Universitat Politècnica de Catalunya

Bachelors in Computer Science and Engineering

Graph and matrix algorithms for visualizing high dimensional data

Director:
Dr. Ricard Gavalda Mestre

Co-Director:
Dr. Marta Arias Vicente

Bachelors Thesis of :
Abhinav Shankaranarayanan Venkataraman

June 20,2016


UPC

To my Mother, Father, Professors and Friends. With a heart filled with Joy and Dedication, I attribute all my effort, work and knowledge to my professors Ricard Gavalda and Marta Arias and to Babaji at Gurudwara

Abstract

Motivated by the problem of understanding data from the medical domain, we consider algorithms for visually representing highly dimensional data so that "similar" entities appear close together. We will study, implement and compare several algorithms based on graph and on matrix representation of the data. The first kind are known as "community detection" algorithms, the second kind as "clustering" algorithms. The implementations should be robust, scalable, and provide a visually appealing representation of the main structures in the data.

Acknowledgement

First of all I would like to thank my university, SASTRA University, Thanjavur, TamilNadu, India for allowing me to work on the project at UPC Barcelona. I thank UPC Barcelona for accepting me as an intern and providing me the necessary infrastructures and administrative support.

I am grateful to Prof.Ricard and Prof.Marta, both of them believed in me that I was doing something interesting and gave me full freedom to try out several new methods, although it took away some time but they still encouraged me. We used to have quite long discussions which always cleared my doubts about the subject and enriched my thoughts. I also thank them for allowing me to participate and also giving me a chance and opportunity to take part in Hackathons, Conferences and Seminars. I thank them for providing me sufficient materials and excellent reference papers. I also thank them for the speedy mail replies too. I will always cherish every moment I spent with both my professors.

Special thanks to Alejandro Fonteboa for extending his help when I was trying to figure out on web framework.

I extend my gratitude to Prof.Dr.P.Swaminathan, Dean, School of Computing,SASTRA University for granting me this opportunity to work in this wonderful environment.

I have deep gratitude for Prof.Dr.A.Umamakeswari, Associate Dean, School of Computing, SASTRA University for supporting me in every effort I take to enrich myself.

I thank Prof.Dr.Sridharan M, Professor,School of Electrical and Electronics Engineering ,SASTRA University for his efforts in connecting to UPC Barcelona and motivating us in every manner possible.

I thank the management Prof.Dr.S.Vaidhyasubramanium,Dean Planning and development,SASTRA University for giving this opportunity to do the project aborad and Prof.Raja S.(Raja Sir), administrative office, Training and Placement, SASTRA University for helping me out in the process for semester abroad program.

I thank the Gurudwara Nanaksar Sahib,Barcelona for supporting us with a happy mind in times of needs, especially I would like to thank Babaji.

I thank my friends, class mates and lab mates along with lots of fun and happinees they gave me some great knowledge too.

Last but never the least , I thank my parents, V Shankaranarayanan and S Gomathi and family members for supporting me all the way.

Contents

Chapter 1

Introduction

In this section an entire overview of the full project is provided. We mention the context of the project we have studied and the goal of the project. We also provide the intended planning, economic estimate and sustainability of the work that has been done.

1.1 Context Of the Project

In the present day scenario, the modern science of algorithms and graph theory has brought significant advances to our understanding of complex data. Many complex systems are representable in the form of graphs. Graphs have time and again been used to represent real world networks. One of the most pertinent feature of graphs representing real system is community structures or otherwise known as clusters. Community can be defined as the organization of vertices in groups or clusters, with many edges joining the vertices of the same cluster and comparatively fewer vertices joining the vertices in another neighbouring cluster. Such communities form an independent compartment of a graph exhibiting similar role. Thus, community detection is the key for understanding the structure of complex graphs, and ultimately deduce information from them.

The networks and highly dimensional data that motivate this problem emerge from the healthcare domain, and particularly from the analysis of complex, chronic disease, which is the major cost factor in modern societies. In the current scenario, a patient does not have one disease but a set of diseases. For example a person with diabetes has a heart disease, kidney

disease, high blood pressure etc. This may vary between sexes, ages etc and thus is a very complex landscape to explore. Visualizing this landscape of diseases would help to analyse the source, the treatment and even the path way of research to be done. Thus, such a visualization would be helpful for the medical experts and health planner to understand the landscape of diseases much better.

This project is carried out within the LARCA research group at UPC [1]. More precisely, researchers within LARCA have in the last two years began collaborations with hospital and health agencies for the analysis of electronic healthcare records [EHR].In previous work within the group [21], they proposed to organize the information in EHR in the form of graphs and hypergraphs, which can then be navigated by experts and mined with graph and network theoretic tools.

Within the perspective of the LARCA project, two kinds of networks could be useful to study in this scenario: one in which nodes are patients and edges indicate their similarity, and another one in which nodes are diagnostics/diseases, and edges indicate their association in a population. Hence, we address this visualization of such high dimensional data using the algorithms and visualization technologies. An auxiliary goal of this project is to help the main LARCA project by investigating a few solutions that could later be incorporated to the main project. A few solutions that are possible to resolve the problem will be analysed and tests will be conducted. The project will also involve study of various algorithms and their respective analysis based on the quality and quantity of data using multiple appropriate experiments. Although, the project is motivated by the real high dimensional data it is not easy to get such data and hence would use simpler ones for testing the project.

1.2 Goal of the Project

The project is built with due recommendations from the directors of the project - Prof.Ricard Gavalda and Prof.Marta Arias. The project is inspired by medical domain and thus slides to the side of implementation which involves faster computation for better visualization. Hence, there are four facets or goals for the project which are enumerated as below:

[1]LARCA - Laboratory for Relational Algorithmics, Complexity and Learning. http://recerca.upc.edu/larca

1. The first objective of the project is to survey a few algorithms that aim in community finding keeping in mind that the input is from the medical domain

2. Next, to choose two algorithms that benefit the purpose of organizing graphs from medical domain and for the purpose of visualization.

3. Implement the algorithms and test the efficiency of the algorithm using variety of graphs.

4. Lastly but more importantly to build a Graphic User Interface (GUI) which enables visualization of the raw input on a web browser by drawing graphs.

1.3 Planning

Planning is essential component of any project. It helps to keep pace with the time. The total duration of the project is five months starting from early February 2016 to the end of June 2016. The following describes the tasks that were planned to be performed in the project.

1.3.1 Task Description

The tasks for the project have been subdivided into various task phases which are enumerated below :

- **Required knowledge acquisition**
  Necessary knowledge to understand the problem needs to be gained in order to deal with the original topic.In this phase we familiarize with the term of community detection, graph theory and understand some of the possible methods that are in practice to deal with the problem. Knowledge about a few visualization methods is also necessary to implement the visualization of the project.

- **Paper Analysis**
  Analysis of paper related to community detection and clustering algorithms over high dimensional graph data is done in this phase of the project. This phase is necessary to understand various functionalities

that the project deals with and to assist in the subsequent phases of the project.

- **Design and Implementation**
  The required functionalities are listed and implemented using a programming language. In this phase the methods of the project are designed and programmed using the chosen language for implementation. The implementation is done for both the community detection algorithm and for the visualization aspect of the project.

- **Testing I**
  In this phase the program is tested using generated test cases and errors are identified and corrected. Multiple recoding is done in this phase of the project In this phase we test the program in order to identify errors in the implementation. It includes the successive recoding.

- **Testing II**
  In this phase, tests are performed on the GUI to ensure the limits of GUI.

- **Report Writing**
  In this phase the report of the project is written.

1.4 Economic Budget

1.4.1 An Introduction to Economic Budget

Economic management is primarily based on an estimate of income and expenditure called as budget. Development of a sustainable budget leads to proper economic management of the project and thus is one of the most important phase of the project management. In this phase we aim at providing an estimate of the project budget and optimize the same. We look at the expenditure from various aspects such as software costs, hardware costs, license costs and human resource costs. Additionally we also account the software for its sustainability. One important factor to note is that the budget that we describe in this section is subject to change and it may increase depending on the unexpected obstacles that we may face. For an instance when we do not get the expected results with a particular software we may have to go in for another software that may incur extra installation and operational charges.

1.4.2 Estimation of Economic Budget

We divide the overall expenditure into three categories namely hardware, software and human resources. One very important factor that we need to consider is that we only get an estimate of the total cost. This may vary depending on the systems in use. To calculate the amortization we consider two factors namely, the overall life of the hardware or software in use and the duration of the project, which is 5 months. Hence the amortization cost turns out to be one eighth of the actual life of each component.

1.4.2.1 Hardware Budget

Hardware budget accounts for the actual and the amortized costs of the hardware elements used by the project. The cost is fictitious as it has not been developed commercially. Table 1. intents to estimate the economic cost of each of the hardware component of the project.

Table 1 - Hardware Budget

Sno:	Hardware Component	Useful Life(in years)	Total Cost(in €)	Amortized Cost(in €)
1	PC System	4	1000€	125 €
	Total		**1000€**	**125 €**

1.4.2.2 Software Budget

The software budget shows an estimate for the various software used in the project along with the estimate of the software costs. Similar to hardware the softwares also gets wornout in time. Maximum performance time is fixed for every software. In addition freeware software and open source software incur no cost. The cost is fictitious as it has not been developed commercially. Table 2 intents to estimate the economic cost of each of the software component of the project.

Table 2 - Software Budget				
Sno:	Software Component	Useful Life(in years)	Total Cost(in €)	Amortized Cost(in €)
1	Linux OS	5	0€	0 €
2	JavaScript Engine	1	0€	0 €
3	Python Components	1	0€	0 €
4	Web.py	1	0€	0 €
5	TexMaker	1	0€	0 €
	Total		**0€**	**0 €**

1.4.2.3 Human Resource Budget

The human resource budget deals with the overall expenditure spent on human resources. Every phase of the project has a cost associated with it in per hour calculation. The cost is fictitious as it has not been developed commercially. Table 3 intents to estimate the economic cost of each of the phases of the project. The cost per hour is intended as an approximation of the current cost per work hour of young analysts and developers in our environment.The cost is fictitious as it has not been developed commercially. Table 3 intents to estimate the economic cost of each of task that involves a human component for the project.

Table 3 - Human Resource Budget					
Sno:	Phase	Deadline	Hours	Cost(per hour in €)	Total(in €)
1	Required Knowledge Acquisition	1 Mar 2016	70	15€/h	1050 €
2	Paper Analysis	1 Apr 2016	150	15€/h	2250 €
3	Design and Implementation	30 Apr 2016	230	20€/h	4600 €
4	Testing I	15 May 2016	75	15€/h	1125€
5	Testing II	31 May 2016	75	15€/h	1125€
6	Report Writing	15 Jun 2016	100	15€/h	1500€
	Total		**600**		**10525 €**

1.4.2.4 Total Budget

The following table, Table 4, summarizes the total budget for the project. This encompasses the hardware, software and human resources budget.

Table 4 - Total Budget		
Sno:	Resource	Total Cost(in €)
1	Hardware Budget	1000 €
2	Software Budget	0 €
3	Human Resource Budget	10525 €
	Total	**11525 €**

1.5 Sustainability

Sustainability is a key factor in any project design. We evaluate the project based on three factors of sustainability namely economic sustainability, social sustainability and environmental sustainability.

1.5.1 Economic Sustainability

From our estimation it can be said that the current budget estimation will be the maximum bound on the budget for the project. This takes into account all the factors namely the hardware costs, software costs and human resource costs. The cost estimated in the project is the least possible cost and hence is a nonpareil project estimate for any indistinguishable project. The budget may exceed our calculations only during unexpected times. When the proposed plan is precisely followed the estimated lower costs gets achieved. Most of the software used in the project is open source which adds no value to product cost. The hardware requires only a computer which is a necessary tool in any project.

1.5.2 Social Sustainability

The project involves making a web application that helps in visualizing raw input data. This will help to analyse the learning characteristics of the patients and provide a feedback both to the medical analyser and health planner. This is going to improve the quality of health analysis in the state. All this requires is a simple computer connected to the internet. Thus adding to the view point of having a great social responsibility.

1.5.3 Environmental Sustainability

The project involves usage of a computer. We can assume that the amount of energy used by a single computer comes to around 250 watts. And given that we spend 500 hours on the project then the energy expended is 125KW. On average, electricity sources emit 1.22lbs CO2 per kWh (0.0006 metric tons = 0.53 Kg of CO2 per kWh). (Source: EPA eGRID Summary Tables and Data Files). This amounts to a upper bound of 67.2 kg of CO_2 considering that the energy was produced by using coal lignite. This can be reduced by reducing the development effort which is possible by reusing the already existing code. The project is environmentally sustainable as it meets with the limits of misusing the environment.

