VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
INFORMATION TECHNOLOGY FACULTY

REPORT ASSIGNMENT OF

COMBINATORICS AND GRAPHS SUBJECT

THE SOCIAL NETWORK ANALYSIS PROBLEMS

Instructing Lecturer: Mr. NGUYEN KHANH TUNG

Student's name: **HUYNH LE THIEN Y – 518H0320**

Class **: 18H50204**

Course **: 22**

HO CHI MINH CITY, YEAR 2020

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
INFORMATION TECHNOLOGY FACULTY

REPORT ASSIGNMENT OF

COMBINATORICS AND GRAPHS SUBJECT

THE SOCIAL NETWORK ANALYSIS PROBLEMS

Instructing Lecturer: Mr. NGUYEN KHANH TUNG

Student's name: **HUYNH LE THIEN Y – 518H0320**

Class **: 18H50204**

Course **: 22**

HO CHI MINH CITY, YEAR 2020

ACKNOWLEDGEMENT

With deep affection, sincerity, we express deep gratitude to all individuals and agencies who have helped us in our study and research. First of all, I would like to express a special appreciation to Ton Duc Thang University's teachers for their conscientious guidance and advices throughout the last semester by gave me their modern outlook and meticulous supervision to carry out the job perfectly. Especially I would like to send my sincere thanks to Mr. Nguyen Khanh Tung has paid attention, help me to complete the report well over the past time. I would like to express my sincere gratitude to the leadership of Ton Duc Thang University for supporting, helping and facilitating me to complete the report well during the study period. With limited time and experience, this report cannot avoid mistakes. I'm looking forward to receiving advice and comments from teachers so that I can improve my awareness, better serve the practical work later. I sincerely thank you!

THE PROJECT WAS COMPLETED AT TON DUC THANG UNIVERSITY

I pledge that this is a product of our own project and is under the guidance of Mr. Nguyen Khanh Tung. The content of research, results in this subject is honest and not published in any form before. The data in the tables used for the analysis, comment, and evaluation were collected by the authors themselves from various sources indicated in the reference section. In addition, many comments and assessments as well as data from other authors and organizations have been used in the project, with references and annotations. If any fraud is found, I am fully responsible for the content of my project. Ton Duc Thang University is not involved in any copyright infringement or copyright infringement in the course of implementation (if any).

Ho Chi Minh city, May 22th 2020

Author

(Signed)

Huynh Le Thien Y

EVALUATION OF INSTRUCTING LECTURER

Confirmation of the instructor

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh city , month day year
(Sign and write full name)

The assessment of the teacher marked

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh city , month day year
(Sign and write full name)

TABLE OF CONTENTS

LIST OF TABLES, DRAWINGS, GRAPHICS

CHAPTER 1: SOCIAL NETWORKS AND SOCIAL NETWORK ANALYSIS PROBLEMS

1.1 Concept

Nowadays, when it comes to the term “social network”, people probably think of the Internet or social media platforms like Facebook, Twitter, etc., but actually, this term has broader applications than that. In general, social networks can be defined as a collection of relationships between social entities and collectively referred to as actors. These social entities are not obligated to be individuals but social groups, organizations, institutions, enterprises and even nations. Relationships between actors can also have many different types such as support, information exchange, goods exchange, service exchange ...

From the above definition of social networks, it can be determined that Social Network Analysis (SNA) is a method for visualizing features in order to detect relationships or structures of social networks. This method maps and measures relationships between social entities to boost communities, identify missing links, and improve connections between groups.

In order to conduct social network analysis, it may be necessary to start by capturing some of the terms and concepts commonly used in this analysis.

- **Actor / Node / Point / Agent:** These terms are used for units in the network. These entities can be people, groups, organizations, companies, countries ...
- **Tie / Link / Edge / Line / Arc:** is used to refer to links between actors in the network.
- **Step / Pas:** The term indicates the distance between two certain agents in the network. For example, if A and B are directly related, the distance between these two agents will be 1 step.

- **Path:** Refers to the total number of steps that an actor must "pass" to reach one / all other actors in the network.
- **Clique:** This term is used to refer to networks where all actors are directly related to each other.
- **Geodetic distance (shortest distance):** The term indicates the shortest distance between two certain agents in the network.
- **Uniplexity:** Used to refer to the relationship of only one "type" between two or more actors. For example, A and B are only related to each other as colleagues.
- **Multichannel / Multiplexing:** When the relationship between two actors has more than one type. For example, A and B relate to each other because they are classmates, countrymen, colleagues (i.e. 3 types).
- **Reciprocity:** The term indicates a two-way relationship between two agents in the network. For example, the relationship between mother and child, friends ...
- **Symmetry:** When two actors have reciprocal relationships and the same kind. For example, A and B see each other as friends.
- **Asymmetry**: When two actors have reciprocal relations but not the same type. For example, A and B are related, but A is a teacher and B is a student, the relationship between A and B is not the same type.

1.2 How Important It Is?

Social networking can help businesses gain more relationships with other companies and customers, or raise public awareness. Even entrepreneurs who run small companies can use this resource to enter the global market.

Social network analysis is important at different levels and for many reasons.

- About the levels:

- *Micro analysis.* At this level, the data analysis is only at the analysis of your data or some small group of social data. This can give suggestions to support your tastes or the tastes of your friends, or to an understanding level for every community found in an extremely large social network, the way they are. affect other communities, or the way people in each interact and become influencers with their peers.
- *Macro analysis*. On a larger scale, one can ask to analyze huge networks, in which interesting measures such as centrality or the use of algorithms such as community search are important.

➢ <u>About the reasons:</u>

- *Marketing.* Social networking is the most important because it helps businesses gain valuable information in marketing, like communities or influencers. All of these interesting aspects can also be useful for Community Managers or Marketing Campaign Designers, targeting products to exactly individual groups in social networks.
- *Crime detection.* Social networks carry a lot of data that can be opened or encrypted. Criminal organizations can use this data to communicate so the analysis of that data is also important.
- *Disease detection.* The spread of disease in diseases is detected through the way people spread disease information on social networks. Such sensitivity analysis may also be important to capture when and where an epidemic is widespread.
- *Relationship and reputation*. It's important to understand that social media is regulating the way people think about each other. Using social networks can also be very important to capture how people trust and how much they trust other individuals, in many situations.

1.3 How To Model Social Network Using A Graph?

Social network is represented as undirected graph G (V, E) where V is that the set of vertices representing actors and E is that the set of edges representing social ties like friendship, common interest, financial exchange, sexual relationships, or relationships of beliefs, knowledge or prestige. Graphs are accustomed represent communication networks or social networks and their basic properties which is analyzed can help to evaluate and improve the performance of networking solution.

1.4 Problems In Social Network Analysis

There are two problems that will be discussed in this report: Measurements and Community Detection.

There are different performance measures that are encountered during any social network analysis in order to understand the fundamental concepts. The four most important concepts used in network analysis which concern structure are closeness, network density, centrality, betweenness and centralization. Additionally, there are four other measures of network performance which concern dynamics that include: robustness, efficiency, effectiveness and diversity. This will be discussed in more detail in Chapter 2.

Community detection is essential to understanding the structure of complex networks and ultimately extracting useful information from them. The applications range from healthcare to regional geography, from human interaction and mobility to economics. Community structure can exist in all matters, so it is necessary to expand the community detection method for all fields. The community detection method applies to any optimization or analysis problem and is scalable, using computationally efficient graph theory algorithms. This will be discussed in more detail in Chapter 3.

CHAPTER 2: SOCIAL NETWORK MEASUREMENTS

2.1 Describe A Social Network Using A Graph

Social network is a structure within which nodes are a group of social actors that are connected together by differing types of relationships.

Here's a technique to represent a social network:

Figure 1. *Example of undirected graph*

A line between the names of two people implies that they know one another. If there is no line between two names, then the people don't know one another. the link "know each other" goes both ways; for instance, because Eric knows Jack, meaning Jack knows Eric. Because "know each other" relationship goes both ways, this graph is undirected.

This social network is a graph denoted by G=(V,E). The names are the vertices (V) of the graph. (If you're talking about only one of the vertices, it is a vertex.) Each line is an edge (E), connecting two vertices.

The relationship between vertices doesn't always go both ways. Here's a graph showing a graph database:

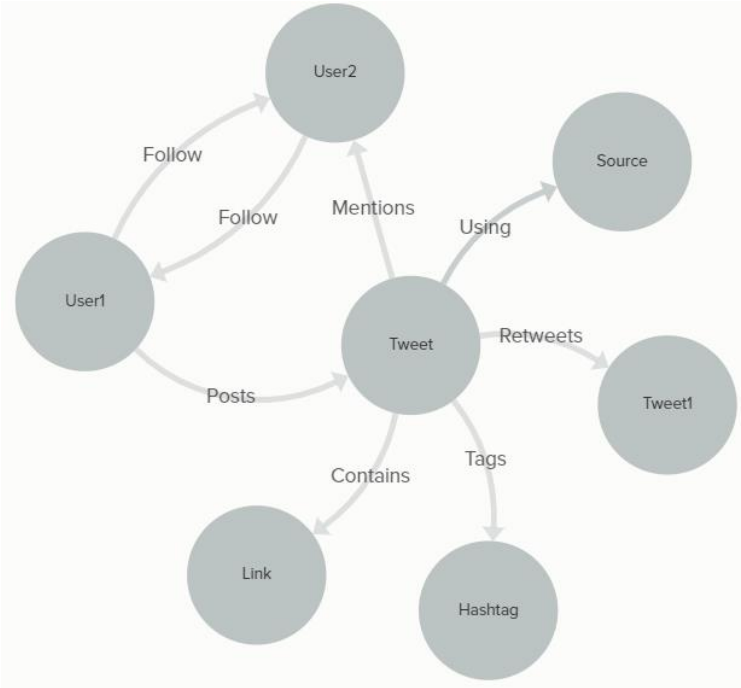


Figure 2. *Example of directed graph*

Now edges, shown with arrows, are directed, and that we have a directed graph.

As you may imagine, graphs—both directed and undirected—have many applications for modeling relationships within the world.

There are several key measurements related to social network, but during this report, we are going to discuss two measurements: Density, Centrality.

2.2 Density

Density is a percentage between the number of connections between existing actors and the maximum number of connections that actors can have in a graph. To calculate the density of a undirected graph, simply as follows:

$$UndirectedDensity = \frac{SumEdges}{SumPossibleEdges} = \frac{Cardinality}{Size} = \frac{m}{n(n-1)/2}$$

Where n is the number of nodes in the network.

The density will range from 0 (without edges) to 1 (with all possible edges).

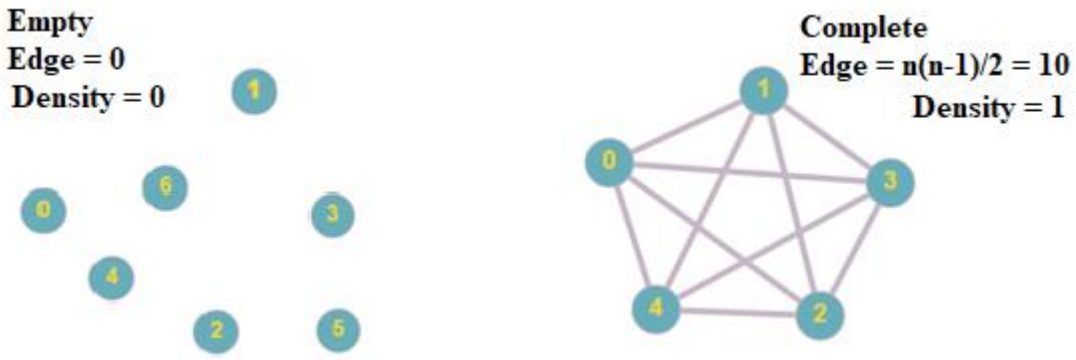


Figure 3. *Example of density*

On account of a directed graph, there is no compelling reason to divide the numerator by two. Thusly, the equation to calculate directed network density is:

$$DirectedDensity = \frac{SumEdges}{SumPossibleEdges} = \frac{Cardinality}{Size} = \frac{m}{n(n-1)}$$

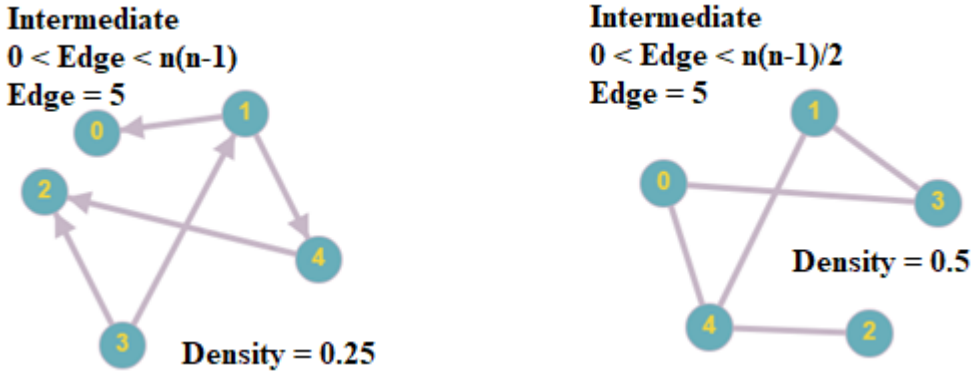


Figure 4. *Example for undirected density and directed density*

The density of a network property is very important to contemplate for 2 reasons. First, is that it can help us understand how connected the network is compared to how connected it would be. Second, when comparing two networks with the identical number of nodes and also the same sort of relationships, it can tell us how the networks are different.

