CRM Master Research Projects

Clustering Algorithms for Anti-Money Laundering Using Graph Theory and Social Network Analysis

Author: Abhishek Awasthi

Advisor: Roc Alabern
Lluís Alsedà

A Master Thesis Project

on

Clustering Algorithms for Anti-Money Laundering Using Graph Theory and Social Network Analysis

Submitted for the fulfillment of the Master of Science degree in Mathematical Modeling in Engineering from Autonomous University of Barcelona under the Erasmus Mundus MathMods Master Program

by

Abhishek Awasthi
MathMods 2010/2012

Supervisors:
Mr. Roc Alabern (AIA)
Prof. Lluis Alseda (UAB)

Acknowledgement

I would like to express my warm thanks to all the people who supported and guided me throughout my thesis work. I am sincerely and heartily grateful to my advisors; Prof. Alseda and Roc Alabern, for their valuable suggestions and guidance. I am sure it would have not been possible without their help. Besides, I would like to thank AIA for providing me an opportunity to work with them and provide me a good working environment. I am also thankful to CRM for providing me the support to complete my thesis successfully.

Special thanks to Prof. Alabert for his immense support without which this work would not have been possible.

Abhishek Awasthi

Abstract

HEMOLIA (a project under European community's 7th framework programme) is a new generation Anti-Money Laundering (AML) intelligent multi-agent alert and investigation system which in addition to the traditional financial data makes extensive use of modern society's huge telecom data source, thereby opening up a new dimension of capabilities to all Money Laundering fighters (FIUs, LEAs) and Financial Institutes (Banks, Insurance Companies, etc.). This Master-Thesis project is done at AIA, one of the partners for the HEMOLIA project in Barcelona. The objective of this thesis is to find the clusters in a network drawn by using the financial data. An extensive literature survey has been carried out and several standard algorithms related to networks have been studied and implemented. The clustering problem is a NP-hard problem and several algorithms like K-Means and Hierarchical clustering are being implemented for studying several problems relating to sociology, evolution, anthropology etc. However, these algorithms have certain drawbacks which make them very difficult to implement. The thesis suggests (a) a possible improvement to the K-Means algorithm, (b) a novel approach to the clustering problem using the Genetic Algorithms and (c) a new algorithm for finding the cluster of a node using the Genetic Algorithm.

Table of Contents

List of Figures

List of Tables

Introduction

Cluster analysis divides data into groups that are meaning well connected communities in a graph. Clusters are meaning groups of objects or entities of the graph which share common characteristics. Cluster analysis has long played an important role in a wide variety of fields: social sciences, biology, statics, pattern recognition, information retrieval, machine learning etc. This work provides an insight to the approach of detecting Money-Laundering using cluster analysis. The idea behind detecting Money-Laundering is to draw a graph of the financial data of the entities and find the clusters in the graph.

We start with a good literature survey of *Graph Theory, Spectral Graph Theory* and *Social Network Analysis (SNA)*. Some standard and useful algorithms which use the above topics have been described e.g. Page-Rank, Hub & Authority algorithm. This lays a good foundation to study the cluster analysis. Metrics' of social network analysis play an important role in detecting the clusters in a graph. They have been explained in chapter 2 in detail. A measure to study the fitness of the clusters in a graph (*Modularity*) has been explained in detail.

In the next chapters we describe some standard algorithms like K-Means and hierarchical clustering algorithm. The chapter on K-Means also describes the use of one of the SNA metric (betweenness) so as to improve the K-Means algorithm with respect to finding the number & centers of clusters. A recent algorithm named SHRINK algorithm has also been explained and implemented. SHRINK algorithm also works on the concept of modularity based on structural similarity (explained in Chapter 6). The later part of this thesis discusses the most important part of this work: implementation of Genetic algorithm for the clustering problem. Most of the standard clustering algorithms work with some input from the user such as the number of clusters or the possible centers of the clusters, while some algorithms are extremely expensive in terms of the time & space complexity of the algorithms. To overcome these issues chapter 7 describe the implementation of genetic algorithms with island model to find the clusters in a graph with a very good accuracy and efficiency. The last chapter describes a genetic clustering algorithm to find the cluster to which a given particular node belongs. Finding the cluster of a node is particularly important while detecting Money-Laundering. Usually, the FIUs (Financial Intelligence Units) have details of some suspicious nodes which are most prone to cause money laundering. In such a scenario finding all the clusters could be an expensive and unnecessary job. Hence the genetic clustering algorithm for a given node comes in very useful.

Chapter – 1

Graph Theory & Spectral Graph Theory

1.1 Introduction

Many real-world situations can conveniently be described by means of a diagram consisting of a set of points together with lines joining certain pairs of these points. For example, the points could represent people, with lines joining pairs of friends; or the points might be communication centres, with lines representing communication links. Notice that in such diagrams one is mainly interested in whether two given points are joined by a line; the manner in which they are joined is immaterial. A mathematical abstraction of situations of this type gives rise to the concept of a *Graph.*

In mathematics and computer science, **graph theory** is the study of *graphs*: mathematical structures used to model pair-wise relations between objects from a certain collection. A "graph" in this context refers to a collection of vertices or 'nodes' and a collection of *edges* that connect pairs of vertices. A graph may be *undirected*, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be *directed* from one vertex to another; see graph (mathematics) for more detailed definitions and for other variations in the types of graphs that are commonly considered. The graphs studied in graph theory should not be confused with "graphs of functions" and other kinds of graphs. Graphs are one of the prime objects of study in Discrete Mathematics [1].

A graph G is an ordered pair $(V(G), E(G))$ consisting of a set $V(G)$ of vertices and a set $E(G),$ disjoint from $V(G),$ of edges, together with an incidence function Ψ_G that associates with each edge of G an unordered pair of vertices of G. If e is an edge and u and v are vertices such that $\Psi_G(e) = \{u, v\},$ then e is said to join u and v, and the vertices and edges in G by $v(G)$ and $e(G);$ those two basic parameters are called the order and size of G, respectively.

A vertex v is *incident* with the edge e if $v \in e$; then e is an edge at v. The two vertices, incident with an edge are its *end-vertices* or *ends*, and an edge *joins* its ends. An edge $\{x, y\}$ is usually written as xy (or yx). If $x \in X$ and $y \in Y$, then xy is an X-Y *edge*. The set of all X-Y edges $E(X, Y)$ in a set E is denoted by $E(X, Y)$; instead of $E(\{x\}; Y)$ and $E(X;\{y\})$ we simply write $E(x, Y)$ and $E(X, y)$. The set of all the edges in E at a vertex v is denoted by $E(v)$. Two vertices x, y of G

are *adjacent*, or *neighbours*, if xy is an edge *adjacent* of G. Two edges $e \neq f$ are *adjacent* if they have an end in common. If all *neighbour* the vertices of G are pair-wise adjacent, then G is *complete*. A *complete* graph on n vertices is a K^n; a K^3 is called a *triangle*. Pair-wise non-adjacent vertices or edges are called *independent*. More formally, a set of vertices or edges is *independent* (or *stable*) if no two of its elements are adjacent [2].

Degree (d_v) of a vertex in the graph is the number of adjacent vertices to the given vertex. D, is a *n by n* diagonal degree matrix (n being the number of vertices in the graph), where the diagonal entries of the matrix contain the degree of the nodes. All the non diagonal entries of the matrix are zero.

The next section describes some of the standard methods to depict a graph in a mathematical manner.

1.2 Matrix Structure of Graphs

There are different ways to store graphs in a computer system. The data structure used depends on both the graph structure and the algorithm used for manipulating the graph. Theoretically one can distinguish between list and matrix structures but in concrete applications the best structure is often a combination of both. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access for some applications but can consume huge amounts of memory.

Some important matrix structures proposed for representing a graph are:

a) Incidence Matrix: The graph is represented by a matrix of size |V| (number of vertices) by |E| (number of edges) where the entry contains the edge's endpoint data. Let G be a graph, with *n* vertices, *m* edges and without self-loops. The incidence matrix I of G is an *n* x *m* matrix $I = [a_{ij}]$ whose n rows correspond to the n vertices and the m columns correspond to m edges such that:

$$a_{ij} = \begin{cases} 1, \; \textit{if the edge } m_j \textit{ is incident on the } i^{th} \textit{ vertex} \\ 0, \; \textit{otherwise} \end{cases}$$

It may be also called vertex-edge incidence matrix and is denoted by I(G).

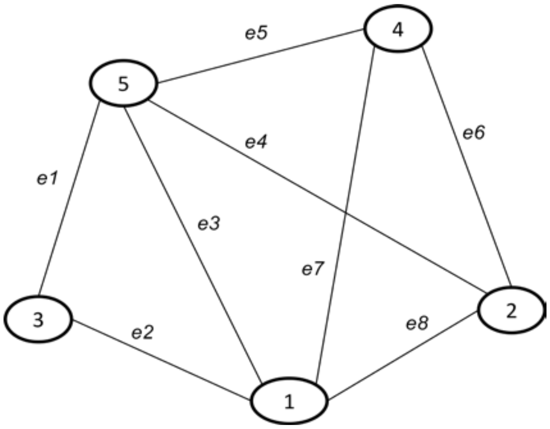


Fig 1.1: Example for Incidence Matrix

As an example, the incidence matrix for the above graph would be given by:

$$I = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

b) Adjacency Matrix: Adjacency matrix is a n by n matrix *Adj,* where n is the number of vertices in the graph. If there is an edge from a vertex x to vertex y, then the element w_{ij} is the edge weight from vertex x to y. Let us consider the same graph with edge-weights

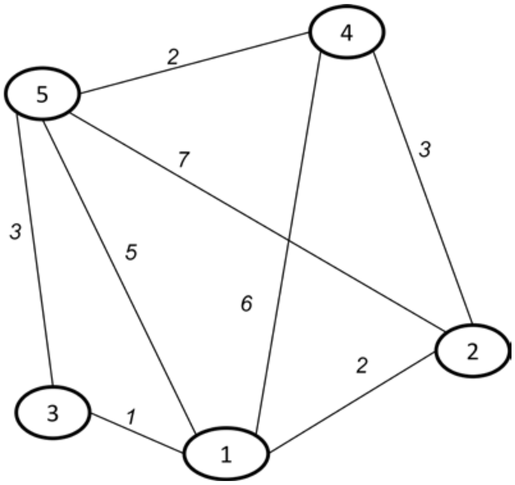


Fig 1.2: Example for the Adjacency Matrix

For a graph with n nodes, the adjacency matrix Adj of G is an n x n matrix $Adj = [w_{ij}]$ where:

$$w_{ij} = \begin{cases} edge\ weight\ of\ the\ edge\ from\ node\ i\ to\ j \\ 0,\ if\ \ no\ edge\ from\ node\ i\ to\ j \\ 0,\ if\ \ i = j \end{cases}$$

Hence, the matrix Adj for the above graph would be:

$$Adj = \begin{bmatrix} 0 & 2 & 1 & 6 & 5 \\ 2 & 0 & 0 & 3 & 7 \\ 1 & 0 & 0 & 0 & 3 \\ 6 & 3 & 0 & 0 & 2 \\ 5 & 7 & 3 & 2 & 0 \end{bmatrix}$$

c) Laplacian Matrix: Laplacian matrix is defined by D-A, where D is the diagonal degree matrix and A is the un-weighted adjacency matrix. Laplacian matrix explicitly contains both the adjacency information and the degree information.

For fig 1.2, the diagonal matrix D, the un-weighted adjacency matrix A and the Laplacian matrix $L = D\text{-}A$, are given by:

$$D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad L = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 3 & 0 & -1 & -1 \\ -1 & 0 & 2 & 0 & -1 \\ -1 & -1 & 0 & 3 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{bmatrix}$$

1.3 Application of Graph Theory

Graphs are among the most ubiquitous models of both natural and human-made structures. They can be used to model many types of relations and process dynamics in physical, biological and social systems. Many problems of practical interest can be represented by graphs. In computer science, graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc. One practical example: The link structure of a website could be represented by a directed graph. The vertices are the web pages available at the website and a directed edge from page A to page B exists if and only if A contains a link to B. A similar approach can be taken to problems in travel, biology, computer chip design, and many other fields. The development of algorithms to handle graphs is therefore of major interest in computer science. There, the transformation of graphs is often formalized and represented by graph rewrite systems. They are either directly used or properties of the rewrite systems (e.g. confluence) are studied. Complementary to graph transformation systems focusing on rule-based in-memory manipulation of graphs are graph databases geared towards transaction-safe, persistent storing and querying of graph-structured data.

Graph-theoretic methods, in various forms, have proven particularly useful in linguistics, since natural language often lends itself well to discrete structure. Traditionally, syntax and compositional semantics follow tree-based structures, whose expressive power lies in the Principle of Compositionality, modeled in a hierarchical graph. Within lexical semantics, especially as applied to computers, modeling word meaning is easier when a given word is understood in terms of related words; semantic networks are therefore important in computational linguistics. Still other methods in phonology (e.g. Optimality Theory, which uses lattice graphs) and morphology (e.g. finite-state morphology, using finite-state transducers) are common in the analysis of language as a graph.

Graph theory is also used to study molecules in chemistry and physics. In condensed matter physics, the three dimensional structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms. In chemistry a graph makes a natural model for a molecule, where vertices represent atoms and edges bonds. This approach is especially used in computer processing of molecular structures, ranging from chemical editors to database searching. In statistical physics, graphs can represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems [3].

Graph theory is also widely used in sociology as a way, for example, to measure actors' prestige or to explore diffusion mechanisms, notably through the use of social network analysis software. Likewise, graph theory is useful in biology and conservation efforts where a vertex can represent regions where certain species exist (or habitats) and the edges represent migration paths, or movement between the regions. This information is important when looking at breeding patterns or tracking the spread of disease, parasites or how changes to the movement can affect other species. In mathematics, graphs are useful in geometry and certain parts of topology, e.g. Knot Theory. Algebraic graph theory has close links with group theory. A graph structure can be extended by assigning a weight to each edge of the graph. Graphs with weights, or weighted graphs, are used to represent structures in which pair-wise connections have some numerical values. For example if a graph represents a road network, the weights could represent the length of each road. A digraph with weighted edges in the context of graph theory is called a network. Network analysis has many practical applications, for example, to model and analyze traffic networks. Applications of network analysis split broadly into three categories:

1. First, analysis to determine structural properties of a network, such as the distribution of vertex degrees and the diameter of the graph. A vast number of graph measures exist, and the production of useful ones for various domains remains an active area of research.
2. Second, analysis to find a measurable quantity within the network, for example, for a transportation network, the level of vehicular flow within any portion of it.
3. Third, analysis of dynamical properties of networks.

The next sections describe two standard algorithms based on graph theory and spectral graph theory: Page-Rank and Hub & Authority Algorithms.

1.4 Page Rank Algorithm

Page Rank algorithm was proposed by Lawrence Page and Sergey Brin, to priorities the web-pages with respect to their relevance and connection with other relevant pages. The general idea of Page Rank algorithm is like the academic citation. If an academic paper has been reference by many other papers, it means this paper has valuable information for certain topics. On the hand, if it has never been referred to or just a few, it means that the paper may not have enough good information for related topics. In this section, we will discuss how the Page Rank algorithm works.

