

UNIVERSITY OF THE PHILIPPINES MANILA
COLLEGE OF ARTS AND SCIENCES
DEPARTMENT OF PHYSICAL SCIENCES AND MATHEMATICS

COMMUNITY DETECTION IN PHILIPPINE CONGRESS
LEGISLATORS

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Kharl Gaebriel A. Agir

May 2018

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled “Community Detection in Philippine Congress Legislators” prepared and submitted by Kharl Gaebriel A. Agir in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D. (*cand.*)
Adviser

EXAMINERS:

	Approved	Disapproved
1. Gregorio B. Baes, Ph.D. (<i>cand.</i>)	_____	_____
2. Avegail D. Carpio, M.Sc.	_____	_____
3. Richard Bryann L. Chua, Ph.D. (<i>cand.</i>)	_____	_____
4. Perlita E. Gasmen, M.Sc. (<i>cand.</i>)	_____	_____
5. Marvin John C. Ignacio, M.Sc. (<i>cand.</i>)	_____	_____
6. Ma. Sheila A. Magboo, M.Sc.	_____	_____
7. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

A program that would detect communities of specific sizes in a network of legislators, as well as compute different centralities of each network nodes, have always been a sought for tool in the field of political science for the entities in that network play vital roles in our government. The Philippine congress is consists of over 300 house members and 58 standing committees, and it is crucial to look on how these entities work together, with the consideration of their political involvements and relationship. The data used contains political involvements of the 17th Congress of The Philippines. Analyzing these data will result to an output that is not commonly obvious to people. For the analysis, community detection by means of clique-finding algorithms is used. Cluster or communities are extracted, as well as the centrality measures of each node, to be able to see whose members have stronger connection towards others. The software also produces a graph visualization of the network and export the results as a PDF file. The software is using Java as its interface and R for some of its functionalities.

Keywords: Community Detection, Weighted Cliques, Centrality, Congress, Political Involvements

Contents

Acceptance Sheet	i
Abstract	ii
List of Figures	v
I. Introduction	1
A. Background of the Study	1
B. Statement of the Problem	3
C. Objectives of the Study	3
D. Significance of the Project	4
E. Scope and Limitations	5
F. Assumptions	5
II. Review of Related Literature	6
III. Theoretical Framework	11
A. Community Detection	11
B. Philippine Congress Structure	12
C. Graph Construction	12
D. Centrality	13
D.1 Degree Centrality	13
D.2 Betweenness Centrality	13
D.3 Closeness Centrality	14
E. Cliques	14
F. Weighted Clique Problem	15
G. 2-Approximation Algorithm for Finding a Clique with Minimum Weight of Vertices and Edges	16

IV.	Design and Implementation	19
A.	Data Specifications	19
B.	System Design	19
B..1	Context Diagram	19
B..2	Use Case Diagram	19
B..3	Flowchart Diagram	21
C.	System Architecture	23
D.	Technical Architecture	23
V.	Results	24
VI.	Discussions	34
VII.	Conclusions	36
VIII.	Recommendations	37
IX.	Bibliography	38
X.	Appendix	40
A.	Source Code	40
XI.	Acknowledgement	62

List of Figures

1	Graphical Representation of a Clique	15
2	Weighted Clique	15
3	An example of a symmetric adjacency matrix	18
4	Context Diagram	19
5	Use Case Diagram	20
6	Flowchart Diagram	21
7	Algorithm Flowchart	22
8	Main Menu	24
9	File Choosing	25
10	Features	26
11	Feature Selection	26
12	Specify Community Parameters	27
13	Sample Results	28
14	Node Centrality	29
15	PDF Input File Name	30
16	PDF Generated	31
17	PDF Contents	32
18	Graph Generated	33

I. Introduction

A. Background of the Study

Representatives or Congressmen/Congress-women are the legislators in the Philippine Congress. They were elected by voters and can serve for a three-year term in their respective legislative districts. In total, there are 238 representatives elected by district and another 20% of the total number of the representatives elected through the party-list system. On the other hand, twenty-four (24) Senators, the legislators in the Senate, are elected at a nationwide election to a six-year term and cannot serve for not more than two consecutive terms [1].

The Philippine Congress is the one responsible for formulating and enabling laws to uphold the constitution in the country and amend or change the constitution itself. This legislative body is opting to produce two main documents: bills and resolutions.

With over 290 house members, 58 standing committees, and 7,778 house bills and resolutions, it is crucial to look on how these entities work together and their relationship background. But with these numbers, it'll be exhausting and will took some time to dig in especially during urgent times. In this kind of situations, a tool that will easily generate a graph that uses legislators as its nodes, different political involvement as their edges would be a great aid. A graph representation of this data will easily provide an instant visualization of how these entities are plotted in a network. Also, graph properties and characteristics can provide significant findings in these network entities. In this study, community detection and node centrality measures were used to analyze this data.

Centrality, in a technical sense, is the measure of how a certain node is connected to other nodes in the graph and how it is positioned in the network. This type of connection can be only readily seen by the use of graphs. Centrality concepts in a network do not differentiate sending from receiveing relations but simply treat all

connections as symmetric. The most central positions in a network are those involving many reciprocated ties to other networks. Network stars acquire power because they are close to many system actors, in effect, by lying between positions that must use them to transmit messages and goods. Thus, centrality prominence is useful for analyzing positional power in symmetric exchange networks such as political network structure. Centrality measure may prove useful for analyzing power and influence in diverse situations.

In a network, not only the nodes and their immediate relationship with each other is important but also much bigger perspective in viewing these entities, such as the formation of groups and clusters. This is where community detection comes in. Community detection, in graph theory, revolves around communities or clusters. They are the group of vertices having higher probability of being connected to each other than to members of other groups. [2]. Political researchers can use these and related measures of network to test theories about how networks affect politics, whether through coercion, agenda setting and interest, or identity changes inside the network. Political analysts define social power and significance primarily on relational terms. Significance and importance of a node in political network is an aspect of the actual or potential interaction between two or more social actors. [3]

Community detection helps researchers understand the entirety of the network by using the nodes individually, or by the presence of clusters and communities. Not only that, network analysis, along with community detection, offers practical tools to measure these information about the politicians and their distributions in any system of politicians. It concerns relationships defined by ties among nodes. This analysis examines the association among nodes in addition to the attributes of particular node because relationships are not properties of nodes but a property of system of nodes.

Community detection can be done in a lot of ways. The traditional methods focused on clustering and divisive algorithms. They use classification and partitioning

in order to bound nodes under certain characteristics. Since the network in this study is already bounded by an underlying feature, it is appropriate to use clique-finding algorithms, for the network under consideration have identical node characteristics.

Cliques, in simple terms, are subgraphs whose vertices are all connected with each other. With this definition, we can infer that clique-finding is exhaustive. As matter of fact, weighted clique problems are considered NP-hard, which means that it can be polynomially reduced [4]and can be approximated under certain circumstances.

There is a lot of approximation algorithm that have immersed from this problem but this discourse tackles specifically a 2-approximation algorithm for finding cliques with minimum weights [4]. This algorithm uses row's subset of symmetric matrix problem where it accounts the sum of the values in the subset of a matrix row given a specific number of elements.

B. Statement of the Problem

As of now, a tool that will aid community detection in Philippine Congress Legislators is still absent. Tools that would show the various political involvements of the legislators and their relationships would be useful to the people who works in the field in order to understand how these involvements are related and also to see how legislators that have worked together in the past or currently working together push through their goals in relation to each other.

C. Objectives of the Study

This study opts to produce a tool that would take in data of Philippine legislators and political involvement variables (Political party, authored/coauthored bills, etc.) from all over the Philippines. The data is taken through a spreadsheet file that contains the name of the legislator, along with the dataset related to these legislators that the political researcher may wish to input. With these data, the tool would generate a

graph based on it with the legislators as nodes and their political relationship, based on the dataset, as the edges and edge weights. The tool is able to:

1. Accept a desired data set of the political involvement of the legislators from the user
2. Display all the communities with the size specified by the user
3. Let the user choose either exact algorithm or approximate algorithm
4. Compute the centrality of each nodes (Degree, Weighted Degree, Betweenness, and Closeness)
5. Display a graph visualization of the political network formed, as well as the communities detected by it.
6. Produce a PDF document containing the list of communities present and the centrality table of legislators.

D. Significance of the Project

The tool can be used to visualize the political involvement of the Philippine legislators with each other. It would help in studying the correlation of different political variables by producing an undirected graph that shows the relationship and connection of each legislator with each other.

The output of this tools may then be useful to the public during election period. They would have an easier way to observe how these legislators worked in the past and consider the political background of each candidate by using the output results as basis, thereby contributing to a better judgment in choosing a much suitable candidate.

E. Scope and Limitations

The tool developed in this study is limited to the following constraints:

1. It can easily process data coming from the same congress only depending on the data set used.
2. The additional data set that will be entered by the user must be from the same set of subset of legislators as the current data set
3. For the approximation algorithm, the optimal communities are only guaranteed if the data in the adjacency matrix follow the metric properties.
4. The data set must follow the data specifications of the tool, saved in .xls, .xlsx, or .csv file extensions.
5. There is no error checking on the data set that will be used as an input

F. Assumptions

The tool produced in this study works under the following assumptions:

1. The political researcher is knowledgeable in interpreting basic graph properties.
2. The machine to be used has a graphics-processing unit.

II. Review of Related Literature

Recent researches over the past years exposed the booming development of graph data modelling and its application [5] thus analyzing and mining knowledge from graphs can be considered significantly meaningful. One of its various applications can be seen in studying and analyzing networks.

A network is a collection of entities that are interconnected with links. Networks are often represented as graphs where entities take on vertices that are connected with edges. They frequently associate with communities, defined as groups of nodes that are densely connected as compared to nodes outside the community [6]. These nodes are somewhat bounded by some common property. Communities in network are represented as subgraphs in graphs. To some extent, they can be considered as separate entities with their own autonomy. It makes sense to evaluate them independently of the graph as a whole.

Social communities can be defined in a very strict sense as a subgroup whose members are all related to each other which can be considered as complete mutuality [7]. In graph terms, this corresponds to a clique, a subset whose vertices are all adjacent to each other. Moreover, a community where all of the entities are connected to each other provides valued information and knowledge, and can now be represented as cliques.

Although community structure may be applied to numerous real-world problems, detecting them, especially a complete one, is one of the most challenging tasks. Through time, various community detection algorithms are present in the field. This applied to both disjoint and overlapping communities. One of it was about clique finding method [7].

According to Fei Hao et al. [5], maximal clique enumeration problem is extensively investigated in many fields. The problem to figure out the subgraph with maximum cardinality is considered NP-hard, where there is no polynomial time solution present.

Their study that involved networks of collaborations between scientist used a formal concept approach that uses equivalence relation between vertices. Their proposed approach is to find bases that are common to maximal cliques. Subgraphs build upon these based to form cliques. The result of their study suggested that this phenomenon is quite important and promising for complex topological structures and other network mining and analysis.

A paper written by Priyanka Saxena et al [8]. studied some of the proposed algorithms that tried to solve the maximum clique problem. Their paper aimed to review and compare famous clique finding algorithm defined to this day. According to them, the clique problem resides with two well-studied problems: The maximum clique problem where it is aimed to find a maximal clique with maximum cardinality or weight and the maximal clique problem that is aimed to find a clique which is not a subset of any other clique. Their study made use of backtracking search algorithm which is the commonly used approach in optimizing the clique problem. Their study showed that almost all of the proposed algorithm present to this day have exponential time complexity, some of them even have no concrete time analysis. In other words, the results of this study showed that even though many algorithms and various approaches have been proposed, still the problem lie the same i.e. to find a cliques in the polynomial time.

In 2015, Rossi et. al [9] presented a parallel maximum clique algorithm for large sparse graphs that is designed to exploit characteristics of social and information networks. The method exhibited a runtime that scales over real-world networks ranging from thousand to million nodes. The algorithm they have used employed a branch-and-bound strategy with novel and aggressive pruning techniques. The pruning techniques included the combined use of core number of vertices along with an initial heuristic solution to remove the vast majority of the search space. They demonstrated the impact of the algorithm on applications using network analysis problems.

The same branch-and-bound approach was used also by Wood [10]. Instead of using pruning techniques, Wood applied a vertex coloring heuristics to determine lower and upper bounds for the size of a maximum clique.

Another branch-and-bound approach was presented by Palsetia et. al. [11], where they presented a clique guided community detection algorithm. Their algorithm was divided into two phases. A heuristic branch-and-bound approach was used on the first phase where they find disjoint cliques that will guide the community detections that is the second phase.

Their papers showed that in order to apply an efficient algorithm to solve the maximum clique problem, different vertex properties must be considered as an algorithm progresses.

Now, imagine the maximum clique problem reformed in a different way where instead of the cardinality, edge weights are the ones being considered. There exists a well-known problem called the Minimum/Maximum Weighted Clique Problem(MWCP) that take this into considerations. The problem inputs a complete weighted undirected graph where it is required to output a complete subgraph(clique) of the graph of order m with the smallest/largest weight. The computational complexity of WCP is determined by the reducibility of the aforementioned clique problem, which is NP-complete in the strong sense [4].

The work of Chang and Wang [12] proposed an efficient algorithm, with runtime $O(n \log(\log n))$, for finding a maximum weight clique and independent set problems on permutation graphs. By taking advantage of the graph property, they were able to solve a weighted clique problem using an efficient algorithm. This study implied that for certain properties being exhibited by a graph, an efficient algorithm may be used and derived.

According to Vassilevska, the k -clique problem is a cornerstone of NP-completeness and parametrized complexity. When k is a fixed constant it is asymptotically bounded

by a factor of k , but this algorithm relies heavily on a fast matrix multiplication [13]. However, Eremin et al. [4] suggested that if the vertex weights are nonnegative and edge weights either satisfy the triangle inequality or are squared pairwise distances for some point configuration of Euclidean space, this natural cases admits a 2-approximation polynomial time algorithm. The proposed algorithm used the Rows subset of symmetric matrix problem for the polynomial time reduction of the MWCP. It was shown through the form of property verification.

With this wide variety of works that approximate and optimize the known clique problem, different efforts in analyzing community structure and the study of community detection will benefit greatly. The seek for link prediction, information diffusion, or even detection of suspicious events in such communities within networks [6] will be much easier.

A traditional view of significance in politics is that it comes from the possession of important resources. The relative possession of resources is thought to provide actors, such as people or organizations with means of coercion or influence over others. This traditional view is highly limiting, since it tackles the ties that link the actors together. These ties, whether material or social, determine an actor's ability to make connections between, or quickly spread resources to, other actors. An actor's relative position in a network formed by these ties thus provides another important source of influence over other. In the work of Burton and Montgomery [14], they explained the advantages of actor's centrality positions along with their advantages and demonstrated that network notions of power and significance that derive from centrality can significantly inform the study of politics.

For example, in 2013, the Philippine government was swindled with almost P10 billion by the Priority Development Assessment Fund (PDAF) scam. The scandal involved ghost projects and dummy non-government organizations being funded by some politicians for their own gains. Multiple kickbacks and diverted portions in their

government-allocated funds are then misused by these recipients. The alleged culprits in multiple instances of these scam are almost the same per ghost projects. With this, we may infer that there must have been an underlying bounding connections between these suspects [15].

On the other hand, linkages between government officials also serves as factors when it comes to drafting house bills for example, under the Philippine Congress. By analyzing these links and connections, we can see how legislators that have political relationship with each other work alongside with each other and how they achieve a common goal.

In instances like this, a tool designed for detecting communities in broad networks would help in analyzing how legislators work with each other and toward their goal. It would ease the public from observing the lawmakers work along with their various political involvements. Lastly, this tool would be beneficial to researchers by providing a more informed visualization of political relationship among legislators.

III. Theoretical Framework

A. Community Detection

Community is a collection of nodes that are more densely connected as compared to the nodes outside the community. The nodes inside the community have some common properties. These properties are shown by the edges between the nodes. A community can be classified into two categories, disjoint and overlapping community. The disjoint community refers to the arrangement where the nodes belong to only one community while the overlapping community is known for its fuzzy assignments of nodes that may overlap between two or more communities.

Detection of such community structure in a complex network is quite hard. Community detection algorithms are available for both disjoint and overlapping communities. Partitioning algorithms for disjoint communities, and clique finding and clustering algorithms for overlapping communities.