2.3 Centrality

How should we define the concept of centrality? We would imagine that somebody “central” to the network is someone who holds some kind of important, advantaged position within the network. The matter is, that there are plenty of specific ways we are able to consider centrality, and thus other ways of mathematically defining the concept.

Centrality measures are often defined in relevancy the various ways within which information is assumed to propagate across nodes, which may occur through:

(1) walks, which follow an unrestricted trajectory through the network.

(2) trails, which may return to a visited node but cannot reuse an edge.

(3) paths, which cannot visit a node or edge quite once.

Hence, paths are a subset of trails which, in flip, are a subset of walks. Most centrality measures developed up to now have focused on walks and paths. In this report, we'll discuss about four individual centrality measures: Degree Centrality, Betweenness Centrality, Closeness Centrality.

In the following are description of each metric and how it relates to the Kite network (Figure 5).

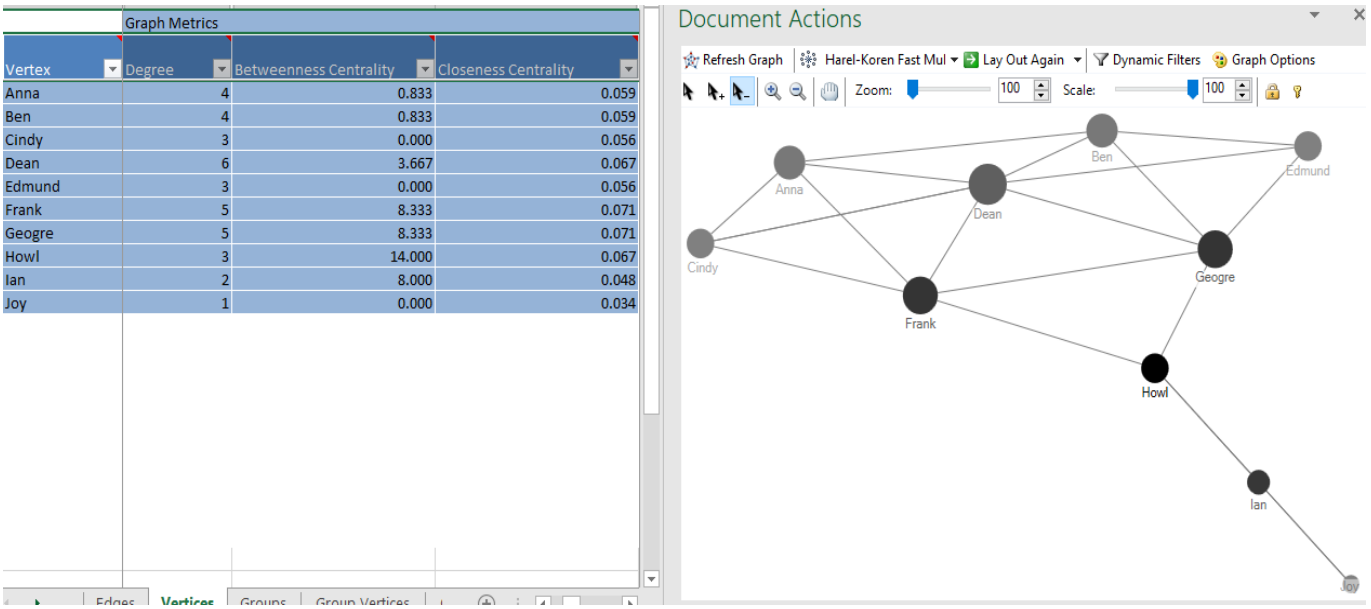


Figure 5. *The Kite network showing centrality metrics for each vertex*

a. *Degree Centrality*

Definition: The degree centrality specifies an important point simply based on the number of links held by each node.

What it tells us: The direct number, a connection that each node must have with other nodes in the network.

When to use it: To locate highly connected individuals, common individuals, individuals with the most information, or individuals who can quickly connect to a wider network.

A little more detail: Degree centrality is the simplest measure of node connection. Sometimes, it is beneficial to appear in-degree (many inbound links) and out-degree (many outbound hyperlinks) as great measures, for example when viewing transaction data or financial interest.

In the Kite network above, Dean encompasses a degree of 6 because she is directly connected to 6 other individuals. compared, Joy encompasses a degree of only one because she is connected to just one other person. If the perimeters represented the strong

friendship ties of people in an exceedingly class, we'd say that Dean is that the hottest person within the class and Joy is that the least popular.

b. Betweenness Centrality

Definition: Betweenness centrality measures the number of times a node is on the shortest path between other nodes.

What it tells us: This measure shows which nodes are 'bridges' between nodes in an extremely network. It does this by identifying all the shortest paths then counting the percentage of times each node falls into one.

When to use it: To locate individuals that affect the flow around a system.

A little more detailed: Betweenness is useful for analyzing communication dynamics, but should be used with care. The high number of mid-high numbers may indicate that someone is in charge of different clusters in an extremely network, or simply they are on the periphery of both clusters.

Although popularity is vital, it's not everything. Consider Howl within the Kite network. She is directly associated with only three people (i.e., she incorporates a degree of 3). Despite her relatively low degree, her position as a "bridge" between Ian (and indirectly Joy) to the remainder of the group could also be of utmost importance. If, as an example, information were passed from one person to a different, Howl would be vital for assuring that Ian and Joy could communicate with the remainder of the group. In fact, if she were aloof from the network, Ian and Joy would be disconnected from the opposite class members. Thus, Howl has high betweenness centrality. In contrast, Edmund incorporates a betweenness centrality of 0. Notice that if he were aloof from the graph, everyone would still be connected to everyone else, and their shortest communication paths wouldn't even be altered.

c. Closeness Centrality

Another characteristic you'll care about is how close every person is to the opposite people within the network. If information needed to flow through the network, some people would be able to get a message to any or all the opposite people relatively quickly (i.e., in few steps), whereas others may require many steps. The lowest possible Closeness Centrality score is 1, which might indicate an individual is directly connected to everyone within the network.

Definition: Closeness centrality scores each node supported their ‘closeness’ to any or all other nodes within the network.

What it tells us: This measure calculates the shortest paths between all nodes, then assigns each node a score supported its sum of shortest paths.

When to use it: for locating the individuals who are best placed to influence the whole network most quickly.

A bit more detail: Closeness centrality can help find good ‘broadcasters’, but in an exceedingly highly-connected network, you may often find all nodes have an identical score. What could also be more useful is using Closeness to seek out influencers in an exceedingly single cluster.

In the Kite network, Frank and Geogre have fewer connections than Dean, yet the pattern of their direct and indirect ties allow them to access all the nodes within the network more quickly than anyone else. They have the shortest paths to any or all others -- they're near everyone else. They're in a wonderful position to observe the knowledge flow within the network -- they have the most effective visibility into what's happening within the network.

2.4 Clustering Coefficient

The clustering coefficient metric is different from the centrality measurements. it's sort of a measurement of density for the complete network, but specializing in natural networks. Specifically, the clustering coefficient may be a measure of the density of the

natural network of 1.5 degrees for every vertex. When these connections are dense, the clustering ratio is high. If all of your friends and other friends know one another, you have got a high clustering coefficient . If your friends, others (who change) do not know one another, then you have got a low clustering coefficient. People have different measures for his or her clustering coefficient counting on how they cultivate reference to others and therefore the environment they're in.

There are two formal definitions of the Clustering Coefficient: “global” and “local”. Though they're slightly different, they both cope with the probability of two nodes that are connected to a standard node being connected themselves.

a. Global Clustering Coefficient

A triplet is three nodes that are connected by either two or three undirected ties. A triangle graph therefore includes three closed triplets, one centered on each of the nodes . The global clustering coefficient is the number of closed triplets over the total number of triplets .

Defined as: $C = \frac{the\ amount\ of\ closed\ triplets}{the\ amount\ of\ all\ triplets}$

b. *Local Clustering Coefficient*

Local clustering coefficient is sort of a local version of betweenness centrality: where betweenness centrality measures a vertex's control over information flowing between all pairs of nodes in its component, local clustering measures control over flows between just the immediate neighbors of a vertex.

The clustering coefficient C_i of a vertex i is that the frequency of pairs of neighbors of i that are connected by an edge, that is, it’s the ratio of the amount m_i of pairs of neighbors of i that are connected and also the number of possible pairs of neighbors of i, which is $\frac{k_i(k_i-1)}{2}$where k_i is the degree of i:

$$C_i = \frac{2m_i}{k_i(k_i - 1)}$$

In other words, C_i is the probability that a pair of neighbors of i are connected. The values of C_i ranges from 0 to 1: the floor represent thing where no pairs of neighbors of i are connected, while the ceil holds when all pairs are linked. In the latter case, the set of neighbors of i extended with i itself forms a complete graph or clique, that is a graph where each pair of node is connected by an edge.

For example, consider the subsequent undirected graph. Node 0 has 4 neighbors, and 4 pairs over 6 among these neighbors are linked. Hence $C_0 = \frac{4}{6} = 0.66$. Node 1 (as well as nodes 2, 3, and 4) has 3 neighbors, and 2 pairs over 3 among them are connected, hence $C_1 = \frac{2}{3} = 0.66$.

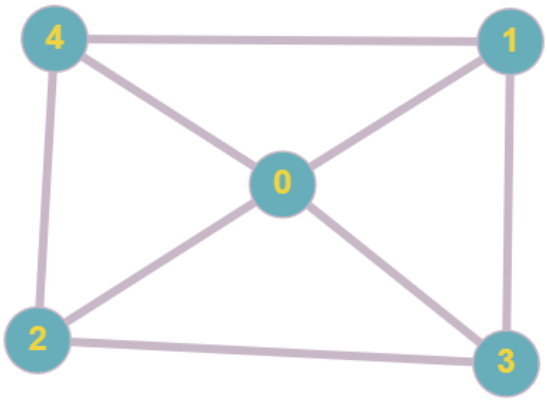


Figure 6. *Example for local clustering coefficient*

c. Average Local Clustering Coefficient

If C_i is that the proportion of two nodes connected to node i that also are connected to every other (i.e., the Local Clustering Coefficient), then Average Local Clustering Coefficient= $\frac{1}{n}\sum_{i=1}^{n} C_i$.

2.5 Key Players And How To Detect Key Players

a. Concept

Key players are nodes within the network that are considered to be the foremost important with several criteria. In general, the importance of a node is measured in many ways. These nodes have a heavy impact on the cohesion and communication patterns that occur within the network.

b. How to Detect

Key player detection is an important step while analyzing covert networks. The proposed framework for key player detection consists of centrality measures for each node followed by hybrid classifier for accurate detection of key players. Key players normally appear as most central nodes in any network; so, they have significant values of centrality measures. Centrality measures offer valuable clues for identifying influential actors, but there are other measures that add effective solutions adapted to meet specific needs for detecting key players. The concept of centrality has been applied not only to single individuals within a network but also to groups of individuals. Measures for degree centrality, closeness, and betweenness are defined for a group. Using these measures, groups having high centrality will be the key players. It must be remarked that group centrality can be used not only to measure how «central» or important a group is, but also in constructing groups with maximum centrality within an organization.

For instance, a team of experts can be distributed within an organization in such a way that it has high group centrality. The method is based on the concept of optimal inter-centrality. Inter-centrality measure takes into account a player's own centrality and its contribution to the centrality of others. The individual optimal inter-centrality measure is then generalized to groups of players. The group with the highest inter-centrality measure is the key group.

2.6 Signed Graph, Its Problems, And Applications

In describing the pattern of who describes whom as a close friend, we could have asked our question in several different ways. This kind of data is called "signed" data. The graph with signed data uses a + on the arrow to indicate a positive choice, a - to indicate a negative choice, and no arrow to indicate neutral or indifferent. This would give us information about the value of the strength of each choice on a ratio level of measurement.

a. Concept

The signed graph $\Gamma = (G, \sigma)$ is a graph with edges that carry a positive or negative sign (+/-). It is a graph G=(V,E) together with function $\sigma: E \rightarrow \{+, -\}$ represents the positive and negative sign for each edge. The three basic questions about a signed graph are:

- Is it balanced?
- What is the maximum size of an equilibrium edge placed in it?
- What is the minimum number of vertices to be deleted to make it balanced?

The first question is easy to solve quickly; second and third are computable.

Cycles and properties of cycles play an important role in many properties of signed graphs. Cycle is the process of traversing a graph so that there is no repeat of a vertex or edge but the beginning and ending vertices must be the same. A signed graph is balanced if the product of the edge signs around each cycle is positive. Here is an example of signed graph.