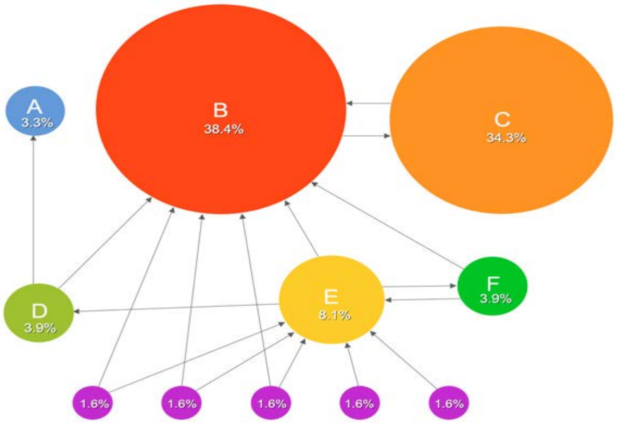


Fig 1.3: Page Ranks for a web-network

Let $PR(p_i)$ denote the pager rank of be page in a given web network; then the Page Rank formula can be stated as given below:

$$PR(p_i) = \frac{1-d}{N} + d * \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where p_i is the page of which the ranking is sought, $M(p_i)$ is the set of pages that link to p_i, $L(p_j)$ is the number of links from page p_j and d is the residual probability, $0<d<1$. The residual probability is considered to take care of those pages which have no forward links (page A in the above graph). According to Brin & Page, they have done many experiments and they suggest the value of d=0.15. [4, 5]

The above equation is recursive but it may be computed by starting with any set of ranks and iterating the computation until it converges. The convergence of the equation can be proved by representing the above equation in matrix form. The above equation is a probability distribution which can also be thought of as modeling the behavior of a *Markov Chain,* with some random initial distribution. Hence the page rank values are nothing but the *stationary distribution* of the above equation represented in a matrix form.

$$PR = \frac{d}{N} + (1-d) * Adj$$

The above equation is the matrix representation of the iterative equation; N is the number of pages (or nodes) in the network, *Adj* is the row stochastic adjacency matrix. Additionally, all the rows of *Adj* with only zeros are replaced by *1/N.* A small example below explains it better:

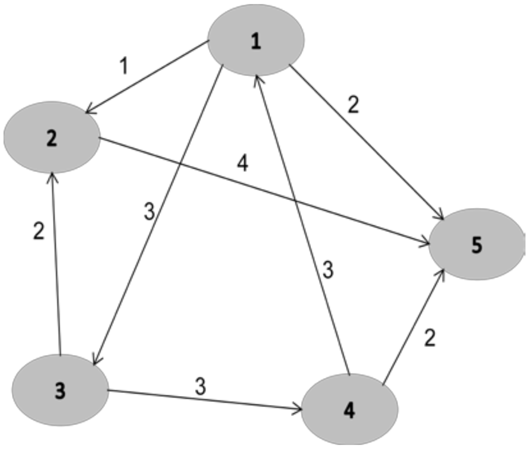


Fig 1.4: Example for Page-Rank Algorithm

Then, the adjacency matrix A and the row stochastic matrix for the above graph are given as:

$$A = \begin{bmatrix} 0 & 1 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 2 & 0 & 3 & 0 \\ 3 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad Adj = \begin{bmatrix} 0 & 1/6 & 3/6 & 0 & 2/6 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 2/5 & 0 & 3/5 & 0 \\ 3/5 & 0 & 0 & 0 & 2/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

Hence PR is an irreducible and aperiodic transition probability matrix, which possesses a unique *stationary distribution.* This stationary distribution of the PR is the page rank vector for all the nodes present in the graph. [6]

The table below shows the page ranking values for the above graph (fig 1.2).

Nodes	1	2	3	4	5
PR	0.1725	0.1656	0.1600	0.1683	0.3335

Table 1.1: Page Rank Values

As we can see, the page rank value for node 5 is the highest since it is directed to by several nodes.

1.5 Hub & Authority: HITS Algorithm

HITS algorithm was proposed by Jon M. Kleinberg. The main objective of this algorithm is to produce the ordering of the search results i.e. to rank the search results by their relevance to the search query and present the results so that the most relevant results are presented before the rest. HITS is a link based algorithm and unlike the Page Rank algorithm, it uses both the in-degree and out-degree of a node to determine its final ranking. Hubs and authorities exhibit a mutually reinforcing relationship. A good hub is a page that points to many good authorities, and, a good authority is a page that is pointed to by many good hubs. This relationship helps us overcome the difficulties faced in the situations where a web page, that is highly relevant to the search query, does not contain the query text. Earlier algorithms like google might not return such pages but HITS, by utilizing this relationship, will successfully return those [12]. This is because the hub pages tend to pull together authorities on a common topic. The figure below depicts the hub and authority nodes in a small web-pages network.

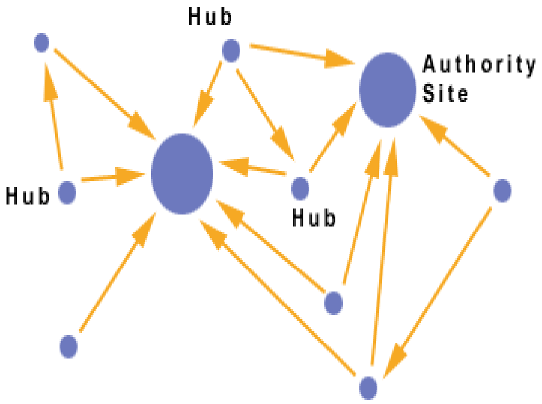


Fig 1.5: Hub and Authority pages in a Web-Network

The mathematically hub and authority values can be represented by its very definition i.e. good hub pages point towards good authority pages and good authority pages are pointed to by good hub pages. Hence we can write two coupled equations as:

$$\left\{ \begin{array}{l} h(v) \rightarrow \sum\limits_{v \rightarrow y} a(y) \\ a(v) \rightarrow \sum\limits_{y \rightarrow v} h(y) \end{array} \right\}$$

The above pair of equation shows the mutual relationship between hub and authority pages; *h(v)* is the hub value of page *v*, *a(v)* is the authority value of page *v* and *y* are all the other pages present in the network. The summation in the above equations can be removed by using the adjacency matrix, which can further give us two independent equations for hub and authority values for the graph.

$$\left\{ \begin{array}{l} \vec{h} \rightarrow A\vec{a} \\ \vec{a} \rightarrow A^T\vec{h} \end{array} \right\} \quad \rightarrow \quad \left\{ \begin{array}{l} \vec{h} \rightarrow AA^T\vec{h} \\ \vec{a} \rightarrow A^TA\vec{a} \end{array} \right\}$$

Kleinberg, proved that the values can be calculated by iterating the last two equations till the values converge. The proof of convergence can be found in the original paper []. Hence, with a random initial choice of values for hub and authority, the equations are iterated as given below:

$$\left\{ \begin{array}{l} h^{(n)} = AA^T h^{(n-1)} \\ a^{(n)} = A^T A a^{(n-1)} \end{array} \right\}$$

However, by Perron-Frobenius theorem, hub and authority values calculated using this iterative method over a non-negative matrix (AA^T and A^TA) are also the principle Eigen-vectors for AA^T and A^TA respectively [11].

The Hub and Authority values for the nodes in fig 1.2 are given as:

Nodes	Hub Value	Authority Value
1	0.7866	0.3130
2	1.0000	0.1411
3	0.1037	0.3351
4	0.7347	0.0442
5	0	1.0000

Table 1.2: Hub and Authority Values

It is clear from the graph that node *2* is a good hub as it is pointing towards a good authority (*5*), with the highest edge weight; and the authority value for node 5 is the highest as it is directed towards by many nodes.

Chapter – 2

Social Network Analysis

2.1 Introduction

A **social network** is a social structure made up of individuals (or organizations) called "nodes", which are tied (connected) by one or more specific types of interdependency, such as friendship, kinship, common interest, financial exchange, dislike, sexual relationships, or relationships of beliefs, knowledge or prestige.

Social network analysis views social relationships in terms of network theory consisting of *nodes* and *ties* (also called *edges*, *links*, or *connections*). Nodes are the individual actors within the networks, and ties are the relationships between the actors. The resulting graph-based structures are often very complex. There can be many kinds of ties between the nodes. Research in a number of academic fields has shown that social networks operate on many levels, from families up to the level of nations, and play a critical role in determining the way problems are solved, organizations are run, and the degree to which individuals succeed in achieving their goals. In its simplest form, a social network is a map of specified ties, such as friendship, between the nodes being studied. The nodes to which an individual is thus connected are the **social contacts** of that individual. The network can also be used to measure social capital – the value that an individual gets from the social network. These concepts are often displayed in a social network diagram, where nodes are the points and ties are the lines.

Social network analysis (related to *network theory*) has emerged as a key technique in modern sociology. It has also gained a significant following in anthropology, biology, communication studies, economics, geography, information science, organizational studies, social psychology, and sociolinguistics, and has become a popular topic of speculation and study. People have used the idea of **"social network"** loosely for over a century to connote complex sets of relationships between members of social systems at all scales, from interpersonal to international. In 1954, J. A. Barnes started using the term systematically to denote patterns of ties, encompassing concepts traditionally used by the public and those used by social scientists: bounded groups (e.g., tribes, families) and social categories (e.g., gender, ethnicity) [11].

The shape of a social network helps determine a network's usefulness to its individuals. Smaller, tighter networks can be less useful to their members than networks with lots of loose connections (weak ties) to individuals outside the main network. More open networks, with many weak ties and social connections, are more likely to introduce new ideas and opportunities to their members than closed networks with many redundant ties. In other words, a group of friends who only do things with each other already share the same knowledge and opportunities. A group of individuals with connections to other social worlds is likely to have access to a wider range of information. It is better for individual success to have connections to a variety of networks rather than many connections within a single network. Similarly, individuals can exercise influence or act as brokers within their social networks by bridging two networks that are not directly linked (called filling structural holes).

A mathematical study of a social network can be made using the metrics' involved in the social network analysis. The next section explains these metrics' and their significance.

2.2 Metrics' of Social Network Analysis

Social network analysis [SNA] is the mapping and measuring of relationships and flows between people, groups, organizations, computers, URLs, and other connected information/knowledge entities. The nodes in the network are the people and groups while the links show relationships or flows between the nodes. SNA provides both a visual and a mathematical analysis of human relationships. Management consultants use this methodology with their business clients and call it Organizational Network Analysis [ONA].

To understand networks and their participants, we evaluate the *location of actors in the network*. Measuring the network location is finding the *centrality* of a node. These measures give us insight into the various roles and groupings in a network -- who are the connectors, mavens, leaders, bridges, isolates, where are the clusters and who is in them, who is in the core of the network, and who is on the periphery? Hence, the metrics' of social network analysis are of utmost importance as they guide us to judge the importance and influence of all the nodes in the network [11].

2.2.1 Degree Centrality

Social network researchers measure network activity for a node by using the concept of degrees -- the number of direct connections a node has. Actors who have more ties to other actors may be advantaged positions. Because they have many ties, they may have alternative ways to satisfy needs, and hence are less dependent on other individuals. Because they have many ties, they may have access to, and be able to call on more of the resources of the network as a whole. Because they have many ties, they are often third-parties and deal makers in exchanges among others, and are able to benefit from this brokerage. So, a very simple, but often very effective measure of an actor's centrality and power potential is their degree. If a graph has n vertices, the degree centrality ($CD(v)$) of any vertex v is defined as:

$$CD(v) = \frac{\deg(v)}{n-1}$$

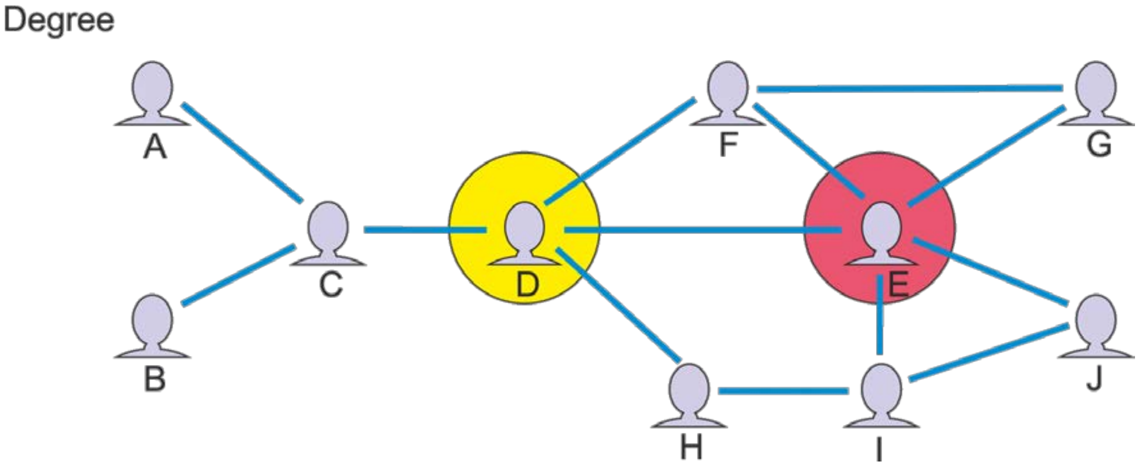


Fig 2.1: Measure of Degree Centrality

The table below shows the degree centrality values of all the vertices in fig 2.1.

Nodes	E	D	C	F	I	G	H	J	A	B
CD	0.556	0.444	0.333	0.333	0.333	0.222	0.222	0.222	0.111	0.111

Table 2.1: Degree Centralities

2.2.2 Closeness Centrality

The measure of closeness of a node to all other nodes is called its closeness centrality. The calculation of closeness centrality requires the shortest distance from any node to all other nodes in the graph.

Given a graph *G* with *n* vertices, $d_G(v,t)$ be the shortest distance between vertices *v* & *t* and V be the set of all the vertices in the graph; then the *closeness centrality* (*CC*) of any node *v* is given by:

$$CC(v) = \frac{\sum_{t \in V \setminus v} d_G(v,t)}{n-1}$$

The table below shows the closeness centrality values for the graph in fig 2.1.

Nodes	A	B	G	J	I	H	C	F	E	D
CC	28.889	28.889	24.444	23.333	22.222	21.111	20.000	18.889	16.667	15.556

Table 2.2: Measure of Closeness Centrality

2.2.3 Betweenness Centrality

The extent to which a node lies between other nodes in the network, called its *betweenness centrality.* This measure takes into account the connectivity of the node's neighbors, giving a higher value for nodes which bridge clusters. The measure reflects the number of people who a person is connecting indirectly through their direct links. [13]

Consider fig 2.1, suppose that vertex G wants to influence vertex B by sending it some information, or make a deal to exchange some resources. But, in order to talk to vertex B, G must go through an intermediary vertices *F, D & C.* Thus as per the above definition the betweenness centralities of vertices *F, D & C* will be greater than that of vertex *B* or *G.*

Let G be graph, V its vertex set; d_{st} be the number of shortest paths between vertex s & t and $d_{st}(v)$ be the number of shortest paths between vertex s & t, that pass through vertex v; then *betweenness centrality* (*CB*) of a vertex *v* is given by:

$$CB(v) = \sum_{s \neq t \neq v \in V} \frac{d_{st}(v)}{d_{st}}$$

The table below shows the betweenness centralities of all the vertices in the previous graph.

Nodes	D	C	E	F	H	I	A	B	G	J
CB	403.33	300.00	253.33	46.667	40.000	36.667	0	0	0	0

Table 2.3: Measure of Betweenness Centrality

As we can see the CB values for vertices *A, B, G and J* are equal to zero. This is because they do not lie in any of the shortest paths in the whole graph. The betweenness centrality index is essential in the analysis of social networks, but costly to compute. Currently, the fastest known algorithms require $\Theta(n^3)$ time and $\Theta(n^2)$ space, where n is the number of actors in the network. Motivated by the fast-growing need to compute centrality indices on large, yet very sparse, networks, new algorithms for betweenness are introduced by Ulrik Brandes [13], which requires $O(n + m)$ space and run in $O(nm)$ and $O(nm + n^2 log\ n)$ time on un-weighted and weighted networks, respectively, where m is the number of links.

Betweenness Centrality is especially important measure to study the clusters in a network. It would be discussed in depth in chapter 5 on KMeans algorithm.

Chapter – 3

Modularity: Graph Clusters

3.1 Introduction

Graph Clustering is an efficient methodology to detect entities in a network or graph, possessing certain similar characteristics. Many networks of interest in the sciences, including social networks, computer networks, and metabolic and regulatory networks, are found to divide naturally into communities or modules. The problem of detecting and characterizing this community structure is one of the outstanding issues in the study of networked systems. One highly effective approach is the optimization of the quality function known as ''Modularity'' over the possible divisions of a network.