The motivation for studying communities in networks is to somehow visualize the relationship of the entities that compose that community given a certain property or condition. These visualizations may help understand in these node relationships and see how these nodes work together along their characteristic similarities.

This motivation can be applied in the field of politics. Networks help to visualize the entirety of a certain political society. Enabling the researchers witness the branching connections of each node that represents a politician will immediately give them a hunch on how this politician interacts with other politicians under different bounding characteristics. With community detection in present, immediate conclusions can now be drawn not just by individual nodes but also as a cluster and a more connected group.

B. Philippine Congress Structure

The Philippine Legislature, or the Congress of the Philippines, is composed of legislators that provides backbone in imposing different laws governed by the current government administration. Congress is divided into two chambers or houses the House of Representatives and the Senate.

Lawmakers in the House of Representatives are called Representatives or Congressman/Congresswomen. They are given a three-year term if theyre elected by voters in their respective legislative districts. There are a total of 212 legislative districts in the Philippines along with their representatives. In addition, there are Representatives elected through the party-list system who constitute not more than 20% of the total number of Representatives.

Lawmakers in the Senate are called Senators, who are elected at large or nationwide by voters to a six-year term. Senators can server for not more than two consecutive terms. This house has twenty-four (24) Senators.

Committees, or small groups of Representatives, headed by committee chairpersons, study proposed laws, called bills, and other measures relating to issued and concerns of our society.

C. Graph Construction

The graph is constructed by using a spreadsheet file of legislators. Each legislator is assigned as a node and an edge is drawn in between them if there exist a similarity in their political involvement in the same column of the file. Each similarity between the same involvement and legislator will add a unit in their edge weight. After the data has been parsed, a graph is then produced, where the weight between the edges is the total number of similarities the legislators have with the others.

D. Centrality

Centrality is the measure of which nodes are the most important in the network. The centrality of a node in a network can take numerous forms. In this paper, three different centrality measures were discussed – degree, betweenness, and closeness.

D..1 Degree Centrality

Degree central nodes are nodes that have the most edges connecting to other nodes. The node's, let's say node n , degree centrality is just the totality of number of edges, x , connecting to it and other nodes.

$$C_d (n_i) = \sum_j x_{ij} \quad (1)$$

A variation of this measure is also present in this paper, the weighted degree centrality. It is the total sum of the weights, w , of the edges connected to a node.

$$C_d (n_i) = \sum_j w_{ij} \quad (2)$$

Higher degree centrality gives the node more access to other nodes in the network. Degree centrality, in general, is just the number and strength of direct connections to a node.

D..2 Betweenness Centrality

Another way to measure node centrality is by the use of betweenness centrality. A node is central if their position in the network lies on the shortest path between many other nodes. This measure assumes that the nodes prefer to make connections

by choosing one of the shortest pathways, and that they are equally likely to choose any of the shortest pathways. If g is the number of nodes, g_{jk} is the number of pathways from node j and k , and $g_{jk}(n_i)$ is the number of pathways containing node i , then the centrality for node i is:

$$C_B(n_i) = \sum_{j < k} g_{jk}(n_i) / g_{jk}$$

The basic betweenness centrality measure assumes that nodes that fall on one of the shortest pathways between other nodes have a network advantage because others depend on them to information or resources.

D.3 Closeness Centrality

Closeness centrality considers the distance between actors. If a node has the highest closeness centrality because they have the shortest mean pathway to the other node, then by contrast, the longer the pathway to another nodes means that it has a lower closeness centrality. If $d(n_i, n_j)$ is the number of paths linking i and j , then the closeness of node i is:

$$C_C(n_i) = \left[\sum_{i \neq j}^g d(n_i, n_j) \right]^{-1}$$

E. Cliques

In given a graph $G = (V, E)$ where V and E are the set of vertices and the set of edges connecting the vertices, respectively, a clique is defined as a complete subgraph generated from G . It is a subgraph whose vertices are adjacent to each one of the vertices in that subgraph. A clique of size k , where k is the number of vertices in the clique, is called a k -clique. A clique of largest possible size is referred to as a

maximum clique. The figure below shows the graphical representation of a clique with the largest size which is represented as the blue entities.

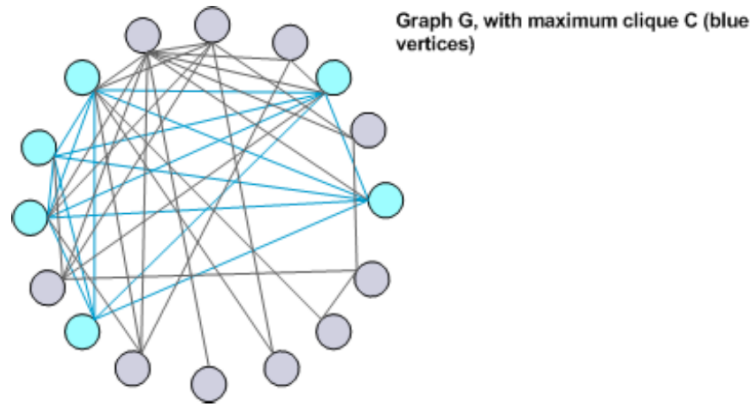


Figure 1: Graphical Representation of a Clique

F. Weighted Clique Problem

Consider a complete simple weighted undirected graph $G = (V, E, a, c)$ where a and c are weight functions that take on integer values for vertices and edges on V and E respectively.

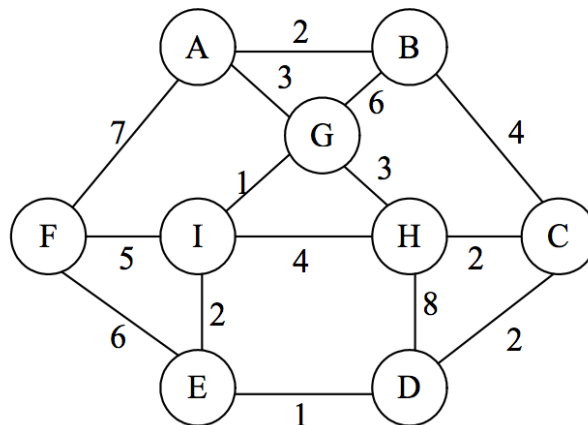


Figure 2: Weighted Clique

$$\text{The sum } \sum a_v + \sum c_e, \text{ for } v \in V \text{ and } e \in E \quad (3)$$

is called the total weight function of the graph G , thus the weight of the clique formed by the vertices F , I , and E is equal to 13. The weighted clique problem is defined as the problem of finding a complete subgraph (clique) of the graph G of order m with the smallest/largest weight, given the complete weighted undirected graph G . The computational complexity of the Weighted Clique Problem is determined by the known combinatorial clique problem, which is NP-complete. The known clique problem is just a question of the existence of a clique of order m given an undirected graph G .

G. 2-Approximation Algorithm for Finding a Clique with Minimum Weight of Vertices and Edges

The problem of finding a minimum weighted clique, with respect to the total weight of its vertices and edges of fixed size in a complete undirected weighted graph is always considered along with its subclasses. It is proven that finding a complete subgraph is NP-hard. Not only that, the inapproximability of the problem is also proven for the general case. Eremin et al. [4] suggested a 2-approximation efficient algorithm with time complexity $O(n^2)$ for the cases when vertex weights are nonnegative and the edge weights either satisfy the triangle inequality or are squared pairwise distances for some point configuration of Euclidean space.

One of the possible motivations in the problem of grouping similar objects and cluster analysis is the existence of identical elements that have equal values of a measured feature in a fixed collection of significant features from a set of objects under analysis. Sometimes, the data in these problems are given by a matrix of

pairwise comparison of object and their comparison criteria may be varied. This approach gave importance to those entries of the input matrix where:

1. They satisfy the triangle inequality
2. They are combinations of the weights of pairwise compared objects and Euclidean distances between them
3. They are combinations of the weights of pairwise compare objects and squared Euclidean distances between them
4. They are arbitrary nonnegative values

This general analysis is then applied to a more specific problem which is the weighted clique problem. The weighted clique problem utilizes an adjacency matrix of a graph where each element of the matrix is edge weight between each node that takes on non-negative values. Propositions presented proved that this problem is NP-hard in strong sense. It is also proven that if there exist an approximation algorithm, the solution to the problem will be bounded by a scalar multiple of the number of edges on the k-sized clique.

The search for an approximation algorithm was tackled in the point of view of the input matrix. As the definition said, a clique is a complete subgraph generated from an underlying graph. This means that in the counterpart adjacency matrix of this graph, a clique can be represented as a subset of this matrix. This is where the rows subset of symmetric matrix problem comes in. The problem is defined by an input weighted adjacency matrix, which is symmetric in nature, has nonnegative entries except for the diagonal entries which take on the value of 0. Using this input, we shall be able to find the sum of a row subset of this matrix with cardinality of m, a positive integer, and bounded by a positive value D.

$$F(C) = \frac{1}{2} \sum \sum w_{ij} \leq D , \text{ for } i \in C \text{ and } j \in C \quad (4)$$

$$W = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 & 2 & 0 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 0 & 1 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 & 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 0 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 2 & 0 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 1 & 2 & 1 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Figure 3: An example of a symmetric adjacency matrix

The RSSM problem is then proven as a polynomial time equivalent to the Minimum-Edge Weight Clique Problem in the form of property verification problem. An approximation algorithm is then presented for the RSSM problem. Step 1. For each $j = 1, \dots, n$, find a set B_j that consist of indices of m smallest entries in the j th row of the matrix W including j itself, Define

$$S(B_j) = \sum_{i \in B_j} w_{ij} \quad (5)$$

Step 2. Denote by k^* the value of j for which $S(B_j)$ takes the minimum value

$$S^* = S(B^*) = \sum_{i \in B_j} w_{ik} \quad (6)$$

Take $C = B^*$ as an approximate solution of RSSM. This algorithm is then proven as an approximate solution for the RSSM with the approximation guarantee 2 in time $O(n^2)$ and is asymptotically achievable.

They have showed that the general case of the problem is NP-hard and in-approximable. Nevertheless, in some rather natural cases, the problem admits 2-approximation polynomial time algorithms.

IV. Design and Implementation

A. Data Specifications

To find an exact or approximate maximum weighted clique or legislators, the researcher inputs a file. The format of the file should strictly comply with the following:

- The input file must be in a spreadsheet file
- The first column shall correspond the list of legislators.
- The subsequent columns must represent their labeled political involvements.

B. System Design

B.1 Context Diagram

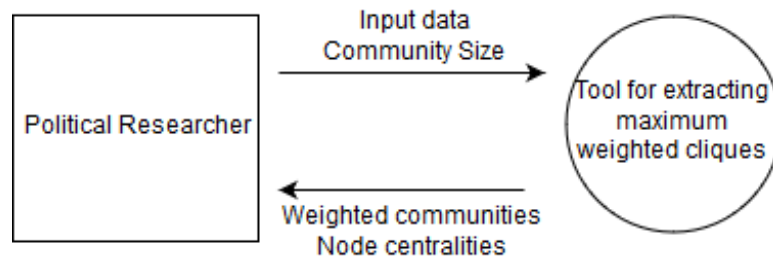


Figure 4: Context Diagram

The succeeding illustrations show how the system is implemented. The input and output requirements of the system is defined in the context diagram above. The researcher inputs legislator data along with the size of the clique to be found. The tool produces the maximum weighted clique as result.

B.2 Use Case Diagram

The researcher may choose to use an existing legislator data to be viewed as a graph or input another type of legislator data, along with the desired size of the community

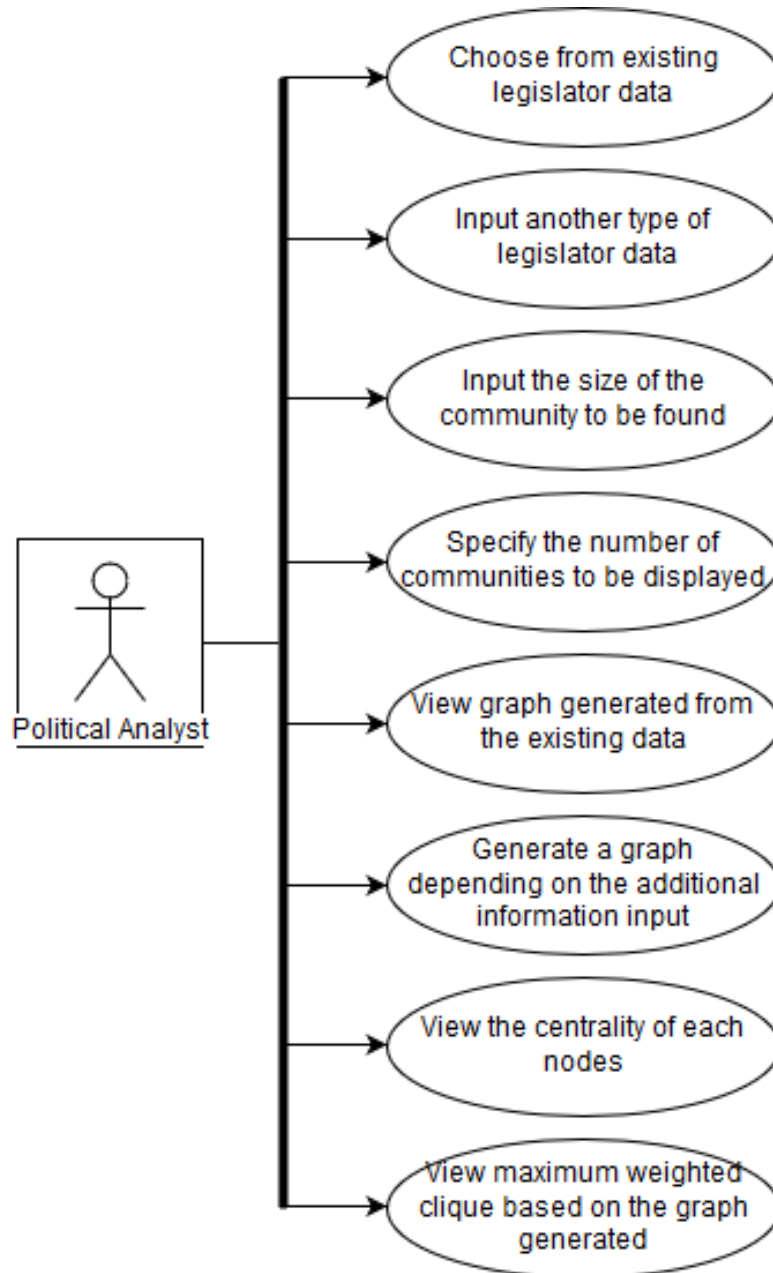


Figure 5: Use Case Diagram

to be found and the number of communities to be displayed. The tool generates a graph depending on this new data type. After the graph has been generated along its adjacency matrix, the clique finding algorithm finds a maximum weighted k-clique based on the generated graph and input data, and display the legislators that are members of this clique.