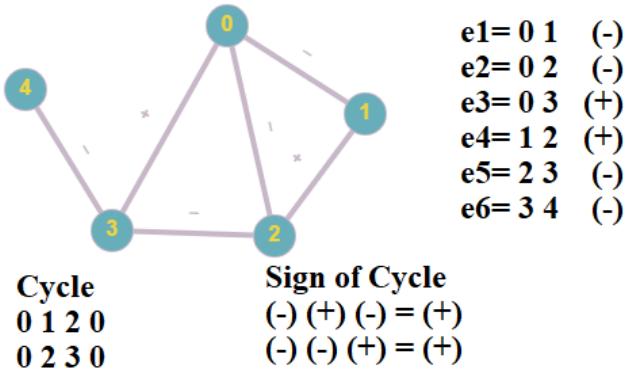


Figure 7. *Example of signed graph*

b. Some Problems

Balance

The first characterization of balance is from Harary:

Theorem: A signed graph is balanced iff its vertex set can be divided into two sets (either of which may be empty), so that each edge between the sets is negative and each edge within a set is positive.

The above theorem indicates that balance is a generalization of the ordinary bipartiteness in (unsigned) graphs.

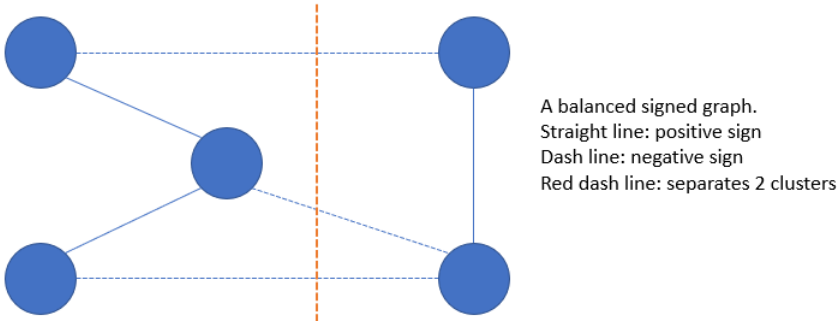


Figure 8. *Example of balanced signed graphs*

Signature Switching

Definition: Let Γ = (G, σ) be a signed graph and U ⊆ V(G). The signed graph Γ U obtained by negating the edges in the cut [U;U_c] is a (sign) switching of Γ. We also say that the signatures of Γ^U and Γ are equivalent.

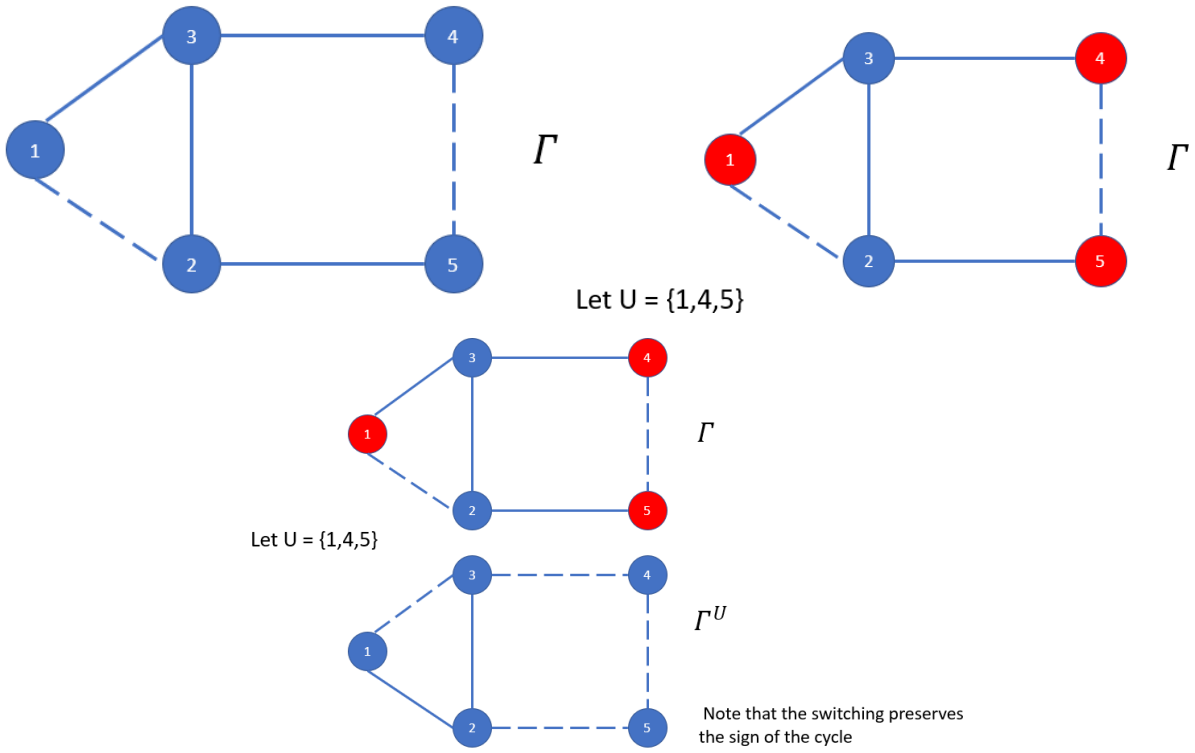


Figure 9. *Example of switching equivalent graphs*

Laplacian Of Signed Graph

The Laplacian matrix of $\Gamma = (G, \sigma)$ is defined as:

$L(\Gamma) = D(G) - A(\Gamma) = (l_{ij})$

$$l_{ij} = \begin{cases} \deg(v_i), & if\ i = j \\ -\sigma(v_i v_j), & if\ i \neq j \end{cases}$$

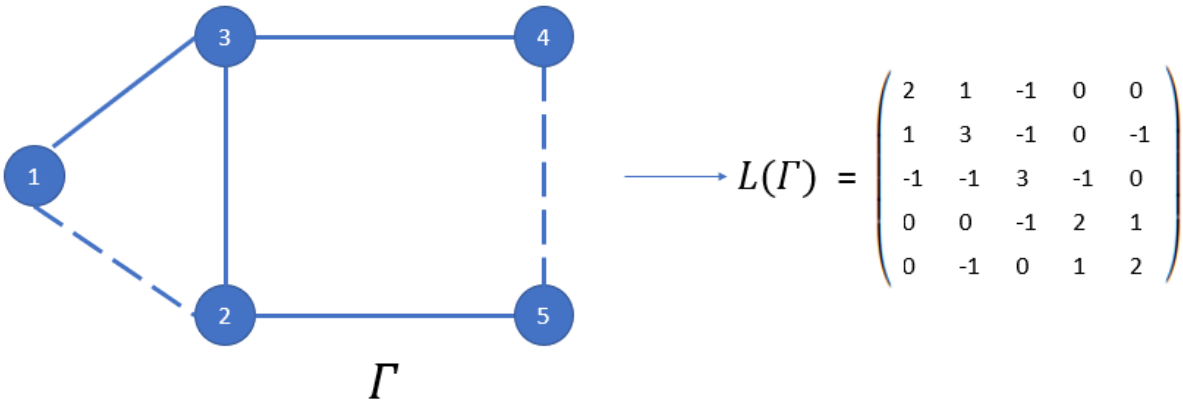


Figure 10. *Example for Laplacian of signed graph*

c. Applications

Signed graphs is used to illustrate good and bad relationships between humans. A positive relationship (family, dating) is denoted by a positive edge between two nodes and a negative relationship (hatred, anger) is denoted by a negative edge between two nodes. Signed social network graphs is used to predict the longer term evolution of the graph. In signed social networks, there's the concept of "balanced" and "unbalanced" cycles. A cycle where the product of all the signs are positive defines a balanced cycle. In keeping with balance theory, balanced graphs represent a group of individuals who are unlikely to vary their opinions of the opposite people within the group. Unbalanced graphs represent a group of people who are very probable to differ their opinions of the people of their group. As an example, an unbalanced cycle is a group of three people (A, B, and C) where both A,B and A,C have a positive relationship, but B,C have a negative relationship. This group is incredibly likely to morph into a balanced cycle, like one where B only encompasses a good relationship with C, and both B and C have a negative relationship with A. By using the concept of balanced and unbalanced cycles, the evolution of signed social network graphs is predicted.

CHAPTER 3: COMMUNITY DETECTION

A community within a network is a very similar group of nodes and is different from the rest of the network. It is often considered a bundle where nodes are heavily linked and sparsely connected to other parts of the network. Recent efforts have been made to identify, identify and identify communities in real world networks. The goal of the community detection algorithm is to find groups of nodes of interest in a very specific network. The motivation behind community detection is unique: it can help a brand understand different opinion groups for its products, target certain groups of people or

identify influencers. It can also help an ecommerce site build a recommendation system based on generic purchases, numerous examples.

One of the most common problems when analyzing a social network is the detection of communities according to certain properties. There are many approaches to the problem of community detection. The following are the main approaches:

- Node-Centric Community: Each node in the community will meet a certain property
- Group-Centric Community: Considering all links within a group to be a unique link within the community, each group will meet certain properties. We do not consider each node but a group of nodes.
- Network-Centric Community: Divide social networks into subsets of non-connected nodes.
- Hierarchy-Centric Community: Build a hierarchical structure from social networks.

The problem of detecting community structure in the network is the most concerned problem in mathematics, biology and sociology. There are many algorithms proposed to solve the following:

- The algorithm detects a structure based on the similarity of vertices: Girvan-Newman's Algorithm
- The algorithm detects community structure based on node similarity: Node Similarity Based Algorithm.
- Label Propagation Algorithm.

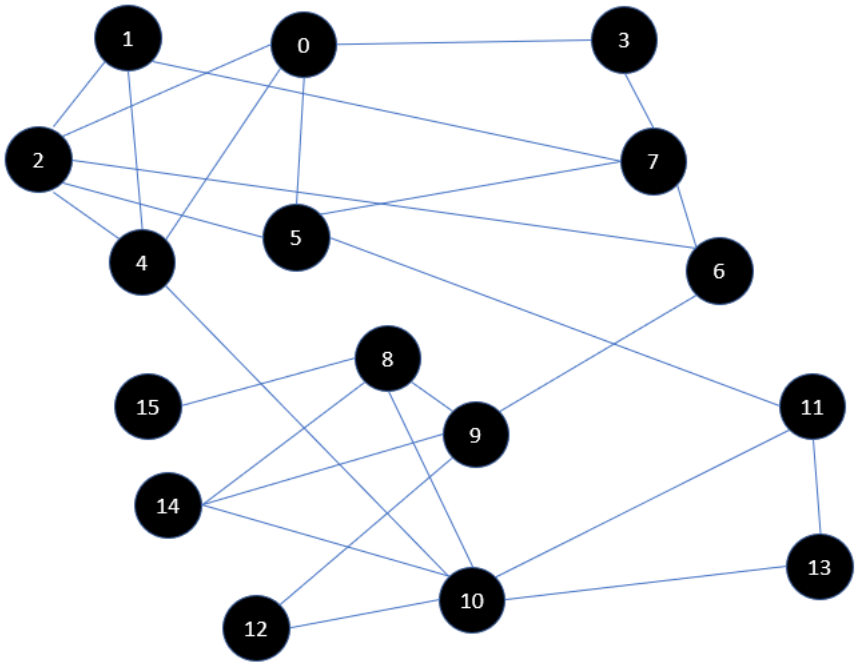


Figure 11. *A sample social network*

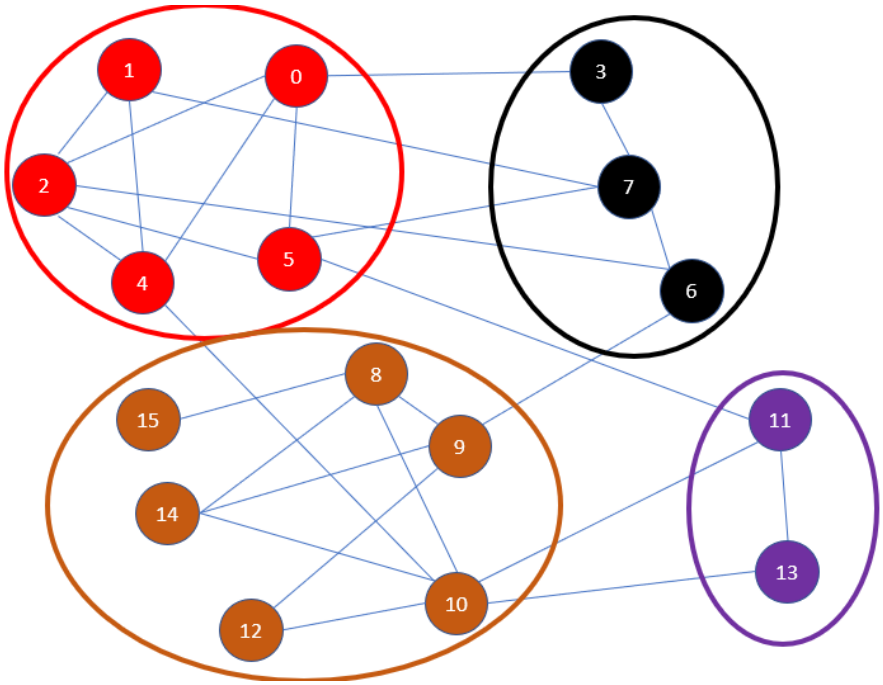


Figure 12. *Detected Communities on the network*

3.1 Modularity And Cut

a. Modularity

Modularity (Q) is a measure of the structure of a network or graph. It's used to measure the strength of division of a community into modules (additionally known as businesses, clusters or communities). Highly modular networks have dense connections between nodes in modules but sparse connections between nodes in multiple modules. In optimization methods to detect community structure in networks, modularity is often used. However, modularity is subject to resolution limits and therefore, it cannot detect small communities, which has been proven. Biological networks, including the animal brain, represent a high level of modules.