Many systems of scientific interest can be represented as networks, sets of nodes or vertices joined in pairs by lines or edges. Examples include the internet and the worldwide web, metabolic networks, food webs, neural networks, communication and distribution networks, and social networks. The study of networked systems has a history stretching back several centuries, but it has experienced a particular surge of interest in the last decade, especially in the mathematical sciences, partly as a result of the increasing availability of accurate large-scale data describing the topology of networks in the real world. One issue that has received a considerable amount of attention is the detection and characterization of community structure in networks, meaning the appearance of densely connected groups of vertices, with only sparser connections between groups (Fig. 3.1). The ability to detect such groups could be of significant practical importance. For instance, groups within the worldwide web might correspond to sets of web pages on related topics; groups within social networks might correspond to social units or communities, etc. Merely finding that a network contains tightly knit groups at all can convey useful information: if a metabolic network were divided into such groups, for instance, it could provide evidence for a modular view of the network's dynamics, with different groups of nodes performing different functions with some degree of independence. [14, 15]

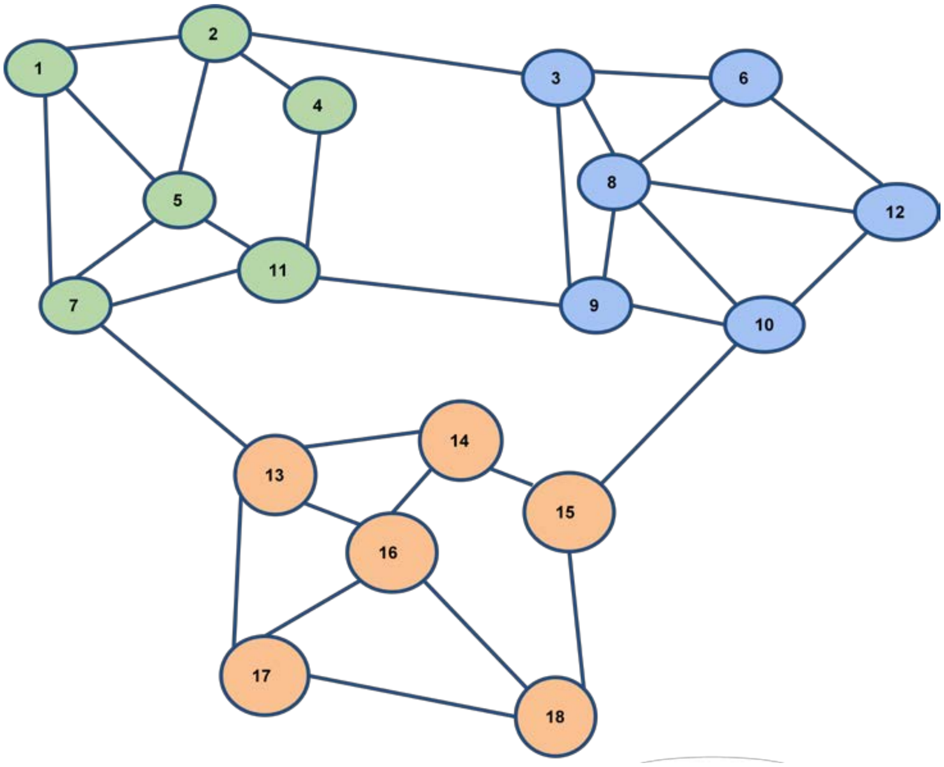


Fig 3.1: Example of a Graph with Clusters

Past work on methods for discovering groups in networks divides into two principal lines of research, both with long histories. The first, which goes by the name of graph partitioning, has been pursued particularly in computer science and related fields, with applications in parallel computing and integrated circuit design, among other areas. The second, identified by names such as block modeling, hierarchical clustering, or community structure detection, has been pursued by sociologists and more recently by physicists, biologists, and applied mathematicians, with applications especially to social and biological networks. It is tempting to suggest that these two lines of research are really addressing the same question, albeit by somewhat different means. There are, however, important differences between the goals of the two camps that make quite different technical approaches desirable.

Community structure detection, by contrast, is perhaps best thought of as a data analysis technique used to shed light on the structure of large-scale network data sets, such as social networks, internet and web data, or biochemical networks. Community structure methods normally assume that the network of interest divides naturally into subgroups and the experimenter's job is to find those groups. The number and size of the groups are thus determined by the network itself and not by the experimenter. Moreover, community structure methods may explicitly admit the possibility that no good division of the network exists, an outcome that is itself considered to be of interest for the light it sheds on the topology of the network.

The next section of this chapter discusses the measure proposed by Newman & Girvan, for detecting the clusters in a graph, called Modularity.

3.2 Modularity

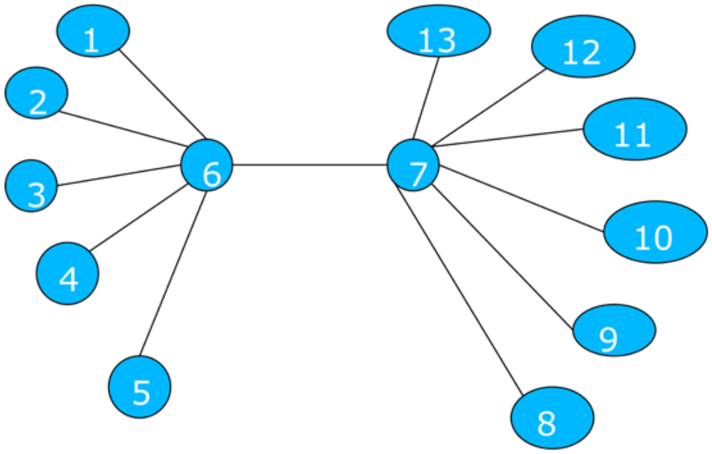
In 2002, Girvan and Newman highlighted the relevance of community structure in complex networks and proposed a method to detect it [18]. This work opened a new scenario that has attracted a great deal of attention in recent years, especially because they identified structures which have meaning; they revealed information about the roles of groups of nodes. This is the case, for example, in the worldwide airports network, the World Wide Web, biological networks, social networks and the Internet, among others. The information revealed by the community structure of real networks can be very valuable and make scientists aware of accuracy and reliability of the method used to detect this substructure. The most important advancement in community detection from the previous hit was given by the same authors, proposing a quality measure, *Modularity* (Q) that allows quantification of the modular structure.

Given a network partitioned into communities or modules, C_i being the community to which node i is assigned, the mathematical definition of modularity is expressed in terms of the weighted adjacency matrix w_{ij}, which represents the value of the weight in the link between the nodes i and j (0 if no link exists), and the strengths $w_i = \Sigma_j w_{ij}$, as,

$$Q = \frac{1}{2w}\sum_{i}\sum_{j}\left(w_{ij} - \frac{w_i w_j}{2w}\right)\delta\left(C_i - C_j\right)$$

where, the kronecker delta function $\delta(C_i ,C_j)$ takes the value 1 if the nodes i and j are in the same module, 0 otherwise, and the total strength is $2w = \Sigma_i w_i$. For unweighted networks, 'w_i' becomes the degree of node i, and 'w' the total number of links of the network. [16, 17, 18]

The modularity of a given partition is the probability of having edges falling within modules in the network minus the expected probability in an equivalent (null case) network with the same number of nodes, and edges placed at random preserving the nodes' strength. The larger the modularity, the better the partitioning is, because the more it deviates from the null case. As a small example, the figure below illustrates the Modularity of a graph with different possible clusters present.



Clusters	Modularity
[1,12,13,5,6,7]; [2,3,4,8,9,10,11]	-0.1701
[1,3,6,8,12]; [2,4,5,7,9,10,11,13]	-0.0139
[1,2,3,4,5,6,7,8]; [9,10,11,12,13]	-0.0868
[1,2,3,4,5,6,7]; [8,9,10,11,12,13]	-0.1250
[1,2,3,4,5,6]; [7,8,9,10,11,12,13]	**0.4132**
[8,3,7,5,12,2]; [1,4,6,9,10,11,13]	-0.1667
[3,6,2,7,1,9,4,5]; [8,10,11,12,13]	-0.0868

Fig 3.2: Modularity values for different clusters

As the above figure suggests the modularity of the graph is highest when the clusters are divided as [1,2,3,4,5,6] &[7,8,9,10,11,12,13].

3.2.1 Resolution Limit of Modularity

Modularity optimization seems, therefore, to be a very effective method to detect communities, both in real and in artificially generated networks. However, Modularity itself has not yet been thoroughly investigated, and only a few general properties are known. For example, it is known that the modularity value of a partition does not have a meaning by itself, but only when compared with the corresponding modularity expected for a random graph of the same size, as the latter may attain very high values due to fluctuations.

Recently, *Fortunato* and *Barthelemy* presented a critical analysis and applicability of Modularity optimization to the problem of community detection [23]. They showed that Modularity contains an intrinsic scale that depends on the total number of links in the network. Modules that are smaller than this scale may not be resolved, even in the extreme case where they are complete graphs connected by single bridges. The resolution limit of modularity actually depends on the degree of interconnectedness between pairs of communities and can reach values of the order of the size of the whole network. The problem of resolution limit of Modularity arises if in a huge graph there are smaller communities present. In that case Modularity can detect them only if they have only a few connections with other communities due to the null case assumption. Some rectifications have been proposed by researchers, suggesting a weighted null-case. However, the resolution limit of Modularity comes into play only if there are clusters of variably different sizes. The resolution limit of Modularity is not accounted for in this work and the graphs considered in this project do not encounter this problem. Moreover the Genetic Algorithm proposed for finding the cluster of a node is not at all affected by the resolution limit of the Modularity. This would be clarified in chapter 8.

With the knowledge of Graph Theory, Spectral Graph Theory, Social Network Analysis & it metrics' and the concept of Modularity; the coming chapters describe some standard and genetic algorithms for Graph Clustering.

3.3 Results: Modularity Values

The table below shows the Modularity value for some graphs, with the number of clusters present.

#	Number of Nodes	Number of Clusters	Modularity
1	13001	13	0.9220
2	14001	14	0.9275
3	12001	12	0.9156
4	1001	10	0.8900
5	2001	20	0.9400
6	3001	30	0.9566
7	5001	25	0.9550
8	6001	30	0.9616
9	4801	24	0.9533
10	7001	70	0.9757
11	7201	72	0.9761

Table 3.1: Modularity Values of some Graphs

Chapter – 4

Hierarchical Clustering

4.1 Introduction

Hierarchical clustering is a sequence of partitions in which each partition is nested into the next partition in the sequence unless all the data points are agglomerated into a single cluster. Hierarchical clustering algorithms can be categorized in three types:

- Single Linkage Clustering
- Complete Linkage Clustering
- Average Linkage Clustering

Hierarchical clustering can also be used to find community structures in a network. The technique again arranges the network into a hierarchy of groups according to a specified weight function. The data can then be represented in a tree structure known as a dendrogram. Hierarchical clustering can either be agglomerative or divisive depending on whether one proceeds through the algorithm by adding links to or removing links from the network, respectively [20, 21].

The components at each iterative step are always a subset of other structures. Hence, the subsets can be represented using a tree diagram, or dendrogram. Horizontal slices of the tree at a given level indicate the communities that exist above and below a value of the weight. In this chapter I would discuss about Single-Linkage Clustering with an illustrative example. The remaining two methodologies work on the same lines as the single-linkage [20].

4.2 The Agglomerative Hierarchical Clustering

Given a graph with N nodes to be clustered, and the N*N distance matrix, the basic process of hierarchical clustering is stated below:

1. Start by assigning each node to a different cluster i.e. if we have N items, we start with N clusters, with each cluster containing just one node. Let the distances between the clusters the same as the distances between the nodes they contain.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now we have one cluster less.

3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster with all the N nodes.

Step 3 can be done in different ways, which is what distinguishes *single-linkage* from *complete-linkage* and *average-linkage* clustering.

In *single-linkage* clustering (also called the *connectedness* or *minimum* method), we consider the distance between one cluster and another cluster to be equal to the shortest distance from any member of one cluster to any member of the other cluster. If the data consist of similarities, we consider the similarity between one cluster and another cluster to be equal to the greatest similarity from any member of one cluster to any member of the other cluster [20].

In *complete-linkage* clustering (also called the *diameter* or *maximum* method), we consider the distance between one cluster and another cluster to be equal to the greatest distance from any member of one cluster to any member of the other cluster [21].

In *average-linkage* clustering, we consider the distance between one cluster and another cluster to be equal to the average distance from any member of one cluster to any member of the other cluster.

This kind of hierarchical clustering is called A*gglomerative* because it merges clusters iteratively. There is also a *divisive* hierarchical clustering which does the reverse by starting with all objects in one cluster and subdividing them into smaller pieces. Divisive methods are not generally available, and rarely have been applied. The next section presents the formal Agglomerative Hierarchical Clustering algorithm.

4.2.1 The Algorithm

The algorithm is an agglomerative scheme that erases rows and columns in the proximity matrix as old clusters are merged into new ones. The N*N proximity matrix is D = [d(i,j)]. The clusters are assigned sequence numbers 1, 2... (n-1) and *iter* is the level of the $iter^{th}$ clustering. A cluster with sequence number m is denoted by (m) and the proximity between clusters (r) and (s) is denoted by d [(r),(s)]. The algorithm is composed of the following steps:

Algorithm

```
begin
    iter =1;
    do {
        Merge two closest clusters (r) & (s); i.e. $d[(r),(s)] = \min d[(i),(j)]$
        Update the Matrix D {
            ▪ delete rows and columns of clusters (r) and (s)
            ▪ add a row and column corresponding to the new cluster.
            ▪ distance between the new cluster, (r,s) and the old cluster (t) is
            defined as:    $d[(t), (r,s)] = \min( d[(t),(r)], d[(t),(s)] )$
        }end
        iter = iter+1;
    } while (iter~=N)
end
```

Let's now see a simple example: a hierarchical clustering of distances in kilometers between some Italian cities. The method used is single-linkage.

Input distance matrix:

	BA	FI	MI	NA	RM	TO
BA	**0**	**662**	**877**	**255**	**412**	**996**
FI	**662**	**0**	**295**	**468**	**268**	**400**
MI	**877**	**295**	**0**	**754**	**564**	**138**
NA	**255**	**468**	**754**	**0**	**219**	**869**
RM	**412**	**268**	**564**	**219**	**0**	**669**
TO	**996**	**400**	**138**	**869**	**669**	**0**

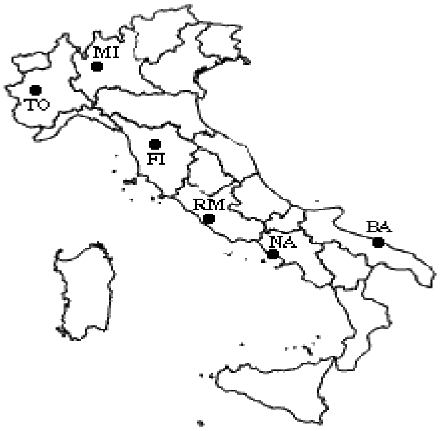


Fig 4.1 (a) : Initial Distance Matrix

The nearest pair of cities is MI and TO, at distance 138. These are merged into a single cluster called "MI/TO". The new sequence number is *iter = 1*. After merging MI with TO we obtain the following matrix:

	BA	FI	MI/TO	NA	RM
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0

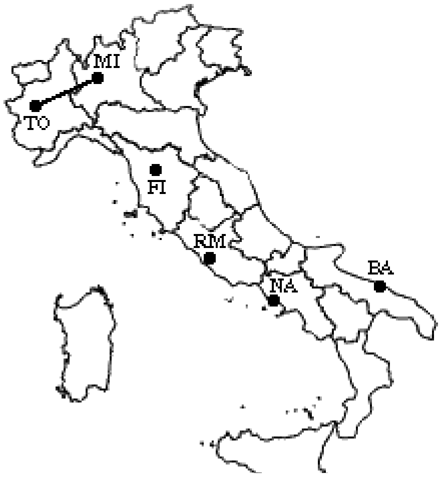


Fig 4.1 (b): First Iteration: Hierarchical Clustering

Then we compute the distance from this new compound object to all other objects. In single link clustering the rule is that the distance from the compound object to another object is equal to the shortest distance from any member of the cluster to the outside object. So the distance from "MI/TO" to RM is chosen to be 564, which is the distance from MI to RM, and so on. For *iter =2*; min d(i,j) = d(NA,RM) = 219, hence we merge NA and RM into a new cluster called NA/RM.

	BA	FI	MI/TO	NA/RM
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0

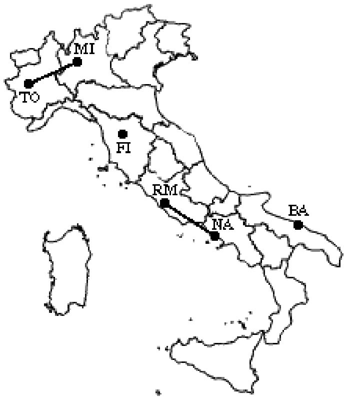


Fig 4.1 (c): Second Iteration: Hierarchical Clustering

For *iter* = *3*, min d(i,j) = d(BA,NA/RM) = 255, hence we merge BA and NA/RM into a new cluster called BA/NA/RM.

	BA/NA/RM	**FI**	**MI/TO**
BA/NA/RM	**0**	**268**	**564**
FI	**268**	**0**	**295**
MI/TO	**564**	**295**	**0**

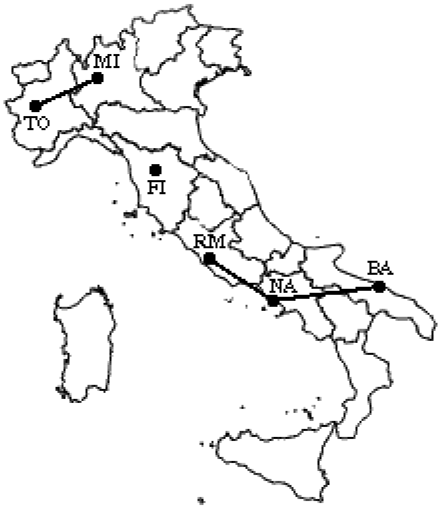


Fig 4.1 (d): Third Iteration: Hierarchical Clustering

For *iter = 4*, min d(i,j) = d(BA/NA/RM,FI) = 268, merge BA/NA/RM and FI into a new cluster called BA/FI/NA/RM.