Chapter 2

Background Knowledge

In this section we present the background knowledge required to understand and solve the problem

2.1 Graph Notion

Many real-world problems can be solved by describing them by means of a diagram that consists of a set of points in which a few or all the pairs of points are joined together by lines. It is interesting to find whether any two given points are joined by lines or not. A mathematical abstraction of this situation is termed as graphs [5]. In the project of concern where we deal with representing the medical data in this manner it becomes necessary to talk about graphs.

2.2 Graph Definition

A Graph G is formed by two finite sets, the set $V = \{ v_1, v_2, \ldots, v_n \}$ of vertices(also called nodes) and the set $E = \{ e_1, e_2, \ldots, e_n \}$ of edges where each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k represented as G=(V,E). In other words E $\subset V^2$, which is the set of all unordered edges. The vertices (v_j and v_k) that represent an edge are called *endpoints* and the edge is said to be adjacent to each of its end points.

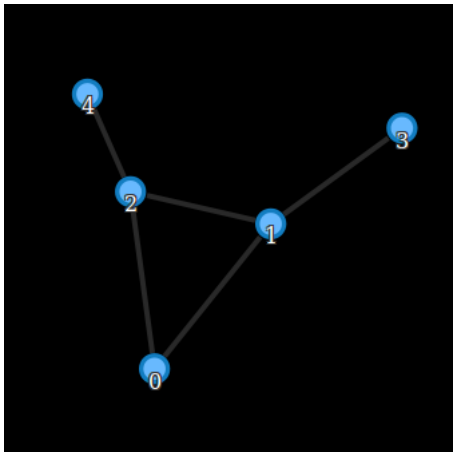


Figure 2.1: Example graph created using the project. This is called the "Bull Graph".

The neighbourhood of a node or vertex v_i is the set of nodes v_i is connected to, $\mathrm{N}(v_i) = \{v_j | (v_i, v_j) \in E, v_i \neq v_j, 1 \leq j \leq n\}$. The degree of a node v_i, or the size of the neighbourhood connected to v_i, is denoted as $\mathrm{d}(v_i) = |N(v_i)|$.

A degree sequence, D, specifies the set of all node degrees as tuples, such that $\mathrm{D} = (v_i, d(v_i)$ and follows a probability distribution called the *degree distribution* with mean d_m [19].

2.3 Graph Matrix Notation

The matrix is commonly used to represent graphs for computer processing. The advantage of using matrix is usually that matrix algebra can be readily applied to study the structural properties of graphs. There are number of ways in which one can represent the graph in its matrix form for example, adjacency matrix and Laplacian matrix.

Let G=(V,E) be a simple graph with vertex set V and edge set E, then the adjacency matrix is square $|V|^2$ matrix M such that its element $M_{i,j}$ is *1* when there is an edge from v_i to v_j,where $v_i \in V$, $v_j \in V$ and *0* when there is no edge. The number of rows and columns that a matrix has is called its order. The adjacency matrix of a graph of order n entitles the entire the topology of a graph. The diagonal elements of the adjacency matrix are all *0* for undirected graphs M.

The sum of the elements of i-th row or column yields the degree of node i. If the edges are weighted, one defines the weight matrix W, whose element W_{ij} expresses the weight of the edges between vertices i and j.

The *spectrum* of a graph G is the set of eigenvalues of it's adjacency matrix M. If D is the diagonal matrix whose element $D_{i,i}$ equals its degree of vertex i ($v_i \in V$) [10].

2.4 Approaches

In this section we discuss the various approaches that are involved in dealing with the input to the project for community identification, for clustering and for visualization purposes.

2.4.1 For Community Identification

Virtually in every scientific field dealing with empirical data, primary approach to get a first impression on the data is by trying to identify groups having "similar" behaviour in data. There are numerous methods to achieve this objective of which

- Community Detection
- Clustering

2.4.1.1 Community Detection

2.4.1.2 Definition of a Community

Communities are a part of the graph that has fewer ties with the rest of the system. Community detection traditionally focuses on the graph structures while clustering algorithms focuses on node attributes. This definition is admittedly somewhat informal. Different applications may require slightly different definitions, and certainly different algorithms formalize them in different ways. So there is not a unique, universally accepted, decomposition of a graph in communities.

Several types of community detection algorithms can be distinguished:

2.4.1.2.1 Divisive algorithms Divisive algorithms detect inter-community links and remove them from the network

2.4.1.2.2 Agglomerative algorithms Agglomerative algorithm merges similar nodes or communities in a recursive manner.

2.4.1.2.3 Optimization Methods Optimization methods are mainly based on maximization of an objective function, informally the "goodness" of a community decomposition.

2.4.1.3 Clustering

According to the paper "Community detection in graph" [11] there are four major traditional clustering methods namely :

- Graph Partitioning
- Hierarchical Clustering
- Partitional Clustering
- Spectral Clustering

2.4.1.3.1 Graph Partitioning This problem deals with dividing graph into groups of predefined size such that the number of edges between the groups is minimized. The paper [11] also defines cut size as the number of edges lying between the clusters. Figure 2.2 shows a problem with 14 vetices and presents a solution for spliting into 2 groups.

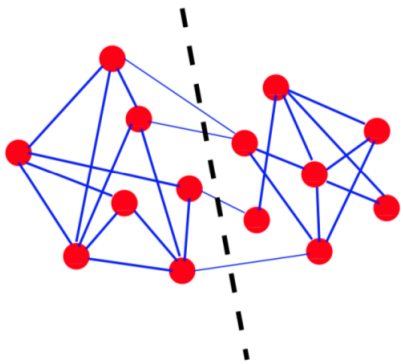


Figure 2.2: Graph Paritioning [11]

Minimum Bisection Problem, is a special problem case that considers partitioning the network into 2 groups of equal size. This problem is an NP-Hard problem. Intutively, to obtain ful partitioning we need to iteratively find all the minimum partition. This is not of significant use in the current problem of finding communities.

2.4.1.3.2 Hierarchical Clustering Hierarchical clustering aims to identify groups of vertices with high similarities. It can be calssifies into two categories:

1. *Agglomerative algorithm* : in one in which Agglomerative algorithms, in which clusters are iteratively merged if their similarity is sufficiently high

2. *Divisive algorithms*, in which clusters are iteratively split by removing edges connecting vertices with low similarity. The figure 2.3 demonstrates the hierarchcal clustering in a diagramatic manner.

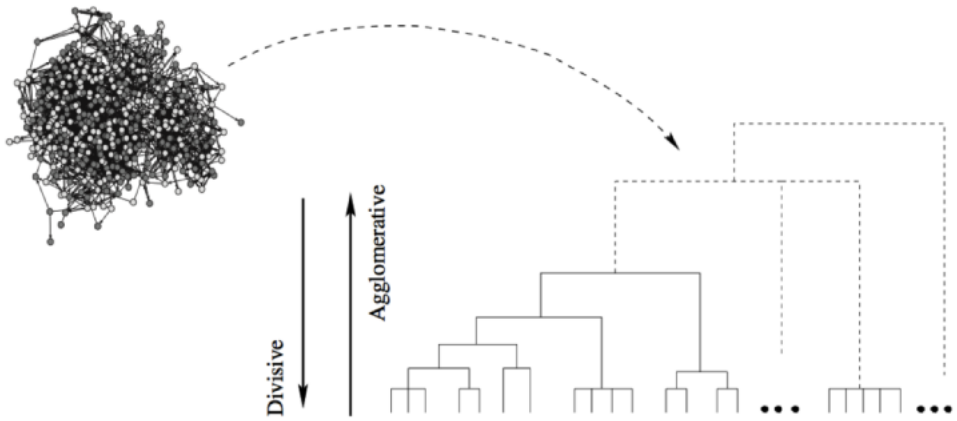


Figure 2.3: From a thickly knit graph to a dendogram [This intuitive diagram was taken from a powerpoint presentation for a data mining class]

2.4.1.3.3 Partitional Clustering Partitional clustering is method to find the clusters as a set of data points. The number of clusters is preassigned. let us call this number as k. The vertex is a point on the metric space with a defined distance measure between the pair of points in the space. The

distance represents the difference in dissimilarity between the vertices. The main objective in this method is to separate the points in k clusters such to maximize ((or) minimize) a given cost function based on distances between points and from points to *centroids* that are suitably defined positions in space. Some of the most used functions are : *Minimum k-clustering* , *k-clustering sum*, *k-center* and *k-median*. One of the most popular partitional technique in literature is *k-means clustering* where the cost function is total intra-cluster distance [11]. This method of clustering is out of scope for the project in concern.

2.4.1.3.4 Spectral Clustering According to the paper [11], let us suppose to have a set of n objects $x_1, x_2, \ldots, x_n$ with a pairwise similarity function S defined between them, which is symmetric and non-negative (i. e., $S(x_i, x_j) = S(x_j, x_i) \geq 0$, $\forall$i,j $= 1, ..n$). Spectral clustering includes all methods and techniques that partition the set into clusters by using the eigenvectors of matrices, like S itself or other matrices derived from it. In particular, the objects could be points in some metric space, or the vertices of a graph. Spectral clustering consists of a transformation of the initial set of objects into a set of points in space, whose coordinates are elements of eigenvectors: the set of points is then clustered via standard techniques, like *k-means clustering*.

2.4.2 For Visualization

Graph visualization is a important task in various scientific application. Visualizing these data as graphs provides the non-experts with an intuitive means to explore the content of the data, identify interesting patterns, etc. Such operations require interactive visualizations (as opposed to a static image) in which graph elements are rendered as distinct visual objects; e.g., DOM objects in a web browser. This way, the user can manipulate the graph directly from the UI, e.g., click on a node or an edge to get additional information (metadata), highlight parts of the graph, etc. Given that graphs in many real-world scenarios are huge, the aforementioned visualizations pose significant technical challenges from a data management perspective [3].

2.5 Computational Complexity

The estimate of the amount of resources required for by the algorithm to perform a task is defined as computational complexity. The humongous amount of data on the real graphs or real networks that are available in the current scenario causes the efficiency of the clustering algorithm to be crucial.

In a brief, algorithms that have polynomial complexity describe the Class **P**. Problems whose solutions can be verified in a polynomial time span the class **NP** of *non–deterministic polynomial time* problems, which includes **P**. problem is **NP**-hard if a solution for it can be translated into a solution for any **NP**-problem. However, a **NP**-hard problem needs not be in the class **NP**. If it does belong to **NP** it is called **NP**-complete. The class of **NP**-complete problems has drawn a special attention in computer science, as it includes many famous problems like the Travelling Salesman, Boolean Satisfiability (**SAT**), Integer Programming, etc. The fact that **NP** problems have a solution which is verifiable in polynomial time does not mean that NP problems have polynomial complexity, i. e., that they are in **P**. In fact, the question of whether **NP**=**P** is the most important open problem in theoretical computer science. **NP**-hard problems need not be in **NP** (in which case they would be **NP**-complete), but they are at least as hard as **NP**-complete problems, so they are unlikely to have polynomial complexity, although a proof of that is still missing. Reference to this has been imbibed from the paper "Community detection in graphs" [11].