Consider two node v, w. Let the adjacency matrix for the network be represented by A. The strength of division of a community is calculated by the following formula:

$$\sum_{v,w} A_{vw} - \frac{d_v d_w}{2m}$$

Modularity: $$Q = \frac{1}{4m} \sum_{vw} (A_{vw} - \frac{d_v d_w}{2m})$$

where:

m : the amount of edges in the graph

d_v, d_w: node degree

$\frac{1}{4m}$: normalization

An alternative formulation of the modularity, useful specifically in spectral optimization algorithms, is as follows. Define S_v to 1 if vertex v belong to group g and zero otherwise.

$$Q = \frac{1}{4m} \sum_{vw} (A_{vw} - \frac{d_v d_w}{2m}) S_v S_w = \frac{1}{4m} S^T B S$$

where S is the (non-square) matrix having elements S_v and B is the so-called modularity matrix, which has elements $B_{vw} = A_{vw} - \frac{d_v d_w}{2m}$

Consider the following graph to detect communities:

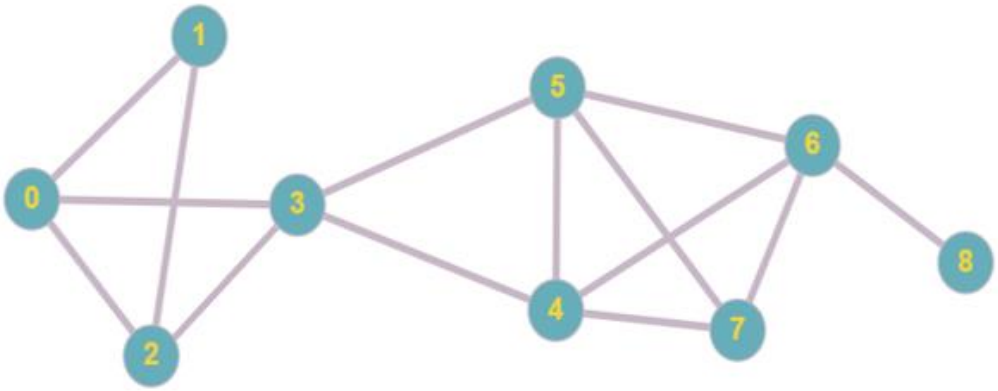


Figure 13. *Graph to detect communities*

$$B = \begin{bmatrix} -0.32 & 0.79 & 0.68 & 0.57 & -0.43 & -0.43 & -0.43 & -0.32 & -0.11 \\ 0.79 & -0.14 & 0.79 & -0.29 & -0.29 & -0.29 & -0.29 & -0.21 & -0.07 \\ 0.68 & 0.79 & -0.32 & 0.57 & -0.43 & -0.43 & -0.43 & -0.32 & -0.11 \\ 0.57 & -0.29 & 0.57 & -0.57 & 0.43 & 0.43 & -0.57 & -0.43 & -0.14 \\ -0.43 & -0.29 & -0.43 & 0.43 & -0.57 & 0.43 & 0.43 & 0.57 & -0.14 \\ -0.43 & -0.29 & -0.43 & 0.43 & 0.43 & -0.57 & 0.43 & 0.57 & -0.14 \\ -0.43 & -0.29 & -0.43 & -0.57 & 0.43 & 0.43 & -0.57 & 0.57 & 0.86 \\ -0.32 & -0.21 & -0.32 & -0.43 & 0.57 & 0.57 & 0.57 & -0.32 & -0.11 \\ -0.11 & -0.07 & -0.11 & -0.14 & -0.14 & -0.14 & 0.86 & -0.11 & -0.04 \end{bmatrix}$$

Figure 14. *Modularity matrix*

b. Cut

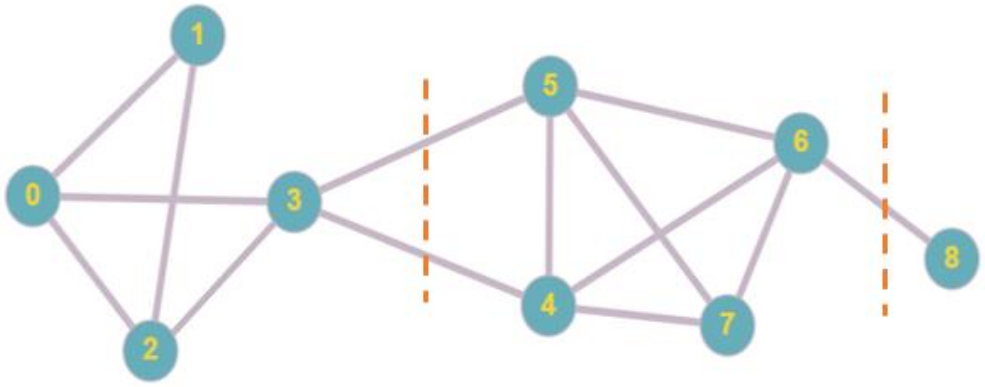


Figure 15. *Partitions with cut method*

One of the oldest algorithms for dividing networks into elements is that the minimal cut approach (and versions like ratio cut and normalized cut). This technique sees use, as an instance, in load balancing for parallel computing on the way to lessen communique among processor nodes. Within the minimum-cut technique, the community is break up right into a predetermined variety of parts, typically of approximately the identical size, chosen targeted the amount of edges among groups is minimized. However, the method regularly returns an imbalanced communities, with one being trivial or singleton (for instance node 8 in Fig.12). Therefore, the goal characteristic is modified so that the group sizes of communities are consider. Two normally variants are ratio cut and normalized cut.

Let $\pi = (C_1, C_2, \ldots, C_k)$ be a graph partition. The ratio cut and normalized cut are defined as follow:

Ratio Cut

$$Ratio\ Cut\ (\pi) = \frac{1}{k}\sum_{i=1}^{k}\frac{cut(C_i, \bar{C_i})}{|C_i|}$$

Normalized Cut

$$Normalized\ Cut\ (\pi) = \frac{1}{k}\sum_{i=1}^{k}\frac{cut(C_i, \bar{C_i})}{vol(C_i)}$$

where:

C_i: community

$\bar{C_i}$: the complement of community C_i

$|C_i|$: the number of node in community C_i

$vol(C_i)$: the total number of nodes degree in the community C_i

We use Figure 12 as an example.

Suppose we divide that network into 2 communities: $C_1 = \{8\}$; $C_2 = \{0,1,2,3,4,5,6,7\}$.

Call that partition π_1. Then, $cut(C_i, \bar{C_i}) = 1,\ k\ =\ 2,\ |C_1| =\ 1, |C_2|\ =\ 8,\ vol(C_1) =$ $1,\ vol(C_2) = 27.$

$$Ratio\ Cut\ (\pi_1) = \frac{1}{2}\left(\frac{1}{1} + \frac{1}{8}\right) = \frac{9}{16} = 0{,}5625$$

$$Normalized\ Cut\ (\pi_1) = \frac{1}{2}\left(\frac{1}{1} + \frac{1}{27}\right) = \frac{14}{27} = 0{,}5185$$

Let's divide that network into another 2 communities: $C_1 = \{0,1,2,3\}$; $C_2 = \{4,5,6,7,8\}$. Call that partition π_2. Then, $cut(C_i, \bar{C_i}) = 2,\ k\ =\ 2,\ |C_1| =\ 4, |C_2|\ =\ 5,\ vol(C_1) =$ $12,\ vol(C_2) = 16.$

$$Ratio\ Cut\ (\pi_2) = \frac{1}{2}\left(\frac{2}{4} + \frac{2}{5}\right) = \frac{9}{20} = 0{,}45$$

$$Normalized\ Cut\ (\pi_2) = \frac{1}{2}\left(\frac{2}{12} + \frac{2}{16}\right) = \frac{7}{48} = 0{,}15$$

Since the measurement of $Normalized\ Cut\ (\pi_2)$ is smaller, we choose the π_2 partition.

3.2 Girvan-Newman's Algorithm

a. Concept

Betweenness centrality is used to detect community structure in networks, especially in the Girvan-Newman's algorithm. The idea of the algorithm is to calculate the "edge betweenness" of each arc and then remove the arcs with the highest "edge betweenness" measures. The measure of arc e is calculated by the ratio of the shortest paths of the pairs of pairs passing through e and the shortest paths between the pairs of nodes in the network. Using the following formula to calculate "edge betweenness":

$$BC(e) = \sum_{\substack{u,w \in V \\ u \neq w}} \frac{\sigma_{uw}(e)}{\sigma_{uw}}$$

The algorithm performs the following:

- Input: a social network G = (V, E) where V is a set of vertices, E is a set of edges.
- Output: Set of communities

Implementation process:

- Step 1: Calculate "edge betweenness" of all sectors in the social network according to the formula presented.
- Step 2: Find the arc that has the highest "edge betweenness" and delete it.
- Step 3: Recalculate the affected arcs's "edge betweenness" .
- Step 4: Repeat step 2 until the bow is no longer present.

b. Example

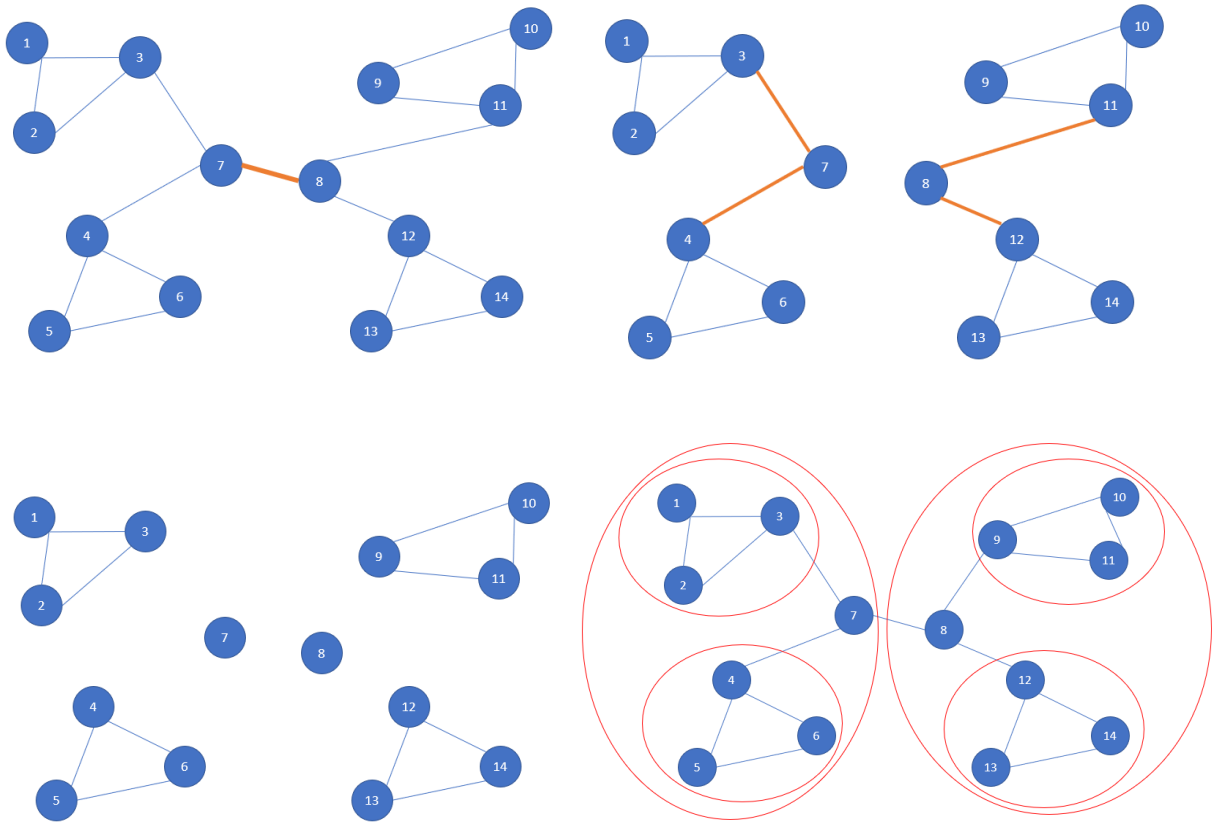


Figure 16. *An general example for Girvan-Newman's Algorithm*

An example write in word

To find "edge betweenness", we need to calculate the shortest path from passing through each edge.

Girvan - The Newman algorithm accesses each X node once and calculates the shortest number of paths from X to each other node passing through each edge.