	BA/FI/NA/RM	**MI/TO**
BA/FI/NA/RM	**0**	**295**
MI/TO	**295**	**0**

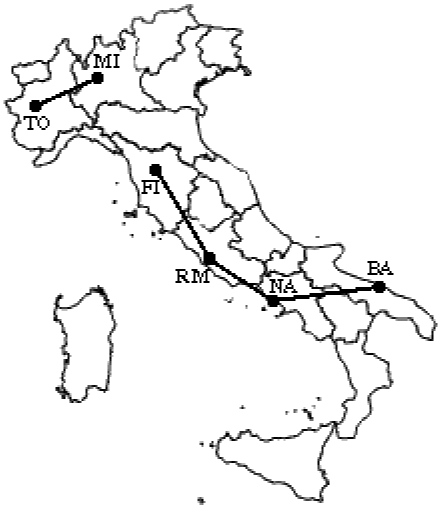


Fig 4.1 (e): Fourth Iteration: Hierarchical Clustering

Finally for *iter = 5*; we merge the last two clusters at level 295. The process is summarized by the following dendrogram (hierarchical tree):

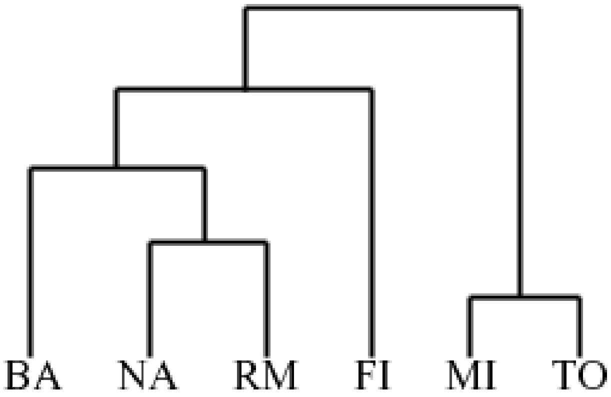


Fig 4.2: Dendrogram: Hierarchical Clustering

Once we have the final dendrogram as shown above, decision is still to be made of the well-knit clusters present in the network. One usual technique is to calculate the Modularity of the graph with clusters. These clusters are obtained by cutting the dendrogram as shown below and the cut with the maximum modularity values (Q1, Q2,…Q5) is the best fit cluster of the graph.

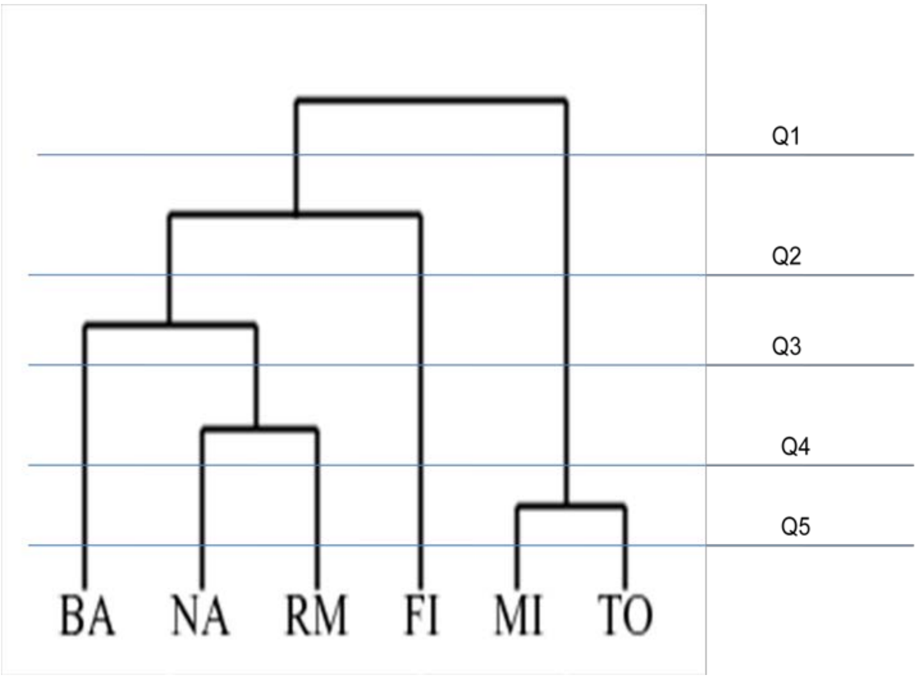


Fig 4.3: Cutting the Dendrogram at each branch

The Hierarchical Clustering is an easy algorithm but has some major issues with its implementation on a large graph. First drawback is that it does not scale well as the time complexity is at least of at least $O(n^2)$. Moreover, all the iterative outcomes have to stored, i.e. the clusters of all sizes; and this implies a space complexity of $O(n)$ at each iteration. Secondly, hierarchical clustering is a greedy algorithm and they can never undo what was done previously [22].

The above explained kind of hierarchical clustering is called agglomerative because it merges clusters iteratively. There is also a divisive hierarchical clustering which does the reverse by starting with all objects in one cluster and subdividing them into smaller pieces. Divisive methods are not generally available, and rarely have been applied.

Chapter – 5

K-Means Algorithm

5.1 Introduction

K-Means [reference] algorithm is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centers, one for each cluster. These centers should be placed in an efficient way as different locations of these initial centers fetches different results. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centre. When no point is pending, the first step is completed and an early grouping is done. At this point we need to re-calculate k new centers as the new centers of the clusters resulting from the previous step. After we have these k new centers, a new binding has to be done between the same data set points and the nearest new centre. Iterations are carried out and as a result the k centers change their locations step by step until no more changes occur. In other words centers do not move any more. The algorithm mainly aims at minimizing an objective function, which will be described in the coming sections [25].

The two important and decisive parameters for K-means algorithm are the number of clusters and the centers of the clusters. Once these parameters are fixed correctly to a considerable tolerance, K-means algorithm is one the most accurate algorithm for the clustering problem. Unfortunately though, guessing the number of clusters and the centers of the clusters in a huge graph is an almost impossible task. Experiments on even some small scale graphs (~50 nodes) show that a random choice of these parameters leads to a non-optimal solution.

In this chapter I am going to propose an improvement for the K-means algorithm in regards to the two parameters:

i) **Number of Clusters**

ii) **Centers of the Clusters**

5.2 The Basic K-means Algorithm

The technique behind the K-means algorithm is very simple and easy to understand. The algorithm begins by first choosing the number of clusters required ***'k'*** and the centers of those clusters. Each point is then assigned to the closest centre, and each collection of points assigned to a centre is a cluster. The centre of each cluster is then updated based on the points assigned to the cluster. This process of assignment and update is repeated until the centers of the clusters do not change their position. The K-means algorithm basically works to minimize an objective function (***J***), which is based on the distance of all the nodes to its cluster centre [25].

$$J = \sum_{j=1}^{K} \sum_{i=1}^{N} d\left(x_i^j, c_j\right)$$

where,

N: Number of Nodes present in the graph

K: Number of Clusters present in the graph

x_i^j: A node '***i'*** in cluster '***j'***

c_j: Centre of Cluster ***j***

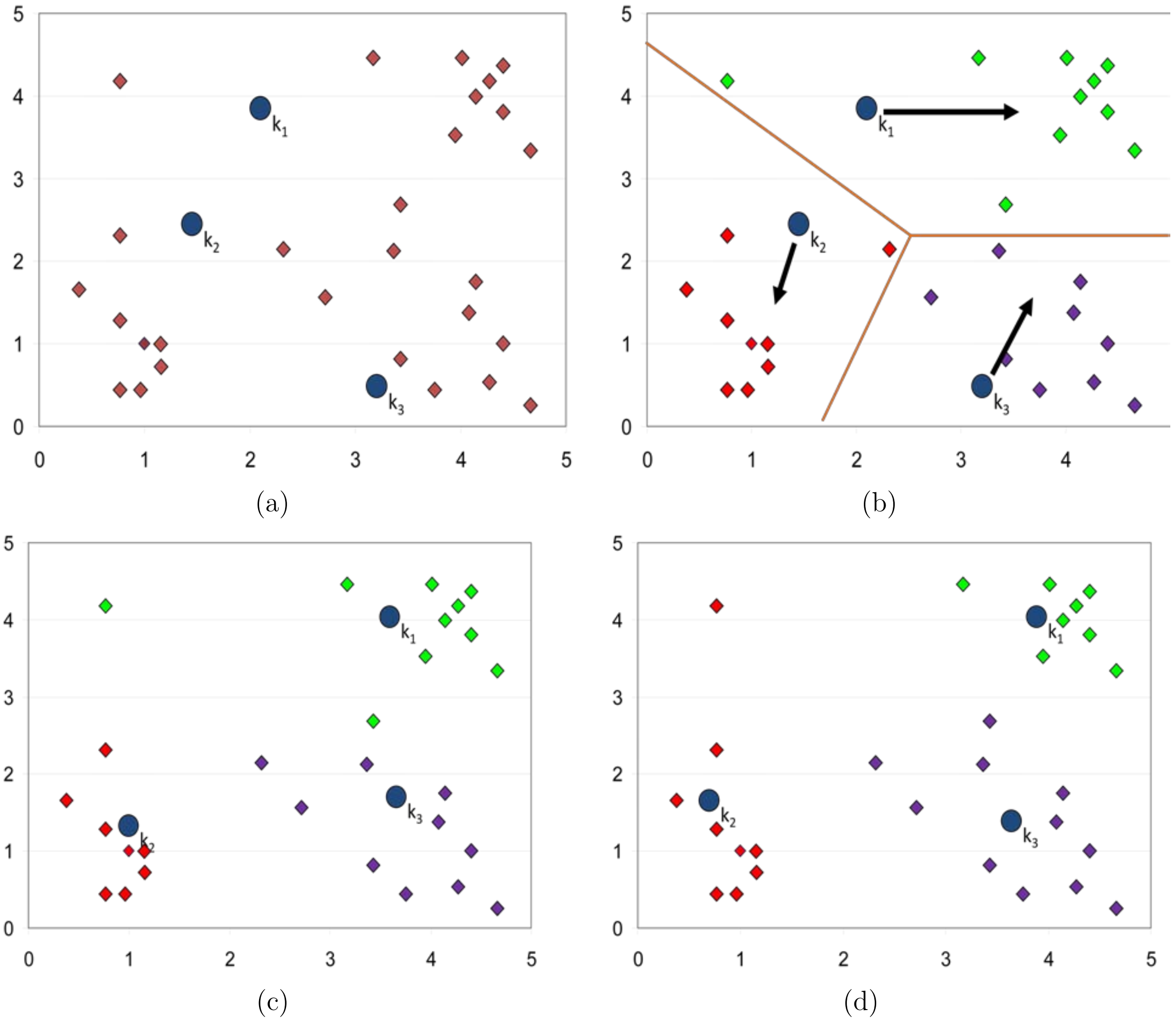
d(x, c): distance between point ***x*** and ***c***

K-means algorithm is formally described below:

ALGORITHM: PSEUDO CODE

```
Select K points as initial centers
do {
        Form K clusters by assigning each point to its closest centre
        Re-compute the centre of the each cluster
    } while (location of centers does not change)
```

The working of the algorithm is straight forward, once the initial K centers are chosen. The steps of the algorithm are illustrated in the next page with the help of a small example.



(a)

(b)

(c)

(d)

Fig 5.1: Steps in K-means algorithm

In the first step, shown in Fig 5.1(a), the centers to the clusters are chosen, equal to the number of clusters required. Once the cluster centers are chosen, the next step assigns all the nodes to one of the centre depending on the distance of the nodes from all the centers, Fig 5.1(b). In the next step the centers of the clusters are re-computed, with reference to the clusters formed in the previous step, Fig 5.1(c). And the last step gives the final result, with the appropriate centers and the clusters, Fig 5.1(d). The algorithm comes to an end as the locations of the cluster centers do not change any further.

But as stated in the beginning, the crucial step in the K-means is the very first step i.e. choosing K cluster centers. A common approach is to choose the initial centers randomly, but the resulting clusters are often poor. Hence the implementation of K-means can come to use only when one has the appropriate number of clusters and the centers. The next section describes two techniques which are often used to overcome the above mentioned problem.

5.3 Choosing Initial Centers

Randomly selected initial centres may be poor. One technique that is commonly used to address the problem of choosing initial centres is to perform multiple runs, each with a different set of randomly chosen initial centres, and then select the set of clusters with the minimum objective function value. While simple, this strategy may not work very well, depending on the data set and the number of clusters sought.

As the number of clusters becomes larger, it is increasingly likely that at least one pair of clusters will have only one initial centre. In this case, because the pairs of clusters are farther apart than clusters within a pair, the K-means algorithm will not redistribute the centres between pairs of clusters, and thus, only a local minimum will be achieved. Because of the problems with using randomly selected initial centres, which even repeated runs may not overcome, other techniques are often employed for initialization [22].

One effective approach is to take a sample of points and cluster them using a hierarchical clustering technique. K clusters are extracted from the hierarchical clustering, and the centres of those clusters are used as the initial centres. This approach often works well, but is practical only if:

- The sample is small, e.g., a few hundred (hierarchical clustering is expensive)
- K is relatively small compared to the sample size.

The following procedure is another approach to selecting initial centres. Select the first point at random or take the centre of all points. Then, for each successive initial centre, select the point that is farthest from any of the initial centres already selected. In this way, we obtain a set of initial centres that is guaranteed to be not only randomly selected but also well separated. Unfortunately, such an approach can select outliers, rather than points in dense regions (clusters). Also, it is expensive to compute the farthest point from the current set of initial centres. To overcome these problems, this approach is often applied to a sample of the points. Since outliers are rare, they tend not to show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small. Also, the computation involved in finding the initial centres is greatly reduced because the sample size is typically much smaller than the number of points. In view of these issues, a random selection or multiple processing of the algorithm does not seem to be a good solution. Hence in the next section, a social network metrics, *Betweenness Centrality* is used to overcome these issues.

5.4 K-means: Betweenness Centrality

As is clear from the above discussion, the initial step of the K-means algorithm is the most vital one. If one has a method to identify the possible centres of the clusters, the K-means algorithm could be an extremely accurate algorithm to solve the hard clustering problem. To identify the centres of the clusters, one should keep in mind that the centre of a cluster is the node which is at shortest distance from all other nodes in the cluster. In other words, the centre of a cluster is a node which lies in the path joining every pair of nodes in the cluster. If in a cluster, a node is visited the most, while travelling from any node to another that is a possible good centre of that cluster. This is where; Betweenness centrality plays a very important role. Betweenness Centrality as discussed in chapter 2, is a metrics which calculates the fraction of shortest paths in the graph on which a particular nodes lies. Hence the nodes with high Betweenness-centrality value are more prone to be the centres of the clusters. But the problem of finding the number of clusters still remains a concern. However, if the clusters in a graph are well defined, it is very likely that there are more edges between the nodes of the same cluster and fewer edges between the nodes of different clusters. In other words, if the centres of the clusters are well defined, then the values of Betweenness-centralities are expected to drop or rise suddenly. The plot of the Betweenness-centrality values (in sorted order) vs. the node numbers shows exactly this phenomenon. The example considered is a graph of 15 nodes with 3 clusters, as shown below.

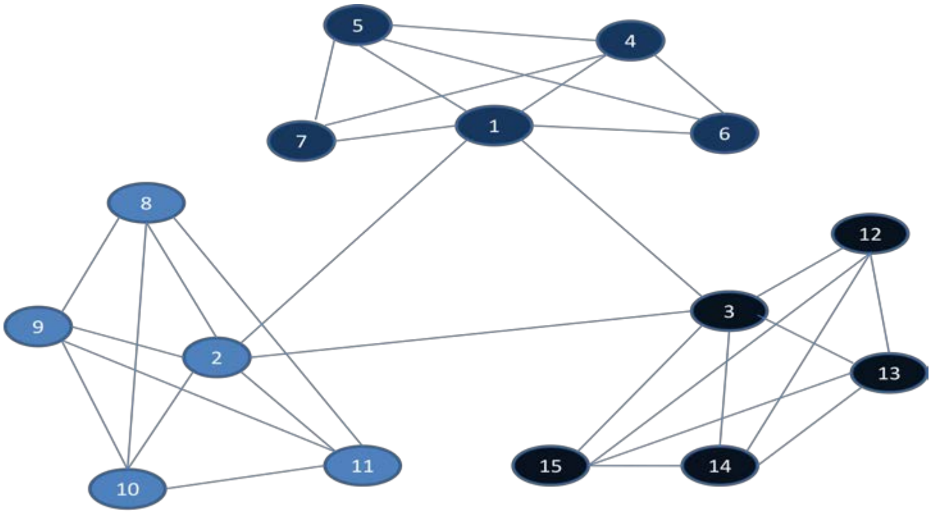


Fig 5.2: Graph with 15 nodes and 3 clusters

The Betweenness centralities of all the nodes are sorted in decreasing order and are plotted vs. the number of nodes.

