]^T \text{ with } f_i = \begin{cases} \frac{1}{\text{vol}(A)} & \text{if } i \in A \\ -\frac{1}{\text{vol}(B)} & \text{if } i \in B \end{cases}$$

$$\text{Relaxation: } \min \frac{f^T L f}{f^T D f} \quad \text{s.t.} \quad f^T D 1 = 0$$

Solution: f – second eigenvector of generalized eval problem

$$\boxed{Lf = \lambda Df}$$

Obtain cluster assignments by thresholding f at 0

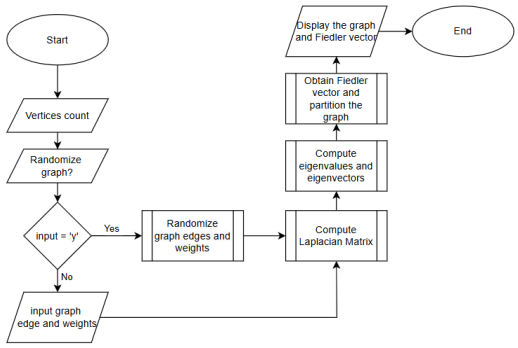
Picture 2.7. Ncut Relaxation to Eigenvalue Problem
Source: Spectral Clustering[5]

Those eigenvalues will then be sorted in ascending order. The first eigenvalue will always be zero, because it represents uniform clustering (there is nothing to cluster when dividing a graph to a single partition).

After obtaining the eigenvalues, the smallest eigenvalue will be zero, and the second smallest eigenvalue will correspond to the Fiedler vector, which is key to dividing the graph to two partitions, as the vector will contain positive and negative values, and they correspond to a partition based on their sign.

III. Implementation

To demonstrate spectral clustering for community detection in social networks, a program implementation was made.



Picture 3.1. Overall Flowchart of the Program
Source: Author

A. Tools and Libraries

The program is made and implemented in Python, considering the amount of tools and libraries available which assisted the implementation of this spectral clustering algorithm. Those tools and libraries include:

1. **NumPy**: For numerical operations, such as matrix manipulation.
2. **SciPy**: For efficient computation of eigenvalues and eigenvectors.
3. **NetworkX**: To create and manipulate the graph representation of the social network.
4. **Matplotlib**: For visualizing the graph and its clusters.

B. Functions

a. create_graph

First, the user can construct the graph, by inputting the amount of vertices in the graph. After that, the program will generate a randomized graph (with the probability of an edge forming between all nodes being 0.5). Then, the user will be asked whether to randomize the weights of the formed edges or to input them manually. Finally, the graph is constructed.

```
def create_graph(vertices, randomize_weights):
    random_graph = nx.erdos_renyi_graph(n=vertices, p=0.5, seed=100)
    for (u, v) in random_graph.edges():
        if randomize_weights:
            random_graph[u][v]['weight'] = np.random.randint(1, 20)
        else:
            random_graph[u][v]['weight'] = int(input(f"Enter the weight of edge ({u}, {v}): "))
    return random_graph
```

Picture 3.2. create_graph function
Source: Author

b. compute_laplacian_matrix

In the next step, the matrix representations of the graph will be computed. First, the adjacency matrix is obtained, and then the degree matrix is computed from the sum of the weights of all the edges connected to each node. Lastly, the Laplacian matrix is obtained from subtracting the degree matrix with the adjacency matrix.

```
def compute_laplacian_matrix(graph):
    adjacency_matrix = nx.to_numpy_array(graph, weight='weight')
    degree_matrix = np.diag(np.sum(adjacency_matrix, axis=1))
    laplacian_matrix = degree_matrix - adjacency_matrix
    return laplacian_matrix
```

Picture 3.3. compute_laplacian_matrix function
Source: Author

c. spectral_clustering

Next, the eigenvectors are extracted from the Laplacian matrix. The Fiedler vector is then obtained from the second smallest eigenvector, and based on the Fiedler vector, the graph is divided into two clusters, negative values gets grouped to cluster_1, and positive to cluster_2.

```
def spectral_clustering(laplacian_matrix):
    _, eigenvectors = eigh(laplacian_matrix)
    fiedler_vector = eigenvectors[:, 1]
    cluster_1 = np.where(fiedler_vector < 0)[0]
    cluster_2 = np.where(fiedler_vector >= 0)[0]
    return fiedler_vector, cluster_1, cluster_2
```

Picture 3.4. spectral_clustering function
Source: Author

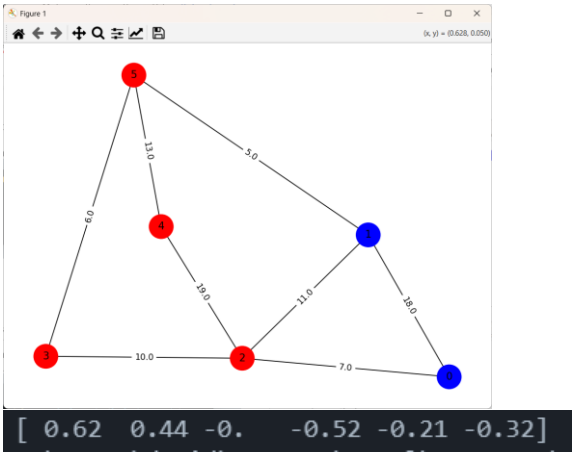
d. visualize_graph

Lastly, the graph is visualized based on the clusters made, with cluster_1 being colored red, and cluster_2 blue. The graph nodes will be scattered, with the edge weight being present.