The algorithm starts by performing the first search according to the width [BFS] of the graph, starting at node X.

The edges going between nodes at the same level can never be part of the shortest path from X.

Consider the graph as follows:

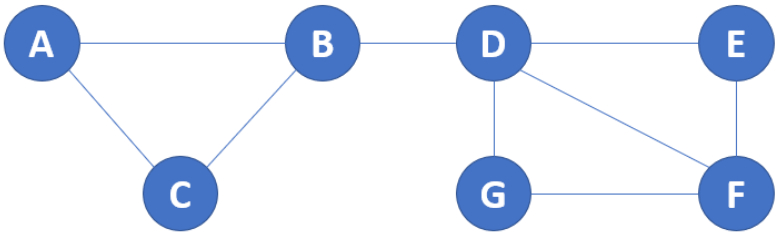
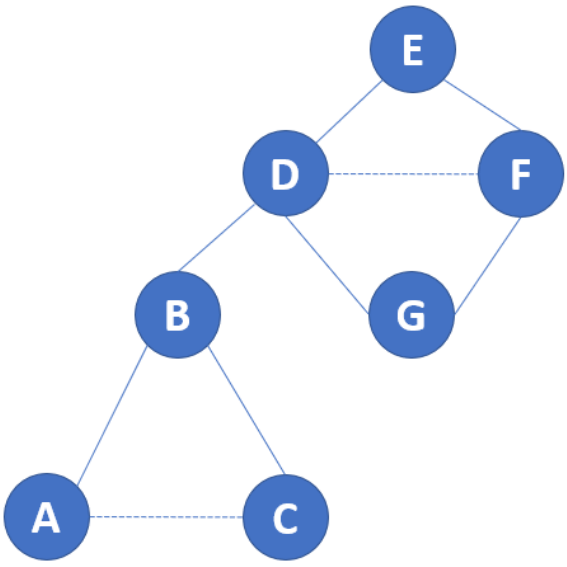


Figure 17. *Sample graph for Girvan-Newman's Algorithm*

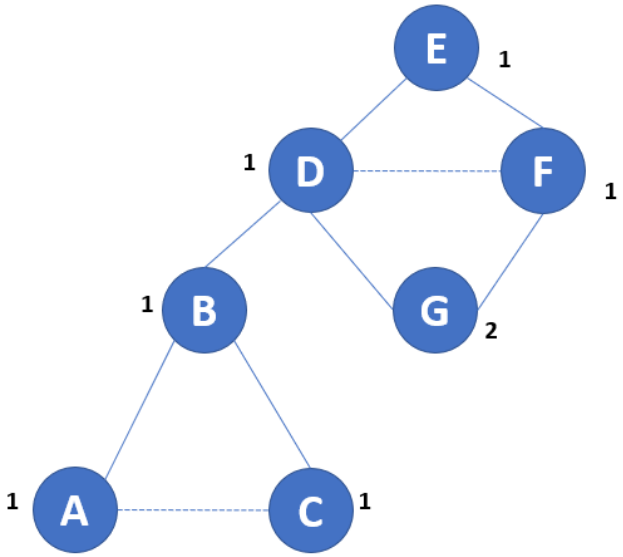
The following figure shows the BFS representation starting at node E.



The second step of the Girvan - Newman algorithm is to label each node according to the shortest number of paths approaching it from the root.

Start by labeling root 1, then, from top to bottom, label each node Y with its parent's total number of labels.

The labeling of the nodes is shown in the image below.



The final step is to calculate for each edge *e* sum on the Y nodes of the portion of the paths displayed from the root X to Y rather than passing through *e*.

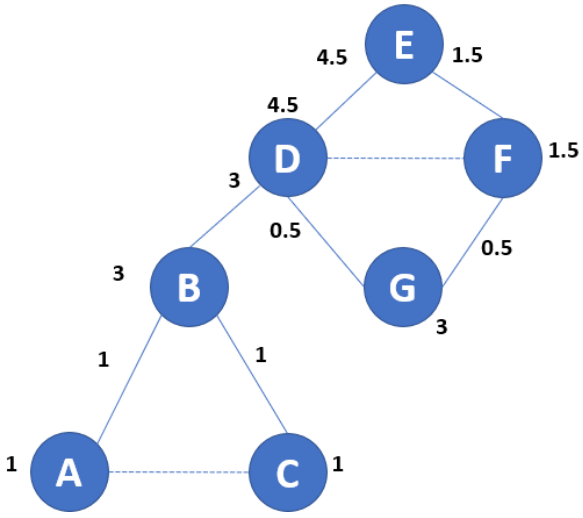
Every node other than root is given a credit of 1.

This credit can be split between nodes and top edge.

The rules for calculation are as follows:

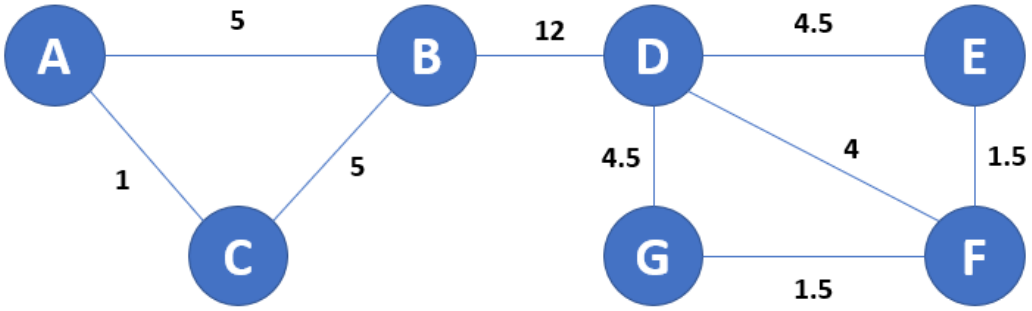
- Since each shortest path will be detected twice, we have to divide the result for each edge by 2.

The following graph shows the calculation starting from node E.



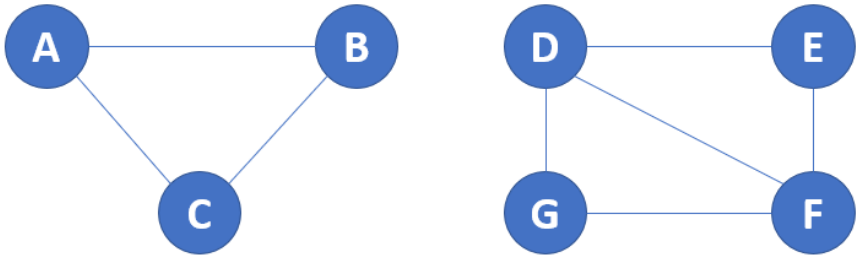
To complete the betweenness calculation, we must repeat this calculation for every node as root and sum the contributions.

After calculation, the following graph shows the values between the final values:



We can cluster by taking the in order to increasing betweenness and add them to the graph at a time. We can remove edges with the highest value to cluster the graph.

In the example graph, we removed the BD edge to have the following two communities:



C1 C2

The worst-case algorithm is $O(t^2n)$, where t is the number of a network and n is the number of nodes. In a complete graph, $t = \frac{1}{2}n(n-1)$, the worst case complexity is $O(n^5)$. After each step, the number of connections and nodes to consider decreases, especially for community-structured networks. This algorithm is still slow processing for large networks with millions of nodes.

3.3 Node Similarity Based Algorithm

a. Concept

The Node Similarity algorithm compares a collection of nodes based on the nodes they're connected to. Two nodes are considered similar if they share many of the identical neighbors. Node Similarity computes pair-wise similarities supported the Jaccard metric, also referred to as the Jaccard Similarity Score.

Jaccard Similarity is computed using the subsequent formula:

$$jaccard(v_i, v_j) = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$$

Similarity measure between nodes reflects on the proximity of nodes. Consider node i and node j. Node i can send resources to node j thanks to neighboring nodes of node i acting as intermediaries to keep moving. Similarity measure of node i and node j is calculated according to the following formula:

$$S_{ij} = \sum_{z \in \Gamma(i) \cap \Gamma(j)} \frac{1}{k(z)}$$

Where $\Gamma\ (i)$ is a set of neighboring nodes of node i and node z is a common node of $\Gamma\ (i) \cap \Gamma\ (j)$. And $k\ (z)$ is the degree measurement of node z. S_{ij} is 0 when node i is not connected directly to node j.

For networks with n nodes, we first calculate the similarity of the pair of nodes (i, j) according to the above formula and store this value in the matrix S level of nxn, where S_{ij} is the similarity of node i and node j.

The idea of the algorithm: The algorithm repeatedly merges a community containing a node with communities containing the largest similar node to that node to create a new community.

Input: a social network $G\ (V, E)$ where V is a node set, E is an edge set, and the S matrix measures the similarity of nodes in the network.

Output: Set of communities.

Implementation process:

Step 1: Initially, each node is a community, the algorithm chooses a random node as the first node.

Step 2: Merge the community containing this node to the community containing the node that has the greatest similarity to this node to create a new community.

Step 3: Identify the next node to handle: select the node that has the greatest similarity to the current node. If this new node is not in the current community, go to step 2. Otherwise, randomly select a new node (not considered) as the first node and perform step 2 to merge the community.

Step 4: Go back to step 2 and step 3 until there are no nodes that haven't gone through yet.

b. Example

Consider a social network following:

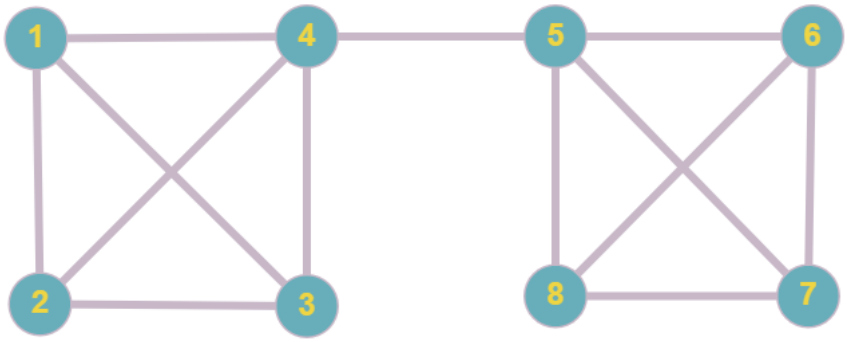


Figure 18. *Sample graph for explain Node Similarity Algorithm*

Initially, calculate the similarity of the nodes.

Considering node 1 and 2, we have $\tau(1) \cap \tau(2) = \{4,3\}$. We have degree (4) = 3 and degree (3) = 3. Therefore, we have $S_{12} = \frac{2}{3}$.

Consider vertices 1 and 6, since there is no connection, so $S_{16} = 0$.

Similarly for other vertices.

Finally, we have the following matrix S:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	2/3	7/12	7/12	0	0	0	0
2	2/3	0	2/3	2/3	0	0	0	0
3	7/12	2/3	0	7/12	0	0	0	0
4	7/12	2/3	7/12	0	0	0	0	0
5	0	0	0	0	0	2/3	2/3	2/3
6	0	0	0	0	2/3	0	7/12	7/12
7	0	0	0	0	2/3	7/12	0	7/12
8	0	0	0	0	2/3	7/12	7/12	0

Figure 19. *Matrix show node similarity measurement result*

Step 1: We have 8 communities {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8} and randomly select node 1.

Step 2: Nodes 1 and 2 have the highest similarity, so they merge into {1,2}.

Step 3: Nodes 1, 3, 4 also have the same similarity compared to node 2. Randomly select node 3 to continue the algorithm. Because node 3 is not in the community {1,2}, go back to step 2. Community merge {1,2} and {3} as {1,2,3}. There is node 2 with the highest similarity to node 3 but node 3 belongs to the community {1,2,3} so we choose a new random node to consider. Select node 4 and we find that node 2 has the highest similarity with it. The community should be merged into {1,2,3,4}.

The algorithm continues to work until there are no more nodes that have not been checked and then stopped.

At the end of the algorithm we find 2 communities: {1,2,3,4} and {5,6,7,8}

The computational costs of this algorithm include: The cost of similarity, finding the next node to handle, and community consolidation. Similarity calculates only neighboring nodes. The complexity for this is $O(k)$. The complexity to calculate the similarity of a network with k nodes is $O(nk)$.

3.4 Label Propagation Community Detection (LPA)

a. Concept

In this section we investigate a simple label propagation algorithm that uses the network structure alone as its guide and requires neither optimization of a predefined objective function nor prior information about the communities. In our algorithm every node is initialized with a unique label and at every step each node adopts the label that most of its neighbors currently have. In this iterative process densely connected groups of nodes form a consensus on a unique label to form communities. We validate the algorithm by applying it to networks whose community structures are known. We also demonstrate that the algorithm takes an almost linear time and hence it is computationally less expensive than what was possible so far.

The LPA algorithm is summarized as follows:

Vertex x has neighbors $x_1,\ x_2,\ x_3,\ldots,x_n$ and each neighbor is labeled to indicate which community the vertex belongs to.

Each vertex in the social network graph selects a community to join and the maximum number of neighbors belongs to a random community.