B.3 Flowchart Diagram

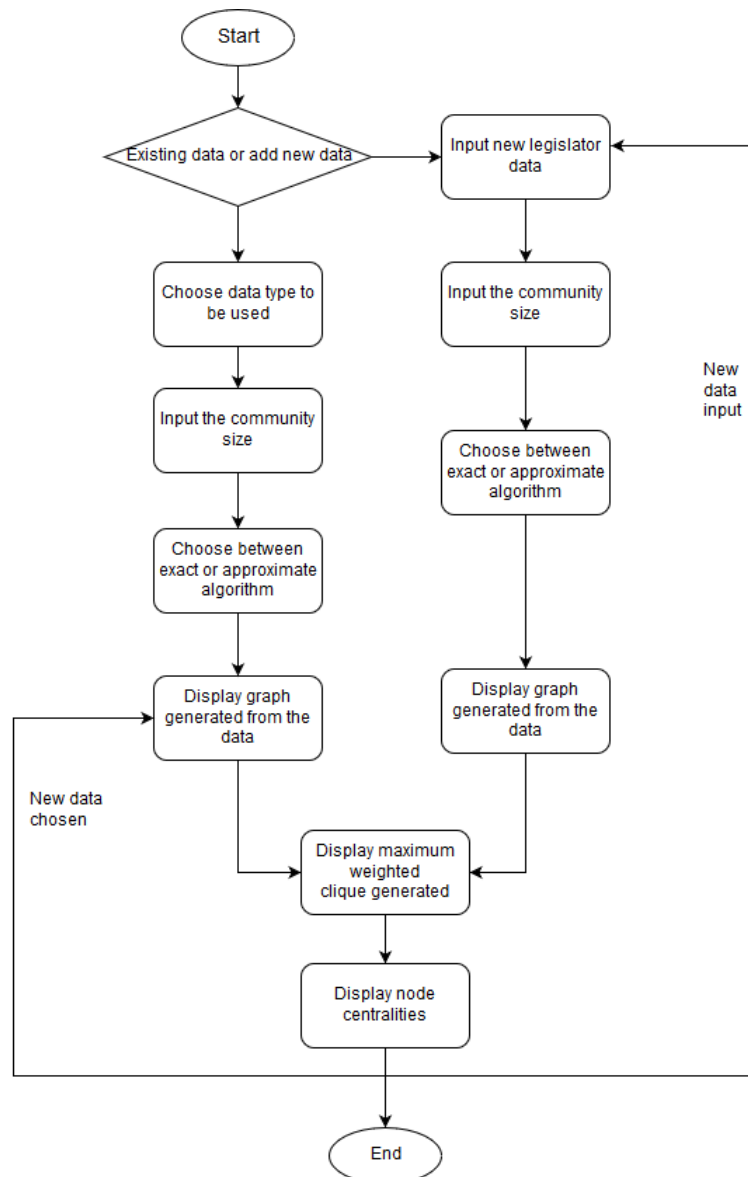


Figure 6: Flowchart Diagram

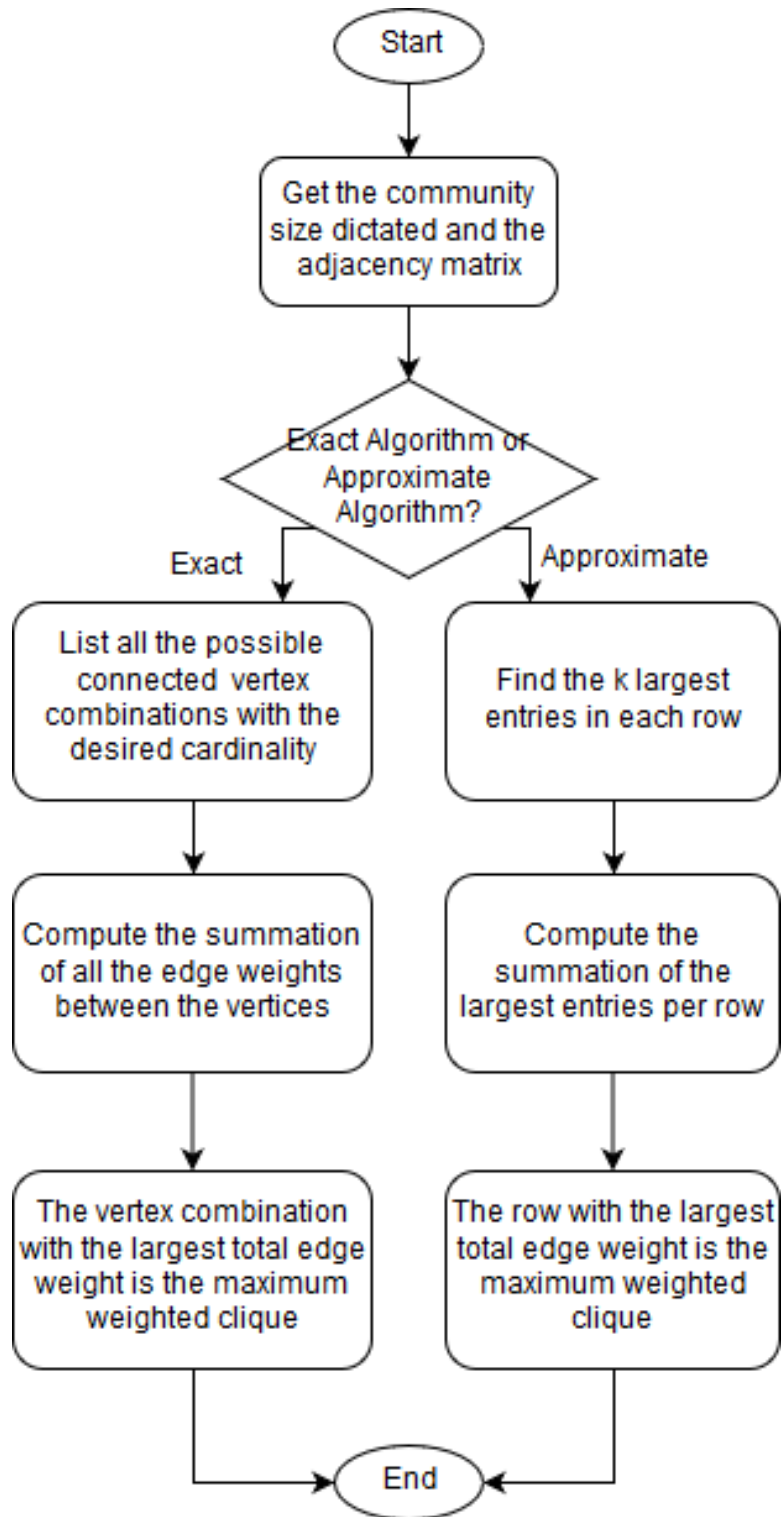


Figure 7: Algorithm Flowchart

First step in using the tool is to choose between using an existing, built-in data in the tool or input your own data type for the legislators, and enter the desired size of the clique to be found. The data determines how well-connected each of the legislator is to each other. After the data has been parsed, the tool generates a graph that visualizes the connectedness of each entity with each other. Along this generated graph is its adjacency matrix that is a tabular form of the node connections and the weights of the edges that connect them to each other.

The output of this tool is based on the generated graph and matrix from the input data. The tool finds the maximum weighted k-clique on the graph and compute for the centrality of each nodes. The tool displays the communities of legislators. This shows how much each legislator interact with each other under different circumstances and factors. The results also show whose legislators work together and are bounded by a common goal.

C. System Architecture

The tool uses R to construct the graph of legislators and it is integrated with Java that will handle the user interface along with the other functionalities. The user can run the program with an installed Java Runtime Environment, R program and RServe() package, and CUDA toolkit, and can run on any operating system.

D. Technical Architecture

Specifications required for the software to run the exact algorithm smoothly includes a graphics processing unit (GPU), preferably a NVIDIA GTX, if the approximate algorithm will be chosen, it will run smoothly on a 4 GB RAM and a processor speed of 2.40 ghz.

V. Results

As the tool runs, the user is presented with the main menu interface, where the user can choose between starting the analysis or seek information about the tool by clicking the "help" button.



Figure 8: Main Menu

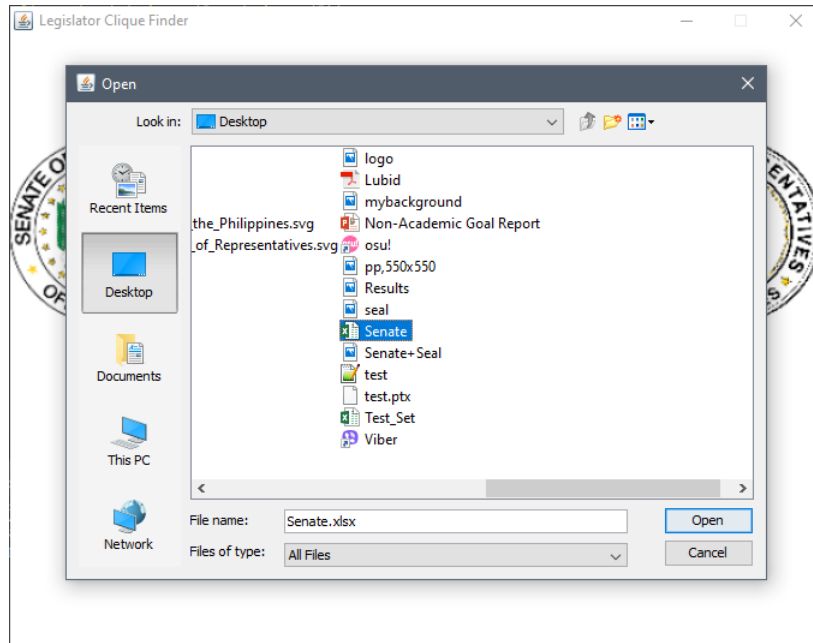


Figure 9: File Choosing

If the user chooses to add a file, the file chooser interface will immediately pop and direct the user to add the legislator data file it wants.

The user may input any file following the data set format specified in this discourse (Figure 9).

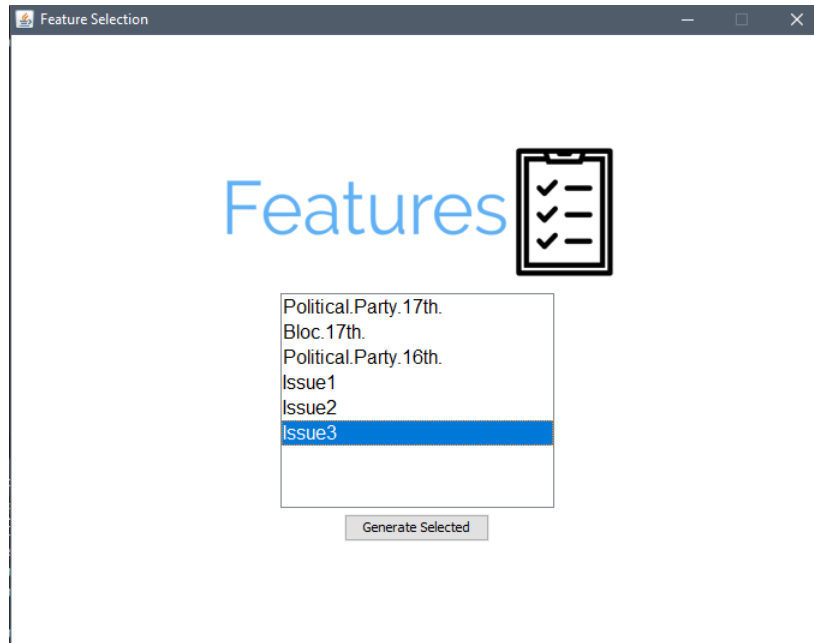


Figure 10: Features

After the file has been read and parsed, a list of legislator features specified in the data set is shown and the user may choose which of the legislator features will be used to create and evaluate the network of legislators.

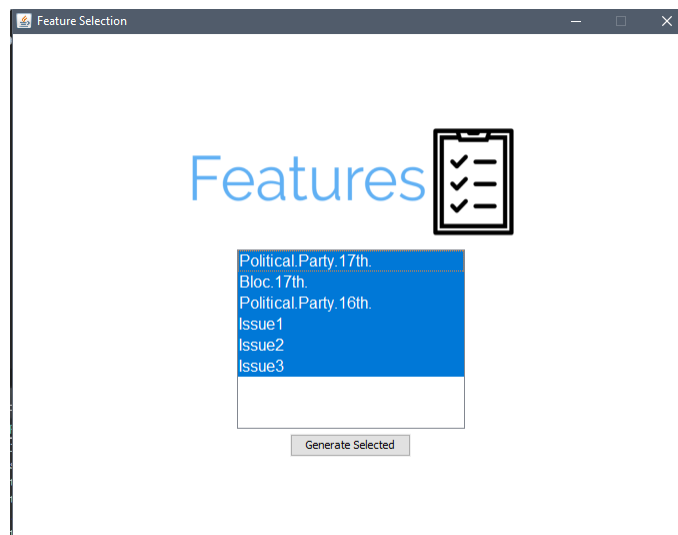


Figure 11: Feature Selection

The user may opt to use one, two, or all of the features in the feature list (Figure

11). An error of empty feature set is prompted if the user has not chosen any feature.

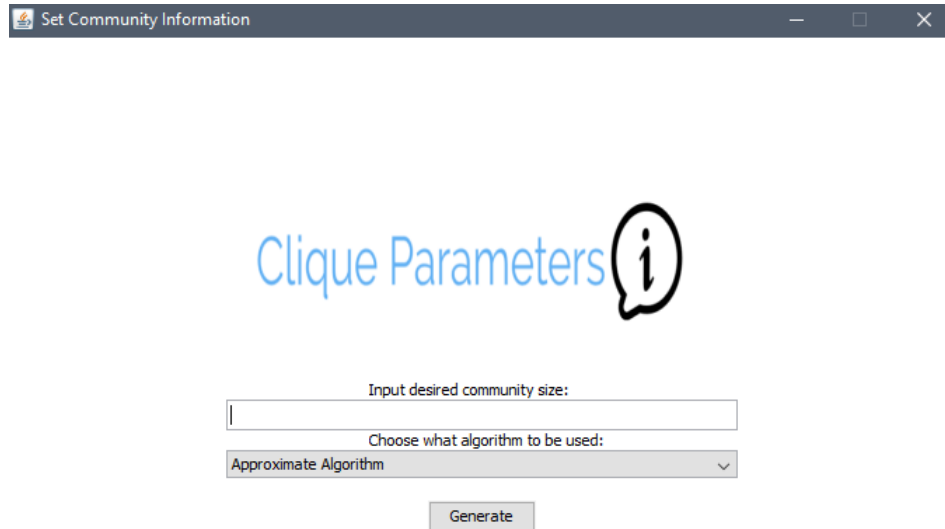


Figure 12: Specify Community Parameters

A graph is generated using these feature parameters. An edge is drawn from legislator node to another if there exist a similarity on their features and a unit is added to their edge weight for every similarity thereof. With the graph represented as an adjacency matrix inside the tool, the user is prompted to input the specifications of communities he/she wants to show. An input parameter interface helps this action.

The text field is for the user to input the size of each maximum weighted communities to be displayed .

After specifying the input parameters, the user may choose between approximate algorithm or exact algorithm to be used in community detection. The approximate algorithm is much faster than the exact algorithm but the results of it may not be optimal. In the exact algorithm, if a GPU device is present in the host, the user may choose where it wants to run the exact algorithm, in the GPU device or the host's

CPU. Clicking on the generate button yields the following results.

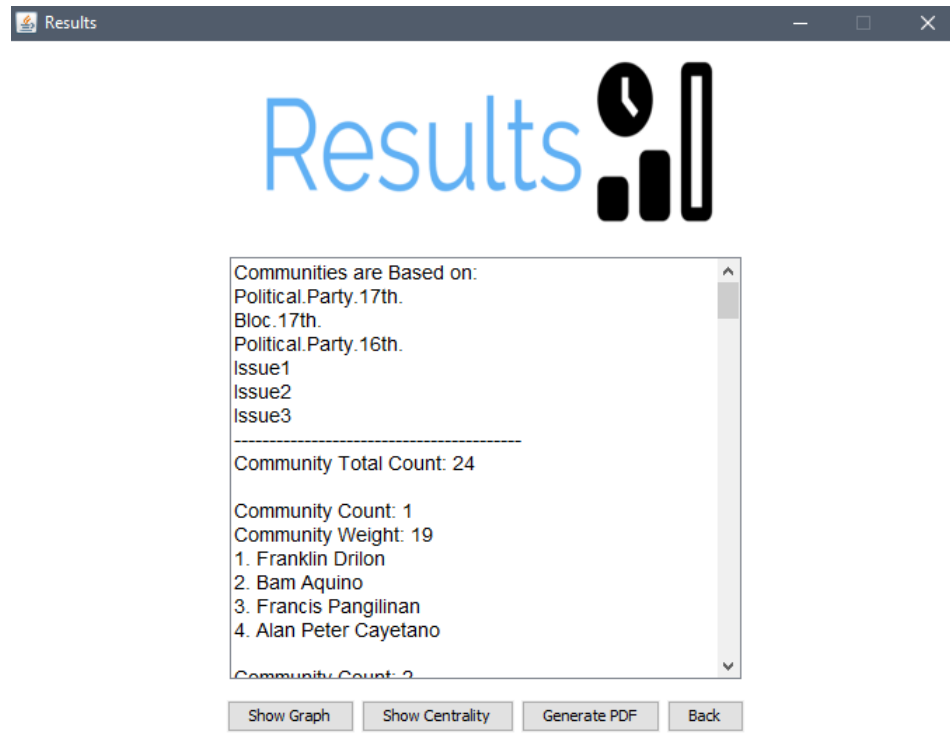


Figure 13: Sample Results

First, the list of the communities and the members under is displayed in a descending order, from highest weight to lowest, which are also displayed beside each community. The feature/s under consideration for each communities are also displayed. Then the user can choose either view the centralities of each nodes, generate a PDF containing the list of communities and the centrality table, or display the graph visualization generated from the input data file.