Many clustering Algorithms or problems related to clustering are **NP**-hard. This makes it hopeless to look for an exact algorithm, in which case we may look for an approximation algorithm. Approximation algorithm are methods that do not deliver the exact solution but an approximate solution but with an advantage of lower complexity. [11]

2.6 State-of-the-art in Community Detection

Modularity is the objective function that is widely used both as a measure and as a optimizing method for partitioning community. As said before there are various algorithms that can be used for community detection . Paper [20] discusses six different community detection algorithms namely:

- Louvain Method

- Le Martelot
- Newman's greedy algorithm (NGA)
- Newman's spectral algorithm with refinement
- simulated annealing
- extremal optimization

The following figure 2.4 from paper [20] the average normalized performance rank of each algorithm in terms of partitioning quality and speed. The main

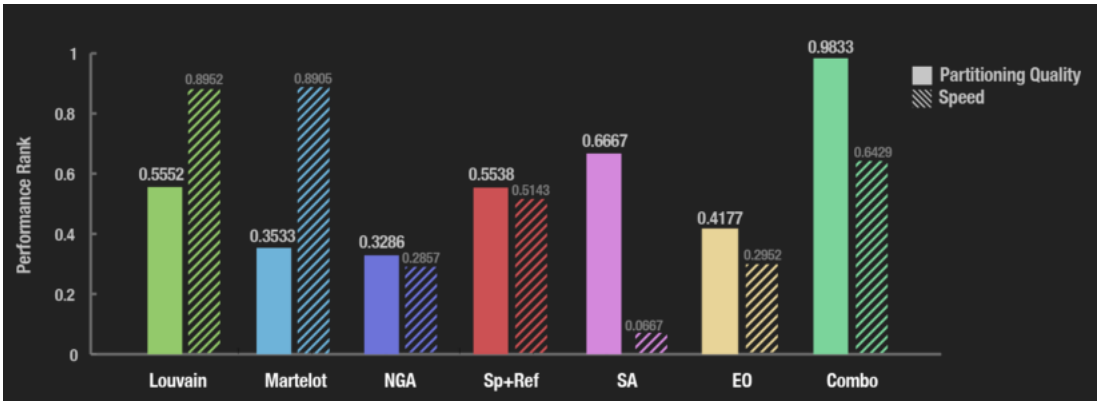


Figure 2.4: Average normalized performance rank of each algorithm in terms of partitioning quality and speed(Taken from the website of the paper that proposed Combo algorithm [20].

objective of the project is to visualize the data on screen thus needs an algorithm that is fast and should be effective. Hence Louvain algorithm was choose for the implementation. The implementation can be found in the Chapter 3 of the report.

Louvain algorithm algorithm is considered as state-of-the art algorithm for community detection [4]. The algorithm is fast and more effective than the other algorithms in real-world graphs. Due to our goal of projecting medical domain we require an algorithm that gives a better trade off between being effective and being fast.

2.7 Degree Distribution

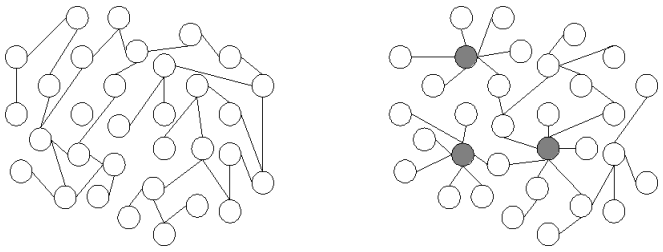
Degree of a node(v_i) in the graph $G = (V, E)$ where V is the set of vertices and E is the set of edges is the number of edges that a node has to other nodes. Usually denoted as $\deg(v_i)$. *Degree distribution* can be thus described as the probability distribution of there degrees over the entire graph. Degree distribution is significant in the study of community networks and hence bringing it into consideration. It is usually denoted as $P(k)$ of a graph which is the fraction of nodes in the network with degree k.

$$P(k) = \frac{N_k}{N} \tag{2.1}$$

where N_k is the number of nodes with degree k and N is the total number of nodes in the graph.

2.7.1 Scale-Free Graph

The graphs whose degree distribution follows power law are called as Scale-Free graphs or scale free networks, fo examples of Scale-Free graphs include Social network graph,protein-protein interaction network etc.
Scale graphs have a number of interesting properties, both theoretical and practical. For example, according to the paper "Resilience of the Internet to Random Breakdowns" [1], removing randomly any fraction of nodes from scale free network will not destroy the network which is in contract to Erdos-Renyi graphs which make them interesting for building networks of several types resilient to failures or attacks. The figure 2.5 demonstrates how the random graph is different from a scale free graph. The highlighted spots are known as the hubs. For example: In a large community the celebrities or politicians serve as the hub. In scale free graphs another interesting feature is that as the clustering coefficient decreases with increase in the node degree. This type if distribution will be used in the experimental phase of the project for generating test cases.



(a) Random network (b) Scale-free network

Figure 2.5: Random network (a) and scale-free network (b). In the scale-free network, the larger hubs are highlighted.(Image Source: Wikipedia)

2.8 State-of-the-art in Graph Visualization

Researchers have designed various tool-kits for the purpose of information visualization. Newer visualization techniques are introduced by creating new components or sub-classing the existing ones. The main aim of the project will be analyse the way inwhich the visual frameworks can be used and what kind of intermediate trasitions that are possible between the Python program and the visualization framework. The project director's sugestion was to go with a JavaScript Library.

Advantage of JavaScript Libraries [18]

1. *Easy to use* : JavaScript is very easy to use, write and learn. It also saves time for the developer to write the code.

2. *Reliable and standardized* : JavaScript libraries are more reliable and standardized and remains widely supported by several major web browsers.

3. *JSON is Standard*: JSON or JavaScript Object Notation is a standard input/output format for data sharing.

4. *Simple and light-weight animation possible in JavaScript* JavaScript libraries need knowledge of CSS, HTML and editor to write the program. Contacting with Java or Adobe Flash , JavaScript animation require less learning and less time to make a cheap and effective animation

Due to these advantages JavaScript was suggested as the visualization as it is light, fast and data is easier to represent in JavaScript which satisfies the goal of the project. Next task is to find a JavaScript visualization framework.

In the paper "Effectiveness of JavaScript Graph Visualization Libraries in Visualizing Gene Regulatory Networks (GRN)" [18] "TABLE I" gives an summary of a few different JavaScript graph visualization libraries. Considering all the factor four of the visual techniques were chosen for the purpose of study and analysis of which one will be chosen for implementation.

2.8.1 Protovis

Protovis is a domain-specific language for constructing visualizations by composing simple graphical marks such as bars, lines and labels. In Protovis, designers specify visualizations as a hierarchy of marks with visual properties defined as functions of data. Inheritance of properties across composed marks similar to cascading of style sheets used in web design enables concise visualization definitions with a large expressive range and a minimum of intervening abstractions. Protovis is implemented in JavaScript, with rendering support for HTML 5 canvas, SVG, and Flash. The above extract is as given in the main paper about Protovis [6].

2.8.2 D3.js

D3.js or simply D3 is a JavaScript library for manupulating documents based on data. D3 takes full advantage of cascading Style Sheets(CSS), HTML5 and Scalable vector graphics (SVG). D3 is viewed as the successor of Protovis. D3 targets at animation,interaction and complex and dynamic visualization in web. It uses pre-built JavaScript functions in its kernel in order to select elements, create SVG objects, style them or add transitions and dynamic effects to them. [18]. Paper "D3: Data-Driven Documents" [7] describes d3in its rightful way. D3 provides a wide set of example of which these were of great interest to the project:

1. For graph: Force-Directed Graph, Labeled Force Layout,Forced Graph Editor , Directed Graph Editor, CodeFlower, University Program Transfers - Interactive, Better force layout node selection

2. For matrix visualization : Day/Hour Heatmap

2.8.3 Gephi

Gephi is an open source software which use 3D render engine for visualization of graphs. A flexible and multi-task architecture brings new possibilities to work with complex data sets and produce valuable visual results. Gephi involves many key features in the context of interactive exploration and interpretation of networks. spatializing, filtering, navigating, manipulating and clustering are allowed in Gephi.Dynamic network visualization is also possible in Gephi [2]. Although gephi is not JavaScript it is cited and considered for its externsive use in graph visualization. It is interesting to note that Gephi has JavaScript viewer such as JavaScript GEXF available on github.

2.8.4 Alchemy.js

Alchemy.js is a JavaScript framework build entirely on d3. The goal of Alchemy.js is help the developer build and run graph visualizations with minimal overhead. A developer in Alchemy.js does not have to write much code to visualize the data. Configuring Alchemy.js to visualize the data will be the only task to be done, along with whatever features that are wanted to be include in the application.

Chapter 3

Louvain Community Detection Algorithm

In this section we describe the community detection algorithms such as Louvain and various tests that were performed to choose the algorithm.

3.1 Modularity

The quality of partitioning that results from application of method is often measured using modularity. The *modularity* of a partition is hence a scalar value between -1 and 1 that is used to measure the density of the links inside the communities as compared to the density of the links between the communities. This concept was first put forward by Newman [15].

Modularity not only serves as a quality measure for detecting the quality of split or -partition, but also acts as an objective function to optimize. Exact modularity optimization is **NP-Complete** in the strong sense [8].

3.1.1 Definition

Let G=(V,E) be a simple graph,where V is the set of vertices and E is the set of undirected edges. Let $n = |V|$ and $m = |E|$. Let degree of a vertex v be, deg(v) where $v \in$ V. Let C be the community, $C \subseteq V$, be the subset of vertices. A *clustering* $C_s = \{C_1, C_2, \ldots, C_k\}$ of G is a partition of V such each vertex is present exactly in one cluster. We thus define *modularity* as follows: [8]

$$Q(C_s) = \sum_{C \in C_s} \left[\frac{|E(C)|}{m} - \left(\frac{|E(C) + \sum_{k \in C_s} |E(C,k)|}{2m} \right)^2 \right] \tag{3.1}$$

where E(I,J) is set of all edges between vertices in cluster I and J. E(C) = E(C,C). The above equation can be continently rewritten as follows:

$$Q(C_s) = \sum_{C \in C_s} \left[\frac{|E(C)|}{m} - \left(\frac{\sum_{v \in C} deg(v)}{2m} \right)^2 \left(t \right] \tag{3.2}$$

In simpler terms the value is of Q can be expressed as

$$Q = \text{(Number of Intra-Cluster Communities)} - \text{(Expected number of Edges)} \tag{3.3}$$

As given in [4]

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - P_{ij} \right) \delta(C_i, C_j) \tag{3.4}$$

$$\delta(C_i, C_j) = \begin{cases} 1, & if C_i = C_j \\ 0, & otherwise \end{cases} \tag{3.5}$$

where, P_{ij} is the expected number of edges between nodes v_i and v_j. P_ij is $\frac{k_i k_j}{2m}$ where k_x is sum of the weights of the edges attached to the vertex v_x for a given random graph G (This is otherwise called as a null model).

3.1.2 Properties of Modularity

1. Q depends on nodes in the same clusters only.
2. Larger modularity implies better Communities.
3. $$Q(C_s) \leq \frac{1}{2m} \sum_{ij} A_{ij} \delta(C_i, C_j) \leq \frac{1}{2m} \sum_{ij} A_{ij} \leq 1 \tag{3.6}$$
4. Value taken by Q can be negative

3.2 Implementation of the Louvain Community detection Algorithm

Louvain algorithm is considered as the state-of-the art algorithm for community detection for identifying community structures [4]. Louvain method developed by Blondel *et al* [4] finds high modularity partitions of large networks in short time. It unfolds a complete hierarchy community.

The Algorithm has two phases that are repeated iteratively to bring the final solution to the problem. The following figure 3.1 visualizes the steps in the algorithm and Algorithm 1 defines a pseudocode for the Louvain method.

Algorithm 1 Louvain Algorithm Pseudocode

Require: A graph G = (V,E)
Ensure: Local optimum community split has happened
  while *LocalOptimumReached* **do**
    Phase1 : Split or partition the graph by optimizing modularity greedily
    Phase2 : Agglomerate the found clusters into new nodes
  end while

3.2.1 First Phase : Optimizing Modularity

In the first phase the algorithm assigns a different community to each node in the network. The number of nodes is equal to the number of communities in the graph. Let v_i be be a node such that $v_j \in \mathrm{N}(v_i)$. The gain of modularity is then calculated by removing v_i and placing it in community of v_j. If the gain is positive the v_i is moved to the community of v_j else v_i stays in it's original community. This procedure is iterated and the phase one stops when a local maxima of the modularity is achieved, that is when no more move of nodes from one community to another is possible. The ordering of the nodes can affect or effect the computation time which can be a part of future works.