At first, each vertex is initialized with a unique community label. Labels and community labels are spread through the network. At each label propagation step, each vertex will update its community label based on neighboring label as propagation label. At the end of the label propagation process, most labels disappear and only a few community tags still exist in social networks. The label propagation algorithm converges when each vertex with the label is the label of its neighbor.

At the end of the process, the end vertices with the same label become a community.

According to the LPA algorithm, communities are identical vertices at the end of the convergence process.

Implementation process:

Step 1: Initialize the labels at all vertices in the network. For a given vertex x, $C_x(0)\ =\ x$.

Step 2: Set $t\ =\ 1$.

Step 3: Arrange vertices in the network in random order and give vertices and X.

Step 4: For each x belonging to X selected in a particular order, do:

$$C_x(t) = f(C_{x,1}(t), \ldots . C_{x,m}(t), C_{x,m+1}(t-1), \ldots C_{x,k}(t-1))$$

The function f returns the label that occurs at the highest frequency among randomly divided neighbors and links.

Step 5: If each vertex has a label that is the label that most of its neighbors, then stop the algorithm, otherwise, set $t\ =\ t\ +\ 1$ and return to step 3.

b. Example

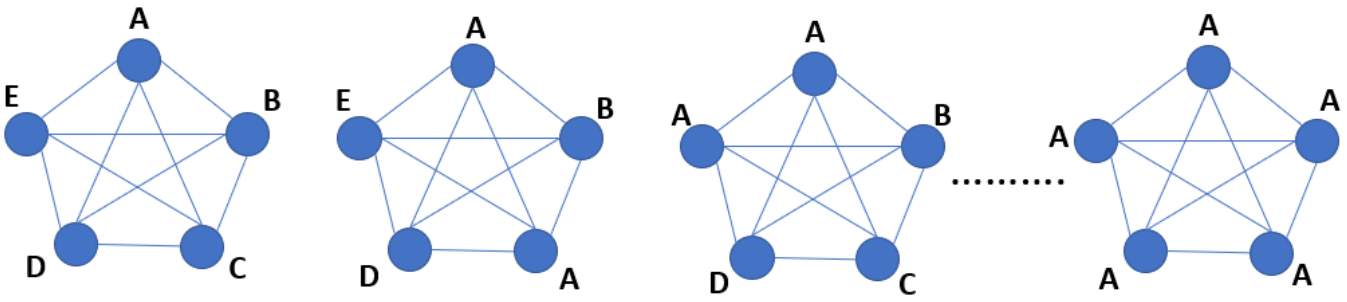


Figure 20. *The vertex labels are updated from left to right*

CHAPTER 4: DEMO

In the report there are many formulas and algorithms are presented. In this chapter, I will perform 5 formulas and algorithms: Degree Centrality, Betweenness Centrality, Closeness Centrality, Clustering Coefficient, Girvan-Newman's Algorithm.

Zachary's karate club is a social community of a college karate club. The network became a popular example of community structure in networks after its use by Michelle Girvan and Mark Newman in 2002.

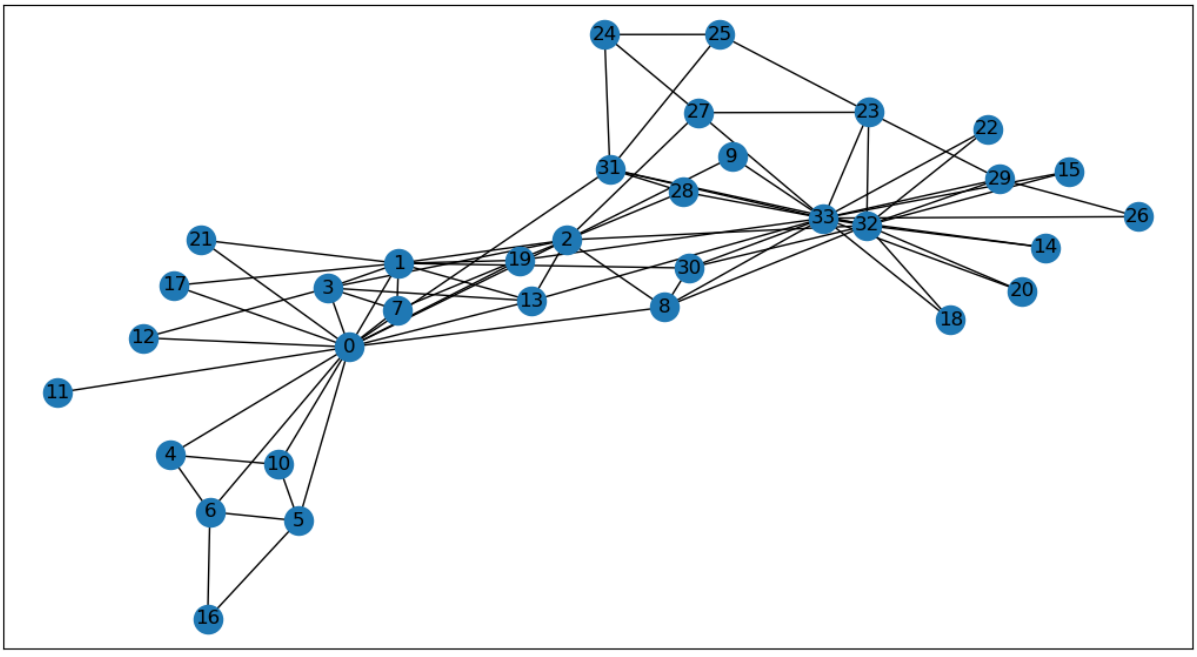
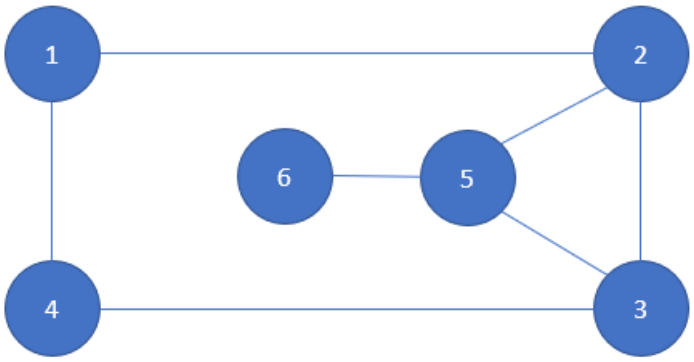


Figure 21. *Zachary's karate club network*

4.1 Degree Centrality

For graph $G = (V, E)$, the degree centrality of a given node v is

$$C_D(v) = degree\ (v)$$



v	$C_D(v)$
1	2
2	3
3	3
4	2
5	3
6	1

Figure 22. *Example of degree centrality*

When the network is a directed graph, we have two variants: in-degree and out-degree. So that, we can normalize C_D to make it comparable between graphs.

$$C_D(v) = \frac{degree\ (v)}{n-1}$$

Where n is the set of all nodes of the graph.

Next, I use the Python language to perform the degree of calculating degree centrality. I use the matplotlib library to show graphs and the networkx library to get an available social network (here I use the Zachary's karate club network) in python to do the calculation of degree centrality.

Calculate centrality using the normalized formula above.

```python
import matplotlib.pyplot as plot
import networkx as net

Graph = net.karate_club_graph()

def degree_centrality(G):
    file = open("degree.txt","w")
    centrality={}
    nodes=1.0/(len(G)-1.0)
    centrality=dict((n,d * nodes) for n,d in dict(G.degree()).items())
    file.write('\n'.join('{} {}'.format(k,v) for k,v in centrality.items()))
    file.close()


# Display network
plot.figure(figsize =(10, 20))
net.draw_networkx(Graph, with_labels = True)
plot.show()

# Graph is the Karate Club Graph
degree_centrality (Graph)
```

Figure 23. *Code for degree centrality*

Parameters in the code:

- Input:
  - G:
    - Data type: graph
    - Explain: It is a bipartite network
  - nodes :
    - Data type: list or container
    - Explain: Container with all nodes in one bipartite node set.
- Output:
  - centrality :
    - Data type: dictionary
    - Explain: Contain an dictionary keyed by node with bipartite degree centrality as the value.

To run this code, I run file degree.py on the command line. Python language needs to be preinstalled on the machine.

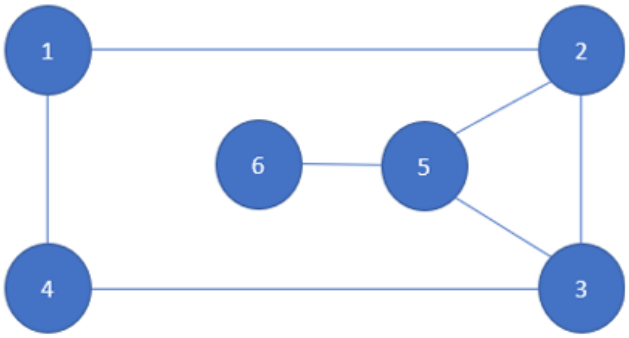
The result of code is shows below and it will be generate to a file name degree.txt when run file degree.py:

```
0 0.48484848484848486
1 0.2727272727272727
2 0.30303030303030304
3 0.18181818181818182
4 0.09090909090909091
5 0.12121212121212122
6 0.12121212121212122
7 0.12121212121212122
8 0.15151515151515152
9 0.06060606060606061
10 0.09090909090909091
11 0.030303030303030304
12 0.06060606060606061
13 0.15151515151515152
14 0.06060606060606061
15 0.06060606060606061
16 0.06060606060606061
17 0.06060606060606061
18 0.06060606060606061
19 0.09090909090909091
20 0.06060606060606061
21 0.06060606060606061
22 0.06060606060606061
23 0.15151515151515152
24 0.09090909090909091
25 0.09090909090909091
26 0.06060606060606061
27 0.12121212121212122
28 0.09090909090909091
29 0.12121212121212122
30 0.12121212121212122
31 0.18181818181818182
32 0.36363636363636365
33 0.5151515151515151
```

Figure 24. *Result for running file degree.py*

4.2 Betweenness Centrality

Define g_{ij} as the number of shortest paths from node $v_i \, to \, v_j$. Let $g_{ij}(v_k)$ be the number of shortest paths from node $v_i \, to \, v_j$ that contain node v_k. Then the betweenness centrality of node v_k is: $C_B(v_k) = \sum_i^n \sum_j^n \frac{g_{ij}(v_k)}{g_{ij}}$



v	$C_B(v)$
1	½
2	½ +1+1 = 5/2
3	½ +1+1 = 5/2
4	½
5	1+1+1 = 3
6	5

Figure 25. *Example of betweenness centrality*

Next, I use the Python language to implement the betweenness centrality. I use the matplotlib library to show graphs and the networkx library to get an available social network (here I use the karate club network) in python to do the calculation of betweenness centrality. The graph is Figure 21 above.

```
import matplotlib.pyplot as plot
import networkx as net
from heapq import heappush, heappop
from itertools import count
import random

Graph = net.karate_club_graph()

def betweenness_centrality (G, n=None, normalized=True, weight=None, endpoints=False, seed=None):
    file = open("betweenness.txt","w")
    betweenness = dict.fromkeys(G, 0.0)  # b[v]=0 for v in G
    if n is None:
        nodes = G
    else:
        random.seed(seed)
        nodes = random.sample(G.nodes(), n)
    for k in nodes:
        # single source shortest paths
        if weight is None:  # use BFS
            S, P, sigma = shortest_path(G, k)
        else:  # use Dijkstra's algorithm
            S, P, sigma = dijkstra_path(G, k, weight)
        # accumulation
        if endpoints:
            betweenness = endpoints(betweenness, S, P, sigma, k)
        else:
            betweenness = basic(betweenness, S, P, sigma, k)
    # rescaling
    betweenness = _rescale(betweenness, len(G), normalized=normalized, directed=G.is_directed(),n=n)
    file.write('\n'.join('{} {}'.format(k,v) for k,v in betweenness.items()))
    file.close()
```

Figure 26. *Main code for betweenness centrality*

These following code are the function helper for the betweenness_centrality function, include:

- Shortest_path: To compute betweenness for a node N, we select a pair of nodes and find all the shortest paths between those nodes.
- Dijkstra_path: To count shortest paths we using traversal algorithms. In this code, Dijkstra's algorithm is used.
- Endpoints: Use to include or exclude a specific node.
- Basic: Combine all the data into betweenness variable.
- Rescale: Use to calculate betweenness centrality with normalized formula.

```
32
33  # helpers for betweenness centrality
34
35  def shortest_path(G, k):
36      S = []
37      P = {}
38      for v in G:
39          P[v] = []
40      sigma = dict.fromkeys(G, 0.0)     # sigma[v]=0 for v in G
41      D = {}
42      sigma[k] = 1.0
43      D[k] = 0
44      Q = [k]
45      while Q:   # use BFS to find shortest paths
46          v = Q.pop(0)
47          S.append(v)
48          Dv = D[v]
49          sigmav = sigma[v]
50          for w in G[v]:
51              if w not in D:
52                  Q.append(w)
53                  D[w] = Dv + 1
54              if D[w] == Dv + 1:   # this is a shortest path, count paths
55                  sigma[w] += sigmav
56                  P[w].append(v)  # predecessors
57      return S, P, sigma
58
59
```

```
59
60  def dijkstra_path(G, k, weight='weight'):
61      # modified from Eppstein
62      S = []
63      P = {}
64      for v in G:
65          P[v] = []
66      sigma = dict.fromkeys(G, 0.0)    # sigma[v]=0 for v in G
67      D = {}
68      sigma[k] = 1.0
69      push = heappush
70      pop = heappop
71      seen = {k: 0}
72      c = count()
73      Q = []   # use Q as heap with (distance,node id) tuples
74      push(Q, (0, next(c), k, k))
75      while Q:
76          (dist, _, pred, v) = pop(Q)
77          if v in D:
78              continue  # already searched this node.
79          sigma[v] += sigma[pred]  # count paths
80          S.append(v)
81          D[v] = dist
82          for w, edgedata in G[v].items():
83              vw_dist = dist + edgedata.get(weight, 1)
84              if w not in D and (w not in seen or vw_dist < seen[w]):
85                  seen[w] = vw_dist
86                  push(Q, (vw_dist, next(c), v, w))
87                  sigma[w] = 0.0
88                  P[w] = [v]
89              elif vw_dist == seen[w]:  # handle equal paths
90                  sigma[w] += sigma[v]
91                  P[w].append(v)
92      return S, P, sigma
93
94
```

```
95  def basic(betweenness, S, P, sigma, k):
96      delta = dict.fromkeys(S, 0)
97      while S:
98          w = S.pop()
99          coeff = (1.0 + delta[w]) / sigma[w]
100         for v in P[w]:
101             delta[v] += sigma[v] * coeff
102         if w != k:
103             betweenness[w] += delta[w]
104     return betweenness
105
106
107 def endpoints(betweenness, S, P, sigma, s):
108     betweenness[s] += len(S) - 1
109     delta = dict.fromkeys(S, 0)
110     while S:
111         w = S.pop()
112         coeff = (1.0 + delta[w]) / sigma[w]
113         for v in P[w]:
114             delta[v] += sigma[v] * coeff
115         if w != k:
116             betweenness[w] += delta[w] + 1
117     return betweenness
```

```
118
119 def _rescale(betweenness, N, normalized, directed=False, n=None)
120     if normalized is True:
121         if N <= 2:
122             scale = None  # no normalization b=0 for all nodes
123         else:
124             scale = 1.0 / ((N - 1) * (N - 2))
125     else:  # rescale by 2 for undirected graphs
126         if not directed:
127             scale = 1.0 / 2.0
128         else:
129             scale = None
130     if scale is not None:
131         if n is not None:
132             scale = scale * N / n
133         for v in betweenness:
134             betweenness[v] *= scale
135     return betweenness
136
137 # Display network
138 plot.figure(figsize =(10, 20))
139 net.draw_networkx(Graph, with_labels = True)
140 plot.show()
141
142 # Graph is the Karate Club Graph
143 betweenness centrality (Graph)
```

Figure 27. *Helper code for betweenness centrality*

Parameters in the code:

- Input:
  - G:
    - Data type: graph
    - Explain: It is a NetworkX graph.
  - n:
    - Data type: int, optional (default=None)
    - Explain: If n is not None use n node samples to estimate betweenness. The value of n <= N where N is the number of nodes in the graph. Higher values give better approximation.
  - normalized :
    - Data type: bool, optional
    - Explain: If True the betweenness values are normalized by $\frac{2}{(N-1)(N-2)}$ for graphs, and $\frac{1}{(N-1)(N-2)}$ for directed graphs where N is the number of nodes in G.

- weight :
  - Data type: None or string, optional (default=None)
  - Explain: If None, consider all edge weights are equal. Otherwise holds the name of the edge attribute used as weight.
- endpoints :
  - Data type: bool, optional
  - Explain: If True, in the shortest path counts, include it.

- Output:
  - betweenness :
    - Data type: dictionary
    - Explain: Contain an dictionary of nodes with betweenness centrality as the value.

To run this code, I run file betweenness.py on the command line. Python language needs to be preinstalled on the machine. The result of code is shows below it will be generate to a file name betweenness.txt when run file betweenness.py:

```
0 0.43763528138528146
1 0.053936688311688304
2 0.14365680615680618
3 0.011909271284271283
4 0.0006313131313131313
5 0.02998737373737374
6 0.029987373737373736
7 0.0
8 0.05592682780182781
9 0.0008477633477633478
10 0.0006313131313131313
11 0.0
12 0.0
13 0.04586339586339586
14 0.0
15 0.0
16 0.0
17 0.0
18 0.0
19 0.03247504810004811
20 0.0
21 0.0
22 0.0
23 0.017613636363636363
24 0.0022095959595959595
25 0.0038404882154882154
26 0.0
27 0.02233345358345358
28 0.0017947330447330447
29 0.0029220779220779218
30 0.014411976911976909
31 0.13827561327561325
32 0.145247113997114
33 0.30407497594997596
```

Figure 28. *Result for running file betweenness.py*

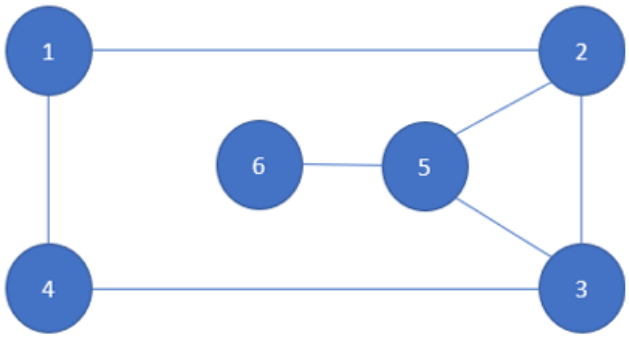
4.3 Closeness Centrality

Let $d(v_i, v_j)$ be the amount of edges in the shortest path between nodes v_i, v_j.

Note that as v_i, v_jget farther apart (“less central”), $d(v_i, v_j)$ is the shortest-path distance between v_i, v_j gets larger.

The closeness centrality is defined as: $C_C(v_k) = \frac{1}{\sum_i^N d(v_i,v_j)}$

C_C can be normalized: $C'_C = \frac{C_C(v_k)}{n-1}$



v	$C_C(v)$
1	1/(1+2+1+2+3) = 1/9
2	1/(1+1+2+1+2) = 1/7
3	1/(2+1+1+1+2) = 1/7
4	1/(1+2+1+2+3) = 1/9
5	1/(2+1+1+1+2) = 1/7
6	1/(3+2+3+2+1) = 1/11

Figure 29. *Example of closeness centrality*

Next, I use the Python language to implement the closeness centrality. I use the matplotlib library to show graphs and the networkx library to get an available social network (here I use the karate club network) in python to do the calculation of closeness centrality. The graph is Figure 21 above.

```python
import functools
import networkx as net
import matplotlib.pyplot as plot

Graph = net.karate_club_graph()

def closeness_centrality(G, u=None, distance=None, normalized=True):
    file = open("closeness.txt","w")
    if distance is not None:
        # use Dijkstra's algorithm with specified attribute as edge weight
        path_length = functools.partial(net.single_source_dijkstra_path_length, weight=distance)
    else:
        path_length = net.single_source_shortest_path_length

    if u is None:
        nodes = G.nodes()
    else:
        nodes = [u]
    closeness = {}
    for n in nodes:
        sp = path_length(G,n)
        totsp = sum(sp.values())
        if totsp > 0.0 and len(G) > 1:
            closeness[n] = (len(sp)-1.0) / totsp
            # normalize to number of nodes-1 in connected part
            if normalized:
                s = (len(sp)-1.0) / ( len(G) - 1 )
                closeness[n] *= s
        else:
            closeness[n] = 0.0
    if u is not None:
        return closeness[u]
    else:
        file.write('\n'.join('{} {}'.format(k,v) for k,v in closeness.items()))
        file.close()

# Display network
plot.figure(figsize =(10, 20))
net.draw_networkx(Graph, with_labels = True)
plot.show()

# Graph is the Karate Club Graph
closeness_centrality (Graph)
```

Figure 30. *Code for closeness centrality*

Parameters in the code:

- Input:
    - G:
        - Data type: graph
        - Explain: It is a NetworkX graph.

- u:
  - Data type: node, optional (default=None)
  - Explain: If n is not None the algorithm return only the value for node u.
- normalized :
  - Data type: bool, optional
  - Explain: If True, normalize by the number of nodes in the connected part of the graph. G.

- Output:
  - closeness :
    - Data type: dictionary
    - Explain: Contain an dictionary of nodes with closeness centrality as the value.

To run this code, I run file closeness.py on the command line. Python language needs to be preinstalled on the machine. The result of code is shows below it will be generate to a file name closeness.txt when run file closeness.py:

```
0 0.5689655172413793
1 0.4852941176470588
2 0.559322033898305
3 0.4647887323943662
4 0.3793103448275862
5 0.38372093023255816
6 0.38372093023255816
7 0.44
8 0.515625
9 0.4342105263157895
10 0.3793103448275862
11 0.36666666666666664
12 0.3707865168539326
13 0.515625
14 0.3707865168539326
15 0.3707865168539326
16 0.28448275862068967
17 0.375
18 0.3707865168539326
19 0.5
20 0.3707865168539326
21 0.375
22 0.3707865168539326
23 0.39285714285714285
24 0.375
25 0.375
26 0.3626373626373626
27 0.4583333333333333
28 0.4520547945205479
29 0.38372093023255816
30 0.4583333333333333
31 0.5409836065573771
32 0.515625
33 0.55
```

Figure 31. *Result for running file closeness.py*

4.4 Clustering Coefficient

Compute the clustering coefficient for nodes.

For unweighted graphs, the clustering of a node u is the fraction of possible triangles through that node that exist $C_v = \frac{2T(v)}{\deg(v)\deg(v-1)}$

where $T(u)$ is the number of triangles through node u and $deg(u)$ is the degree of u.

Next, I use the Python language to implement the clustering coefficient. I use the matplotlib library to show graphs and the networkx library to get an available social network (here I use the karate club network) in python to do the calculation of clustering coefficient. The graph is Figure 21 above. Below code is valid for unweighted graphs.

```
import matplotlib.pyplot as plot
import networkx as net
import random

Graph = net.karate_club_graph()

def clustering_node(G,v):
    neighbors = G[v].keys()
    if len(neighbors) == 1: return -1.0
    links = 0
    for w in neighbors:
        for u in neighbors:
            if u in G[w]: links += 0.5
    clustering_node = 2.0*links/(len(neighbors)*(len(neighbors)-1))
    return clustering_node

def clustering_coefficient(G,v=None):
    file = open("clustering.txt","w")
    clustering ={}
    nodes = G
    for k in nodes:
        if v is None:
            clustering[k]=clustering_node(G,k)
        else:
            return clustering_node(G,v)

    file.write('\n'.join('{} {}'.format(k,v) for k,v in clustering.items()))
    file.close()

# Display network
plot.figure(figsize =(10, 20))
net.draw_networkx(Graph, with_labels = True)
plot.show()

# Graph is the Karate Club Graph
clustering_coefficient(Graph)
```

Figure 32. *Code for clustering coefficient*

Parameters in the code:

- Input:
  - G:
    - Data type: graph
    - Explain: It is a NetworkX graph.
  - v:
    - Data type: node
    - Explain: The algorithm return only the value for node v.
- Output:
  - clustering:
    - Data type: dictionary
    - Explain: Contain an dictionary of nodes with closeness centrality as the value.

To run this code, I run file clustering.py on the command line. Python language needs to be preinstalled on the machine. The result of code is shows below it will be generate to a file name clustering.txt when run file clustering.py:

```
0 0.15
1 0.3333333333333333
2 0.24444444444444444
3 0.6666666666666666
4 0.6666666666666666
5 0.5
6 0.5
7 1.0
8 0.5
9 0.0
10 0.6666666666666666
11 -1.0
12 1.0
13 0.6
14 1.0
15 1.0
16 1.0
17 1.0
18 1.0
19 0.3333333333333333
20 1.0
21 1.0
22 1.0
23 0.4
24 0.3333333333333333
25 0.3333333333333333
26 1.0
27 0.16666666666666666
28 0.3333333333333333
29 0.6666666666666666
30 0.5
31 0.2
32 0.19696969696969696
33 0.11029411764705882
```

Figure 33. *Result for clustering coefficient*

4.5 Girvan-Newman's Algorithm

I will demonstrate community detection using the Girvan-Newman Algorithm use Zachary's karate club Graph (Fig.21). The code in file name GNA.py.

We understand that within the Girvan-Newman approach, the edges of the graph are eliminated one-by-one based totally at the Edge Betweenness Centrality(EBC) score.