Node Number	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Betweenness	**130**	**80**	**80**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Table 5.1: Betweenness-Centrality & Nodes

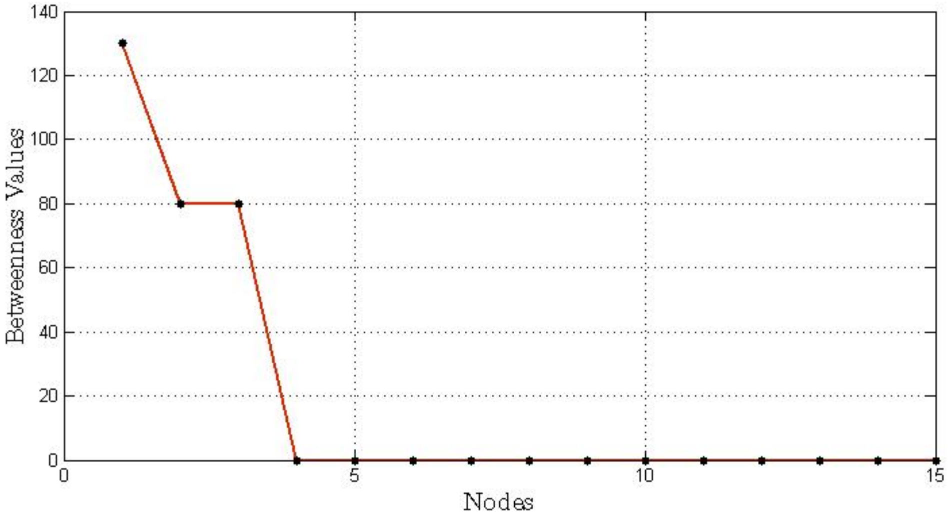


Fig 5.3: Betweenness-Centrality vs. Nodes

The figure above shows that there is a sudden dip (knee-point) in the plot. The point where the dip occurs is the number of clusters in the graph and the corresponding nodes give a precise upper-bound to the number of clusters. Hence the next step is to find this knee-point in the plot.

Some research has been done in finding this knee point in the graph, using the Bayesian Information Criteria (BIC) and the discrete second derivative of the plot. Due to the calculation complexity of the BIC approach, the methodology using discrete second derivative has been adopted [26].

Given a discrete valued plot, the second derivative of the plot gives information how the three consecutive values of the plot are varying. The formula for the discrete second derivative is:

$$\Delta^2 B(n) = B(n\text{-}1) - 2 * B(n) + B(n+1)$$

where, B(n) are the *Betweenness-Centrality* value for the all nodes.

As an example, the table below shows the Betweenness-Centrality values and the second derivative values for a graph of 15 nodes.

Nodes	2	1	3	4	5	6	7	8	9	10	11	12	13	14	15
$B(n)$	**130**	**80**	**80**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\Delta^2 B(n)$	**50**	**-80**	**80**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Table 5.2: The values of Betweenness and the discrete second derivative

The plot of the discrete second derivative of the Betweenness-Centrality vs. the Nodes is shown below.

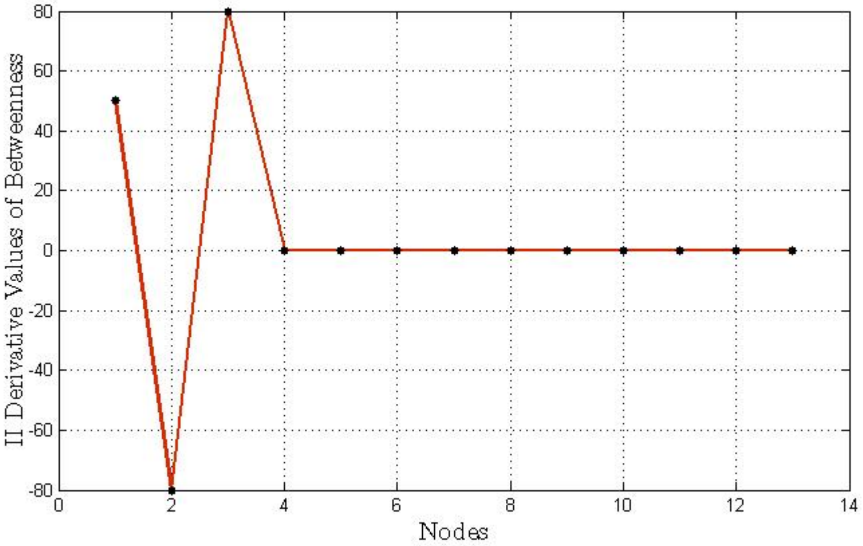


Fig 5.4: 2nd Derivative of Betweenness vs. Nodes

As is suggested from the above plot and several experiments conducted over huge graphs up to 10,000 nodes, the maximum point of the second derivate values is the knee point of the graph.

With the certain centres of the clusters and the upper bound on the number of clusters, K-means can be improved a good deal. With the upper bound of the number of clusters not so high than the optimum number, K-means can be run a few times to get the clusters with the optimum value of the objective function.

5.5 Results: K-means Improved Algorithm

With this information and improvement, graphs with large number of nodes have been used to implement the suggested algorithm and the optimum results are attained within a considerable amount of time.

#	Number of Nodes	Number of Clusters	Run-Time (seconds)
1	801	8	0.370
2	8801	11	55.273
3	1801	18	2.490
4	1801	90	29.520
5	1601	32	3.580
6	2801	28	7.830
7	2601	130	106.370
8	4001	40	21.220
9	4201	42	24.660
10	4201	84	75.700
11	4401	88	87.910
12	6001	60	71.540
13	6801	68	106.640
14	13001	13	130.260
15	14001	14	169.700

Table 5.3: Results for K-Means Algorithm

The time complexity of the proposed algorithm is higher than the standard K-Means algorithm due to the involvement of betweenness-centrality. However, the accuracy of the algorithm is enhanced incredibly.

Chapter – 6

SHRINK Algorithm

6.1 Introduction

SHRINK Algorithm, (A Structural Clustering Algorithm for Detecting Hierarchical Communities in Networks) is a very recent algorithm proposed to identify the *hubs* and *outliers* in a graph along with the clusters. Generally, there are several different kinds of nodes in a network which are cluster nodes densely connected within communities, as well as some special nodes like *hubs* bridging multiple communities and *outliers* marginally connected with a community [29].

SHRINK algorithm is based on the structural connectivity information. Moreover, it overcomes the sensitive threshold problem of density-based clustering algorithms and the resolution limit possessed by other modularity-based methods.

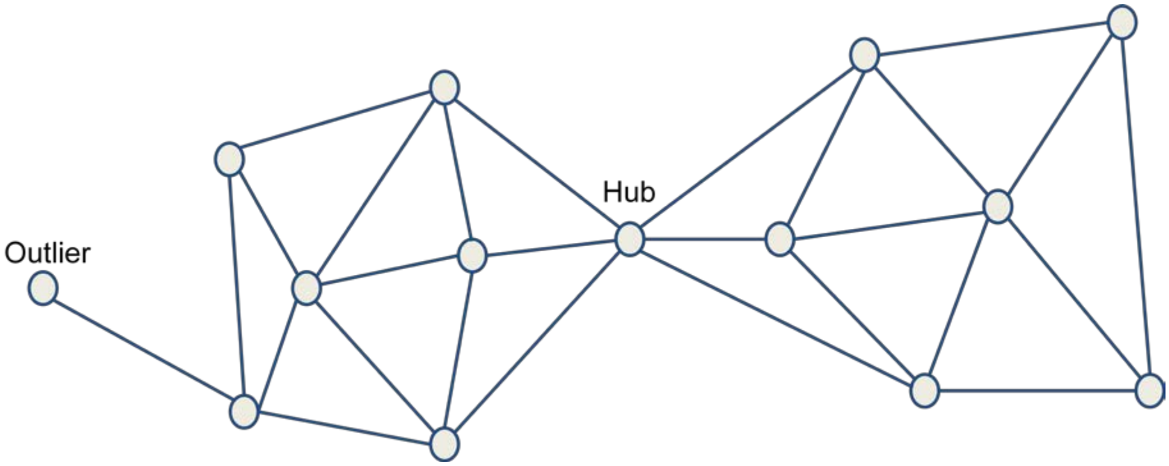


Fig 6.1: A Graph with Hub and Outliers

Hubs play special and important roles in many real-world networks. For example, hubs in the WWW could be utilized to improve the search engine rankings for relevant authoritative Web pages, and hubs in viral marketing and epidemiology could be central nodes for spreading ideas or diseases. Furthermore, there are some nodes that are marginally connected with the community members. Since outliers have little or no influence in a community, they may be isolated as noise in the network. Therefore, how to detect hierarchical communities as well as hubs and outliers in a network becomes an interesting and challenging problem. The algorithm is a parameter-free hierarchical network clustering algorithm and combines the advantages of density-based clustering and modularity-based methods. This chapter describes the algorithms in detail with some results implemented on the graphs.

6.2 SHRINK Algorithm

Before stating the algorithm we first need to study about some important measures related to the algorithm. These measures are defined below.

1) Structural Similarity

Let *G = (V,E,w)* be a weighted undirected graph and *w(e)* be the weight of edge *e* such that for a node $u \epsilon V$, *w({u,u})=1.* Moreover, the structure neighborhood of a node u is the set $\Gamma(u)$ containing u and its adjacent nodes:

$$\Gamma(u) = \{v \epsilon V | \{u, v\} \epsilon E\} \cup \{u\}$$

The structural similarity between two adjacent nodes u and v is given by:

$$\sigma(u,v) = \frac{\sum_{x\epsilon\Gamma(u)\cap\Gamma(v)} w(u,x).w(v,x)}{\sqrt{\sum_{x\epsilon\Gamma(u)} w^2(u,x)} \, . \sqrt{\sum_{x\epsilon\Gamma(v)} w^2(v,x)}}$$

2) Dense Pair

Given a graph *G = (V, E), σ(u,v)* be the structural similarity of nodes u and v. If *σ(u,v)* is the largest similarity between nodes *u, v* and their adjacent neighbor nodes such that:

$$\sigma(u,v) = \max\{\sigma(x,y) | (x = u, y\epsilon\Gamma(u) - \{u\}) \vee (x = v, y\epsilon\Gamma(v) - \{v\})\}$$

Then {u,v} is called a dense pair in the graph G, denoted by $u \underset{\varepsilon}{\leftrightarrow} v$ where $\varepsilon = \sigma(u,v)$ is the density of pair {u,v}.

3) Micro-Community

Given a graph *G = (V, E), C(a) = (V, E, ε)* is a sub-graph of *G* represented by a node *a*. C(a) is a local micro-community iff:

$i)\ a \in V^{'};$

$ii) \forall\, u \in V^{'}, \exists v \in V^{'} \left(u \underset{\varepsilon}{\leftrightarrow} v\right);$

$iii) \nexists u \in V (u \underset{\varepsilon}{\leftrightarrow} v \wedge u \in V' \wedge v \notin V')$

4) Similarity-Based Modularity

The Modularity function used in the SHRINK algorithm is different than suggested by Newman. SHRINK algorithm uses similarity based modularity proposed by Feng [28]. Given a cluster $CR = \{C_1, C_2, \ldots, C_k\}$ of the graph $G = (V, E)$, the function Q_s (similarity based modularity) is defined as:

$$Q_S = \sum_{i=1}^{k} \left[\left(\frac{IS_i}{TS} \right) - \left(\frac{DS_i}{TS} \right)^2 \right]$$

where,

$\boldsymbol{k}$ is the number of clusters,

$IS_i = \sum_{u,v \in C_i} \sigma(u, v)$, is the total structural similarity of nodes within cluster C_i

$DS_i = \sum_{u \in C_i, v \epsilon V} \sigma(u, v)$, is the total similarity between nodes in cluster C_i and any node in the graph.

$TS = \sum_{u,v \epsilon V} \sigma(u, v)$, is the total similarity between any two adjacent nodes in the graph.

We also define incremental modularity gain:

$$\Delta Q_{S_{C_i \cup C_j}} = Q_S^{C_i \cup C_j} - Q_S^{C_i} - Q_S^{C_j}$$

If the modularity gain $\Delta Q_{S_{C_i \cup C_j}}$ of a micro-community C are positive, the nodes of C should be clustered in the same community. Thus, given a network G, the task of the algorithms is to find a higher modularity solution under the principle of density-based clustering, rather than to search a partition greedily maximizing the modularity Q_S.

With the above definitions, we can now understand the SHRINK algorithm. The formal algorithm as suggested by the researchers who proposed it is given below:

SHRINK Algorithm

1. *Initialize by placing every vertex 'v' into its own cluster C_i, and calculates $Q_S^{C_i}$ for each cluster.*
2. *Merge every adjacent pair of clusters C_i and C_j, $(i \neq j)$ and record $\Delta Q_{S_{C_i \cup C_j}}$.*
3. *Choose the merge(micro-community) with the largest $\Delta Q_{S_{C_i \cup C_j}}$ (dense pair) and update Q_S.*
4. *Update all affected values of $\Delta Q_{S_{C_i \cup C_k}}$ and $\Delta Q_{S_{C_j \cup C_l}}$ $(k \neq i;\ l \neq j)$ using the above formula.*
5. *Repeat step 3 and 4 until all vertices are grouped into one cluster.*
6. *Retrieve the partitioning with the highest Q_S value as the best result.*

The pictorial representation of the algorithm, as given in the paper, is shown below.

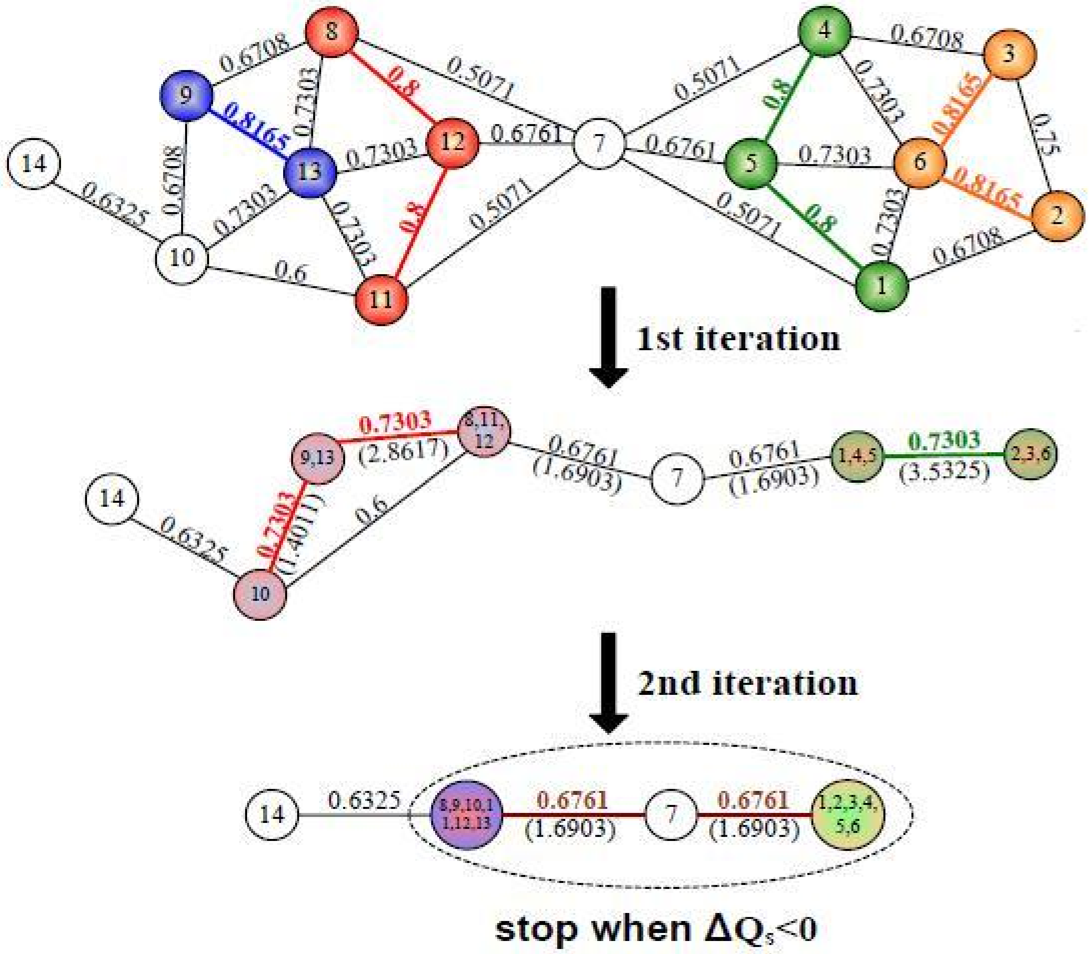


Fig 6.2: Pictorial Representation of SHRINK algorithm through a small example

The above figure demonstrates the algorithm. The numbers associated to every edge in the first graph are the strutural similarities of all the adjacent edges. The nodes in similar colours are the dense pairs present in the graph(or in other words, they exhibit positive $\Delta Q_{S_{C_i \cup C_j}}$ values). These dense pairs are clubbed together in one cluster. This process is iterated until there is no increase in the value of $\Delta Q_{S_{C_i \cup C_j}}$ is negative.