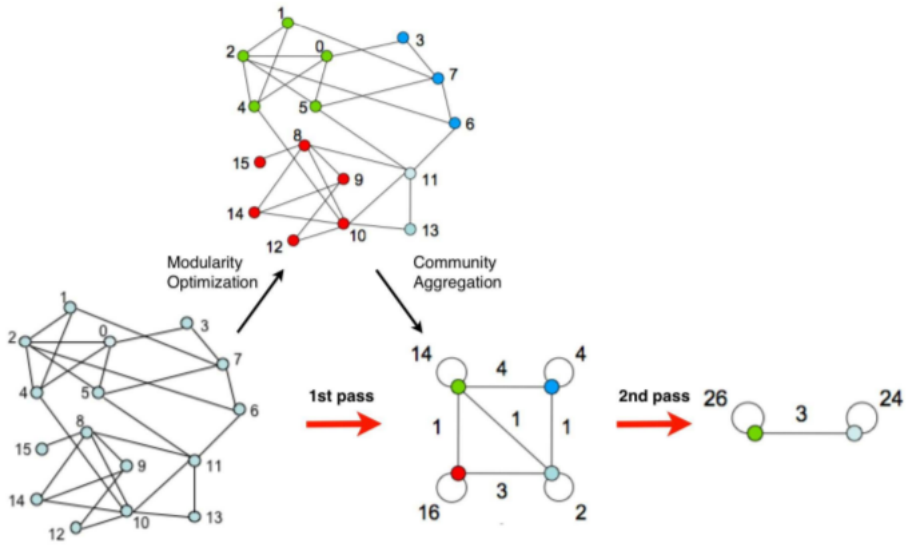


Figure 3.1: Visualization of the steps of our algorithm. Each pass is made of two phases: one where modularity is optimized by allowing only local changes of communities; one where the found communities are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible. This was taken from the paper "Fast unfolding of communities in large networks" [4]

Algorithm 2 Phase 1 in Louvain Algorithm Pseudocode

Require: A graph G = (V,E)
Ensure: Partition network greedily using modularity
  Assign a different community to each node
  while *LocalOptimumReached* **do**
    for all Each node v_i **do**
      For each node $v_j \in \mathrm{N}(v_i)$, consider removing v_i from community of v_i and place it in the community of v_j
      Calculate the modularity gain
      if *ModularityGain* is Positive **then**
        remove v_i from community of v_i and place it in the community of v_j
      else
        No Change
      end if
    end for
  end while

The main algorithm relies on the calculation of modularity. Listing 3.1 demonstrated the calculation using a Python snippet. The first phase of the algorithm has been written into a Python code snippet and is presented in the Listing 3.2. In the paper it is stated that the gain in modularity as $\triangle$Q

$$\triangle Q = \left[\frac{\sum_{in}+k_{i,in}}{2m} - \left(\frac{\sum_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{3.7}$$

where $\sum_{in}$ is the sum of the weights of the links inside C and $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C, k_i is the sum of weights of the links incident to node i, $k_{i,in}$ is the sum of the weights of all the links in the network.

3.2.2 Second Phase : Agglomerating the communities found in first phase into new nodes

In the second phase the algorithm builds the new network. The communities that are found during the first phase are now the nodes here. According to the paper [4], the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. The edges between nodes of the same community lead to seld-loops for this community in the new network. The resulting new weighted network is then subjected to first phase and this process is iteratively done.

Algorithm 3 Phase 2 in Louvain Algorithm Pseudocode

Require: A graph G = (V,E)
Ensure: Agglomeration of nodes
Every community C_i forms a new node v_i
$W_{ij} = \sum$ {All edges between C_i and C_j} where W_{ij} is the edge between newly formed nodes v_i and v_j

3.3 Observations of Louvain

1. The final output of the Louvain algorithm forms a complete hierachical structure.

2. Resolution limit problem [12] has been resolved in the algorithm stated in the current paper under discussion [4] due to the multi-level nature of Louvain algorithm.

3. Modularity can be redefined for weighted graphs and Louvain works well with weighted graphs.

3.4 Mode of implementation of the Louvain Algorithm

In the project the above algorithm has been implemented in Python taking inspiration and reusing some part of the pyLouvain program API for implementation [16]. pyLouvain cannot be directly used in the project as

Using Python was a requirement of the project directors, in case the code has to be reused in a larger ongoing project which is programmed in Python and Javascript. Hence Python was used to implement the Louvain Algorithm.

3.5 Experiments

In performing experiments we analyse the speed and capability of the Python program to handle more input and also to see pattern of behaviour of the data. For this purpose test case inputs have to be generated or test cases that are available online must be used. In this project we utilize both method to see the performance of the program. All the tests were run on 2.50GHz Intel Core i5-3210M with 8 GB RAM (Sony Vaio) running gnome-ubuntu 14.04 operating system.
In the paper [13] the author suggest a few benchmarks for evaluvating the performance of community detection algorithms:

1. *real-world benchmarks*, such as Zachary's Karate Club, where a dataset based on a social system includes a natural set of ground-truth communities;

2. *synthetic benchmarks*, where data is artificially created according to some model which includes

3. *a pre-defined set of ground-truth communities*; and task-oriented benchmarks, in which communities are used to help complete some task on real-world data.

Following the above bench marks three kinds of experiments were performed:

3.5.1 Synthetic Benchmarks

3.5.1.1 Nodes vs Time keeping modularity constant

In the paper "Exploring community structure in biological networks with random graphs" [19] [1], authors have developed a generative model to produce undirected connected graphs with a degree and a pattern of communities while maintaining a graph strucuture that is as random as possible. They have given two programs one which contains the degree distribution funtions such as scale-free degree distribution, poisson degree distribution, regular distribution and geometric and the other one that contains the graph generators that return a generated graph in networkX package. In this project we are intersted in scale-free distribution as most real-world networks are scale free in nature [1]. To run this test networkx package is a necessary component.

For the purpose of testing two programs in Python were written. First program is a generator program that sets the degree distribution to scale-free, Q=0.4 [Average Q value] and N=150 to N=2300. This program generate a set of files contains graphs with nodes in the range of N. Second program is the plotting program in which the files that were generated in the first program was supplied as input to the second program and Louvain algorithm was applied. Time of run for Louvain was calculated for each input and recorded. This was then plotted into graphs using the matplotlib function. The graphs are displayed in figure 3.2, 3.3, 3.4 and 3.5 for respective range of nodes mentioned in the caption of the graphs.

[1] Random Modular Network Generator `https://github.com/bansallab/modular_graph_generator/`

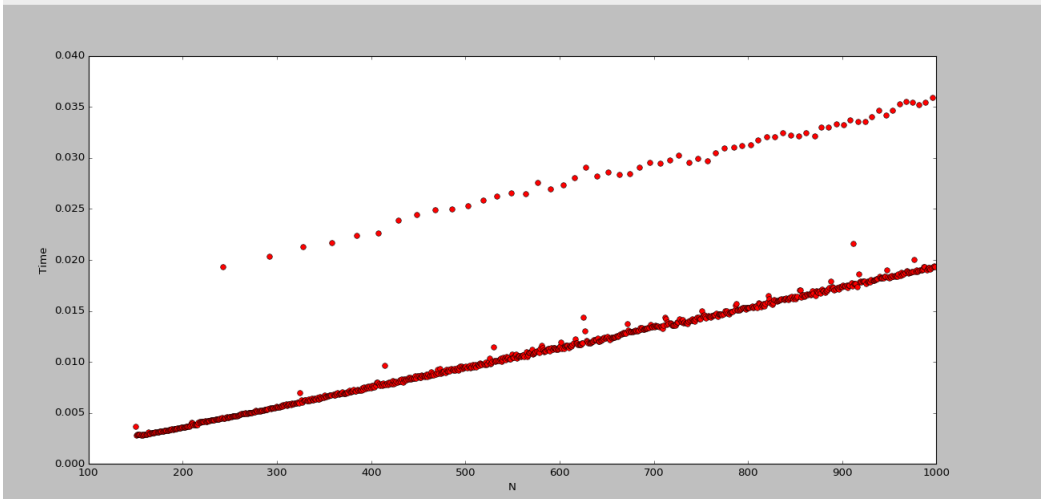


Figure 3.2: Experiment showing the plot of time vs number of nodes for 1000 nodes

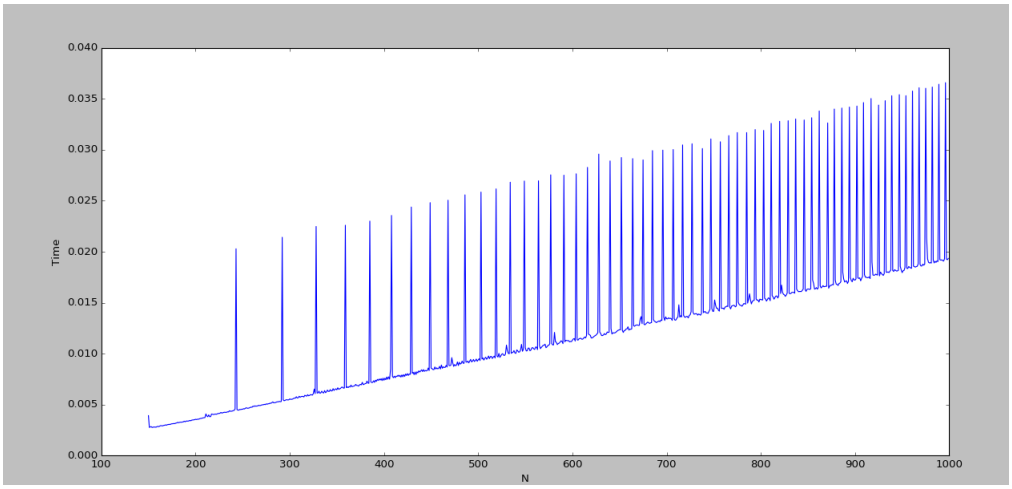


Figure 3.3: Experiment showing the plot of time vs number of nodes for 1000 nodes points joint by lines

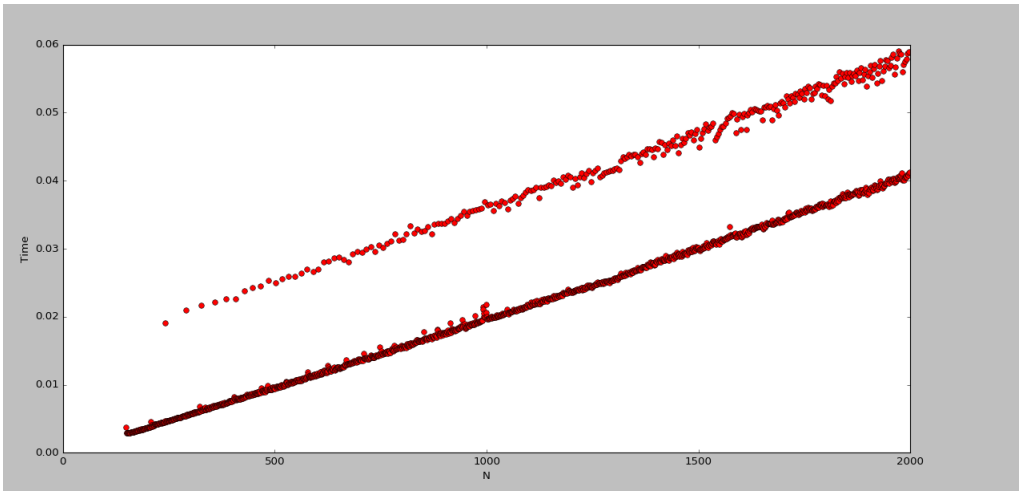


Figure 3.4: Experiment showing the plot of time vs number of nodes for 2000 nodes

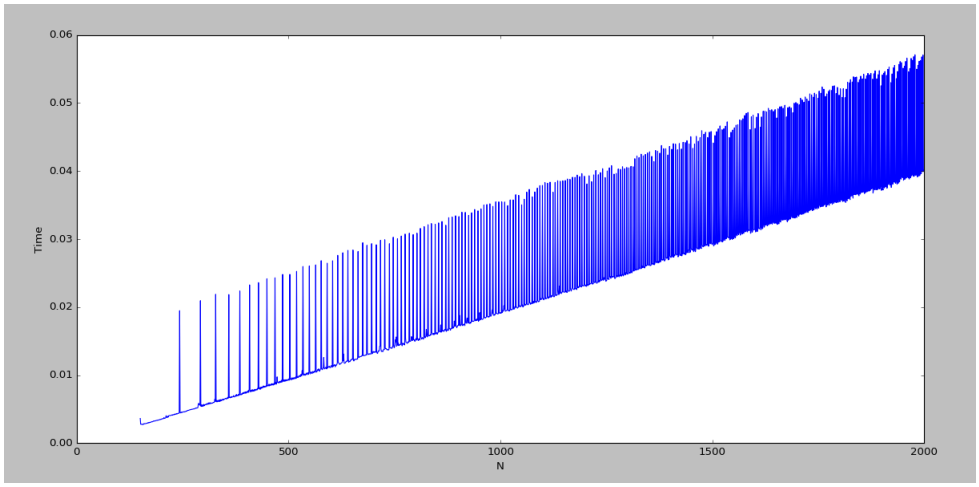


Figure 3.5: Experiment showing the plot of time vs number of nodes for 2000 nodes points joint by lines

It is interesting to note that for steady raise in the number of inputs there was a steady raise in the time. There were very few jitters that were caused by the computer.