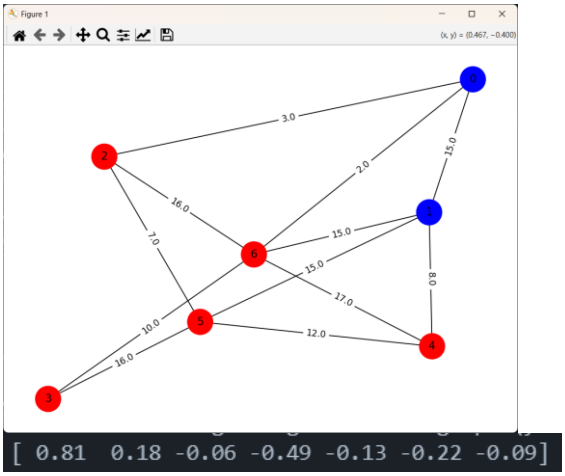
```
def visualize_graph(adjacency_matrix, fiedler_vector, cluster_1, cluster_2):
    graph = nx.from_numpy_array(adjacency_matrix)
    node_colors = ['red' if node in cluster_1 else 'blue' for node in range(len(fiedler_vector))]
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(graph)
    nx.draw(graph, pos, with_labels=True, node_color=node_colors, node_size=800, cmap=plt.cm.rainbow)
    edge_labels = nx.get_edge_attributes(graph, 'weight')
    nx.draw_networkx_edge_labels(graph, pos, edge_labels=edge_labels)
    plt.title("Spectral Clustering of Social Network with Edge Weights")
    plt.show()
```

Picture 3.5. visualize_graph function
Source: Author

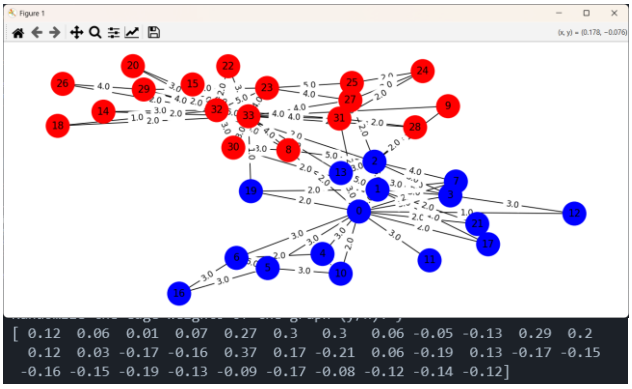
IV. Results and Analysis



Picture 4.1. Program Execution Results with 6 Vertices



Picture 4.2. Program Execution Results with 8 Vertices



Picture 4.3. Program Execution Results with 34 Vertices (nx.karate_club_graph)

The sign of the Fiedler vector's values determined the cluster membership of each node. Nodes with positive values were grouped in one cluster, while those with negative values formed another. Other than that, the magnitude of the Fiedler vector's values indicates how strongly a node was associated with its cluster. Nodes with values close to zero were observed to have weaker connections, being more neutral to neither clusters, and serving as boundary points between clusters. This insight is critical in understanding the structure of social networks, where such nodes may represent individuals with connections across communities.

By examining the difference between the largest and smallest Fiedler vector values, the natural partitionability of the graph can be assessed. Larger differences suggests a clearer natural partitioning, whereas smaller differences indicates less distinct clusters. This metric can serve as a quantitative measure of the clustering's quality.

However, this program also comes with its limitations, as it is only able to partition the graph to two clusters only, because it only relies on the Fiedler vector. The visualization is also quite poor on graphs which are more crowded in nodes and weights.

V. Conclusion

Spectral Graph Theory proves to be a powerful mathematical help for community detection in social networks, especially spectral clustering, offering a computationally feasible and effective method for partitioning graphs. By utilizing the eigenvalues and eigenvectors of the Laplacian matrix, this approach identifies natural divisions in the graph, making the grouping of users based on shared relationships or interests possible. The Python implementation highlights the method's effectiveness and provides a practical example of its application. Future work could be explored extending this method to dividing to more than two clusters, dynamic graphs, or enabling real-time community detection in evolving social networks.

VI. Acknowledgements

I would like to acknowledge and express my gratitude to the IF2123 - Linear and Geometric Algebra lecturers, Ir. Rila Mandala, M.Eng., Ph.D., and Dr. Ir. Rinaldi Munir, M.T., for their lectures throughout the semester, which helped contributed a lot on making this paper possible. Their expertise has greatly enriched my understanding of the subject matter.

I am also grateful to my peers and colleagues in the who provided constructive feedback and support during the research and writing process.

References

[1] Munir, Rinaldi. 2024. Graf (Bagian 1), https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, accessed on December 28, 2024.

[2] Munir, Rinaldi. 2024. Nilai Eigen dan Vektor Eigen (Bagian 1), https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2023.pdf, accessed on December 28, 2024.

[3] Singh, Aarti. 2010. Spectral Clustering, https://www.cs.cmu.edu/~aarti/Class/10701/slides/Lecture21_2.pdf, accessed on December 28, 2024

[4] Patwary, Md Anwarul & Garg, Saurabh & Battula, Sudheer & Kang, Byeong. (2021). SDP: Scalable Real-time Dynamic Graph Partitioner. 10.48550/arXiv.2110.15669, accessed on December 29, 2024.

[5] Veni, Gopalkrishna. n.d., Normalized Graph Cuts, https://www.sci.utah.edu/~gerig/CS7960-S2010/handouts/Normalized%20Graph%20cuts.pdf accessed on December 29, 2024.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 28 Desember 2024

David Bakti Lodianto 13523083