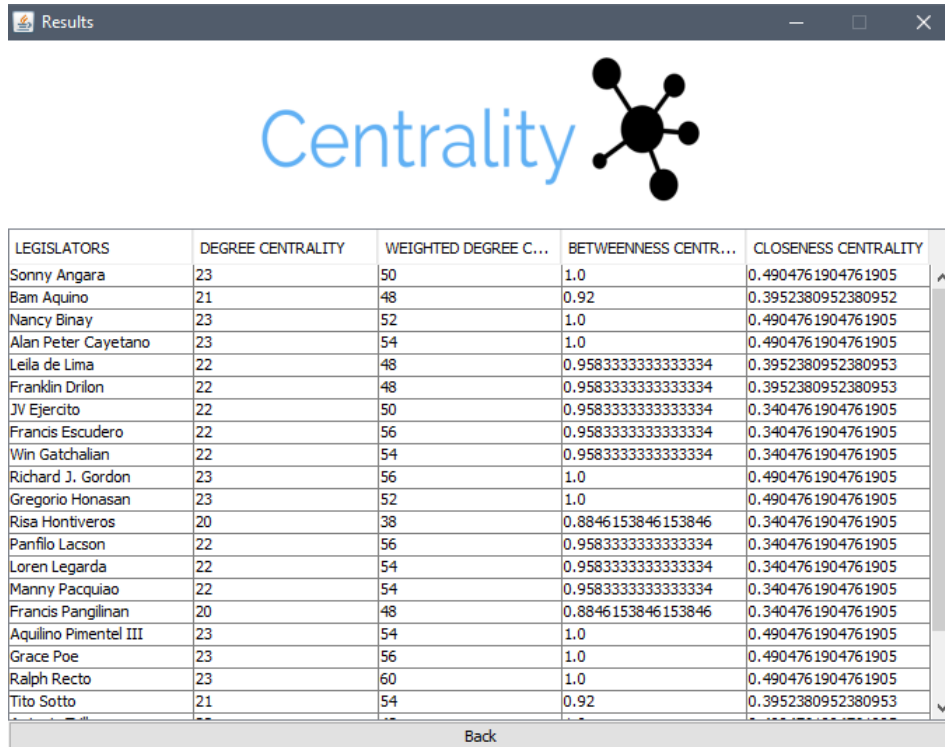


Figure 14: Node Centrality

The node centralities (Degree, Weighted Degree, Betweenness, and Closeness Centrality) is viewed as a table for easier comparison and reading. The table can be sorted according to the desired centrality.

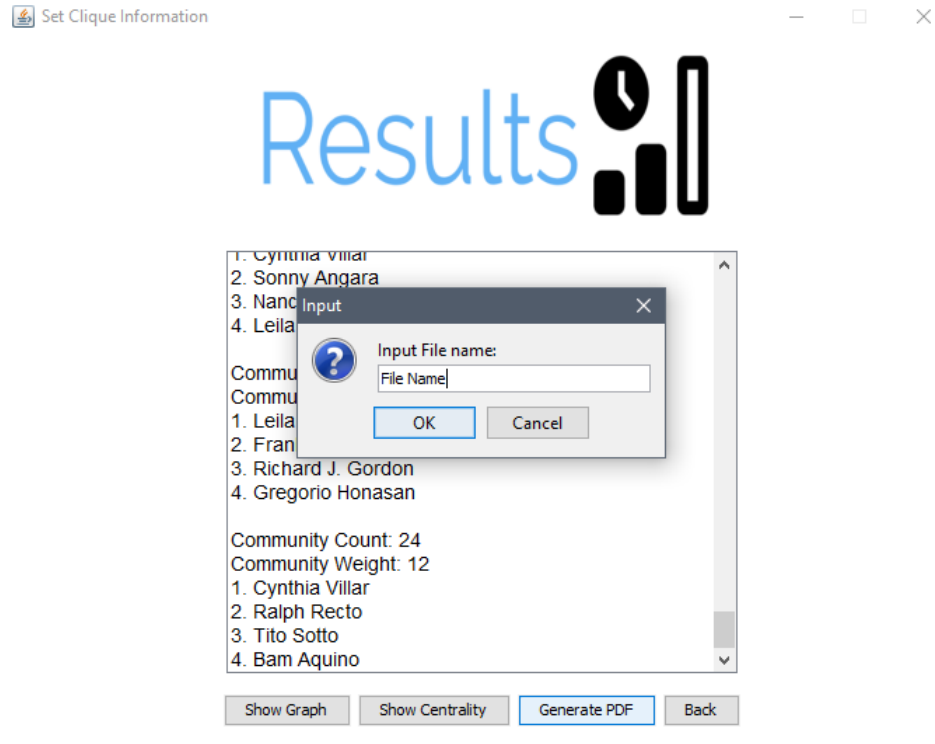


Figure 15: PDF Input File Name

If the user chooses to export the result as a PDF, he/she is prompted to enter a desired file name for the PDF file. The file name generated includes the tag of the tool appended in the beginning.

Results

The screenshot shows a software window titled "Set Clique Information" with a "Results" header. A notification dialog box titled "Success!" with an information icon and "PDF Generated!" text is centered over the main content. The main content displays a list of community members and statistics. At the bottom, there are four buttons: "Show Graph", "Show Centrality", "Generate PDF", and "Back".

1. Cynthia Villar
2. Sonny Angara
3. Nanc
4. Leila

Comm
Comm
1. Leila
2. Fran
3. Richard J. Gordon
4. Gregorio Honasan

Community Count: 24
Community Weight: 12
1. Cynthia Villar
2. Ralph Recto
3. Tito Sotto
4. Bam Aquino

Show Graph Show Centrality Generate PDF Back

Figure 16: PDF Generated

A notification pops up suggesting a successful file saving.



Figure 17: PDF Contents

The resulting PDF contains the logo of the software, the author name, and the date and time of creation. After that page, the results of the software is listed as well as the centrality table.

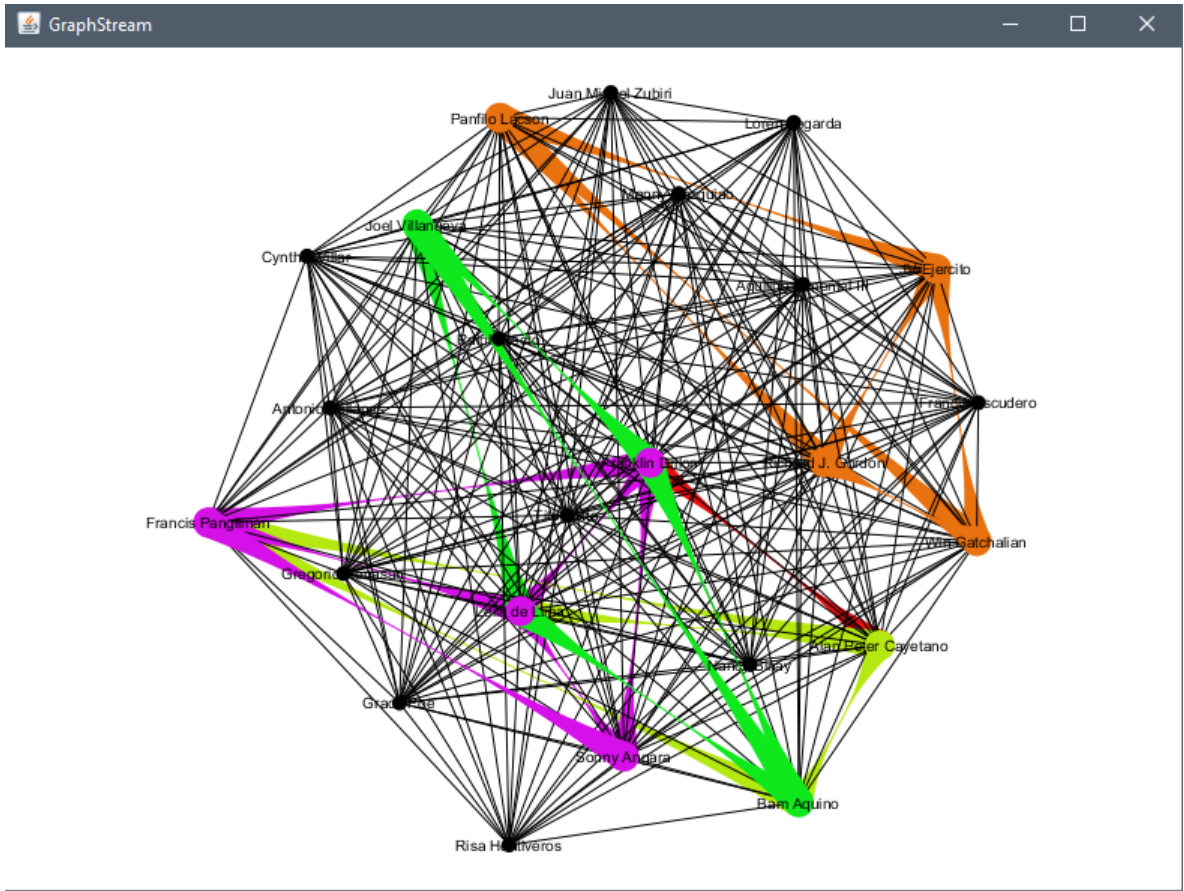


Figure 18: Graph Generated

The graph generated can be also viewed as the visual representation of the data input. The colored entities are the top 5 communities present in the network. The user can pan the graph by using the arrow keys and zoom using the page up/down keys.

VI. Discussions

Legislator Clique Finder is a Java-based application that aims to perform community detection in Philippine Congress Legislators by means of clique-finding algorithms. The basis of the relationship of the members of the community is defined by the similarities present in their different political involvement.

As the data input is placed, the user is given a chance to select the features he/she wants to use to define the relationship of the legislator nodes. After that, the tool generates a graph by building an edge and adding an edge weight for every similarity in the feature that is present between the legislator nodes.

The application provides its users the power to dictate the size of each community to be shown. Other than that, the application also allows the user to choose what algorithm to be used in detecting the said communities, approximate algorithm or the exact algorithm.

The approximate algorithm implemented by the application is the 2-approximation algorithm proposed in [4] that uses the same approach for the RSSM problem and guarantees an optimal result if the edge weights in the adjacency matrix generated follows the metric property. Meanwhile, the exact algorithm uses an exhaustive yet more effective approach by passing through all of the possible node combinations and return the combination with the maximum weight between its nodes. The application provides a GPU implementation for a much faster computation, given that a GPU device is available in the host machine otherwise, a CPU implementation of the exact algorithm is available.

These algorithms are applied and implemented by the tool to generate and detect the number of desired communities by the user. The application produces three outputs from user-defined specifications and data. The first one is the community list where user-defined number of community is displayed along with its members, where the first community has the largest weight between them. The larger the weight

between nodes in a community means more interaction and similarities happened between them.

The application also provides the user a table of centralities of each nodes. The degree centralities defines the number of direct connections of the nodes, the betweenness centrality provides the information on how a node is being used as a bridge between parties, and the closeness centrality defines how close a node is to a more central node. These pieces of information may conclude a more concrete position of each politician in a network.

These results can be exported and saved in a PDF file by choosing the option of generating the PDF file and specifying a file name for the output file.

The GraphStream library of Java provides the visualization of the graph generated from the user input and parameters. It enables a different and convenient point of understanding in viewing and visualizing the said data set by showing some community detected from the network and its ability to be panned and zoomed as well as for its nodes to be moved and placed in a different manner.

There's an absence of a community detection tool that focuses on political network that uses clique-finding algorithms. The process of finding one happens to be exhaustive and needs a chunk of the host memory, especially when a network has numerous nodes. This application provides the initial step in building a wider understanding of political networks and all of its components.

VII. Conclusions

Legislator Clique Finder is a stand-alone software produced in this study that aims to provide political researchers and analysts a way to view the occurrence of highly interactive nodes that forms a community. These community detection can be done using a choice of an algorithm provided and implemented in the software. Aside from that, the software provides a simple graph property extraction by means of node centralities that provides certain information on how each nodes works and interacts in a political network.

This application provides the initiative in helping user visualize and map different politician involvements for further understanding of relationship as well as discover interactions, provided by an empirical data, that may be hidden from the public. It also aims to provide assistance in visualizing and interpreting their gathered data, and provide a conclusive result that may be used for researches or just provide new insights and information regarding a political network.

VIII. Recommendations

The proponent suggests that the data set that will be used for further improvements involves a wider variety of political involvements and relationships, such as voting patterns in different congress sessions, for visible diversity of connections that are not easily seen in public data. The data under consideration may be expanded to different political aspects, such as foundations funded, or businesses handled by the legislators, for the software may yield other results that may be useful in studying political relationships.

In terms of network analysis, the proponents suggests researching on community modularity for it measures the strength of network modules such as communities. Additional implementations of clique-finding algorithms can be also added to simply compare different approaches and results to community detection in Philippine legislators. The proponent also suggests that the GPU implementation of algorithms be maximized for the potential of this approach can be useful for faster output generation and analysis.

Lastly, for the visualization aspect, the author recommends viewing the graph visualization of individual communities, and not just as a part of the network, for better component analysis and inspection.

IX. Bibliography

- [1] H. of Representatives, “Legislative information,”
- [2] F. Santo and D. Hric, “Community detection in networks: A user guide,” *Physics Reports*, vol. 659, pp. 1–44, 2016.
- [3] D. Knoke, *Political networks: the structural perspective*, vol. 4. Cambridge University Press, 1994.
- [4] E. Gimadi, M. Khachay, and A. V. Kel’manov, “2-approximation algorithm for finding a clique with minimum weigh of vertices and edges,” *Proceedings of the Steklov Institute of Mathematics*, pp. 87–97, 2014.
- [5] F.Hao, D. Park, and Z. Pei, “Exploiting the formation of maximal cliques in social networks,” *Symmetry*, 2017.
- [6] M. Ahuja, J. Singh, and Neha, “Practical applications of community detection,” *International Journal of Advanced Research in Computer Science and Software Engineering*, 2016.
- [7] F. Santo, “Community detection in graphs,” *Physics Reports*, vol. 486, pp. 75–174, 2016.
- [8] P. Saxena and D. Thakur, “Complexity analysis of clique problem,” *International Journal of Computer Science and Engineering*.
- [9] R. Ross, D. F. Gleich, and A. H. Gebremedhin, “Parallel maximum clique algorithms with applications to network analysis,” *Society for Industrial and Applied Mathematics*, 2015.
- [10] D. Wood, “An algorithm for finding a maximum clique in a graph,” *Operations Research Letters*, vol. 21, pp. 211–217, 1997.

- [11] D. Palsetia, M. A. Patwary, W. Hendrix, A. Agrawal, and A. Choudhary, “Clique guided community detection,” *IEEE International Conference on Big Data*, 2014.
- [12] M. S. Chang and F. H. Wang, “Efficient algorithms for the maximum weight clique and maximum weight independent set problems of permutation graphs,” *Information Processing letters*, vol. 43, pp. 293–295, 1992.
- [13] V. Vassilevska, “Efficient algorithms for clique problems,” *Information Processing Letters*, vol. 109, no. 4, pp. 254–257, 2009.
- [14] E. M. Hafner-Burton and A. H. Montgomery, “Centrality in politics: How networks confer power,” *OpenSIUC*, 2010.
- [15] “What went before: The 10-b pork barrel scam,” *Philippine Daily Inquirer*, 2017.