3.5.1.1.1 Minor bugs In the results obtained it is interesting to note that there is a set of slow runs that occur at regular intervals. These are due to the effect of garbage collection done by Python automatic garbage collector after a number of runs of Louvain algorithm had exhausted available memory This issue was found by manually turning of the garbage collection and explicitly calling it every time after each run. Therefore, the "bottom" line of the plot should be taken as the true running time of the Louvain algorithm.

3.5.1.2 Modularity vs Time keeping Number of nodes constant

Using the same code we retain the distribution and make the number of nodes as a constant and vary the Q value. This experiment helps to find out the change in run time of the algorithm varying Q. Figure 3.6 and 3.7 show a sample of the test.

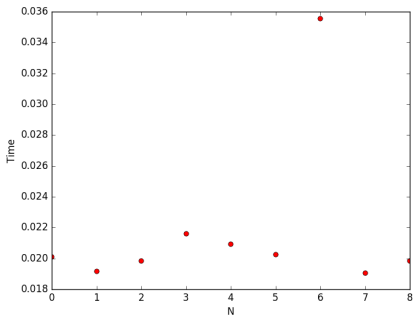


Figure 3.6: Experiment showing the plot of time vs number of nodes for 1000 nodes but Q is varying(X axis is scaled 10 times, that is 0.1 is 1 in the graph)

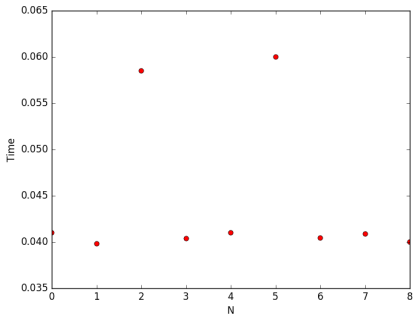


Figure 3.7: Experiment showing the plot of time vs number of nodes for 2000 nodes but Q is varying (X axis is scaled 10 times, that is 0.1 is 1 in the graph)

3.5.2 Task Oriented Benchmark: Stanford Network Analysis Project(SNAP) data

SNAP[2] collection of more than 50 large network datasets from tens of thousands of nodes and edges to tens of millions of nodes and edges. In includes social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks [14]. An instance of the facebook data having 5000 nodes and intersections was tested and the resulting output on screen. Time taken was 0.38 seconds.Figure 3.8 shows the community graph of the facebook graph taken. Arxiv data of physics consiting of 18772 nodes and 396160 edges took 1.78s. Amazon ungraph with 334863 nodes and 925872 edges took 5.13s.

[2]Stanford Network Analysis Project `http://snap.stanford.edu/`

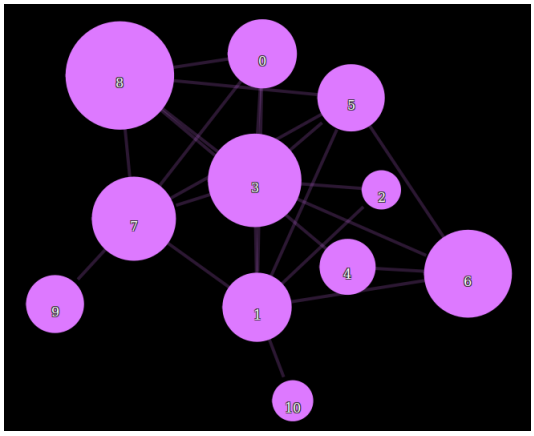


Figure 3.8: Community graph of facebook data taken from SNAP

3.5.3 Some Real-world Benchmarks : igraph tests

Python provides an exciting package on igraph. igraph provides a functionality where famous graphs such as Zachary karate graph, bull graph etc can be obtained. A program was written to generate 28 famous graphs and time for running Louvain algorithm on the 28 graphs was calculated and recorded [9].

The names of the 28 graphs are :
"Zachary","Chvatal","Coxeter","Cubical","Diamond",
"Dodecahedral","Folkman","Franklin","Frucht","Grotzsch",
"Heawood","Herschel","House","HouseX","Levi","McGee","Meredith",
"Noperfectmatching","Nonline","Octahedral","Petersen","Tutte","Robertson",
"Smallestcyclicgroup","Tetrahedral","Thomassen","Uniquely3colorable","Walther"

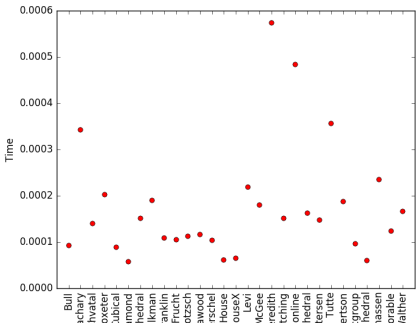


Figure 3.9: Famous graphs and their timing

3.6 Result

The algorithm was found to fast in terms of computation and effective in split. In medical domain the graph size is expected to be around 10^5 to 10^6 nodes with nearly 1 million edges. In the tests performed with Amazon data from SNAP this range was achieved. Hence the experiments and the analysis of the algorithm prove that Louvain algorithm implemented in the project is effective in its execution.

Chapter 4

Visualization Module

The visualization module was developed keeping mind that it should be Javascript based and that it must interface with Python. As already said these were the requirements that were set by the project directors. In this section an analysis of the selected software and method of choice by exploring all the possible ways will be explained.

4.1 Exploring the state-of-the-art

In Chapter 2, section 2.8 , a few options on graph visualization were discussed which includes Protovois, D3.js, Gephi and Alchemy. Amongst these state of the art graph visualization algorithms we need to choose the one that suits the projects needs.

Gephi is a tool used for data analysis for understanding and exploring graph based data,however it has some drawbacks in the logistics of the project such as Gephi is not JavaScript based although it has a JavaScript viewer. Gephi has more overhead being a software than a simple framework. For a similar reason we refrain from using softwares such as HyperTree, HyperGraphs as they did not have a Python API or are not based on JavaScript but instead has a Java Code based APIs. D3 provided the necessary tools and was JavaScript and could be linked to Python and was the best option for the project's logistics.

Alchemy.js was built using D3. Alchemy.js required minimal code to generate the graphs as most of the customization could be done by just overriding or altering the "config" [Configuration part of the Alchemy.js]

instead of implementing it entirely using JavaScript. Alchemy.js also provides a feature in which the core application can be further extended with any other feature of D3. Having D3 to be the base, with minimal code and maximum customization Alchemy.js was chosen for visual representation of the graph in this project.

Apart from graph visualization, matrix visualization was needed. Protovis provides a matrix drawing component called matrix diagram to represent the graph in matrix format.

Thus the project directors considered alchemy.js as a visualization software as it fits the bigger project for graph visualization and Protovis was used to visualize matrix representation.

4.2 Alchemy.js

Alchemy.js is a graph drawing library built to provide graph visualization with little overhead. It is built on the d3 library, written in Javascript, which runs in most web browsers.

4.3 Dependencies for Alchemy.js

Alchemy needs three main units to form as an application namely: *alchemy.css*, *alchemy.js* and *data*. CSS and JavaScript are major dependencies in Alchemy.js. Installation of *jQuery* and *d3* is also useful. *alchemy.min.js*, *alchemy.css* and *alchemy.min.css* will be updated in the CDN (Content Delivery netwrok)

4.4 Steps to use Alchemy.js

In this project we have uploaded the *alchemy.min.js*, *alchemy.css* and *alchemy.min.css* into the project's file repository http://abhinavsv3.github.io/javascriptsal and hence will be using the link in the following explanation. The following describes the steps that are followed in use of Alchemy.js :

1. *Include the files in this format*
   <link rel="stylesheet" href="http://abhinavsv3.github.io/javascriptsal/alchemy.min.css" />
   ...
   <script src="http://abhinavsv3.github.io/javascriptsal/alchemy.min1.js"><
   /*script* >

2. *Include an element with "alchemy" ID as the id and class*
   The alchemy class is used to apply styles while the alchemy id is used programatically. By default Alchemy.js looks for the alchemy div but this can be overridden.

3. *Provide Alchemy.js with a JSON dataSource*

4. *Begin Alchemy.js* <script> alchemy.begin({"*dataSource*" : *someData*}) </script>

4.5 Protovis for Matrix Display

Protovis is considered as the predecessor D3.js. In Protovis there is an example for Matrix Diagrams. This example is of keen interest for the project for representing the data.It is JavaScript and can be visualized with just the JavaScript matrix as input.

Since this is the only component that is taken from Protovis a Protovis template was adopted and from the Protovis repository and integrated with the project.

4.6 Getting the data from the Louvain Python code to Alchemy.

Alchemy.js takes a simple data format called GraphJSON. GraphJSON serves as a light weight and flexible representation of graph data, easily consued locally or over the web.

GraphJSON is a JSON Object which has two kinds of objects namely: *nodes* and *edges* . These are individual arrays that represent the nodes and edges that will be represented in the graph visualization.

Nodes : id key is the only unique value that should be present in the nodes

Edges : source and target key are the only unique value that should be present in the edges.

A Python program was implemented to convert the output of the Louvain program into JSON object using the Python JSON package. This converts the output of the Louvain program into two JSON object. one JSON object

This can be seen as the advantage of system. We compute once and reuse the computed result for various visualizations.

For the matrix representation the JavaScript file containing the matrix is created which is used by the protovis program to display and reorder the matrix.

4.7 Tests

For the purpose of visualization various test were performed for analysing the compatibility of Alchemy.js and to explore and exploit all the functionalities of alchemy.js and choose the best ones for the project and implement them.Protovis for the matrix visualization was used.

4.7.1 Alchemy.js Tests

The following will enumerate the tests that were performed one after another. Github pages were used to test host the web visualization.

1. Test1 and Test2

   Test1 consists of providing the JSON object as input to the Alchemy.js to build a normal graph. Test 1 is performed to see if the nodes can be selected, viewed and seperated.

   Test 2 Test2 uses the same JSON object as input.In Test2 the Test1 graph is presented with custom graph weight, graph width and link distances. This makes it easier to integrate it with larger application. Figure 4.1 shows the graph that was used to perform test1 and test2.

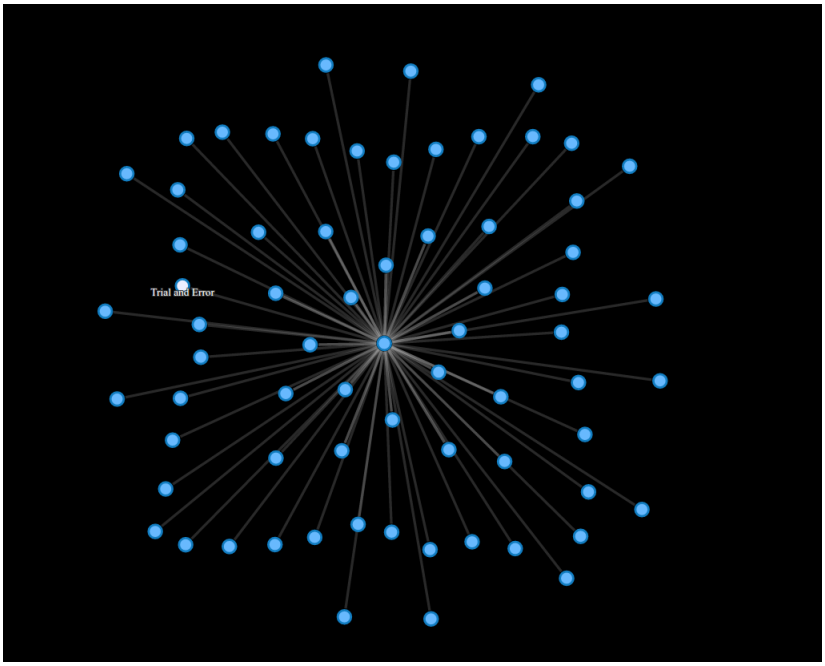


Figure 4.1: Test 1 and Test 2 look the same. The JSON object has been taken from Alchemy.js Examples. One named node "Trial and Error" has been selected and viewed.