The final output of the algorithm fetches us with the clusters in the graph and the hubs & outliers. In the above graph, node 7 is a hub node and node 14 is an authority node.

6.3 Results: SHRINK Algorithm

#	Number of Nodes	Number of Clusters	Run-Time (Seconds)
1	**201**	**10**	**2.38**
2	**201**	**5**	**3.30**
3	**401**	**20**	**22.78**
4	**401**	**10**	**38.29**
5	**801**	**20**	**308.14**
6	**801**	**10**	**599.20**
7	**1001**	**20**	**738.40**
8	**1001**	**10**	**1493.00**
9	**1201**	**20**	**1554.40**
10	**1201**	**10**	**3222.00**
11	**1401**	**20**	**2973.00**
12	**1401**	**10**	**6371.60**

Table 6.1: Some Results with implementation of the Shrink Algorithm

Chapter – 7

Genetic Clustering Algorithm

7.1. Introduction

Genetic Algorithms are one very important class of algorithms for solving optimization problems. Since their invention, they have been extensively used for solving NP-hard optimization problems in various fields. Several variations of the algorithm are proposed and implemented depending upon the nature of the problem, yet the basic idea of the algorithm remains the same i.e. Evolution. Genetic algorithms can be viewed as advanced level random selection of the best fit solutions over the entire search space.

Genetic algorithms (GAs) are search methods based on principles of natural selection and genetics. They encode the decision variables of a search problem into finite length strings of alphabets (or binary, decimal) of certain cardinality. The strings which are the candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes and the values of genes are called alleles [30, 31].

For example, in a problem such as the traveling salesman problem, a chromosome represents a route, and a gene may represent a city. In contrast to traditional optimization techniques, GAs work with coding of parameters, rather than the parameters themselves. To evolve good solutions and to implement natural selection, we need a measure for distinguishing good solutions from bad solutions. The measure could be an objective function that is a mathematical model or a computer simulation or it can be a subjective function where humans choose better solutions over worse ones. In essence, the fitness measure must determine a candidate solution's relative fitness, which will subsequently be used by the GA to guide the evolution of good solutions.

Another important concept of GAs is the notion of population. Unlike traditional search methods, genetic algorithms rely on a population of candidate solutions. The population size, which is usually a user-specified parameter, is one of the important factors affecting the scalability and performance of genetic algorithms. For example, small population sizes might lead to premature convergence and yield substandard solutions. On the other hand, large population sizes lead to unnecessary expenditure of valuable computational time [31].

Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones is chosen, we can start to evolve solutions to the search problem using the following steps:

1) Fitness Function: Genetic algorithms are generally implemented to optimize the maximum or the minimum value of a given function, to find the optimum solution of an equation, etc. This function called the *fitness function* varies from problem to problem and is the pivot around which all the genetic operations are dependent. As an example, to find the maxima of the function given below,

$$f(x) = -\frac{1}{4}x^2 + 2x + 5, \quad x \in [0,10]$$

2) Initialization: The initial population of candidate solutions is usually generated randomly across the search space for all individuals in the population. However, domain-specific knowledge or other information can be easily incorporated. That is to say, for some optimization problems a GA works well with a binary representation of the solutions while for some a decimal representation is a better choice for an efficient GA algorithm. Often both the representations work as per the correctness of the algorithm is concerned, but the choice is made for a representation which may yield an efficient algorithm. For the clustering problem, a decimal representation of population yields a better result, as will be explained in the later section.

3) Evaluation: Once the fitness function is obtained and the population is initialized; the fitness values of the candidate solutions are evaluated. Better the fitness value of an individual, higher are its chances to move to the next generation. However the individuals are selected as per the separate selection procedures, as explained in the next section.

4) Selection: Selection of the individuals is the next necessary step. Selection process allocates more copies of those solutions which have higher fitness values and thus imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to prefer better solutions over worse ones. After the selection of the individuals is made, genetic operations are implemented on the individuals. Many selection procedures have been proposed, and the most common are the Roulette-Wheel selection and the Elitist selection.

a) Roulette-Wheel Selection: The basic idea behind the roulette-wheel selection is to stochastically select individuals from one generation to create the basis of the next generation. The requirement is that the fittest individuals have a greater chance of survival than weaker ones. The fitter individuals tend to have a better probability of survival and go forward to form the mating pool for the next generation. Weaker individuals are discarded. In nature such individuals may have genetic coding that may prove useful to future generations. As an example, the following table lists a sample of population with 5 individuals for the above fitness-function *f(x)*.

No.	Chromosomes	Values	x=Values/100	Fitness *f(x)*	% of Total
1	0001101001	105	1.05	6.82	31
2	1111000010	962	9.62	1.11	5
3	0011111111	255	2.55	8.48	38
4	1110001011	907	9.07	2.57	12
5	1101110111	887	8.87	3.08	14
Total				**Σ *f(x)*=22.05**	**100**

Table 7.1: Roulette-Wheel Selection

To incorporate many possible solutions in the interval [0, 10], the search space is multiplied by 100 i.e. [0, 1000] and converted to binary representation. The percentage contribution to the total fitness in calculated for each individual (last column). We can see from the table that 3rd individual is the fittest, with fitness value of 8.48 and percentage contribution of 38%; 2nd individual being the weakest. The percentage fitness values are essential and are used to configure the roulette wheel as shown below. Fig 7.1 highlights that individual no. 3 has a segment cover of 38% on the whole wheel i.e. the 3rd individual has high probability of getting selected for the next step.

The number of times the roulette wheel is spun is equal to size of the population. As can be seen from the way the wheel is divided, each time the wheel stops, the fitter individuals possess a greater probability of being selected for the next generation and subsequent mating pool.

What is possibly more interesting from this example is that as the generation's progress and the population gets fitter the gene pattern for individual no. 3: 0011111111_2 will become more prevalent in the general population because it is fitter, more apt to the function we are trying to optimize.

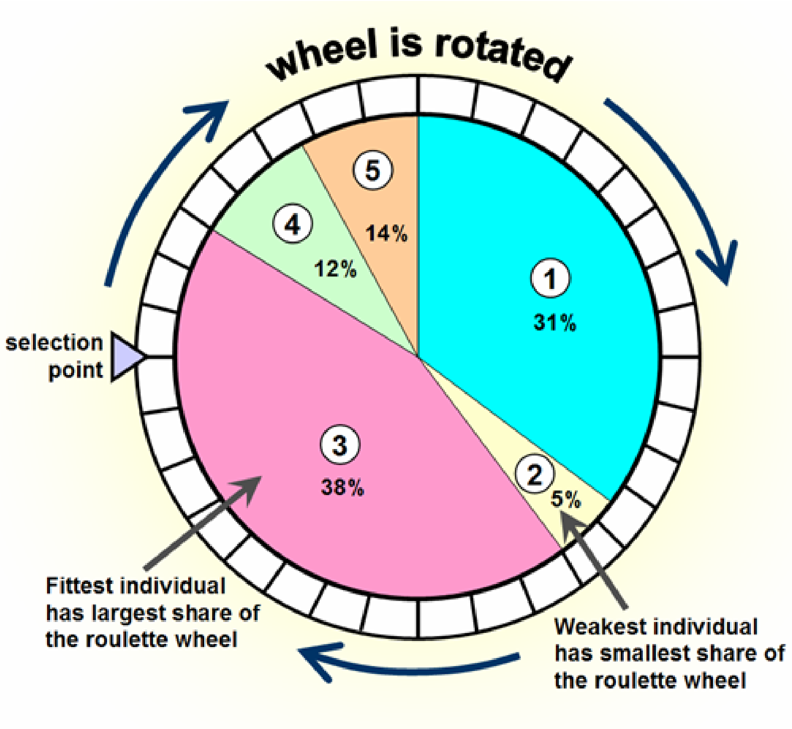

wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Fig 7.1: Roulette-Wheel selection

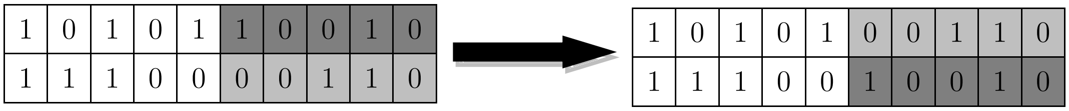
b) Elitist Selection: Elitist selection is another selection methodology used for genetic algorithms. In this particular method the individuals with the best fitness values are moved to the next generation and the remaining individuals are considered for genetic operations i.e. go to the mating pool. As an example for the above function *f(x)* the elitist selection can be explained by the table below.

No.	Chromosomes	Values	x=Values/100	Fitness *f(x)*
1	0001101001	105	1.05	6.82
2	1111000010	962	9.62	1.11
3	0011111111	255	2.55	8.48
4	1110001011	907	9.07	2.57
5	1101110111	887	8.87	3.08
6	0101000100	324	3.24	8.85
7	1101000100	836	8.36	4.25
8	0101101001	361	3.61	8.96

Table 7.2: Elitist Selection

Once the fitness values of the individuals are calculated, the a given number of bet fit individuals are moved to the next generation without any genetic operations and the rest individuals move to the mating pool wherein they undergo genetic operations, crossover and mutation. In the example above, if we perform elitist of 2 individuals, then individual no. **8** & **6** are retained in the next generation and the rest form a mating pool for the genetic operations.

5) Crossover: Crossover combines parts of two or more parental solutions to create new, possibly better solutions (i.e. offspring). There are many ways of accomplishing this operation, such as a single point crossover, two point crossovers, random crossover etc. The competent performance depends on a properly designed crossover mechanism. The offspring under crossover will not be identical to any particular parent and will instead combine parental traits in a novel manner (Goldberg, 2002). The figure below illustrates a single point crossover [30, 31].



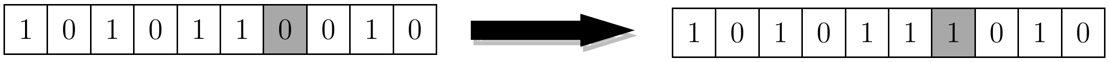
1	0	1	0	1	1	0	0	1	0
1	1	1	0	0	0	0	1	1	0

1	0	1	0	1	0	0	1	1	0
1	1	1	0	0	1	0	0	1	0

Fig 7.2: Crossover

The crossover operation is an essential operation when the problem to be optimized needs to search over a huge sample space. Crossover makes the genetic algorithm explorative with respect to the solutions. Crossover produces a set of two new individuals which are the combination of its parent's genotypes. The off-springs thus produced contain gene characteristics of its both parents.

6) Mutation: While crossover operates on a pair of parental chromosomes, mutation modifies a solution locally. Again, there are many variations of mutation, but it usually involves one or more changes being made to an individual's trait or traits. In other words, mutation performs a random walk in the vicinity of a candidate solution. Mutation is a necessary operation as it brings diversity to the individuals in the population. In this operation, each individual undergoes a genetic change with a certain probability. The example below explains better.



1	0	1	0	1	1	0	0	1	0

1	0	1	0	1	1	1	0	1	0

Fig 7.3: One point Mutation

Mutation makes the genetic algorithm more exploitative in the solution search. Several strategies could be adopted for the mutation and one of them is depicted in the figure above (single point Mutation).

7) Replacement: The offspring population created by selection, crossover, and mutation replaces the original parental population. Many replacement techniques such as elitist replacement, generation wise replacement and steady-state replacement methods are used in GAs.

8) Iterate: Repeat steps 2–6 until a terminating condition is met.

7.2. Genetic Operations for Clustering Problem

The Genetic Clustering Algorithm which I am proposing in the coming section is based on the same basic guidelines of evolution phenomenon. The difference from the standard Genetic Algorithm lies in the population initialization and incorporation of the concept of Island Model. The algorithm takes the adjacency matrix of the graph as input and returns the best fit clusters present in the graph. To improve the efficiency and accuracy of the algorithm an Island Model of the Genetic Algorithm has been used with migration at specific number of generations. In the next sections I will describe the population initialization, fitness function, selection criteria, methodologies for all the genetic operations implemented and the implementation of the Island Model [32].

7.2.1. Population Initialization

The kind of population initialization in a Genetic Algorithm is of utmost importance for the efficiency and accuracy of the algorithm as it is one of the many solutions the algorithms starts to optimize. The selection of the type of population is very much problem specific. For example, an algorithm designed to find an extrema of a (multi-modal) function, would work the best with a binary initial population. Like-wise, many standard optimization problems such as Traveling Salesman Problem would require the initial population to be of decimal type where a decimal number would define a particular city. Several other problems may require the initial population to be of alphabetical nature and so on.

The Genetic Algorithm for clustering problem can be solved using both the Binary and Decimal representation of the population. But considering the fact that the size of graph could be really huge, and also from my own experiments with both the kind of population types; the Decimal representation of the population seems to yield the results in a much less time.
Hence, in the genetic clustering algorithm proposed, decimal representation of the population has been considered, wherein a decimal number for a node indicates to which cluster the node belongs.

To better explain the population representation, below is a small example. Let us say, we have an undirected graph with 12 nodes then one may represent an individual of the population as:

Nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Population	1	2	3	1	1	1	1	2	2	2	2	3	3	3	3

Table 7.3: Population Representation

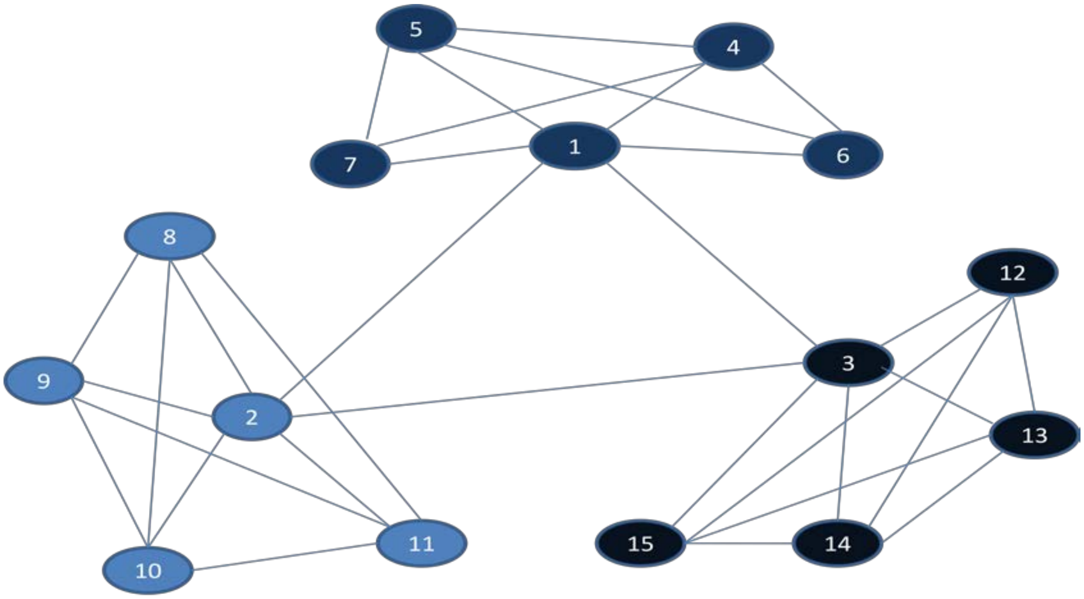


Fig 7.4: A Graph with three clusters

The above representation indicates that there are 3 clusters in the graph and each node belongs to one of the three clusters. The table below shows which node belongs to which cluster.

Cluster number	**Nodes in the Cluster**
1	1, 4, 5, 6, 7
2	2, 8, 9, 10, 11
3	3, 12, 13, 14, 15

Table 7.4: Cluster of Nodes

Hence each individual in the population is a vector of size *'n'* where n is the total number of nodes in the graph and each entry of the vector indicates to which cluster the nodes belongs.

7.2.2. Fitness Function

Fitness function of problem is used to identify and select the best fit solutions (individuals) in the entire population. For the Clustering problem, I am using Modularity [Newman, 2002] as the fitness function. As stated in the earlier section, Modularity is a well accepted criterion and measure to judge the goodness of a cluster. Hence it makes a perfect sense to accept it as the fitness function for the Genetic Clustering Algorithm.