X. Appendix

A. Source Code

CliqueFinder_GUI.java

```
import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.filechooser.FileSystemView;
import org.rosuda.REngine.REXPMismatchException;
import org.rosuda.REngine.Rserve.RserveException;
import org.graphstream.graph.*;
import org.graphstream.graph.implementations.*;
import java.io.File;
import java.io.IOException;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Insets;
import java.awt.Toolkit;
import java.awt.event.*;
public class CliqueFinder_GUI{
    String R_Path_Parsed = "";
    JFrame frame;
    ArrayList<Integer> feature_column = new ArrayList<Integer>();
    R.Integration_RSERVE r;
    int [] selected;
    JTable centrality_table;
    JScrollPane scrollPane = new JScrollPane();
    JList featureList;
    boolean isSelected;
    boolean isAvailDev;
    JPanel feature_panel = new JPanel();
    Graph g;
    String [][] data;
    String [] column_header = {"LEGISLATORS", "DEGREE CENTRALITY", "WEIGHTED DEGREE CENTRALITY",
        "BETWEENNESS CENTRALITY", "CLOSENESS CENTRALITY"};
    ArrayList<int []> colors = new ArrayList<int []>();
    public CliqueFinder_GUI() throws RserveException, REXPMismatchException {
        int [] color1 = {232,16,16};
        int [] color2 = {181,232,16};
        int [] color3 = {16,232,30};
        int [] color4 = {214,16,232};
        int [] color5 = {232,113,16};

        colors.add(color1);
        colors.add(color2);
        colors.add(color3);
        colors.add(color4);
        colors.add(color5);

        r = new R.Integration_RSERVE();
        isSelected = false;
        Toolkit tk = Toolkit.getDefaultToolkit();
        int window_size = tk.getScreenResolution();
        int window_x = (int)(tk.getScreenSize().width * 0.50);
        int window_y = (int)(tk.getScreenSize().height * 0.70);
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException
            | UnsupportedLookAndFeelException e) {
            e.printStackTrace();
        }

        //Containers and Layouts
        Container box;
        BorderLayout bor = new BorderLayout();
        GridLayout grid = new GridLayout(2,0);
        CardLayout card = new CardLayout();
        GridBagLayout gb = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(window_y/1000, window_x/100, 0, window_x/100);

        //Icons and Labels
```

```

Image logo = null;
try {
    logo = ImageIO.read(getClass().getResource("/Image/logo.PNG"));
    getScaledInstance(window_x/(6/5), window_y/(2), Image.SCALE.SMOOTH);
} catch (IOException e2) {
    // TODO Auto-generated catch block
    e2.printStackTrace();
}
ImageIcon icon = new ImageIcon(logo);

JLabel logoD = new JLabel(icon);

//logoD.setIcon(icon);
//logo.setSize(new Dimension(window_x *,window_y - 40));

//Frames
frame = new JFrame("Legislator Clique Finder");
JFrame feature_frame = new JFrame("Choose the Features");
JFrame centrality_frame = new JFrame("Centralities");
box = frame.getContentPane();
centrality_frame.setSize(new Dimension(window_x, window_y));
feature_frame.setSize(600, 500);

//TextFields and TextAreas
JTextField comm_field, size_field;
JTextArea result_field = new JTextArea();
JTextArea help_area = new JTextArea();
result_field.setSize(600, 500);
comm_field = new JTextField();
comm_field.setText("Number of Communities");
size_field = new JTextField();
size_field.setText("Size per Community");

help_area.setText("> Click Add file then choose the data set. \nMake sure that
the data set follows the data set format specified \n> Choose the features
you want to visualize (At least 1) \n> Specify the community size and number
of community to be displayed \n> Choose between Approximate and Exact
Algorithm \n> View Results and Graph");

//Panels
JPanel panel = new JPanel();
JPanel setInfo_panel = new JPanel();
JPanel results_panel = new JPanel();
JPanel centrality_panel = new JPanel();
JPanel help_panel = new JPanel();
JPanel button_panel = new JPanel();

//Buttons
JButton generate_button = new JButton("Generate");
JButton generate_all, generate_selected, centrality_button, showGraph_button,
back_button;
JButton start_button = new JButton("Start");
JButton help_button = new JButton("Help");
back_button = new JButton("Back");
generate_all = new JButton("Generate All");
generate_selected = new JButton("Generate Selected");
centrality_button = new JButton("Show Centrality");
showGraph_button = new JButton("Show Graph");

start_button.setPreferredSize(new Dimension(window_x/3, window_y/6));
start_button.setFont(new Font("Arial",Font.PLAIN, 16));
help_button.setPreferredSize(new Dimension(window_x/3, window_y/6));
help_button.setFont(new Font("Arial",Font.PLAIN, 16));
//ComboBox
String[] algo_boxString = {"Approximate Algorithm", "Exact Algorithm(GPU)", "Exact
Algorithm(CPU)"};
JComboBox algo_box = new JComboBox(algo_boxString);

JFileChooser jfc = new JFileChooser(FileSystemView.getFileSystemView().
getHomeDirectory());

button_panel.setLayout(grid);
button_panel.setPreferredSize(new Dimension(window_x/3, window_y/3));
button_panel.add(start_button);
button_panel.add(help_button);
//results_panel.setLayout(bor);
results_panel.add(result_field);
results_panel.add(centrality_button);
results_panel.add(back_button);

//results_panel.add(button_panel, bor.PAGE_END);

feature_frame.add(generate_all);
feature_frame.add(generate_selected);

JPanel filler = new JPanel();
JPanel filler2 = new JPanel();
filler.setPreferredSize(new Dimension(window_x/3, window_y/5));
filler2.setPreferredSize(new Dimension(window_x/3, window_y/5));
filler.setBackground(Color.white);
filler2.setBackground(Color.white);

```

```

setInfo_panel.setLayout(grid);
setInfo_panel.add(comm_field);
setInfo_panel.add(size_field);
setInfo_panel.add(algo_box);
setInfo_panel.add(back_button);
setInfo_panel.add(generate_button);
setInfo_panel.setSize(new Dimension(window_x - 100,window_y - 100));

panel.setSize(new Dimension(window_x , window_y));
panel.setBackground(Color.WHITE);
panel.setLayout(new BorderLayout(10,10));
panel.setBorder(BorderFactory.createEmptyBorder(65,0,115,0));
panel.add(logoD , BorderLayout.PAGE_START);
panel.add(button_panel , BorderLayout.CENTER);
panel.add(filler2 , BorderLayout.LINE_START);
panel.add(filler , BorderLayout.LINE_END);

gbc.gridx = 0;
gbc.gridy = 2;
gbc.ipadx = 1;
// panel.add(help_button , gbc);

frame.add(panel);
frame.setSize(new Dimension(window_x , window_y));
frame.setVisible(true);
frame.setResizable(false);
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

help_panel.setLayout(new BorderLayout());
help_panel.add(help_area , BorderLayout.CENTER);
help_panel.add(back_button , BorderLayout.PAGE_END);
help_button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        frame.setContentPane(help_panel);
        frame.revalidate();
        frame.repaint();
    }
});

start_button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        int returnValue = jfc.showOpenDialog(null);
        if(returnValue == JFileChooser.APPROVE_OPTION){
            File selectedFile = jfc.getSelectedFile();
            R_Path_Parsed = selectedFile.getAbsolutePath().replace
                ('\\', '/');
            System.out.println(R_Path_Parsed);
            try {
                r.r_executeConnection(R_Path_Parsed);
            } catch (RserveException | REXPMismatchException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

            Image image = null;
            try {
                image = ImageIO.read(getClass().getResource("/
                    Image/Features.PNG")).getScaledInstance(
                    window_x/2, window_y/4 , Image.SCALE_SMOOTH);
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            ImageIcon icon = new ImageIcon(image);
            JLabel logo = new JLabel(icon);
            JList jList = new JList(r.getFeatureList());
            featureList = jList;
            feature_panel.setPreferredSize(new Dimension(window_x ,
                window_y));
            feature_panel.setBackground(Color.WHITE);
            feature_panel.setLayout(gb);

            JScrollPane featureScroll = new JScrollPane(featureList);
            JPanel featureButtons = new JPanel();
            featureButtons.add(generate_selected);
            featureButtons.setBackground(Color.white);
            //featureButtons.add(back_button);
            featureScroll.setPreferredSize(new Dimension(window_x/3,
                window_y/3));
            featureList.setFont(new Font(" Arial",Font.PLAIN, 16));
            feature_panel.setBackground(Color.white);
            gbc.gridx = 0;
            gbc.gridy = 0;
            feature_panel.add(logo , gbc);

            gbc.gridx = 0;
            gbc.gridy = 1;

```



```

        //gbc.gridheight = 3;
        feature_panel.add(featureScroll, gbc);

        gbc.gridx = 0;
        gbc.gridy = 2;
        //gbc.gridheight = 3;
        feature_panel.add(featureButtons, gbc);
        gbc.gridx = 0;
        gbc.gridy = 3;
        //feature_panel.add(back_button, gbc);

        //feature_frame.setVisible(true);
        //System.out.println(R.Path_Parsed);
        frame.setTitle("Feature Selection");
        //card.show(box, "features");
        frame.setContentPane(feature_panel);
        frame.revalidate();
    }
}
});

generate_selected.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        MWCP_Exact.GPU m = new MWCP_Exact.GPU();

        if(m.getNumOfGPUAvailable() <= 0){
            JOptionPane.showMessageDialog(null, "GPU Absent!", "No GPU
                Error", JOptionPane.WARNING_MESSAGE);
            isAvailDev = false;
        }else{
            JOptionPane.showMessageDialog(null, m.getNumOfGPUAvailable
                () + " GPU Device/s Found!", "GPU Present",
                JOptionPane.INFORMATION_MESSAGE);
            isAvailDev = true;
        }

        if(featureList.isSelectionEmpty() == false){
            selected = new int[featureList.getSelectedIndices().length
                ];
            selected = featureList.getSelectedIndices();
            isSelected = true;
            System.out.println(selected.length);
            for(int s: selected){
                System.out.println("Feature index " + s);
            }
            frame.setTitle("Set Community Information");
            frame.setContentPane(SetInfoPanel(window_x, window_y,
                isAvailDev));
            frame.revalidate();
            frame.repaint();
        }else{
            JOptionPane.showMessageDialog(null, "You have not selected
                a feature", "Empty Feature Set", JOptionPane.
                WARNING_MESSAGE);
        }
    }
});

showGraph_button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        g.display();
    }
});

back_button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        if(frame.getContentPane().equals(setInfo_panel)){
            //frame.removeAll();
            frame.setContentPane(feature_panel);
            frame.revalidate();
            frame.repaint();
        }else if(frame.getContentPane().equals(help_panel)){
            //featureList.clearSelection();
            frame.setContentPane(panel);
            frame.revalidate();
            frame.repaint();
        }else if(frame.getContentPane().equals(scrollPane)){
            frame.setContentPane(setInfo_panel);
            frame.revalidate();
            frame.repaint();
        }
    }
});
//panel.
}

public Graph constructGraph_Stream(ArrayList<Legislator> legist, int [][] ref_matrix){
    Graph g = new SingleGraph("Philippine Legislators Network");
    for(int legist_iter = 0; legist_iter < legist.size(); legist_iter++){

```

```

        g.addNode(legist.get(legist_iter).getName());
    }
    for(int row_iter = 0; row_iter < ref_matrix.length; row_iter++){
        for(int col_iter = row_iter; col_iter < ref_matrix[row_iter].length;
            col_iter++){
            if(ref_matrix[row_iter][col_iter] != 0){
                try{
                    g.addEdge(legist.get(row_iter).getName() + legist.get(
                        col_iter).getName(), legist.get(row_iter).getName(),
                        legist.get(col_iter).getName());
                }catch (EdgeRejectedException e){
                    e.printStackTrace();
                }
            }
        }
    }
    for(Node node: g){
        node.addAttribute("ui.label",node.getId());
    }

    g.addAttribute("ui.antialias");
    return g;
}

public String toDisplayField(ArrayList<Legislator[]> result,ArrayList<Integer> weights,
    int number_of_community, int clique_size){
    String display = "";
    display = display + ("Communities are Based on: ");
    for(String s : r.getFeatureList()){
        display = display + "\n" + s;
    }

    display = display + "\n-----\n";
    display = display + ("Community Total Count: " + result.size());
    for(int iterator_comm = 0; iterator_comm < result.size(); iterator_comm++){
        display = display + ("\n\nCommunity Count: " + (iterator_comm + 1));
        display = display + ("\nCommunity Weight: " + weights.get(iterator_comm));
        for(int iterator_leg = 0; iterator_leg < result.get(iterator_comm).length;
            iterator_leg++){
            display = display + ("\n" + (iterator_leg + 1) + ". " + result.get(
                iterator_comm)[iterator_leg].getName());
        }
    }
    return display;
}

public JPanel SetInfoPanel(int width, int height, boolean isAvailDevice){
    JLabel community, clique, param, algo;
    JPanel setPanel = new JPanel();
    setPanel.setBackground(Color.white);
    Image image = null;
    JTextField community_text, clique_text;
    JComboBox algorithms;
    JButton generate, back;
    setPanel.setLayout(new BorderLayout(10,10));
    setPanel.setBorder(BorderFactory.createEmptyBorder(100,10,150,10));
    setPanel.setForeground(Color.WHITE);
    try {
        image = ImageIO.read(getClass().getResource("/Image/CliqueParam.PNG"));
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    ImageIcon icon = new ImageIcon(image);

    param = new JLabel(icon);
    JPanel infoPanel = new JPanel();
    infoPanel.setBackground(Color.white);
    JPanel buttonPanel = new JPanel();
    buttonPanel.setBackground(Color.white);
    JPanel fillerPanel = new JPanel();
    fillerPanel.setBackground(Color.white);
    JPanel fillerPanel2 = new JPanel();
    fillerPanel2.setBackground(Color.white);

    fillerPanel.setPreferredSize(new Dimension(width/5, height/5));
    fillerPanel2.setPreferredSize(new Dimension(width/5, height/5));
    infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.Y_AXIS));
    generate = new JButton("Generate");
    back = new JButton("Back");
    community = new JLabel("Input desired number of community: ");
    community.setAlignmentX(Component.LEFT_ALIGNMENT);
    clique = new JLabel("Input desired community size: ");
    clique.setAlignmentX(Component.LEFT_ALIGNMENT);
    algo = new JLabel("Choose what algorithm to be used: ");
    algo.setAlignmentX(Component.LEFT_ALIGNMENT);
    community_text = new JTextField();
    clique_text = new JTextField();

    infoPanel.setPreferredSize(new Dimension(width/3, height/5));

```

```

infoPanel.add(clique);
infoPanel.add(clique_text);
infoPanel.add(algo);

if(isAvailDevice){
    String[] algo_boxString = {"Approximate Algorithm", "Exact Algorithm(GPU)",
        "Exact Algorithm(CPU)"};
    algorithms = new JComboBox(algo_boxString);
} else {
    String[] algo_boxString = {"Approximate Algorithm", "Exact Algorithm(CPU)",
        ""};
    algorithms = new JComboBox(algo_boxString);
}

infoPanel.add(algorithms);
buttonPanel.add(generate);

setPanel.add(param, BorderLayout.PAGE_START);
setPanel.add(infoPanel, BorderLayout.CENTER);
setPanel.add(fillerPanel, BorderLayout.LINE_START);
setPanel.add(fillerPanel2, BorderLayout.LINE_END);
setPanel.add(buttonPanel, BorderLayout.PAGE_END);

generate.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        r.setCliqueSize(Integer.parseInt(clique_text.getText()));
        if(((Integer.parseInt(clique_text.getText())) < r.getLegislators().size()){
            try {
                r.r_execute(algorithms.getSelectedItem().toString(), selected);
            } catch (RserveException | REXPMismatchException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            g = constructGraph_Stream(r.toThrowLegislator, r.toThrow_adj);
            //g.display();
            frame.setTitle("Results");
            frame.setContentPane(new ResultsPanel(width, height));
            frame.revalidate();
        } else {
            JOptionPane.showMessageDialog(null, "The clique size entered is greater than the number of nodes", "Invalid Clique Size", JOptionPane.WARNING_MESSAGE);
        }
    }
});

back.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        frame.setContentPane(feature_panel);
        frame.revalidate();
        frame.repaint();
    }
});
return setPanel;
}

public JPanel ResultsPanel(int width, int height){
    JPanel resultsPanel = new JPanel();
    resultsPanel.setBackground(Color.white);
    JButton graph, centrality, back;
    JScrollPane scroller;
    JTextArea resultField = new JTextArea();
    Image image = null;
    try {
        image = ImageIO.read(getClass().getResource("/Image/Results.PNG")).getScaledInstance(width/2,height/4, Image.SCALESMOOTH);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    ImageIcon icon = new ImageIcon(image);
    JLabel logo = new JLabel(icon);
    JPanel buttonPanel = new JPanel();
    buttonPanel.setBackground(Color.white);
    JPanel fillerPanel = new JPanel();
    fillerPanel.setBackground(Color.white);
    JPanel fillerPanel2 = new JPanel();
    fillerPanel2.setBackground(Color.white);
    fillerPanel.setPreferredSize(new Dimension(width/5, height/5));
    fillerPanel2.setPreferredSize(new Dimension(width/5, height/5));
    resultField.setEditable(false);
}