2. *Cluster Assignment Test*

   In this test cluster assignment is tested.Clusters are assigned in the JSON object by setting values for the cluster in the form of cluster number and assign seperate colors for these clusters. Thus, this test focussed on node clustering or node coloring to assign each node into its cluster. Figure 4.2 show the result of the test(Test3).

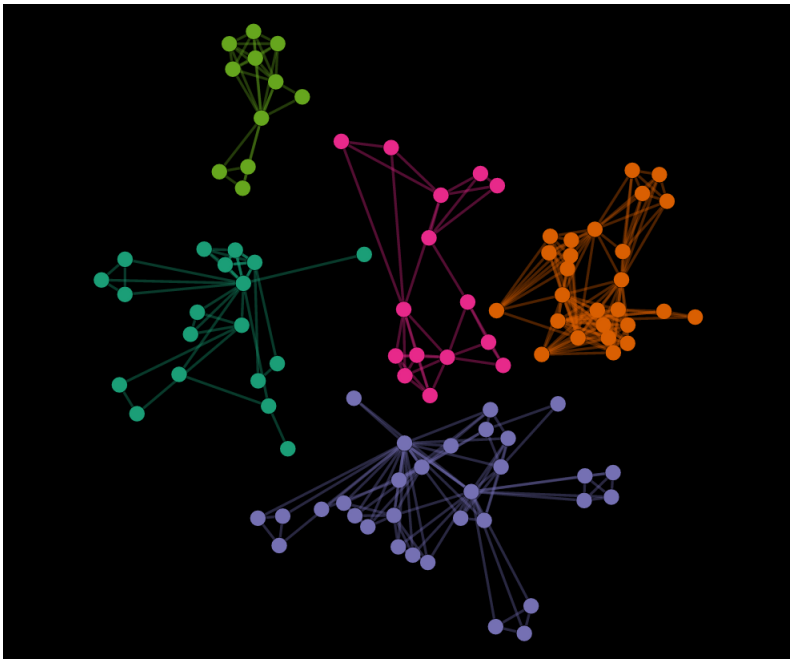


Figure 4.2: Clustering based Alchemy.js. The JSON object has been taken from Alchemy.js Examples.

3. *Directed and weighted edges*

   In this test we explore the special JSON object with directed and weighted edges. The test also shows an instance of controlling edge thinkness, node diameter and node thickness(Test4). Figure 4.3 shows an example of this test.

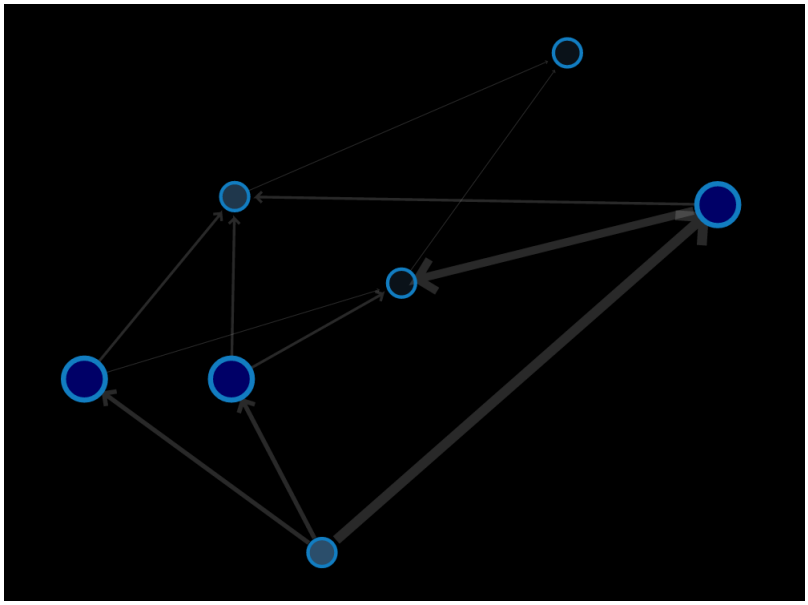


Figure 4.3: Example Directed and weighted edges. The JSON object has been taken from Alchemy.js Examples.

4. *Network*

   A test in which a hairball network is generated having roots and nodes. Figure 4.4 shows an example of seven minutes of communication between members of the AngularJS, EmberJS, and KnockoutJS IRC channels(Test5).

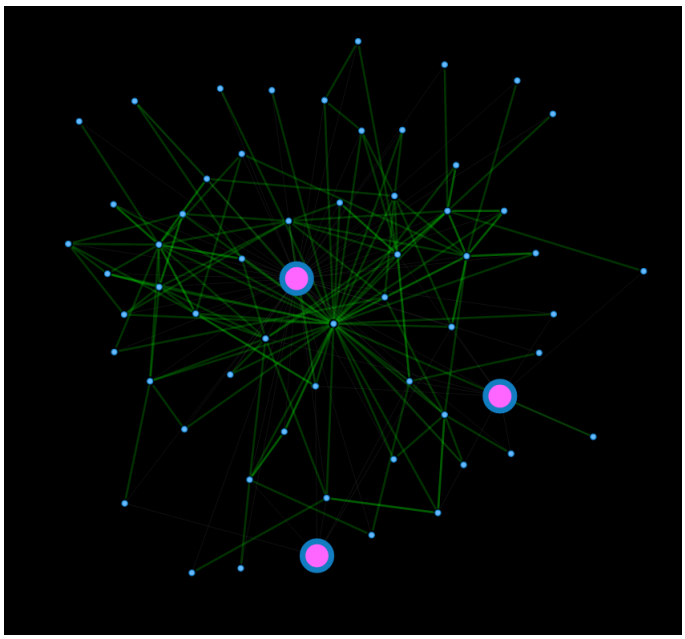


Figure 4.4: Seven minutes of communication between members of the AngularJS, EmberJS, and KnockoutJS IRC channels. JSON object taken from the Alchemy.js examples.

5. *Graph with relatedness*

   A test in which a graph with relatedness was generated. Figure 4.5 shows a sample having relatedness between philosophers. Figure 4.6 shows a zoomed view of one of the philosopher graph (Test6).

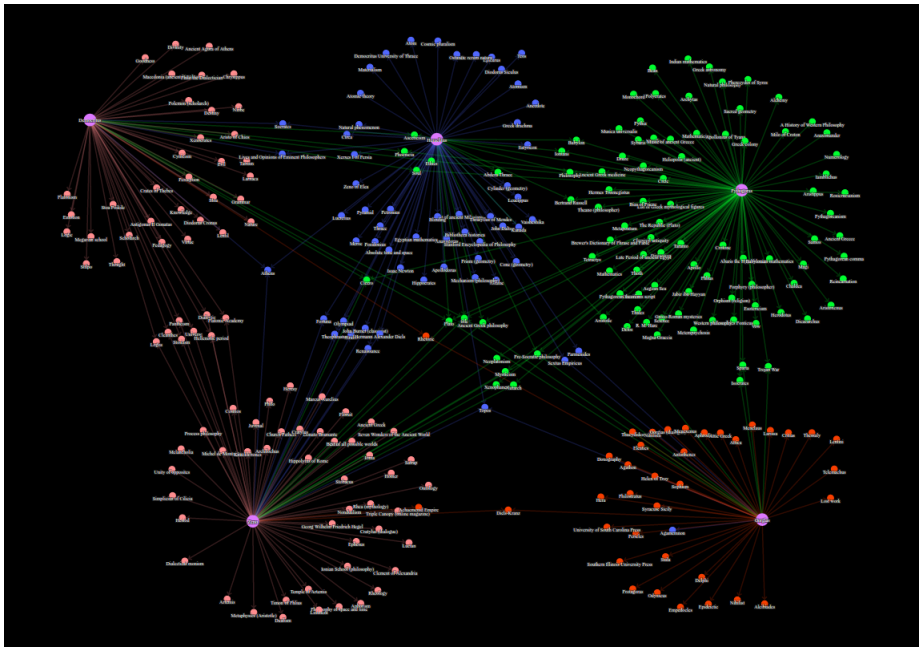


Figure 4.5: A graph in which the relatedness of scientists to the philosopher is shown. This example JSON was take from Alchemy.js example graphs

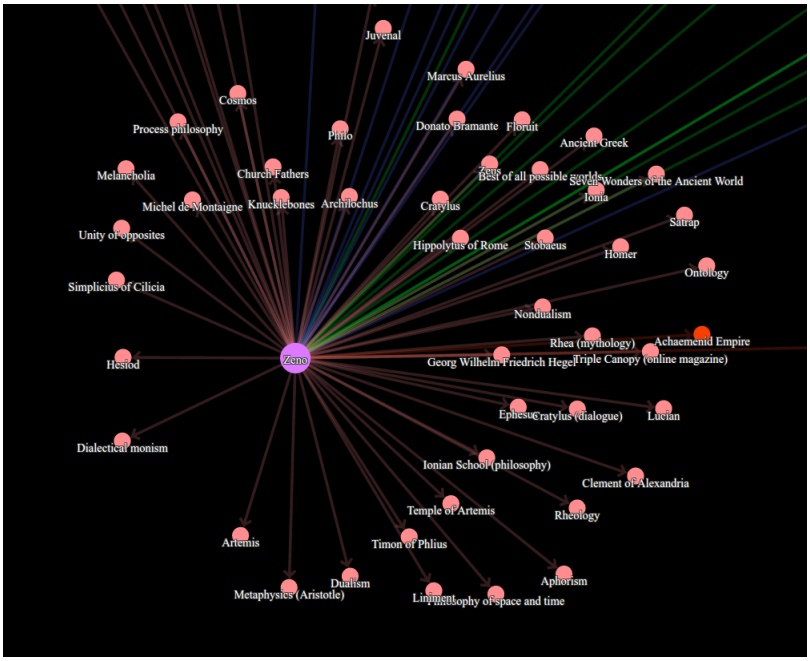


Figure 4.6: Closeup view of one of the philosopher.

6. *Node Area and Color*

   In this test the difference in node are and assignment of colors to nodes was tested. Figure 4.7 presents can example in which the test is performed (Test7).

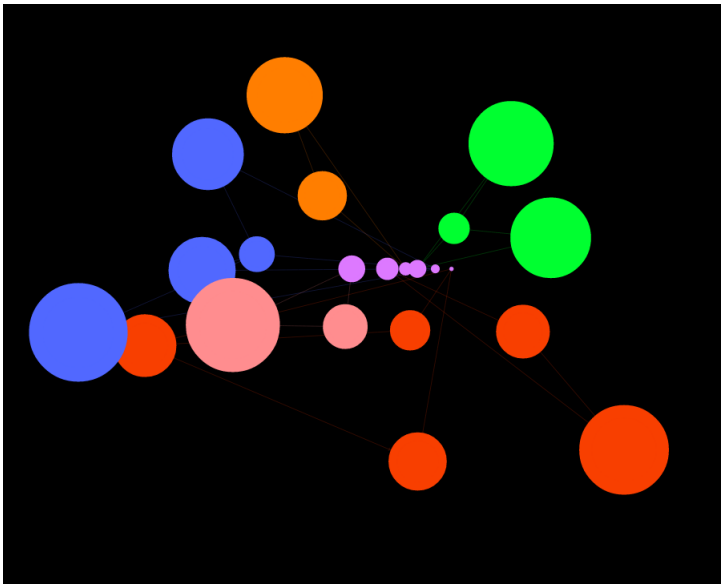


Figure 4.7: Node area and node color.JSON object created manually.

After performing all the test few of the desired features were incorporated to fordisplaying th graph in the project. The project uses test7 for showing the size or the bulkiness of the node in comparison with other node. Test6 relatedness has also been added to the project to mark the edge values between the nodes. Test3, clustering has also been added to the project.

4.7.2 Protovis Tests

For the purpose of matrix visualization, protovis has been implemented in the project. A test run of it is presented in the Figure 4.8.