7.2.3. Selection

The next step is that of the selection of the individuals in the population for mating and producing new off-springs. There are several methods available for selection as earlier but for the clustering problem Elitist selection has been adopted, keeping in mind, that the best fit individuals are more prone to produce better off-springs.Under this methodology, all the possible solutions (individuals) are ranked according to their fitness value and the best fit individuals are considered for the genetic operations (crossover & mutation). One may argue that sometimes the genetic operations between the best fit individuals might as well reduce the fitness values of the off-springs. Let us say the population size is ***N,*** then ***2*** best individuals move to the next generation (to avoid the loss of better fit individuals of the population) but the remaining (***N-2***) individual's move to the mating pool and undergo genetic operations of crossover and mutation. The Flow chart for the Selection Methodology:

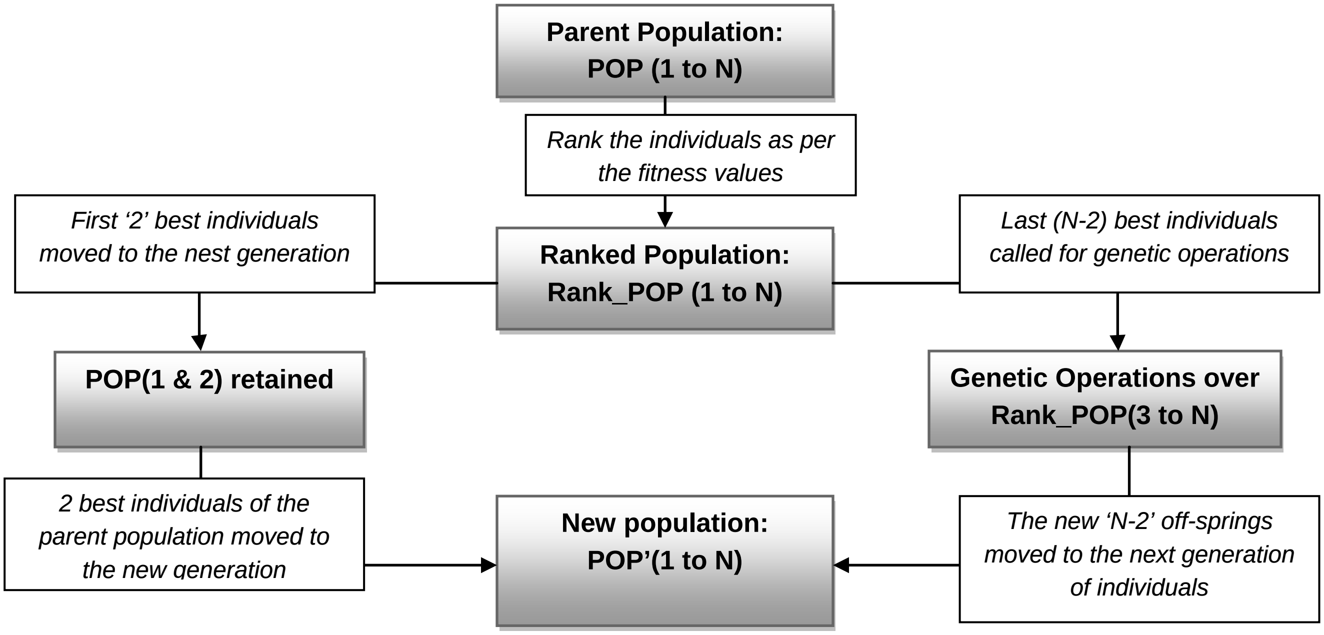
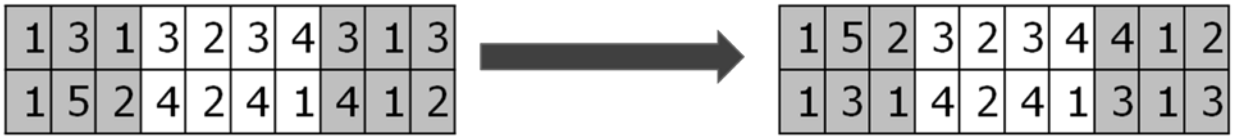

Parent Population:
POP (1 to N)
Rank the individuals as per the fitness values
First '2' best individuals moved to the nest generation
Ranked Population:
Rank_POP (1 to N)
Last (N-2) best individuals called for genetic operations
POP(1 & 2) retained
Genetic Operations over Rank_POP(3 to N)
2 best individuals of the parent population moved to the new generation
New population:
POP'(1 to N)
The new 'N-2' off-springs moved to the next generation of individuals

Fig 7.5: Flow Chart for Elitist Selection

7.2.4. Crossover

Crossover is one of the two standard genetic operations considered in the algorithm. During crossover certain genes of the two mating parent-individuals get swapped forming two new off-springs with mixed characteristics. Crossover is an essential operation if one wants to incorporate a larger search space or in other words, to be explorative. Two types of crossover techniques have been implemented:

1) **2-Point Uniform Crossover (CdecU):** Two equally spaced sections of each of the two mating individuals are swapped with each other forming two new off-springs with mixed characteristics of the parent genes. This is better shown below:

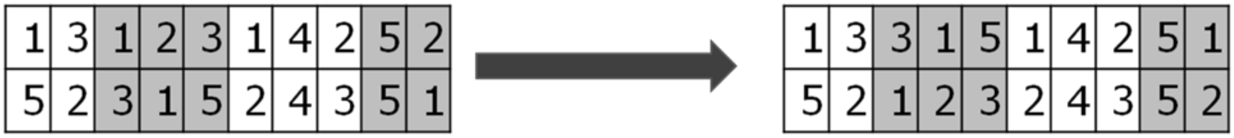


1	3	1	3	2	3	4	3	1	3
1	5	2	4	2	4	1	4	1	2

1	5	2	3	2	3	4	4	1	2
1	3	1	4	2	4	1	3	1	3

Fig 7.6: 2-Point Uniform Crossover

2) **2-Point Random Crossover (CdecR):** Two sections of the individuals are randomly chosen in both the mating parent individuals and are swapped forming two new off-springs with mixed characteristics. This is depicted in the diagram below:



1	3	1	2	3	1	4	2	5	2
5	2	3	1	5	2	4	3	5	1

1	3	3	1	5	1	4	2	5	1
5	2	1	2	3	2	4	3	5	2

Fig 7.7: 2-Point Random Crossover

7.2.5. Mutation

Mutation is an essential genetic operation in the Clustering problem. As the algorithm by itself decides and searches for the number of clusters, it should be able to incorporate the below mentioned possibilities:

1) **Addition of Clusters (MdecA):** It might happen that at certain instant of time the solution after the above operations may contain less than optimal number of clusters. Hence the code should be able to add a new cluster.

2) **Deletion of Clusters (MdecD):** Likewise, another possibility is that the solution may contain more than optimal number of clusters.
3) **Re-Arrangement of Clusters (MdecR):** Another likely possibility is wrong assignment of nodes to clusters. One may need to switch one (or more) node(s) from one cluster to another so as to optimize the solution.

The three above mentioned issues are taken off, via Mutation. Four different types of mutation operations have been considered to encounter the above mentioned problems. Three of these operations take care of the above issues and the fourth operation randomly chooses one of the above issues and performs the operation accordingly. The diagram below shows the mutation operation in a single individual. The green box shows addition of a new cluster, the grey box shows the rearrangement of the clusters, and the blue box shows the deletion of a cluster.

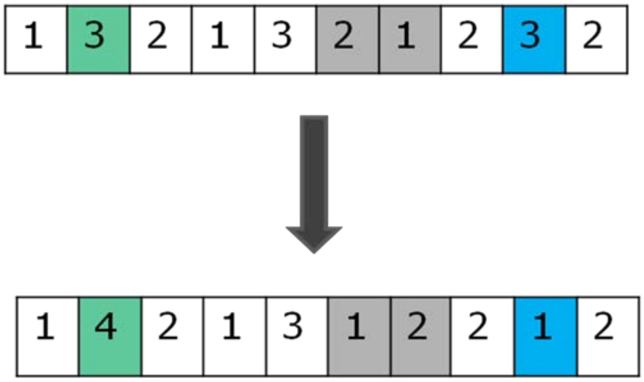


Fig 7.8: Mutation operation

Apart from the above three mutation operations, one more mutation operation is considered which is uniform probabilistic selection of one of the above three methods, **MdecP**.

7.2.6. Island Model & Migration

Island Model is one class of Parallel Genetic Algorithms and it arose out of the desire to exploit coarse grained parallel architectures. Island models use a strategy usually found in a polytypic species, which evolve in isolated sub groups with more interaction within subgroups than between them. The island model typically runs a serial GA on each of the several loosely connected processors. Each GA is identical to but independent of the others. Each GA is usually started with a different random seed. Periodically, individuals are transmitted between the islands in a process called migration. The island model strategy eliminates the global synchronization that is required in the panmictic approach [32]. However, some synchronization is usually required for migration. To make the algorithm more robust the concept of Island model has been implemented other than the standard genetic operations. The main idea behind this concept is to consider some sets of populations (Islands) and carry out different combinations of genetic operations (i.e. 2 types of Crossover Operation & 4 types of Mutation Operations) in each island. In the algorithm 6-islands with equal population size have been considered with six different combinations of crossover and mutation. This way one makes the algorithm a lot more explorative and robust to attain the optimal solution. A further enhancement is the concept of Migration. After certain number of iterations in each island, the best individuals of each island are migrated to all other islands and replaced by the 5 worst individuals as per their fitness value. The concept of Island Model and Migration is a very efficient and accurate enhancement to the Genetic Algorithm. With the advancement of multi-processor computers, this model can incredibly increase the speed of the algorithm by using different cores of the processor on different Islands. The combination of crossover & mutation operations in each island and the Algorithm are depicted below:

Island No.	1	2	3	4	5	6
Crossover	CdecR	CdecU	CdecU	CdecR	CdecU	CdecR
Mutation	MdecA	MdecD	MdecA	MdecD	MdecR	MdecP

Table 7.5: Crossover & Mutation Operations in each Island

Algorithm:	*% Elitist Selection*

1: Initial Population for each Island
2: Crossover & Mutation in each Island
3: Elitist selection in each Island for the next generation
4: Selection of best individuals in each Island
5: Migration to all other Islands
6: Repeat Step 2 through 5
	I1=Pop1(best); I2=Pop2(best);
	I3=Pop3(best); I4=Pop4(best);
	I5=Pop5(best); I6=Pop6(best);
	% Migration
	Pop1(1:6) = [I2;I3;I4;I5;I6;I1];
	Pop2(1:6) = [I3;I4;I5;I6;I1;I2];
	Pop3(1:6) = [I4;I5;I6;I1;I2;I3];
	Pop4(1:6) = [I5;I6;I1;I2;I3;I4];
	Pop5(1:6) = [I6;I1;I2;I3;I4;I5];
	Pop6(1:6) = [I1;I2;I3;I4;I5;I6];

Table 7.6: Genetic Algorithm with Island Model

7.3. Results: Genetic Clustering Algorithm

Below are some results of the Genetic Clustering Algorithm.

	Number of Nodes	Number of Clusters	Run-Time (seconds)
1	8	2	0.06
2	11	2	0.40
3	15	3	0.32
4	21	3	4.82
5	25	4	2.28
6	29	4	1.59
7	42	5	10.70
8	101	5	28.09
9	201	10	179.88
10	301	15	472.50
11	401	10	682.23
12	619	10	859.14

Table 7.7: Results of Genetic Clustering Algorithm

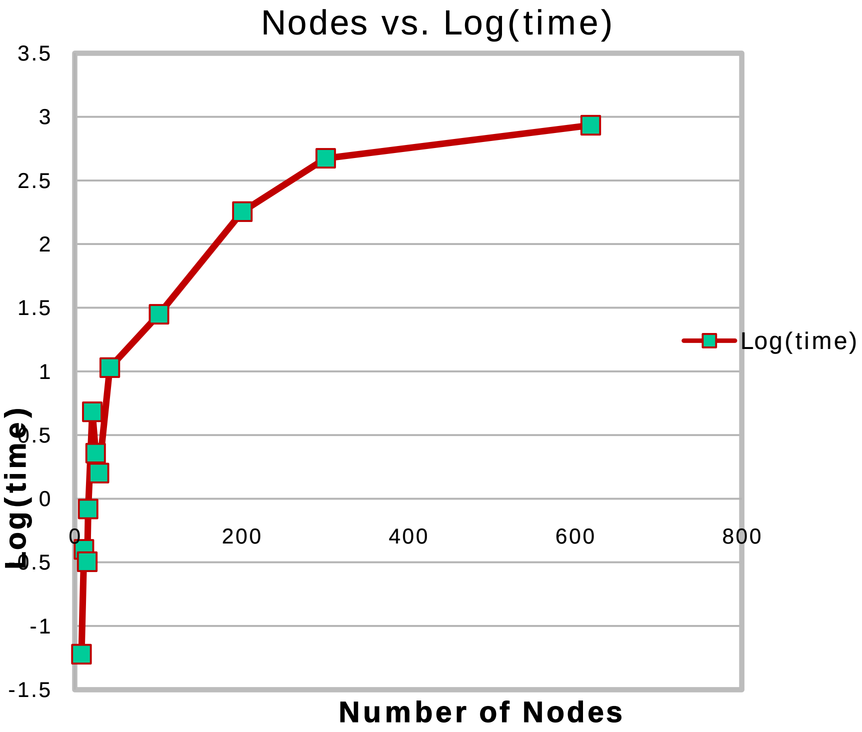


Fig 7.9: Plot of Log of Run time vs. number of nodes

Chapter – 8

Cluster of a Node: Genetic Algorithm

8.1. Introduction

Often, finding all the clusters in a huge graph could be a laborious, unnecessary and a time consuming problem. In situations like these one may be interested to find the cluster of one particular node. One very obvious scenario occurs in the Money Laundering Detection. Financial Intelligence Unit's (FIUs) usually have knowledge of one or more entities which cause Money Laundering or some financial fraud. In such a case it is worth to find only the cluster of that suspicious node (entity) instead of finding all the clusters in the network.

All the standard clustering algorithms focus on finding all the clusters in the graph. Not much of research has been carried out on finding the cluster of one desired node. Most of the metrics' which are available for judging the goodness of the clusters work only when one is aware of all the clusters in the graph. Modularity for instance is a measure for a graph and takes into account all the clusters present. But Modularity can be used in a different manner, discussed later, in finding the cluster of a node.

As stated above in chapter 4, Modularity is a measure which will always divide the graph in a way that the clusters have more edges inside them and fewer edges between different clusters. This property can be exploited to break up the graph and search for the desired cluster. In the next sections, an iterative Hierarchical Genetic Algorithm is proposed to find the cluster of one node, with Modularity as the fitness function. The input to the algorithm is the data to build the graph and a node of which the cluster is sought and the algorithm returns the cluster of this input node.

8.2. Main Idea

The real motivation for the algorithm is its simplicity and robustness. The complexity of the algorithm in comparison to a regular clustering algorithm for finding the cluster is much better. One usually tries to find all the clusters in the network and pick the one desired. In the process a lot memory overhead is consumed in dividing the part of the network which is not even relevant to the problem. Thus discarding the portion of the graph which is irrelevant to the problem makes perfect sense and saves a lot of time and space.

The problem of finding cluster of a node requires an algorithm to find just one desirable cluster in the whole network, irrespective of how big the network is. This is a very simple and yet a very important fact to exploit and detect a part of the graph which may contain the cluster of the desired node. The idea behind the proposed algorithm is to iteratively bisect the graph in two almost equal clusters. The figures below explain the methodology behind the algorithm.

Let us we have a graph containing 3 clusters and we wish to find the cluster of node 7. At the first iteration, genetic clustering algorithm is applied to divide the graph in two best fit clusters. The cluster which does not contain node 7 is discarded and in the next the smaller graph is divided again in two best fit clusters. This procedure is iterated until the desired cluster cannot be divided anymore i.e. the modularity of the graph does not improve.