```

```

resultField.setText(toDisplayField(r.getResults(), r.getResultWeights(), r.
    getNumberOfCommunities(), r.getCliqueSize()));
resultField.setFont(new Font("Arial", Font.PLAIN, 14));
scroller = new JScrollPane(resultField);
graph = new JButton("Show Graph");
centrality = new JButton("Show Centrality");
back = new JButton("Back");
JButton pdf = new JButton("Generate PDF");

buttonPanel.add(graph);
buttonPanel.add(centrality);
buttonPanel.add(pdf);
buttonPanel.add(back);

resultsPanel.setLayout(new BorderLayout(10,10));
resultsPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
resultsPanel.setPreferredSize(new Dimension(width, height));
resultsPanel.setForeground(Color.WHITE);

resultsPanel.add(logo, BorderLayout.PAGE_START);
resultsPanel.add(scroller, BorderLayout.CENTER);
resultsPanel.add(fillerPanel, BorderLayout.LINE_START);
resultsPanel.add(fillerPanel2, BorderLayout.LINE_END);
resultsPanel.add(buttonPanel, BorderLayout.PAGE_END);

graph.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        //int CliqueCounter = 0;
        int bound = 5;
        if (r.getResults().size() < bound){
            bound = r.getResults().size();
        }
        for(int CliqueCounter = 0; CliqueCounter < bound ; CliqueCounter
            ++){
            for(int nodeCounter = 0; nodeCounter < r.getResults().get(0).
                length; nodeCounter++){
                Legislator [] l = r.getResults().get(CliqueCounter);
                g.getNode(l[nodeCounter].getName()).setAttribute("ui.style",
                    " fill-color: rgb(" + colors.get(CliqueCounter)[0] +
                    "," + colors.get(CliqueCounter)[1] + "," + colors.get(
                    CliqueCounter)[2] + ")"; size: 20px;");

                for(int nodeCounter2 = nodeCounter + 1; nodeCounter2 < r.
                    getResults().get(0).length; nodeCounter2++){
                    g.getNode(l[nodeCounter].getName()).getEdgeBetween(l[
                        nodeCounter2].getName()).setAttribute("ui.style", "
                        fill-color: rgb(" + colors.get(CliqueCounter)[0] +
                        "," + colors.get(CliqueCounter)[1] + "," + colors.get(
                        CliqueCounter)[2] + ")"; shape: blob;");
                }
            }
        }
        g.display();
    }
});

pdf.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        int [] dc = r.getDegCentrality();
        double [] cc = r.getCloCentrality();
        double [] bc = r.getBetCentrality();
        int [] dwc = r.getDegWeighted();
        data = new String[r.getLegislators().size()][5];
        for(int row_iter = 0; row_iter < r.getLegislators().size();
            row_iter++){
            data[row_iter][0] = r.getLegislators().get(row_iter).
                getName();
            data[row_iter][1] = Integer.toString(dc[row_iter]);
            data[row_iter][2] = Integer.toString(dwc[row_iter]);
            data[row_iter][3] = Double.toString(cc[row_iter]);
            data[row_iter][4] = Double.toString(bc[row_iter]);
        }
        String name = JOptionPane.showInputDialog("Input File name: ");
        String fileName = "[LCF] " + name + " Results.pdf";
        ReportMaker r = new ReportMaker(fileName, System.getProperty("user.
            name"), column_header, data);
        r.makeReport(resultField.getText());
        JOptionPane.showMessageDialog(null, "PDF Generated!", "Success!",
            JOptionPane.INFORMATION_MESSAGE);
    }
});

centrality.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        frame.setContentPane(centralityPanel(width, height));
        frame.revalidate();
        frame.repaint();
    }
}

```

```

    });
    back.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            frame.setTitle("Set Clique Information");
            frame.setContentPane(SetInfoPanel(width, height, isAvailDev));
            frame.revalidate();
            frame.repaint();
        }
    });
    return resultsPanel;
}

public JPanel centralityPanel(int width, int height){
    JPanel centralityP = new JPanel();
    centralityP.setBackground(Color.white);
    JButton back_button = new JButton("Back");
    centralityP.setLayout(new BorderLayout());
    centralityP.setPreferredSize(new Dimension(width, height));
    centralityP.setForeground(Color.WHITE);

    Image image = null;
    try {
        image = ImageIO.read(getClass().getResource("/Image/Centrality.PNG"));
        getScaledInstance(width/2,height/4, Image.SCALE_SMOOTH);
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    ImageIcon icon = new ImageIcon(image);
    JLabel logo = new JLabel(icon);
    JScrollPane scrollPane;
    int [] dc = r.getDegCentrality();
    double [] cc = r.getCloCentrality();
    double [] bc = r.getBetCentrality();
    int [] dwc = r.getDegWeighted();
    data = new String[r.getLegislators().size()][5];
    for(int row_iter = 0; row_iter < r.getLegislators().size(); row_iter++){
        data[row_iter][0] = r.getLegislators().get(row_iter).getName();
        data[row_iter][1] = Integer.toString(dc[row_iter]);
        data[row_iter][2] = Integer.toString(dwc[row_iter]);
        data[row_iter][3] = Double.toString(cc[row_iter]);
        data[row_iter][4] = Double.toString(bc[row_iter]);
    }

    centrality_table = new JTable(data, column_header){
        /**
         *
         *
         */
        private static final long serialVersionUID = -43591001090799797L;

        public boolean isCellEditable(int row, int column){
            return false;
        }
    };
    centrality_table.setAutoCreateRowSorter(true);
    scrollPane = new JScrollPane(centrality_table);

    centralityP.add(logo, BorderLayout.PAGE_START);
    centralityP.add(scrollPane, BorderLayout.CENTER);
    centralityP.add(back_button, BorderLayout.PAGE_END);
    back_button.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            frame.setTitle("Set Clique Information");
            frame.setContentPane(ResultsPanel(width, height));
            frame.revalidate();
            frame.repaint();
        }
    });

    return centralityP;
}
}

```

CliqueFinder_Main.java

```

import org.rosuda.REngine.REXPMismatchException;
import org.rosuda.REngine.Rserve.RserveException;
public class CliqueFinder_Main {
    public static void main(String [] args) throws RserveException, REXPMismatchException{
        System.out.println(" result="+R_Connector.checkLocalRserve());
        System.setProperty("org.graphstream.ui.renderer", "org.graphstream.ui.j2dviewer.
            J2DGraphRenderer");
    }
}

```

```

        CliqueFinder_GUI g = new CliqueFinder_GUI();
    }
}

```

Graph_Construction.java

```

import java.util.ArrayList;

public class Graph_Construction {
    public Graph_Construction(){
    }

    public int [][] construct_adjacency_matrix(ArrayList<Legislator> legislators){
        int [][] adj_matrix = new int[legislators.size()][legislators.size()];
        for(int index_row = 0; index_row < legislators.size(); index_row++){
            for(int legislator_iterator = 0; legislator_iterator < legislators
                .size(); legislator_iterator++){
                adj_matrix[index_row][legislator_iterator] = 0;
                for(int index_col = 0; index_col < legislators.get(0).
                    getFeatures().size(); index_col++){
                    if((legislators.get(index_row).getFeature(
                        index_col).equals(legislators.get(
                            legislator_iterator).getFeature(index_col)))
                        && (legislator_iterator != index_row)){
                        adj_matrix[index_row][legislator_iterator]
                            += 1;
                    }
                }
            }
        }
        return adj_matrix;
    }

    public int [][] construct_adjacency_matrix_chosen(ArrayList<Legislator> legislators ,
        ArrayList<Integer> feature_column){
        int [][] adj_matrix = new int[legislators.size()][legislators.size()];
        for(int index_row = 0; index_row < legislators.size(); index_row++){
            for(int legislator_iterator = index_row; legislator_iterator < legislators
                .size(); legislator_iterator++){
                adj_matrix[index_row][legislator_iterator] = 0;
                for(int feature_iterator = 0; feature_iterator < feature_column.
                    size(); feature_iterator++){
                    if(legislators.get(index_row).getFeature(feature_column.
                        get(feature_iterator)).equals(legislators.get(
                            legislator_iterator).getFeature(feature_column.get(
                                feature_iterator))) && (legislator_iterator !=
                                index_row)){
                        adj_matrix[index_row][legislator_iterator] += 1;
                        adj_matrix[legislator_iterator][index_row] += 1;
                    }
                }
            }
        }
        return adj_matrix;
    }

    public void printAdj(int [][] adj_matrix){
        for(int index_row = 0; index_row < adj_matrix.length; index_row++){
            for(int index_col = 0; index_col < adj_matrix[1].length; index_col++){
                System.out.print(" " + adj_matrix[index_row][index_col]);
            }
            System.out.print("\n");
        }
    }

    public String graph_exp(int [][] adj_matrix){
        StringBuilder sb = new StringBuilder();
        //int index_col = 0;
        //int row_length = adj

        for(int index_row = 0; index_row < adj_matrix.length; index_row++){
            for(int index_col = index_row; index_col < adj_matrix[1].length; index_col
                ++){
                if((adj_matrix[index_row][index_col] != 0) ){
                    sb.append((index_row + 1) + " " + (index_col + 1) + ",");
                }
            }
        }
        String expression = sb.toString();
        return expression.substring(0,expression.length()-1);
    }
}

```

Legislator.java

```

import java.util.ArrayList;

public class Legislator {

```

```

    String name;
    ArrayList<String> features = new ArrayList<String>();
    public Legislator(){
    }

    public void setName(String i_name){
        name = i_name;
    }

    public String getName(){
        return name;
    }

    public void setFeature(ArrayList<String> i_features){
        features = i_features;
    }

    public ArrayList<String> getFeatures(){
        return features;
    }

    public String getFeature(int index){
        return features.get(index);
    }
}

```

MWCP_Approx_Algorithm.java

```

import java.util.Arrays;
import java.util.stream.IntStream;
import java.util.ArrayList;
//import org.apache.commons.lang3.ArrayUtils;

public class MWCP_Approx_Algorithm {
    public MWCP_Approx_Algorithm(){
    }
    ArrayList<Integer> final_weights = new ArrayList<Integer>();
    public ArrayList<Legislator[]> approx_algorithm(ArrayList<Legislator> legislators ,int [][]
        adj_matrix1 , int clique_size){
        ArrayList<Legislator[]> results = new ArrayList<Legislator[]>();
        ArrayList<int[]> keys = new ArrayList<int[]>();
        ArrayList<Integer> sum_buffer = new ArrayList<Integer>();
        ArrayList<Integer> max_int = new ArrayList<Integer>();
        ArrayList<Integer> sum_int = new ArrayList<Integer>();
        //ArrayList<Integer> keys_list = new ArrayList<Integer>();
        //System.out.println("Comm size: " + number_of_communities);
        Integer[] sum_array = new Integer[adj_matrix1.length];
        int[] key_array = new int[clique_size];
        int[][] adj_buffer = new int[adj_matrix1.length][adj_matrix1.length];

        for(int column = 0; column < adj_matrix1.length; column++){
            for(int row = 0; row < adj_matrix1[column].length; row++){
                adj_buffer[row][column] = adj_matrix1[row][column];
            }
        }

        //Sort each row in the adjacency matrix then get the desired number of vertices
        System.out.println("Adjacency Matrix Sorted");
        for(int clique_iterator = 0; clique_iterator < adj_matrix1[0].length;
            clique_iterator++){
            //adj_buffer = adj_matrix[clique_iterator];
            Arrays.sort(adj_buffer[clique_iterator]);
            int[] row_buffer = new int[clique_size];
            for(int vertex_iterator = 0; vertex_iterator < clique_size;
                vertex_iterator++){
                row_buffer[vertex_iterator] = adj_buffer[clique_iterator][
                    adj_buffer[0].length - vertex_iterator - 1];
            }
            keys.add(row_buffer);
        }

        System.out.println("Adjacency Matrix Sorted");
        System.out.println("Compute array sum Done!");
        //Get the sum of each array and store it on an arraylist for index reference , and
        //integer array for sorting
        for(int sum_buffer_iterator = 0; sum_buffer_iterator < keys.size();
            sum_buffer_iterator++){
            sum_buffer.add(IntStream.of(keys.get(sum_buffer_iterator)).sum());
            sum_array[sum_buffer_iterator] = IntStream.of(keys.get(sum_buffer_iterator)
                ).sum();
        }

        //Sort the sum array
        Arrays.parallelSort(sum_array);
    }
}

```

```

System.out.println("Get all index of the communities done!");
//Get the all index of the /number_of_communities/ largest sum in the array
/*for(int sum_buffer_iterator = 0; sum_buffer_iterator < number_of_communities;
    sum_buffer_iterator++){
    if (!(max_int.contains(getIndexOfAll(sum_array[sum_array.length -
        sum_buffer_iterator - 1], sum_buffer).get(0)))){
        max_int.addAll(getIndexOfAll(sum_array[sum_array.length -
            sum_buffer_iterator - 1], sum_buffer));
    }
}

}*/
System.out.println("Test subject done!* max_int size: " + max_int.size());
System.out.println("sum arr length: " + sum_array.length);

int sum_buffer_iterator = 0;
while((sum_buffer_iterator < sum_array.length)){
    if (!(max_int.contains(getIndexOfAll(sum_array[sum_array.length -
        sum_buffer_iterator - 1], sum_buffer).get(0)))){
        if(sum_array[sum_array.length - sum_buffer_iterator - 1] !=0){
            //System.out.println("Sum array value added: " + sum_array
                [sum_array.length - sum_buffer_iterator - 1]);
            max_int.addAll(getIndexOfAll(sum_array[sum_array.length -
                sum_buffer_iterator - 1], sum_buffer));
        }
    }
    //System.out.println("Sum buff: " + sum_buffer_iterator);
    sum_buffer_iterator++;
}

System.out.println("Test subject done! max_int size: " + max_int.size());
//max_int = limitCommunities(max_int, number_of_communities);

//System.out.println("Test subject done! max_int size: " + max_int.size());

for(int sum_counter = 0; sum_counter < max_int.size(); sum_counter++){
    Legislator[] max_legislators = new Legislator[clique_size];
    ArrayList<Integer> indexes = new ArrayList<Integer>();
    boolean hasZero = false;
    key_array = keys.get(max_int.get(sum_counter));

    for(int vertex_iterator = 0; vertex_iterator < clique_size;
        vertex_iterator++){
        System.out.print(key_array[vertex_iterator] + " ");
        if(key_array[vertex_iterator] == 0){
            hasZero = true;
        }
    }

    System.out.println("");
    for(int vertex_iterator = 0; vertex_iterator < clique_size;
        vertex_iterator++){
        if(indexes.containsAll(getIndexOfAll(key_array[vertex_iterator],
            intToList(adj_matrix1[max_int.get(sum_counter)]))) == false
            ){
            indexes.addAll(getIndexOfAll(key_array[vertex_iterator], intToList
                (adj_matrix1[max_int.get(sum_counter)])));
            //System.out.print(adj_matrix1[sum_counter][vertex_iterator]);
        }
    }
    indexes = limitCommunities(indexes, clique_size);
    for(int index_iterator = 0; index_iterator < clique_size; index_iterator
        ++){
        //System.out.print(indexes.get(index_iterator) + " ");
        max_legislators[index_iterator] = legislators.get(indexes.get(
            index_iterator));
    }
    if(hasZero == false){
        final_weights.add(sum_array[sum_array.length - sum_counter - 1]);
        results.add(max_legislators);
    }
    System.out.println("");
    //for(int leg_iter = 0; leg_iter < results.get(sum_counter).length;
        leg_iter++){
        //System.out.print(results.get(sum_counter)[leg_iter].getName() +
            " ");
    }
}

return results;
}

public ArrayList<Integer> getFinalWeights(){
    return final_weights;
}
}

```



```

public ArrayList<Integer> getIndexOfAll(int int_max, ArrayList<Integer> sum_buffer){
    ArrayList<Integer> indexes = new ArrayList<Integer>();
    for(int buffer_iterator = 0; buffer_iterator < sum_buffer.size(); buffer_iterator
        ++){
        if(int_max == sum_buffer.get(buffer_iterator)){
            indexes.add(buffer_iterator);
        }
    }
    return indexes;
}

public ArrayList<Integer> intToList(int [] int_arr){
    ArrayList<Integer> int_list = new ArrayList<Integer>();
    for(int iterator = 0; iterator < int_arr.length; iterator++){
        int_list.add(int_arr[iterator]);
    }
    return int_list;
}

public ArrayList<Integer> limitCommunities(ArrayList<Integer> index_list, int
    number_of_communities){
    for(int trim_iterator =index_list.size(); trim_iterator > number_of_communities;
        trim_iterator--){
        index_list.remove(trim_iterator - 1);
    }
    return index_list;
}
}