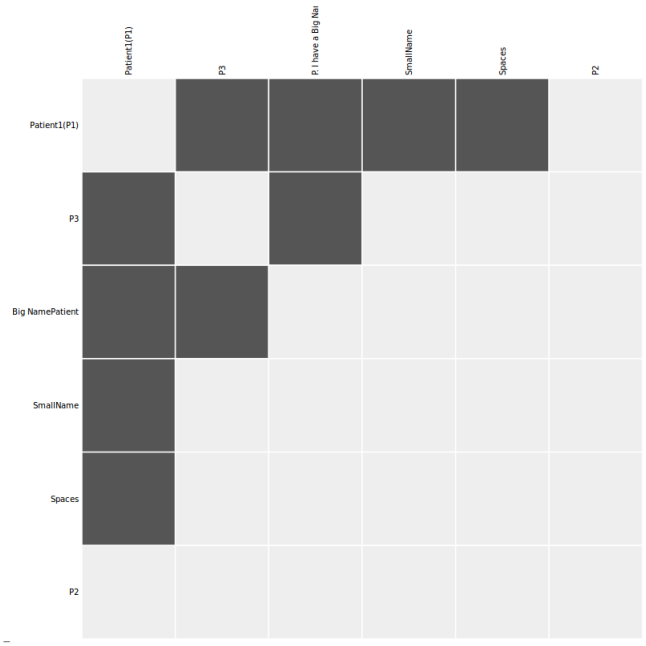


Figure 4.8: Matrix Visualization.

4.7.3 Bug Report

In the project there were minor issues that were to be addressed:

1. Alchemy.js does not render a few edges on screen when using web browsers that such as Google chrome. Mozilla Firefox use Gecko which helps to read web contents like JavaScript, html, css etc better than in Google chrome. Thus, it is recommended to run the project on mozilla firefox or sea monkey than on Google Chrome.

2. *Bug Number 472*: It was found when rendering of edges is an bug that was reported in Alchemy.js issues as Bug number : 472 (`https://github.com/GraphAlchemist/Alchemy/issues/472`). We have also reiterated the issue. The issue is that in the graph where the communities are rendered separately will colors the edges that run between two communities are names and weights are assigned but the edge is not rendered.

4.8 Result and Output

Finally the project adopted alchemy.js for the graph representation and Protovis layout for the matrix representation. The errors were verified and necessary tests were performed to ensure that the visualization process runs smoothly. Sample outputs that were obtained by using the application are shown in the figure 4.9, 4.10, 4.11.

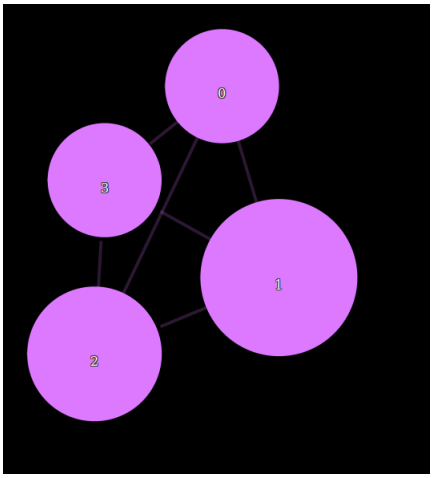


Figure 4.9: Community graph for Karate graph input.

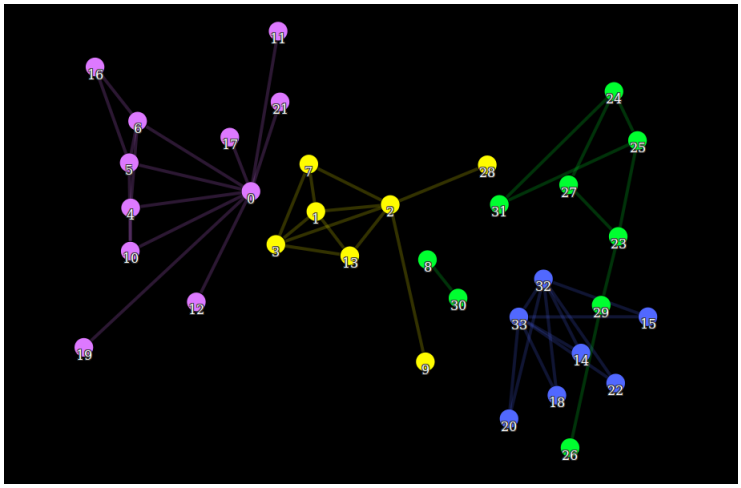


Figure 4.10: Community graph showing all nodes in the community for the Karate graph input

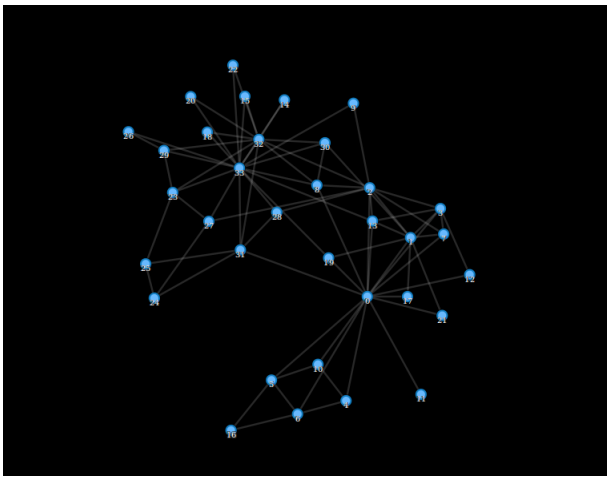


Figure 4.11: Original Karate graph

Chapter 5

Overall System Description

The project uses two complementary technologies of a raw Python code and a JavaScript program that can represent or draw the output of the Python code on screen. This lead to a need to include a new element of a web framework for implementing the Louvain algorithm and creating a JSON object, and JavaScript to represent the graph on a web browser. A web framework is one that aims to remove the overhead associated with common activities performed in the web development.

5.1 Choice of Web.py

An exploration one a few Python web frameworks such as Django, Grok and web.py. A sample application in Django was built to see if Django suits the need of the project. Django was eliminated due to the fact that it was heavier for a simple task that we wish to perform in the project. Web.py was chosen as the web framework for the project as it allowed successful integration of the existing Python code with the web framework over grok.We must note that the larger project at the LARCA group does not use web.py instead another framework called Angular.js. However, Angular.js is designed to be used for large, complex projects, and after some evaluation it was clear that the overhead for this small project did not pay off. Thus, Web.py suffices and was easy to used for the current need of the project that is to perform the integration task.

5.2 Frontend Framework

Bootstrap was an intuitive web front-end frame work that has been implemented in the project. Bootstrap allows to divide the screen into various matrix cubes enabling us to place buttons to run the application. Bootstrap, originally named Twitter Blueprint, was developed by Mark Otto and Jacob Thornton at Twitter as a framework to encourage consistency across internal tools. It is the second most starred project in github and has more than forty thousand forks [17].

5.3 Using the application

The web application thus developed is intutive enough such that even though it performs tasks of computation and integration into the web it looks simple. Minimal and most necessary elements have been implemented. In the following we describe the entire working flow of the Web Application:

1. **"Login Page"** In this part a simple login password is described.

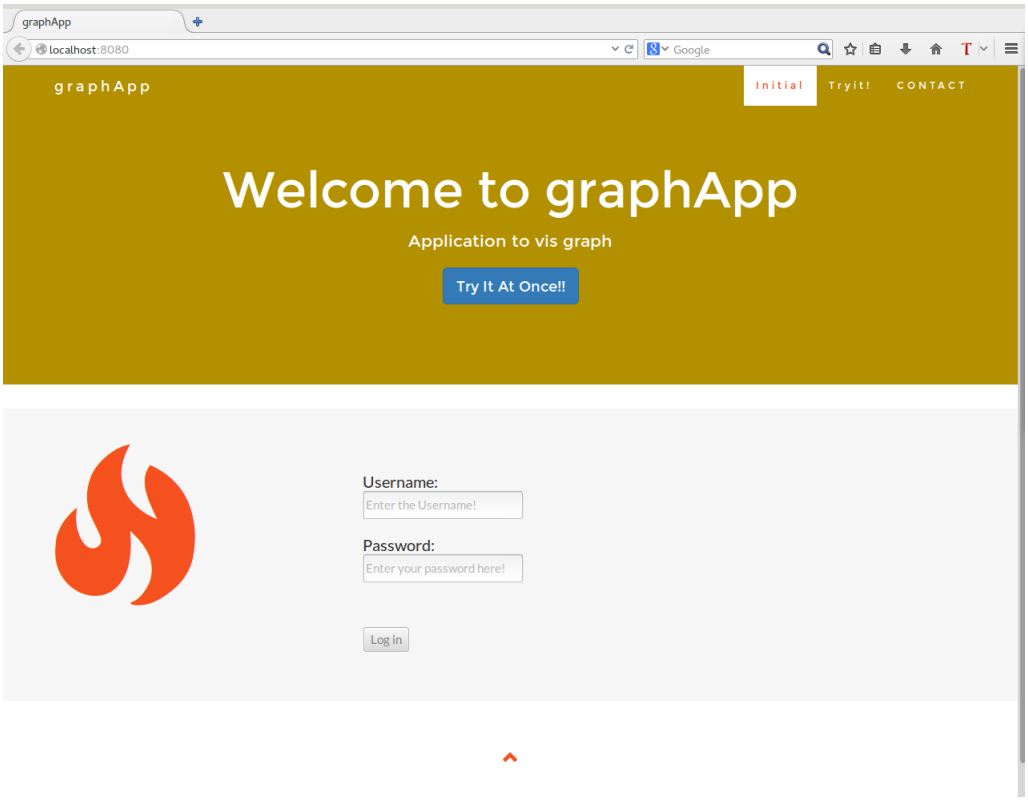


Figure 5.1: Landing page containing login

2. **File Loader Page** After logging in the user lands in a file loader page where provision has been provided to upload a text file containing the adjacency matrix of the graphs are present in the "start" "Destination" or "Start" "Weight" "Destination" format. In this space only numerical values can be accepted for supporting the simplicity of the background process. Another space to provide a key that maps the names of the node numbers. This helps to present another dimension of seeing the data in the form of names. Viewing the names of the node over nodes helps the user to deepen his vision of analysis.

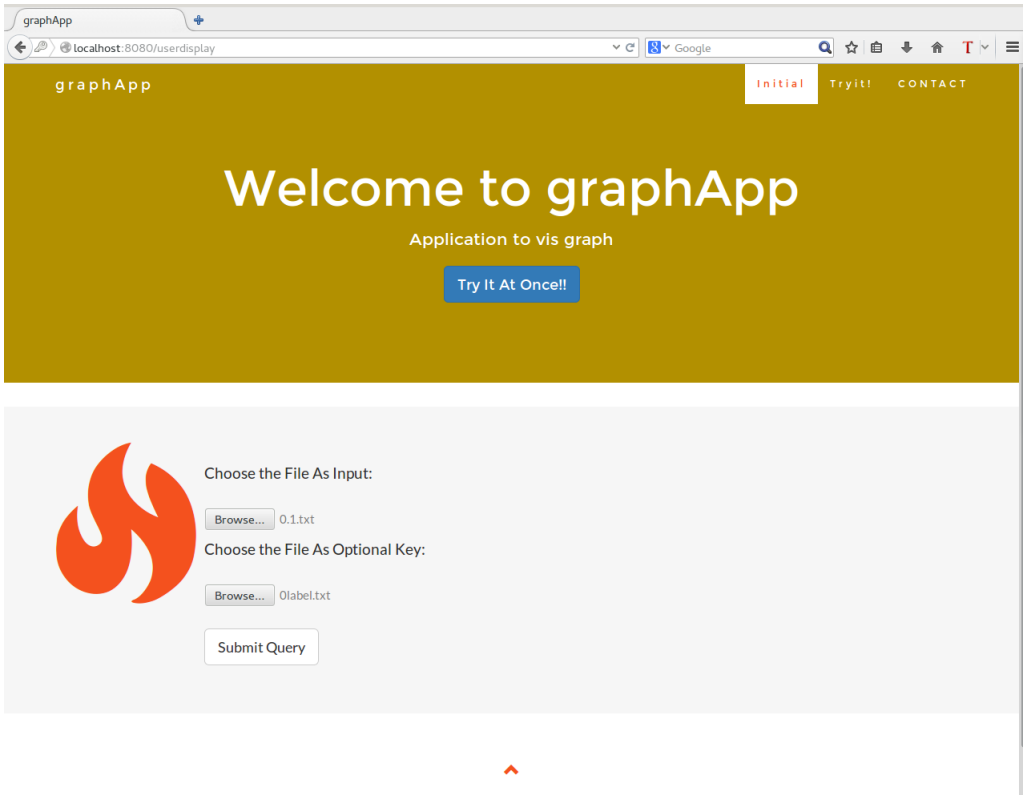


Figure 5.2: After login, the application asks for the data and/or key

3. **Dash Board** In this part of the web application the user lands in a page that presents to the user 4 different variety of visual representation of the input data : Community graphs (Diplaying the links between various communities), Full graph split into communities represented with different colors, Original graph input(Helps to see if the input was proper) and a Matrix view of the graph

4. **On Click Viz.** On clicking on the visualization the user wants to take the web aplication switches to the graph that the user has clicked.