So, the first task is to find the EBC values for all the edges and then take off the edge with the most important cost. The below function would do that:

```
def edge_to_remove(G):
    edge_dict = net.edge_betweenness_centrality(G)
    edge = ()

    # extract the edge with highest edge betweenness centrality score
    for key, value in sorted(edge_dict.items(), key=lambda item: item[1], reverse = True):
        edge = key
        break

    return edge
```

Figure 34. *Code to find and remove highest edge in Girvan-Newman's Algorithm*

Now the following function will use the above function to partition the graph into communities:

```
def girvan_newman(G):
    # find number of connected components
    C = net.connected_components(G)
    L = net.number_connected_components(G)

    while(L == 1):
        G.remove_edge(edge_to_remove(G)[0], edge_to_remove(G)[1])
        C = net.connected_components(G)
        L = net.number_connected_components(G)

    return C
```

Figure 35. *Function to partition the graph into communities of Girvan-Newman's Algorithm*

Let's pass the Karate club graph to the above function:

```
def GNA(G):
    file = open("communities.txt","w")

    # find communities in the graph
    C = girvan_newman(G.copy())

    # find the nodes forming the communities
    communities = []

    for i in C:
        communities.append(list(i))

    file.write("The number of communities are " + str(len(communities)) + "\n")
    file.write('\n'.join([''.join(str(i)) for i in communities]))
    file.close()
```

The result is show below, and in file name communities.txt:

```
The number of communities are 2
[0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 19, 21]
[2, 8, 9, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
```

Figure 36. *Result for the Girvan-Newman's Algorithm*

There are 2 communities in the Karate club graph according to the Girvan-Newman set of rules. Let's visualize the communities discovered within the graph:

```
51      # Draw the communities
52      color_map = []
53      for node in G:
54          if node in communities[0]:
55              color_map.append('yellow')
56          else:
57              color_map.append('red')
58
59      net.draw(G, node_color=color_map, with_labels=True)
60      plot.show()
61
62  GNA(Graph)
```

Figure 37. *Code for visualizing the graph into communities in GNA*

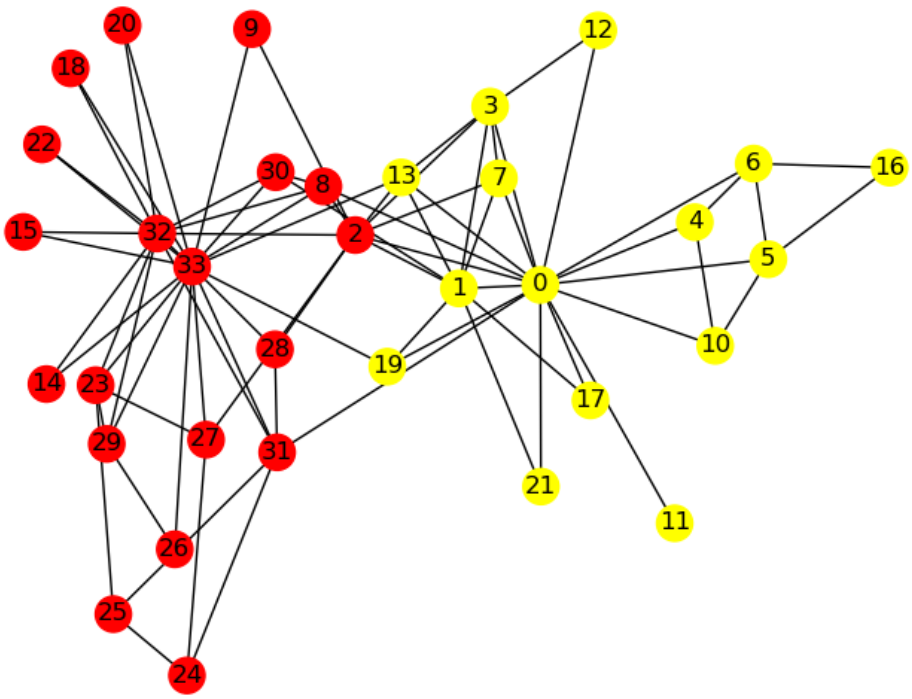


Figure 38. *Graph divided into 2 communities*

```python
import networkx as net
import matplotlib.pyplot as plot

# load the graph
Graph = net.karate_club_graph()
plot.figure(figsize =(10, 20))
net.draw_networkx(Graph, with_labels = True)
plot.show()


def edge_to_remove(G):
    edge_dict = net.edge_betweenness_centrality(G)
    edge = ()

    # extract the edge with highest edge betweenness centrality score
    for key, value in sorted(edge_dict.items(), key=lambda item: item[1], reverse = True):
        edge = key
        break

    return edge


def girvan_newman(G):
    # find number of connected components
    C = net.connected_components(G)
    L = net.number_connected_components(G)

    while(L == 1):
        G.remove_edge(edge_to_remove(G)[0], edge_to_remove(G)[1])
        C = net.connected_components(G)
        L = net.number_connected_components(G)

    return C

def GNA(G):
    file = open("communities.txt","w")

    # find communities in the graph
    C = girvan_newman(G.copy())

    # find the nodes forming the communities
    communities = []

    for i in C:
        communities.append(list(i))

    file.write("The number of communities are " + str(len(communities)) + "\n")
    file.write('\n'.join(['.join(str(i)) for i in communities]))
    file.close()

    # Draw the communities
    color_map = []
    for node in G:
        if node in communities[0]:
            color_map.append('yellow')
        else:
            color_map.append('red')

    net.draw(G, node_color=color_map, with_labels=True)
    plot.show()

GNA(Graph)
```

Figure 39. *Full code for Girvan-Newman's Algorithm*

CHAPTER 5: CONCLUSION AND DISCUSSIONS

In this report, I have accomplished the required goals assigned.

In chapter 1, I introduced the definition of social networking, social network analysis. In addition, I also discuss the importance of social networking as well as its analysis. At the same time, I also showed how to use graphs to model social networks.

In chapter 2, I presented more clearly how to use graphs on social networks. And as required, I also presented the concept as well as giving examples for density, centrality, clustering coefficient. I also defined key players and how to detect it, in a simple way. The concept of signed graph, the problem and its applications are covered in this chapter.

In chapter 3, I described the problem of community detection along with the following concepts and algorithms: modularity and cut, Girvan-Newman's algorithm, node similarity based algorithm, label propagation algorithm along with examples for them.

In chapter 4, I demo 4 formulas and 1 algorithm mentioned in chapters 2 and 3, including: degree centrality, betweenness centrality, closeness centrality, clustering coefficient, Girvan-Newman's algorithm. For demo, I use Zachary’s karate club network to create a computing environment for the above formulas and algorithms.

After completing this report, I also learned a lot more about social networking and how to analyze it. At the same time, I am also aware of the importance of social network analysis in life. Thanks to that, I learned the need to use computers and algorithms to support social network analytics.

REFERENCES

[1] Ehu.eus. http://www.ehu.eus/cuadernosdegestion/documentos/140512ir.pdf. Published 2020. Accessed May 22, 2020.

[2] Vbn.aau.dk. https://vbn.aau.dk/ws/portalfiles/portal/77727140/SNA_Springer_final.pdf. Published 2020. Accessed May 22, 2020.

[3] Clustering Coefficient - an overview | ScienceDirect Topics. Sciencedirect.com. https://www.sciencedirect.com/topics/computer-science/clustering-coefficient. Published 2020. Accessed May 22, 2020.

[4] Clustering Coefficient in Graph Theory - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/clustering-coefficient-graph-theory/. Published 2020. Accessed May 22, 2020.'

[5] Jilbert O. 4.2 Degree Centrality | Social Networks: An Introduction. Bookdown.org. https://bookdown.org/omarlizardo/_main/4-2-degree-centrality.html?fbclid=IwAR021AB7-eztGAsFAbiJazPyIfxavzKD2hrpMvzgqx0llGd783otwuX82C4. Published 2020. Accessed May 22, 2020.

[6] Disney A. Social network analysis: Centrality measures. Cambridge Intelligence. https://cambridge-intelligence.com/keylines-faqs-social-network-analysis/?fbclid=IwAR2UioTaT6G_VDrfUOqJIXLAWyIxEoqCfiAN4tNs7TBXnlFEJwa2BQ7em5E. Published 2020. Accessed May 22, 2020.

[7] Introduction to Social Network Methods: Chapter 3: Using Graphs to Represent Social Relations. Faculty.ucr.edu. https://faculty.ucr.edu/~hanneman/nettext/C3_Graphs.html?fbclid=IwAR3pdjaPNUfa40BTexBJZPElax-saFZ1F2X_LCyRn0_eFvLrQuZxopw9EwI. Published 2020. Accessed May 22, 2020.

[8] Social network analysis. En.wikipedia.org. https://en.wikipedia.org/wiki/Social_network_analysis. Published 2020. Accessed May 22, 2020.

[9] Social Network Analysis: An Introduction by Orgnet,LLC. Orgnet.com. http://www.orgnet.com/sna.html?fbclid=IwAR0NwXmHwMB9O3ZwvjV6IKmBxLlOqgcmDkkaWGlG8K34WJhvDNmKKwXZ5VY. Published 2020. Accessed May 22, 2020.

[10] Yang S, Keller F, Zheng L. Social Network Analysis.

[11] Tang L, Liu H. Community Detection And Mining In Social Media. [San Rafael, Calif.]: Morgan & Claypool Publishers; 2010.

[12] Networks G. Getting Started with Community Detection in Graphs and Networks. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/04/community-detection-graphs-networks/. Published 2020. Accessed May 22, 2020.

[13] Degree Centrality (Centrality Measure) - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/degree-centrality-centrality-measure/?ref=rp. Published 2020. Accessed May 22, 2020.

[14] Closeness Centrality (Centrality Measure) - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/closeness-centrality-centrality-measure/. Published 2020. Accessed May 22, 2020.

[15] Betweenness Centrality (Centrality Measure) - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/betweenness-centrality-centrality-measure/. Published 2020. Accessed May 22, 2020.

[16] Centrality — NetworkX 1.10 documentation. Networkx.github.io. https://networkx.github.io/documentation/networkx-1.10/reference/algorithms.centrality.html?highlight=networkx.algorithms.centrality#module-networkx.algorithms.centrality. Published 2020. Accessed May 22, 2020.

[17] Web.eecs.utk.edu. http://web.eecs.utk.edu/~cphill25/cs594_spring2015_projects/CentralityProject.pdf. Published 2020. Accessed May 22, 2020.

[18] Conferences.matheo.si. https://conferences.matheo.si/event/19/contribution/5/material/slides/0.pdf. Published 2020. Accessed May 22, 2020.

[19] 7.1 The Node Similarity algorithm - Chapter 7. Graph similarity algorithms. Neo4j.com. https://neo4j.com/docs/graph-algorithms/current/algorithms/node-similarity/. Published 2020. Accessed May 22, 2020.

[20] 6.2. The Label Propagation algorithm - Chapter 6. Community detection algorithms. Neo4j.com. https://neo4j.com/docs/graph-algorithms/current/algorithms/label-propagation/. Published 2020. Accessed May 22, 2020.

[21] Wasserman S, Faust K. Social Network Analysis. New York [etc.]: Cambridge University Press; 2019.

[22] Vinh P, Barolli L. Nature Of Computation And Communication. Springer; :Chapter: Graph Methods for Social Network Analysis.

[23] Raghavan U, Albert R, Kumara S. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E. 2007;76(3). doi:10.1103/physreve.76.036106

[24] Girvan M, Newman M. Community structure in social and biological networks. Proceedings of the National Academy of Sciences. 2002;99(12):7821-7826. doi:10.1073/pnas.122653799

[25] Blondel V, Guillaume J, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment. 2008;2008(10):P10008. doi:10.1088/1742-5468/2008/10/p10008

[26] Citeseerx.ist.psu.edu. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.735.9223&rep=rep1&type=pdf. Published 2020. Accessed May 22, 2020.

[27] Abraham A. Computational Social Networks. London: Springer; 2012.

SELF-EVALUATION

Requirements	Score /10	Level 1	Level 2	Level 3	Self-evaluation	Reason(s)
		0 score	1/2 score	Full score		
1/ Report	**8.0**					
In right format	1.0	Wrong format and outlines	Some errors	In right format and outlines, no error	1.0	
Chapter 1	1.0	Not enough content, bad written, no example	Full contents, not very well written, not enough examples	Full contents, well written, with examples	0.5	Not enough examples
Chapter 2	2.0	Not enough content, bad written, no example	Full contents, not very well written, not enough examples	Full contents, well written, with examples	1.0	Not enough examples
Chapter 3	2.0	Not enough content, bad written, no example	Full contents, not very well written, not enough examples	Full contents, well written, with examples	1.0	Not enough examples
Chapter 4	1.0	Not enough content, bad	Full contents, not very well	Full contents, well written,	0.5	Not enough examples

Requirements	**Score /10**	**Level 1**	**Level 2**	**Level 3**	**Self-evaluation**	**Reason(s)**
		0 score	**1/2 score**	**Full score**		
		written, no example	written, not enough examples	with examples		
Chapter 5	0.5	Not enough content, bad written	Full contents, not very well written	Full contents, well written	0.25	Not very well written
References	0.5	No reference	Wrong format, < 3 references	Right format, ≥ 3 references	0.5	
2/ Demo	**2.0**					
Contents	1.5	Implement less than half of the operations	Implement half of the operations	Implement all of the operations	1.5	
Program	0.5	Cannot be compiled	Runtime error for 1 formula or algorithm	Can be run correctly with no error	0.5	
Total	10.0	Result:			6.75	