```

MWCP_Exact_GPU.java

```

import static jcuda.driver.JCudaDriver.cuCtxCreate;
import static jcuda.driver.JCudaDriver.cuCtxSynchronize;
import static jcuda.driver.JCudaDriver.cuDeviceGet;
import static jcuda.driver.JCudaDriver.cuInit;
import static jcuda.driver.JCudaDriver.cuLaunchKernel;
import static jcuda.driver.JCudaDriver.cuMemAlloc;
import static jcuda.driver.JCudaDriver.cuMemFree;
import static jcuda.driver.JCudaDriver.cuMemcpyDtoH;
import static jcuda.driver.JCudaDriver.cuMemcpyHtoD;
import static jcuda.driver.JCudaDriver.cuModuleGetFunction;
import static jcuda.driver.JCudaDriver.cuModuleLoad;
import jcuda.runtime.JCuda;
import jcuda.driver.*;
import static jcuda.driver.CUdevice_attribute.*;
import static jcuda.driver.JCudaDriver.*;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

import jcuda.Pointer;
import jcuda.Sizeof;
import jcuda.driver.CUcontext;
import jcuda.driver.CUdevice;
import jcuda.driver.CUdeviceptr;
import jcuda.driver.CUfunction;
import jcuda.driver.CUmodule;
import jcuda.driver.JCudaDriver;

public class MWCP_Exact_GPU {
    private static int number_of_indices;
    private static int [][] adj_matrix;
    private static int clique_size;
    private static int num_of_communities;
    private static ArrayList<Legislator> legislators;
    static ArrayList<Integer> outputIndices = new ArrayList<Integer>();
    Comparator<queueElement> com = new LegislatorComparator();
    PriorityQueue<queueElement> queue = new PriorityQueue<queueElement>(com);
    ArrayList<Integer> final_weights = new ArrayList<Integer>();
    public MWCP_Exact_GPU(){
    }

    public MWCP_Exact_GPU(ArrayList<Legislator> legislators, int [][] adj_matrix, int
        clique_size, int num_of_communities){
        this.legislators = legislators;
        this.number_of_indices = number_of_indices;
        this.adj_matrix = adj_matrix;
    }
}

```

```

        this.clique_size = clique_size;
        this.num_of_communities = num_of_communities;
    }

    public static int[] setIndexArray(int n){
        int[] index_array = new int[n];
        for(int iter = 0; iter < n; iter++){
            index_array[iter] = iter;
        }

        return index_array;
    }
    public static int getIndexLen(){
        return legislators.size();
    }

    public static int getCliqueSize(){
        return clique_size;
    }

    public static int getCommNum(){
        return num_of_communities;
    }

    public static int computeSum(int[] indices){
        int totalSum = 0;
        for(int iter = 0; iter < indices.length - 1; iter++){
            totalSum += adj_matrix[indices[iter]][indices[iter+1]];
        }

        return totalSum;
    }

    public static int getNumOfGPUAvailable() {
        JCudaDriver.setExceptionsEnabled(true);
        cuInit(0);
        // Obtain the number of devices
        int deviceCountArray[] = { 0 };
        cuDeviceGetCount(deviceCountArray);
        int deviceCount = deviceCountArray[0];
        System.out.println("Found " + deviceCount + " devices.");
        return deviceCount;
    }

    public void runGPUAlgorithm() throws IOException {
        // Enable exceptions and omit all subsequent error checks
        JCudaDriver.setExceptionsEnabled(true);

        // Initialize the driver and create a context for the first device
        cuInit(0);
        CUdevice device = new CUdevice();
        cuDeviceGet(device, 0);
        CUcontext context = new CUcontext();
        cuCtxCreate(context, 0, device);

        // Create the PTX file by calling the NVCC
        //String ptxFileName = preparePtxFile("com.cu");
        byte ptxData[] = toByteArray(getClass().getResourceAsStream("/com.ptx"));
        //String ptxFileName = preparePtxFile("combination.cu");

        // Load the ptx file
        CUmodule module = new CUmodule();
        cuModuleLoadData(module, ptxData);
        //cuModuleLoad(module, ptxFileName);

        // Obtain a function pointer to the "combination" algorithm function
        CUfunction function = new CUfunction();
        cuModuleGetFunction(function, module, "combination");

        // Allocate the host input data
        int[] index_array = setIndexArray(getIndexLen());
        int numOfIndices = getIndexLen();
        int cliqueSize = getCliqueSize();
        int number_comm = getCommNum();
        int outputSize = 99000000;
        int[] data_res = new int[getCommNum()];

        System.out.println("*****");
        System.out.println("Host input indices size: " + numOfIndices);
        //System.out.println("Host input indices of base gene clusters: " +
            indicesOfBaseGeneClusters.length);
        System.out.println("Host output size: " + outputSize);
        System.out.println("*****");

        // Allocate the device input data, and copy the host input data to the device

```

```

CUdeviceptr dev_indices = new CUdeviceptr();
cuMemAlloc(dev_indices, numOFIndices * Sizeof.INT);
cuMemcpyHtoD(dev_indices, Pointer.to(index_array), numOFIndices * Sizeof.INT);

CUdeviceptr dev_res = new CUdeviceptr();
cuMemAlloc(dev_res, cliqueSize * Sizeof.INT);
cuMemcpyHtoD(dev_res, Pointer.to(data_res), cliqueSize * Sizeof.INT);

// Allocate device output memory
CUdeviceptr dev_output = new CUdeviceptr();
cuMemAlloc(dev_output, outputSize * Sizeof.INT);

// Set up the kernel parameters: A pointer to an array of pointers which point to
// the actual values
Pointer kernelParameters = Pointer.to(
    Pointer.to(dev_indices),
    Pointer.to(new int [] { numOFIndices } ),
    Pointer.to(new int [] { cliqueSize } ),
    Pointer.to(dev_res),
    Pointer.to(dev_output)
);

// Call the kernel function
int blockSizeX = 1;
int gridSizeX = 1;
//int blockSizeX = 256;
//int gridSizeX = (int)Math.ceil((double)numOfElements/blockSizeX);
cuLaunchKernel(function,
    gridSizeX, 1, 1, // Grid dimension
    blockSizeX, 1, 1, // Block dimension
    0, null, // Shared memory size and
    stream
    kernelParameters, null // Kernel and extra parameters
);
cuCtxSynchronize();

// Allocate host output memory and copy the device output to the host
int [] output = new int [outputSize];
cuMemcpyDtoH(Pointer.to(output), dev_output, outputSize * Sizeof.FLOAT);

// Print results
System.out.println(" Results: ");
for(int i = 0; i < output.length; i++) {
    if(output[i] == 0 && output[i+1] == 0){
        break;
    }
    if(i%cliqueSize == 0) {
        //System.out.println();
    }

    outputIndices.add(output[i]);
    //System.out.print(output[i] + " ");
}

//outputIndices = output;

// Clean up
cuMemFree(dev_indices);
cuMemFree(dev_output);
cuMemFree(dev_res);
}

public ArrayList<Legislator []> throwResults(){
//Legislator [] clique = new Legislator [getCliqueSize ()];
ArrayList<Legislator []> result = new ArrayList<Legislator []> ();
ArrayList<int []> index_arr = new ArrayList<int []> ();
for(int iter = 0; iter < outputIndices.size (); iter+=getCliqueSize ()) {
    int [] data_res = new int [getCliqueSize ()];
    for(int iter2 = iter; iter2 < iter+getCliqueSize (); iter2++){
        data_res [iter2 - iter] = outputIndices.get (iter2);
        //System.out.print (outputIndices.get (iter2) + " ");
    }
    index_arr.add (data_res);
    //System.out.println (index_arr.get (iter/getCliqueSize ()) [0] + " " +
        index_arr.get (iter/getCliqueSize ()) [1]);
}

for(int queueIter = 0; queueIter < index_arr.size (); queueIter++){
    queueElement dataQ = new queueElement (index_arr.get (queueIter));
    int curr_sum = dataQ.getSum ();
    //System.out.println (dataQ.getIndex () [0] + " " + dataQ.getIndex () [1] + "*"
        + curr_sum);

    /*if (queue.size () < getCommNum ()) {
        queue.add (dataQ);
    } else {
        if (dataQ.getSum () > queue.peek ().getSum ()) {
            System.out.println ("Head removed, new Combination added");
        }
    }
}

```

```

        queue.remove();
        queue.add(dataQ);
    }
}*/
if(dataQ.hasZero == false){
    queue.add(dataQ);
}

}

for(queueElement q : queue){
    //System.out.println(q.getIndex()[0] + " "+q.getIndex()[1]);
}

/*for(int iter = 0; iter < outputIndices.size() - 1; iter += getCliqueSize()){
    int[] data_res = new int[getCliqueSize()];
    for(int iter2 = iter; iter2 < iter +getCliqueSize() - 1; iter2++){
        data_res[iter2 - iter] = outputIndices.get(iter2);
        System.out.print(data_res[iter2 - iter] + " ");
    }*/

/*queueElement dataQ = new queueElement(data_res2);
int curr_sum = dataQ.getSum();
System.out.println(" Current sum: " + curr_sum + " ");
if(queue.size() < getCommNum()){
    queue.add(dataQ);
} else{
    if(dataQ.getSum() > queue.peek().getSum()){
        System.out.println(" Head removed, new Combination added");
        for(int iterR = 0; iterR < queue.element().getIndex().length; iterR++){
            System.out.print(queue.element().getIndex()[iterR] + " ");
        }
        queue.remove();
        queue.add(dataQ);
    }
}*/

while(queue.isEmpty() == false){
    Legislator[] leg_list = new Legislator[queue.element().getIndex().length];
    for(int iterR = 0; iterR < queue.element().getIndex().length; iterR++){
        //System.out.print(queue.element().getIndex()[iterR] + " ");
        leg_list[iterR] = legislators.get(queue.element().getIndex()[iterR]);
    }
    final_weights.add(queue.element().getSum());
    result.add(leg_list);
    System.out.println(" ");
    queue.remove();
}

return result;
}

public ArrayList<Integer> getFinalWeights(){
    return final_weights;
}
public int getSumOfCom(int[] dataArr){
    int totalSum = 0;
    for(int iterator = 0; iterator < dataArr.length - 1; iterator++){
        totalSum += this.adj_matrix[dataArr[iterator]][dataArr[iterator+1]];
    }
    totalSum = totalSum + this.adj_matrix[dataArr[0]][dataArr[dataArr.length - 1]];
    return totalSum;
}

public boolean hasZeroAdj(int[] dataArr){
    for(int iterator = 0; iterator < dataArr.length - 1; iterator++){
        if(this.adj_matrix[dataArr[iterator]][dataArr[iterator+1]] == 0){
            return true;
        }
    }

return false;
}

public class LegislatorComparator implements Comparator<queueElement>{

    public int compare(queueElement x, queueElement y) {
        // TODO Auto-generated method stub
        return Integer.compare(x.getSum(), y.getSum());
    }
}

```

```

        //return 0;
    }
}

class queueElement{
    int [] index;
    int sum;
    boolean hasZero;

    public queueElement(int [] index){
        this.index = index;
        this.sum = getSumOfCom(index);
        this.hasZero = hasZeroAdj(index);
    }

    public int [] getIndex(){
        return this.index;
    }

    public int getSum(){
        return this.sum;
    }
}

/**
 * The extension of the given file name is replaced with "ptx".
 * If the file with the resulting name does not exist, it is
 * compiled from the given file using NVCC. The name of the
 * PTX file is returned.
 *
 * @param cuFileName The name of the .CU file
 * @return The name of the PTX file
 * @throws IOException If an I/O error occurs
 */
private String preparePtxFile(String cuFileName) throws IOException {
    int endIndex = cuFileName.lastIndexOf('.');
    if (endIndex == -1) {
        endIndex = cuFileName.length() - 1;
    }
    String ptxFileName = cuFileName.substring(0, endIndex+1)+"ptx";
    File ptxFile = new File(ptxFileName);
    if (ptxFile.exists()) {
        return ptxFileName;
    }

    File cuFile = new File(cuFileName);
    System.out.println(cuFile.getAbsolutePath());
    if (!cuFile.exists()) {
        throw new IOException("Input file not found: "+cuFileName);
    }
    String modelString = "-m" + System.getProperty("sun.arch.data.model");
    String command = "nvcc " + modelString + " -ptx " + cuFile.getPath() + " -o " +
        ptxFileName;

    System.out.println("Executing\n"+command);
    Process process = Runtime.getRuntime().exec(command);

    String errorMessage = new String(toByteArray(process.getErrorStream()));
    String outputMessage = new String(toByteArray(process.getInputStream()));
    int exitValue = 0;
    try {
        exitValue = process.waitFor();
    }
    catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        throw new IOException(
            "Interrupted while waiting for nvcc output", e);
    }

    if (exitValue != 0) {
        System.out.println("nvcc process exitValue "+exitValue);
        System.out.println("errorMessage:\n"+errorMessage);
        System.out.println("outputMessage:\n"+outputMessage);
        throw new IOException(
            "Could not create .ptx file: "+errorMessage);
    }

    System.out.println("Finished creating PTX file");
    return ptxFileName;
}

/**
 * Fully reads the given InputStream and returns it as a byte array
 *
 * @param inputStream The input stream to read
 * @return The byte array containing the data from the input stream
 * @throws IOException If an I/O error occurs

```

```

*/

private static byte[] toByteArray(InputStream inputStream) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    byte buffer[] = new byte[8192];
    while (true) {
        int read = inputStream.read(buffer);
        if (read == -1)
            break;
        baos.write(buffer, 0, read);
    }
    baos.write(0);
    return baos.toByteArray();
}
}

```

PriorityExact.java

```

import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

public class PriorityExact {

    Comparator<queueElement> com = new LegislatorComparator();
    int counter = 0;
    PriorityQueue<queueElement> queue = new PriorityQueue<queueElement>(com);
    ArrayList<Integer> final_weights = new ArrayList<Integer>();
    //ArrayList<queueElement> test = new ArrayList<queueElement>();
    private int [][] adj_matrix;

    public PriorityExact(int [][] adj_matrix, int number_of_communities){
        this.adj_matrix = adj_matrix;
        //queue.clear();
        //this.queue = new PriorityQueue<queueElement>(com);
        //this.test = new ArrayList<queueElement>();
    }

    public ArrayList<Legislator[]> exact_algorithm(ArrayList<Legislator> legislators, int [][]
        adj_matrix1, int clique_size, int number_of_communities){
        ArrayList<Legislator[]> results = new ArrayList<Legislator[]>();
        System.out.println("Matrix dim: " + adj_matrix.length + " by " + adj_matrix[1].
            length);
        int[] data_res = new int[clique_size];
        int[] testt = {1,2,3,4};
        queueElement tester = new queueElement(testt);

        performCombination(legislators, data_res, 0, legislators.size() - 1, 0,
            clique_size, number_of_communities);

        System.out.println("Queue size: " + queue.size());
        System.out.println("Queue entries: ");
        while(queue.isEmpty() == false){
            Legislator[] leg_list = new Legislator[queue.element().getIndex().length];
            for(int iterR = 0; iterR < queue.element().getIndex().length; iterR++){
                //System.out.print(queue.element().getIndex()[iterR] + " ");
                leg_list[iterR] = legislators.get(queue.element().getIndex()[iterR]
                    );
            }
            results.add(leg_list);
            final_weights.add(queue.element().getSum());
            System.out.println(" ");
            queue.remove();
        }

        return results;
    }

    /*
    *
    *
    *
    *
    */
    public ArrayList<Integer> getFinalWeights(){
        return final_weights;
    }

    public void performCombination(ArrayList<Legislator> legislators, int[] data_res, int
        start_index, int endIndex, int index, int clique_size, int num.com){
        if(index == clique_size){
            counter++;
            queueElement dataQ = new queueElement(data_res);