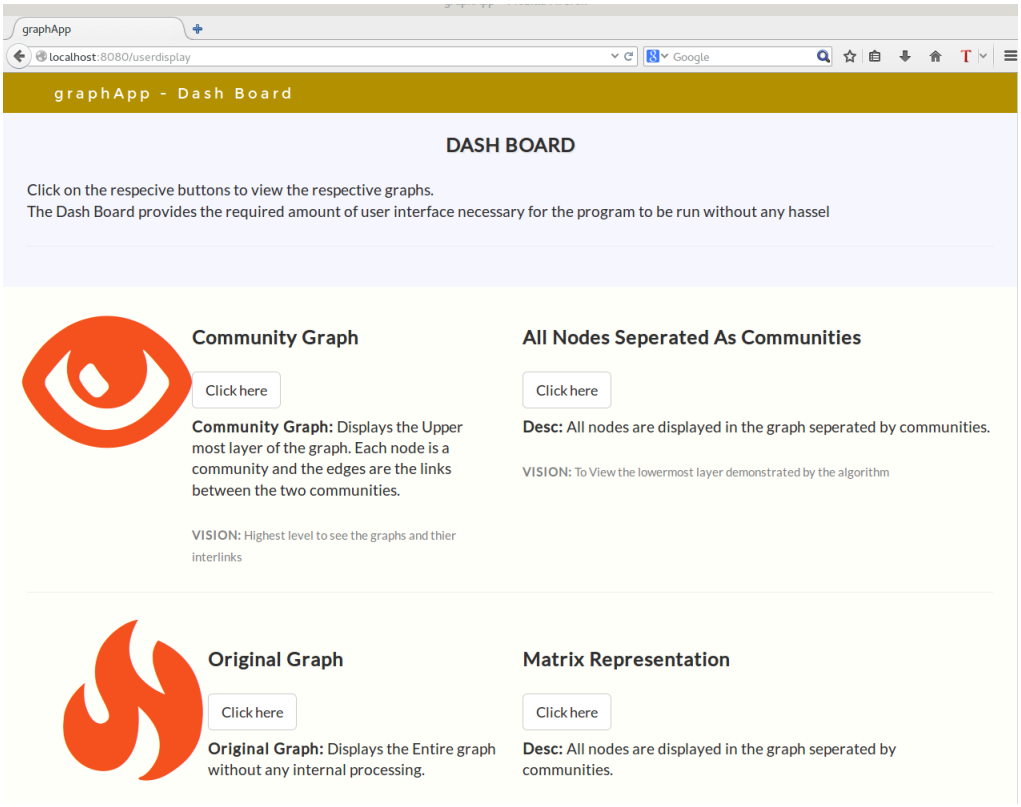


Figure 5.3: This Dash board is displayed after the computation process has been completed.

5. **Error Page** In case of mismatch in the format or the password was wrong or the visualization is not possible due the screen size or the browser is not able to handle the large JSON object Errors pages have been genrated to couter act on exceptions.

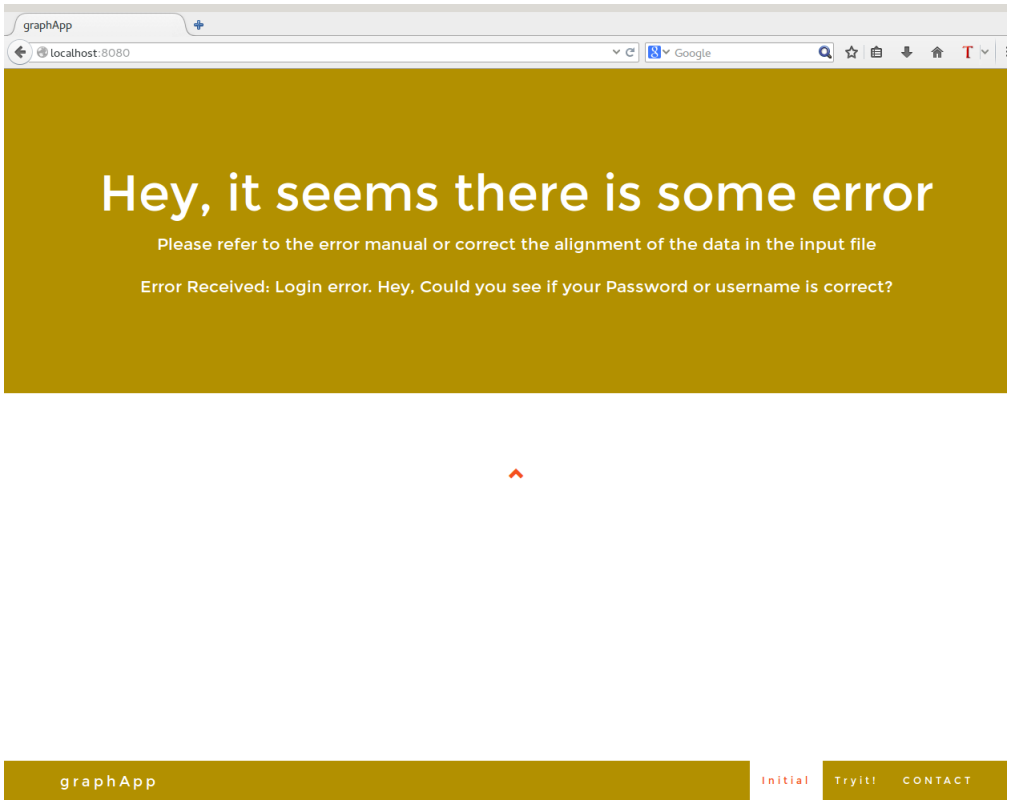


Figure 5.4: Error page showing the appropriate error message

5.4 Implementation Benefits and Drawbacks

5.4.1 Benefits

1. The input given can be a weighted or an unweighted graph. The web input is still accepted and the output is generated.

2. Key labels can be assigned to node numbers. This file if not given the program uses the node number given as id in the input file.

3. Whether the input is a large graph such as Facebook graph or is a small one such as karate graph the community screen displays the community graph in a size that is proportional with the entire size of the graph on one screen.

4. Error management involves a separate class. Hence exact errors can be mentioned and presented to the user in an elegant web interface.

5.4.2 Drawbacks

1. Very large graph needs a bigger screen for the entire graph to be displayed.

2. To display such large graphs a browser that could handle large JSON objects is needed along with bigger screen size

Chapter 6

Conclusion and Future Works

In this section the final conclusion of the project, goals achieved, benefits the project presents to the community and future works are presented.

6.1 Goals Achieved

In the beginning of the project four major goals were set to be achieved. THe following would enlist in the same order on it s achievement.

1. In the project we have surveyed a few algorithms that aim in community finding for instance Combo and Louvain keeping in mind that the input is taken from health care domain.

2. Louvain Community detection algorithm was chosen for community detection and for visualization Alchemy.js and Protovis were selected after considering the input and a few state of the art algorithms.

3. The algorithms and frameworks thus found were implemented and tests were conducted for finding the effciency of the algorithms.

4. A GUI implementing Web.py and Bootstrap was created combining the visualization and the computation.

Thus, the project succesfully achieved the tasks that were set in the early stages of the project.

6.2 Benefits to the community

The system can be used in places where there is difficulty in visualization of a very complex landscape of data such as medical domain. In the medical domain a patient can be a vector of diseases and visualization of such patients (patients graph–which shows relations of how two patients are similar, a graph in which patient-patient edge weight is the similarity value) would be useful for analysing and predicting the disease landscape of a region and in turn multiple regions. In place of patients if disease/diagnosis graphs can be loaded patterns of occurrence of disease or method of diagnosis can be intuitive from the community that are thus formed.

6.3 Future Works

In the span of five months we were able to build a basics project by comparing, contrasting, including the one that the directors suggested and choosing the one that is simple and works well. In this project I would like to suggest a few improvements that we would have done given more time. We would like to enumerate on that:

1. In the Algorihthm part :
    (a) The order in which the information is presented can affect the computation of the Louvain community detection algorithm. Hence the problem of finding specific heuristics to solve this ordering can improve the Louvain algorithm computation time.
    (b) The project relies fully on Louvain. One can speculate on whether modularity is the only measure that exists. Thinking about a completely new measure would be interesting.
2. In the visualization Module:
    (a) In the current project the community graph is presented separately from the main graph. A zoom effect can be introduced to zoom into to community graph to reach different levels of hierarchy.
    (b) Double-click on a node to fade out all but its immediate neighbours.

(c) We deal with large graphs thus it would be nice to have some search functionalities. jQuery can be used to create an autocompleting search box that can be featured on the graph display page which can search the name of the patient or the treatment that is needed.

(d) Fish eye can be introduced. Since alchemy.js is a framework that runs on d3 it can easily be extended o d3. Hence the fish-eye module can help to view every node along with it's neighbours in a more expanded format.

(e) Developing an editor which is a tool to dynamically add nodes and edges after the graph visualization has taken place. D3 has a "forced editor" layout and "Directed forced editor" layout sample which could be useful for further study.

3. In the Overall structure:

   (a) The current dash board is before the graph board. The Dash board can be included in the graph board itself to avoid switching back and front in the web application.

   (b) A simple Database can be set-up for storing passwords and improvisation on sign-ins can deliver a better personalised user experience. The project provides provision for password input which could be used as a starting point to build on further.

6.4 Availability and requirements

1. **Project Name**: Graph and matrix algorithms for visualizing high dimensional

2. **Project Homepage**: `https://github.com/abhinavsv3/webproject`

3. **Operating System**: Platform Independent. Preferably Unix–like operating system

4. **Programming Language**: Python 2.7

5. **Other Requirements** : Alchemy.js, Python Packages, Web.py

6.5 Conclusion

This is one of the greatest project experience.

6.6 Personal Conclusion

The project has made my mind very innovative. I can proudly call myself an engineer. My directors, Prof.Ricard and Prof. Marta gave me a freedom to think freely and understand the project and made me do the project the way I have analysed it. This made me develop an new characteristics of learning stuff in the fly and made me feel enthusiastic about working on projects that I have little knowledge on. I am sure given any project I can now make innovation and work hard to bring the project to a better light.

Bibliography

[1] Albert-László Barabási. Scale-free networks: a decade and beyond. *science*, 325(5939):412–413, 2009.

[2] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[3] Nikos Bikakis, John Liagouris, Maria Krommyda, George Papastefanatos, and Timos Sellis. graphvizdb: A scalable platform for interactive large graph visualization.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[5] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[6] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2009.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[8] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. Maximizing modularity is hard. *arXiv preprint physics/0608255*, 2006.

[9] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[10] David Emms, Edwin R Hancock, Simone Severini, and Richard C Wilson. A matrix representation of graphs and its spectrum as a graph invariant. *Electr. J. Comb*, 13(1), 2006.

[11] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

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

[13] Conrad Lee and Pádraig Cunningham. Benchmarking community detection methods on social media data. *arXiv preprint arXiv:1302.0739*, 2013.

[14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[15] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[16] Julien Odent and Michael Saint-Guillain. Automatic detection of community strucures in networks, November 26,2012.

[17] Mark Otto and Jacob Thornton. Bootstrap.

[18] Graciously Kharumnuid Swarup Roy. Effectiveness of javascript graph visualization libraries in visualizing gene regulatory networks (grn).

[19] Pratha Sah, Lisa O. Singh, Aaron Clauset, and Shweta Bansal. Exploring community structure in biological networks with random graphs. *BMC Bioinformatics*, 15(1):1–14, 2014.

[20] Stanislav Sobolevsky, Riccardo Campari, Alexander Belyi, and Carlo Ratti. General optimization technique for high-quality community detection in complex networks. *Physical Review E*, 90(1):012811, 2014.

[21] M. Zamora, M. Baradad, E. Amado, S. Cordomí, E. Limón, J. Ribera, M. Arias, and R. Gavaldà. Characterizing chronic disease and polymedication prescription patterns from electronic health records. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–9, Oct 2015.