```

```

        System.out.println(" Combination " + counter + " solved!");
        int curr_sum = dataQ.getSum();
        System.out.println(" Current sum: " + curr_sum + " ");
        /*if (queue.size() < num.com){
            queue.add(dataQ);
        }else{
            if (dataQ.getSum() > queue.peek().getSum()){
                System.out.println("Head removed, new Combination added");
                queue.remove();
                queue.add(dataQ);
            }
        }*/

        if (dataQ.hasZero == false){
            queue.add(dataQ);
        }

        return;
    }
    for (int curr_index = start_index; ((curr_index <= endIndex) && ((endIndex -
        curr_index + 1) >= (clique_size - index))); curr_index++){
        int [] data_res2 = new int [clique_size];
        for (int dataIter = 0; dataIter < data_res2.length; dataIter++){
            data_res2[dataIter] = data.res[dataIter];
        }
        data_res2[index] = legislators.indexOf(legislators.get(curr_index));
        performCombination(legislators, data_res2, curr_index + 1, endIndex, index
            + 1, clique_size, num.com);
    }

}

public int getSumOfCom(int [] dataArr){
    int totalSum = 0;
    for (int iterator = 0; iterator < dataArr.length - 1; iterator++){
        totalSum += this.adj_matrix[dataArr[iterator]][dataArr[iterator+1]];
    }

    return totalSum;
}

public boolean hasZeroAdj(int [] dataArr){
    for (int iterator = 0; iterator < dataArr.length - 1; iterator++){
        if (this.adj_matrix[dataArr[iterator]][dataArr[iterator+1]] == 0){
            return true;
        }
    }

    return false;
}

public class LegislatorComparator implements Comparator<queueElement>{

    public int compare(queueElement x, queueElement y) {
        // TODO Auto-generated method stub
        return Integer.compare(x.getSum(), y.getSum());
        //return 0;
    }

}

class queueElement{
    int [] index;
    int sum;
    boolean hasZero;

    public queueElement(int [] index){
        this.index = index;
        this.sum = getSumOfCom(index);
        this.hasZero = hasZeroAdj(index);
    }

    public int [] getIndex(){
        return this.index;
    }

    public int getSum(){
        return this.sum;
    }

}

```

```
}
```

R_Integration_RSERVE.java

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.stream.IntStream;

import org.rosuda.REngine.*;
import org.rosuda.REngine.Rserve.*;

public class R_Integration_RSERVE {
    ArrayList<Legislator> toThrow_Legislator = new ArrayList<Legislator>();
    ArrayList<Integer> result_weights = new ArrayList<Integer>();
    int [][] toThrow_adj;
    int clique_size, number_of_communities;
    String [] featureList;
    String filePath;
    ArrayList<Legislator []> toThrow_community = new ArrayList<Legislator []>();
    ArrayList<Legislator> matrix_L = new ArrayList<Legislator>();
    RConnection c;
    REXP eval_file;
    int [] degree centrality;
    int [] degreeW_centrality;
    double [] closeness_centrality;
    double [] betweenness_centrality;
    int number_of_devices;
    public R_Integration_RSERVE() throws RserveException, REXPMismatchException{
        c = new RConnection();
    }

    public void r_executeConnection(String filePath) throws RserveException,
        REXPMismatchException{
        try {
            // make a new local connection on default port (6311)
            int delimit = filePath.lastIndexOf(".");
            REXP x0 = c.eval("library('xlsx')");
            REXP x_igraph = c.eval("library(igraph)");
            if(filePath.substring(delimit, filePath.length()).equals(".xlsx")){
                eval_file = c.eval("df <-read.xlsx('" + filePath + "', sheetName = 'Sheet1',
                    header = TRUE)");
            }else if(filePath.substring(delimit, filePath.length()).equals(".xls")){
                eval_file = c.eval("df <-read.xls('" + filePath + "', sheet = 'Sheet1')");
            }else if(filePath.substring(delimit, filePath.length()).equals(".csv")){
                eval_file = c.eval("df <-read.csv('" + filePath + "', header = TRUE, sep = ',')")
            }
            ;

            REXP exp_nrow = c.eval("nrow(df)");
            REXP exp_ncol = c.eval("ncol(df)");
            REXP exp_headers = c.eval("colnames(df)");
            featureList = exp_headers.asStrings();
            for(String header : featureList){
                System.out.println(header);
            }
            int nrow = exp_nrow.asInteger();
            int ncol = exp_ncol.asInteger();
            //Parse the Spreadsheet file into an ArrayList of Legislators
            for(int index_row = 1; index_row <= nrow; index_row++){
                Legislator legislator = new Legislator();
                ArrayList<String> features = new ArrayList<String>();
                for(int index_col = 1; index_col <= ncol; index_col++){
                    REXP exp_element = c.eval("df['" + index_row + "', " + index_col + "']");
                    if(index_col == 1){
                        legislator.setName(exp_element.asString());
                    }
                    else{
                        features.add(exp_element.asString());
                    }
                }
                legislator.setFeature(features);
                matrix_L.add(legislator);
            }

            toThrow_Legislator = matrix_L;
        } catch (REngineException e) {
            e.printStackTrace();
        }
    }

    public void r_execute(String Algo, int [] feature_column ) throws RserveException,
        REXPMismatchException{
        Graph_Construction g = new Graph_Construction();
        MWCP_Approx_Algorithm m = new MWCP_Approx_Algorithm();
        MWCP_Exact_GPU egpu = new MWCP_Exact_GPU();
    }
}
```



```

PriorityExact e;
ArrayList<Integer> feature_col = new ArrayList<Integer>();
for(int i : feature_column){
    feature_col.add(i);
}
int [][] adj_matrix;
adj_matrix = g.construct_adjacency_matrix_chosen(matrix_L, feature_col);
toThrow_adj = adj_matrix;
//g.printAdj(adj_matrix);
System.out.println("Number of communities " + number_of_communities);
System.out.println("Algorithm Chosen: " + Algo);
egpu = new MWCP_Exact_GPU(matrix_L, adj_matrix, clique_size, number_of_communities);
if(Algo.equals("Approximate Algorithm")){
    toThrow_community = m.approx_algorithm(matrix_L, adj_matrix, clique_size);
    result_weights = m.getFinalWeights();
} else if(Algo.equals("Exact Algorithm(GPU)")){
    try {
        egpu.runGPUAlgorithm();
        toThrow_community = egpu.throwResults();
        Collections.reverse(toThrow_community);
        result_weights = egpu.getFinalWeights();
        Collections.reverse(result_weights);
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
} else if(Algo.equals("Exact Algorithm(CPU)")){
    e = new PriorityExact(adj_matrix, number_of_communities);
    toThrow_community = e.exact_algorithm(matrix_L, adj_matrix, clique_size,
        number_of_communities);
    Collections.reverse(toThrow_community);
    result_weights = (e.getFinalWeights());
    Collections.reverse(result_weights);
}
REXP exp_name_graph = c.eval("g <- graph(c("+ g.graph_exp(adj_matrix) +"), n = " +
    adj_matrix[0].length+ ", directed = F)");
REXP exp_degree = c.eval("centr_degree(g, mode = 'all', normalized = T)$res");
REXP exp_closeness = c.eval("centr_clo(g, mode = 'all', normalized = T)$res");
REXP exp_betweenness = c.eval("centr_betw(g, directed = F, normalized = T)$res");
degree_centrality = exp_degree.asIntegers();
degreeW_centrality = computeWeightedCentrality(adj_matrix);
closeness_centrality = exp_closeness.asDoubles();
betweenness_centrality = exp_betweenness.asDoubles();
}

public ArrayList<Integer> getResultWeights(){
    return result_weights;
}

public int [] computeWeightedCentrality(int [][] temp_adj){
    int [] degreeW = new int [temp_adj.length];
    for(int iter = 0; iter < temp_adj.length; iter++){
        degreeW[iter] = IntStream.of(temp_adj[iter]).sum();
    }
    return degreeW;
}

public ArrayList<Legislator> getLegislators(){
    return toThrow_Legislator;
}

public int [][] getAdj(){
    return toThrow_adj;
}

public ArrayList<Integer> addIntToList(String [] toInt){
    ArrayList<Integer> featureIndex = new ArrayList<Integer>();
    for(String s : toInt){
        featureIndex.add(Integer.parseInt(s));
    }
    return featureIndex;
}

public String [] getFeatureList(){
    return Arrays.copyOfRange(featureList, 1, featureList.length);
}

public void setNumberOfCommunities(int communityNumber){
    this.number_of_communities = communityNumber;
}

public int getCliqueSize(){
    return clique_size;
}

```

```

    }

    public int getNumberOfCommunities(){
        return number_of_communities;
    }

    public void setCliqueSize(int cliqueSize){
        clique_size = cliqueSize;
    }

    public ArrayList<Legislator[]> getResults(){
        return toThrow_community;
    }
    public void setFilePath(String filepath){
        filePath = filepath;
        //System.out.println("End my misery");
    }

    public int [] getDegCentrality(){
        return degree_centrality;
    }

    public double [] getBetCentrality(){
        return betweenness_centrality;
    }

    public double [] getCloCentrality(){
        return closeness_centrality;
    }

    public int [] getDegWeighted(){
        return degreeW_centrality;
    }
}

```

ReportMaker.java

```

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Date;

import com.itextpdf.text.BadElementException;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Image;
import com.itextpdf.text.List;
import com.itextpdf.text.ListItem;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
public class ReportMaker {
    private String filename;
    private String username;
    private String [] headers;
    private String [][] data_centrality;

    public ReportMaker(String filename, String username, String [] headers, String [][]
        data_centrality){
        this.filename = filename;
        this.username = username;
        this.headers = headers;
        this.data_centrality = data_centrality;
    }

    public void makeReport(String text){
        Document report = new Document();
        try {
            PdfWriter.getInstance(report, new FileOutputStream(this.filename));
        } catch (FileNotFoundException | DocumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        report.open();
        initializeMetadata(report);
        addTitlePage(report, text);
        //addImages(report, null);
        report.close();
    }

    private void initializeMetadata(Document pdf){

```

```

        pdf.addTitle("Legislator Clique Finder Results");
        pdf.addAuthor(this.username);
        pdf.addCreator(this.username);
    }

    private void addTitlePage(Document pdf, String text){
        Font largeFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 16, Font.NORMAL);
        Font mediumFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 14, Font.NORMAL);
        Font smallFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 10, Font.NORMAL);
        Paragraph preface = new Paragraph();
        try {
            Image img = Image.getInstance(getClass().getResource("/Image/logo.PNG"));
            float scaler = ((pdf.getPageSize().getWidth()-pdf.leftMargin()-pdf.
                rightMargin()) / img.getWidth()) * 100;
            img.scalePercent(scaler);
            pdf.add(img);
        } catch (BadElementException | IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (DocumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        addEmptyLine(preface, 2);
        preface.add(new Paragraph("Legislator Clique Finder Result",largeFont));
        addEmptyLine(preface, 1);
        preface.add(new Paragraph("Author: " + System.getProperty("user.name") + ", Date: " + new
            Date(),smallFont));

        Paragraph details = new Paragraph();
        addEmptyLine(details, 1);
        details.add(text);

        try {
            preface.setAlignment(Element.ALIGN_CENTER);
            pdf.add(preface);
            pdf.newPage();
            pdf.add(details);
            pdf.newPage();
            NumberFormat format = new DecimalFormat("##.##");
            PdfPTable table = new PdfPTable(headers.length);

            Paragraph title = new Paragraph();
            title.add(new Paragraph("Centrality Table",mediumFont));
            addEmptyLine(title,1);

            for(String header : headers){
                PdfPCell colHead = new PdfPCell(new Phrase(header,smallFont));
                colHead.setHorizontalAlignment(Element.ALIGN_CENTER);
                table.addCell(colHead);
            }
            table.setHeaderRows(1);

            for(int row=0; row<data_centrality.length; row++){
                table.addCell(new Phrase(data_centrality[row][0]));
                table.addCell(new Phrase(data_centrality[row][1]));
                table.addCell(new Phrase(data_centrality[row][2]));
                table.addCell(new Phrase(format.format(Double.parseDouble(
                    data_centrality[row][3]))));
                table.addCell(new Phrase(format.format(Double.parseDouble(
                    data_centrality[row][4]))));
                /*for(int col=0; col<data_centrality[row].length; col++){
                    table.addCell(new Phrase(format.format(data_centrality[row]
                        ][col]),smallFont));
                }*/
            }

            pdf.add(title);
            pdf.add(table);
        } catch (DocumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        pdf.newPage();
    }

    private void addEmptyLine(Paragraph paragraph, int number) {
        for (int i = 0; i < number; i++) {
            paragraph.add(new Paragraph(" "));
        }
    }
}

```

XI. Acknowledgement

Una sa lahat, gusto kong pasalamatang kung sinuman ang gumagabay sa itaas, pwede na rin yung nasa baba. Gusto ko ring pasalamatang iyong gumabay sa akin dito sa gitna, ang aking SP adviser, Mr. Geoff Solano. Sir, maraming salamat sa pagtitiwala simula noong una pa lamang, at sa walang-sawang pagbibigay gabay at liwanag sa pagtahak sa madalim at masukal na kagubatan ng SP. Maraming salamat sa hindi pagsuko lalo na sa aming kakulitan at minsa'y ka-tigasan ng ulo. Hindi mapapantayan ang pagmamahal na inyong ibinigay sa amin. Maraming salamat po, Sir. Nais ko ring pasalamatang ang mga propesyonal na aking hiningang gabay, Mr. Jalton Taguibao at Mr. Jeff Aborot. Maraming salamat po sa inyong tulong upang mabuo ang SP na ito. Salamat rin sa panel na nagpursiging kilatisin at pagandahin ang SP na ito.

Pangalawa, nais ko ring pasalamatang ang mga sumusunod:

1. Aking mga magulang. Para sa bawat baong ibinigay upang makapag-cafe ako upang matapos ang SP na ito. Sa walang-sawang paggabay sa landas ng buhay at pag tanggap sa kung anuman ang aking paroroonan. Maraming salamat. Para sa inyo ito.
2. Delwyn P. Mendoza. Para sa mga “gusto ko nang mamatay” moments. Para sa panahong pinagtagpo tayo ng tadhana. Para sa bawat tasa ng kapeng itinaob kasabay ang palpitation at anxiety attack. Rak. Lamonayan. Black. Gold lining.
3. Bengemin S. Uy. Para sa mga “ito pa rin yung mali mo dati ah?” moments. Para sa walang sawang pag-intindi sa mga kabaluktutan ng utak ko. Para sa pag mentor at pagdebug ng magulo kong SP. Clutch sir.
4. Princess V. Florendo. Para sa mga “tara inom” moments. Maging milktea man yan o alak. Para sa bawat “hassle pre” coding sessions at “tara cut.” Para sa bawat tansang binulsa at boteng itinumba. Tagay.

5. Bianca L. Silmaro. Para sa mga “pareho naman tayo pero bakit mali” moments. Para sa mga pinagsaluhang “CUDA_ERROR” na linya. Para sa bawat NBA finals na pinagpustahan. GPU pa more.
6. Adrian Sabado at Jed Reyes. Para sa mga nakatatandang walang-sawang nagpagalit sa aming mga nakababata. Sa pag pressure sa amin na gumraduate na kaagad. Para sa mga naudlot na gala. Salamat, daddies.
7. David Dobrik. Sa pagkain ng oras ko sa mga panahong dapat nagcocode ako ng SP, pinapanood ko lang lahat ng vlogs mo ng paulit-ulit. 420 Blaze it. Penge merchs.
8. Doon sa kanta ng One Ok Rock na “The Beginning”. Ito yung anthem ko pag alam kong oras na para patayin na naman ang sarili sa pagod, puyat, at gutom. 2 sems din ituu.
9. MCSU profs especially Ms. Hermie Monterde(Hi ma’am kelan na dinner ng tech staff hihi), Ms. Therese Basco(Para sa bawat hugs noong proposal at defense), Mr. Mong Llarenas(Sa bawat pangaasar, acads man yan o DPSM week), Mr. Marvin John Ignacio(Para sa lahat. Alam mo na yan. Sobra.), Ms. Perl Gasmien(Sa kwentuhan at balitaan ng love life hihi), Mr. Marbert John Marasigan(G na Sir, Challenger na itu), at Mr. John Althom Mendoza(Miss na kita, Sir. See you soon). Mahal q kau lahat.

Para sa mga kamay na kumapit, para sa mga bisig na yumakap, at para sa mga balikat na nagpasandal. Maraming salamat. Atin ‘to, papasok ‘to.

.A1D3300391.X.