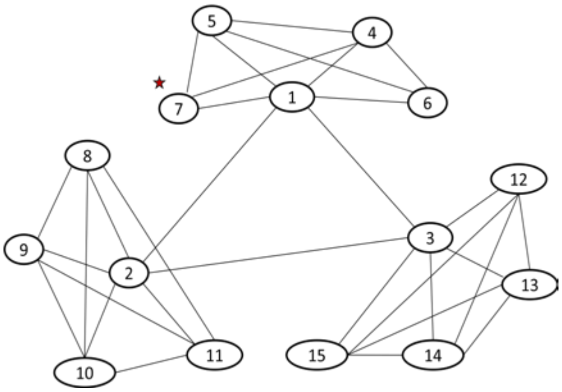


Fig 8.1 (a): Whole Graph with the desired node

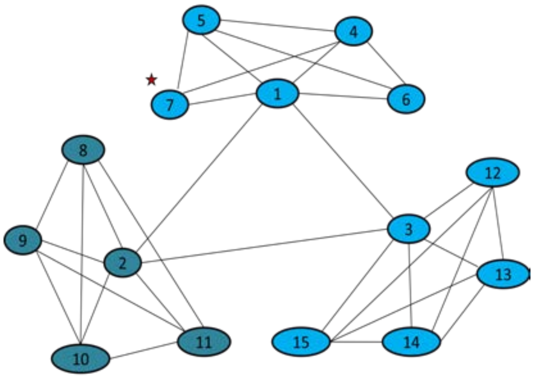


Fig 8.1 (b): Two best clusters in the graph

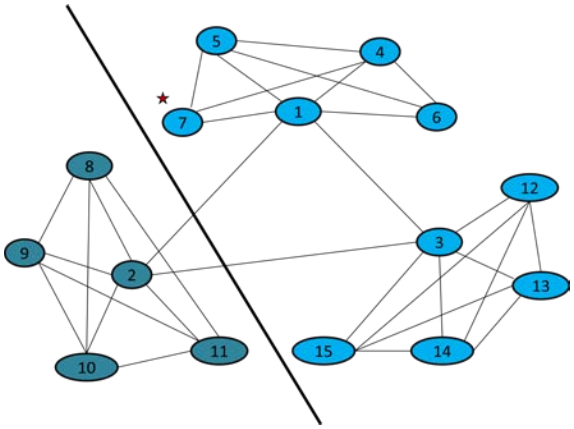


Fig 8.1 (c): Unwanted cluster discarded

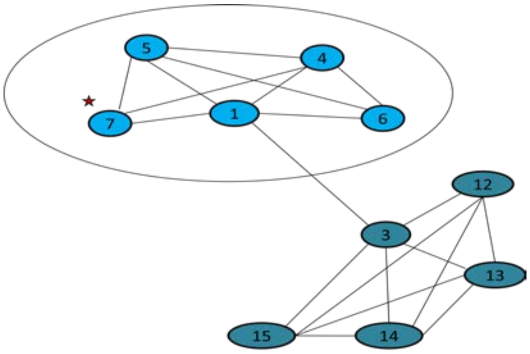


Fig 8.1 (d): Final cluster with the desired node

As shown in the figure, the graph is divided in two best fit clusters in every iteration; using genetic clustering algorithm described in the previous chapter. And the cluster which contains the desired node is retained for the next iteration. The process continues till the desired cluster is not possible to divide to give a better modularity. The stopping criterion is explained in detail in the next section.

8.3. Methodology and Genetic Operations

The basic idea behind this algorithm is to iteratively implement the same genetic algorithm as explained in the previous chapter. The first iteration divides the whole graph in two almost equal sized clusters. For the next iteration only the cluster which contains our desired node is considered, the second cluster which does not contain the desired node is left behind and never considered as if the graph size has reduced to half the previous size. In the second iteration this new half sized graph is considered and divided again in almost two equal sized clusters. In addition to this the number of generation produced in the genetic clustering algorithm are also reduced to half as the graph has reduced considerably. This methodology is iterated over and over again, always considering the cluster which contains the desired node.

As for the stopping criteria, we again go back to the definition of Modularity. As explained earlier, Modularity of a graph is higher for a division in which all the clusters of the graph have more number of edges in between the nodes of the cluster and fewer edges going outside the cluster; than a division which has more number of edges between two clusters and fewer edges in between the nodes of the cluster. This property of Modularity is essentially useful and exploited in the proposed algorithm. Let us say, the proposed algorithm is run for some number of iterations and ultimately we are left with a graph having just one cluster, with the desired node inside it. As per the algorithm, when this graph (with just one cluster i.e. the whole graph) is divided in two clusters, the Modularity value of such a division would be less than the Modularity value of the whole graph as one single cluster. In such a scenario the genetic algorithm will search for the best fit solution and this solution would be nothing but the whole graph as a single cluster. As we know from the earlier discussion of the property of Modularity; its value for the whole graph is always zero. Hence this is a well defined stopping criterion for the iterations of the genetic algorithm.

The Genetic Algorithm which is implemented iteratively is almost the same as the Genetic Clustering Algorithm explained in the previous chapter. But there are a few differences as in this algorithm, at any instance the graph is divided in 2 clusters, no matter how big the clusters are.

The input to the algorithm is the data to build the graph and a node of which the cluster is sought and the algorithm returns the cluster of this input node.

8.3.1. Population Initialization

The population initialization is again considered in decimal representation but unlike the previous algorithm here the nodes are assigned one of cluster 1 or 2. This way, at every new generation the graph is divided in only two clusters and ultimately yields two communities in the graph with highest modularity. As an example let us consider a graph of 12 nodes; the initial population is assigned as:

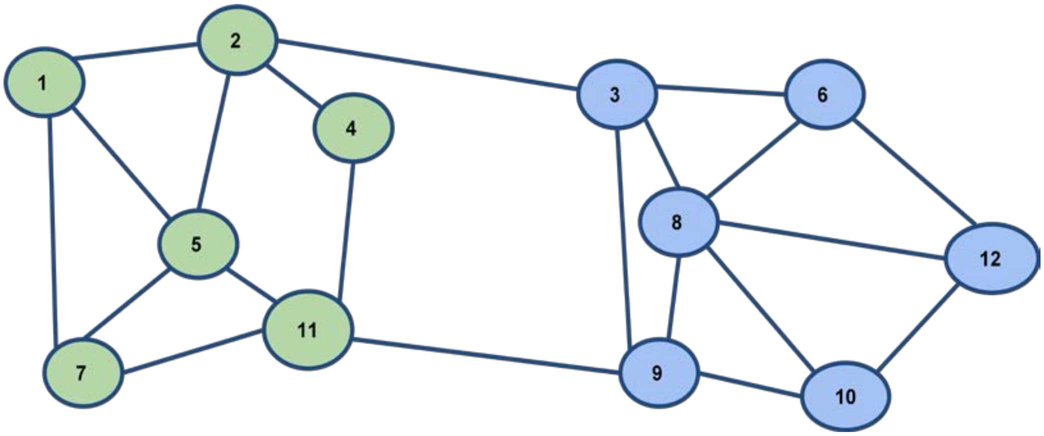


Fig 8.2: Graph containing two clusters

Nodes	1	2	3	4	5	6	7	8	9	10	11	12
Population	1	1	2	1	1	2	1	2	2	2	1	2

Table 8.1: Population Representation

Thus in the above representation there are two clusters in the graph and each node belongs to one of the two clusters. The table below shows which node belongs to which cluster.

Cluster number	Nodes in the Cluster
1	1, 2, 4, 5, 7, 11
2	3, 6, 8, 9, 10, 12

Table 8.2: Cluster of Nodes

Hence each individual in the population is a vector of size ***'n'*** where n is the total number of nodes in the graph and each entry of the vector indicates to which cluster the nodes belongs.

8.3.2. Fitness Function

Fitness function, as always is one very important aspect of any genetic algorithm and it is used to identify and select the best fit solutions (individuals) in the entire population. Here again Modularity has been as the fitness function. Since all the nodes in the graph are always divided in two clusters, the Modularity divides the graph the almost 2 equal sized clusters. It is this fact, that Modularity will not divide a well connected cluster in two or more sub-clusters, which is exploited in this algorithm to find the cluster of a node.

Since at every step of the genetic iteration, the graph is divided in only two clusters, the problem of resolution limit of Modularity does not affect the working of our algorithm.

8.3.3. **Selection**

The selection procedure is exactly the same as in the previous algorithm implementing the Elitist selection methodology as explained in section 8.2.3. Again two best individuals in each generation of every island are retained and moved on to the next generation.

8.3.4. Mutation

Mutation operation is slightly different and relaxed than the previous algorithm. Since at every generation we are only considering two clusters in the graph the Mutation should only incorporate deletion and re-arrangement of the clusters. The deletion of a cluster is again essential because of the last iteration in the algorithm when there is only one well defined cluster in the graph. The diagram below shows the mutation operation in a single individual. The grey box shows the rearrangement of the clusters, and the blue box shows the deletion of a cluster.

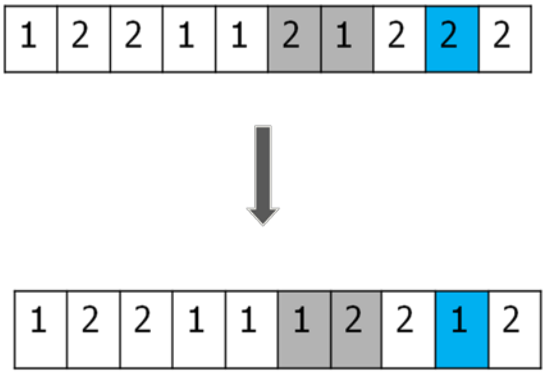


Fig 8.3: Mutation operation

The remaining Genetic operations implemented in the algorithm are the same as in the Genetic Clustering Algorithm, discussed in the previous chapter. The mutation operations are named as in the earlier chapter: ***MdecA, MdecD, MdecP***.

8.4. Pseudo Code: Cluster of a Node

In this section I am going to explain the whole algorithm for finding the cluster of a desired node with a pseudo code. The algorithm takes the data for the whole graph and the desired node of which the cluster is sought and returns the cluster in which the desired node lies. The pseudo code of the algorithm is given below:

ALGORITHM: PSEUDO CODE

```
Input Data for the graph $\rightarrow$ Data;
Input Desired Node $\rightarrow$ Clust_Node;
MODULARITY $\leftarrow$ 1;
while (MODULARITY $\neq$0)
    Clusters $\leftarrow$ Genetic_Clustering (Data);
    Required_Cluster $\leftarrow$ Clusters(Clust_Node);
    Data $\leftarrow$ Data(Clusters);
    Calculate MODULARITY;
end
Output: Required_Cluster
```

The above algorithm uses the algorithm (‘*Genetic_Clustering(Data)*’) described in the previous chapter with a slight modification of the number of islands and the genetic operations. The exact steps of the ‘*Genetic_Clustering(Data)*’ algorithm are given in the table below.

Genetic_ Clustering(Data)
1. *Graph: Data*
2. *Initialize population in 4 islands*
3. *Mutation operation in each island*
4. *Elitist selection in each island*
5. *Select 4 best individuals in each island*
6. *Migration to all other islands*
7. *Repeat steps 2 through 5*
8. *Return best fit Clusters*
	% Elitist Selection
	I1=Pop1(best); I2=Pop2(best);
	I3=Pop3(best); I4=Pop4(best);
	% Migration
	Pop1(1:4) = [I2;I3;I4;I1];
	Pop2(1:4) = [I3;I4;I1;I2];
	Pop3(1:4) = [I4;I1;I2;I3];
	Pop4(1:4) = [I1;I2;I3;I4];

Table 8.3: Genetic_Clustering (Data)

8.5. Results: Cluster of a Node

	Number of Nodes	Number of Clusters	Run-Time (seconds)	RunTime: GA1 *(Table: 7.7)*
1	8	2	0.3276	0.06
2	11	2	0.39	0.40
3	15	3	0.61	0.32
4	21	3	1.03	4.82
5	25	4	1.45	2.28
6	29	4	1.51	1.59
7	42	5	2.26	10.70
8	101	5	11.56	28.09
9	201	10	14.75	179.88
10	301	15	34.10	472.50
11	401	10	37.64	682.23
12	619	10	162.74	859.14

Table 8.4: Results of Genetic Algorithm for Cluster of a Node

We can observe that for graphs of over 25 nodes, the algorithm for finding the desired cluster runs much faster than finding all the clusters in the graph.

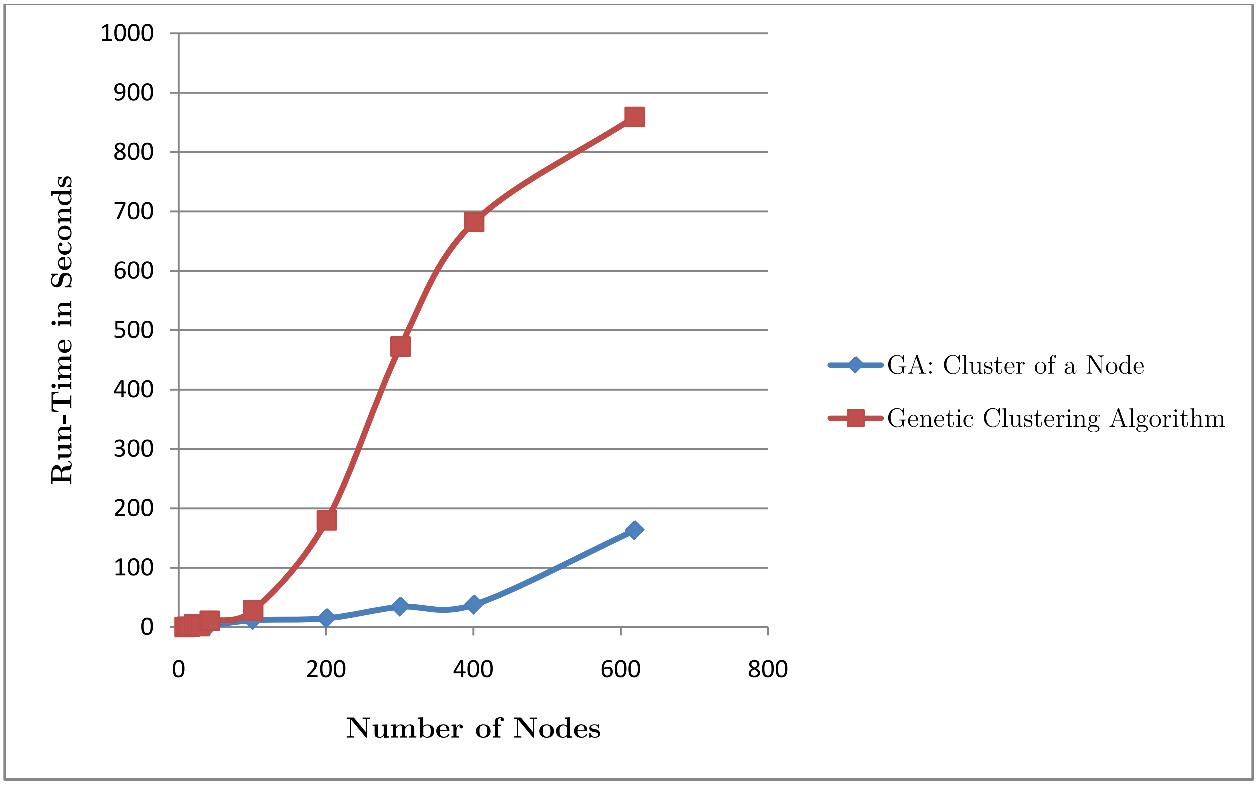


Fig 8.4: Comparison of Genetic Clustering Algorithm & GA for Cluster of a node

As the number of nodes increase the run time for finding the cluster of just one node is much better than finding all the clusters in a node. Specially, in Money-Laundering the proposed algorithm is extremely useful wherein one is usually aware of some suspicious entities.

Conclusion

Basic idea behind detecting Money-laundering is to detect the entities which cause Money-Laundering, by detecting the well connected communities in a network of financial and telecommunication data. Basic workflow steps are (i) to collect the relevant financial data, (ii) to draw a graph of the data representing financial and telecommunication transfer between the entities and lastly (iii) to find the cluster of entities which are suspicious and well connected within each other. However, in Money-Laundering, it is sometimes unnecessary to find all the clusters in the graph of financial data. More often, it is only some communities in the whole network which cause money laundering while the rest of the communities carry out legal financial operations. In such a scenario, given a suspicious entity, finding the cluster of this entity is much more useful. The thesis successfully presents a *Genetic Clustering Algorithm* which can be used for a) detecting all the clusters in a graph and b) the cluster of just one given node. The GA used for the clustering problem also incorporates the Island model with migration making the algorithm much efficient than a single population GA. The implementation of the island model with migration can be enhanced much more by parallel programming. The last topic of this thesis, Cluster of a node using GAs is a very robust algorithm specially for detecting a particular community. In the literature carried out during the thesis, we have not found any well stated algorithm which deals with this problem, efficiently.

The second substantial result of the work is the improvement of the *K-Means Algorithm*. The concept of *betweenness-centrality* has been used to detect the centers and the number of clusters, two necessary inputs of the K-Means algorithm. The complexity of the improved K-Means algorithm is slightly higher than the standard K-Means algorithm, but the accuracy of the algorithm is enhanced many folds.

The resolution limit of Modularity is one area where some improvement is required. With several sizes of clusters present in the graph, Modularity optimization does not always give the optimum solution. All the algorithms have been implemented using MATLAB on a 1.73 GHz dual-core machine with 1.5GB Ram. A possible improvement for the run time of the algorithm can be implementing them in parallel e.g. the genetic operations for different islands, calculation of betweenness-centrality for K-Means algorithm.

References

[1] Bondy J. A., Murty U. S. R.; Graph Theory- Graduate Texts in Mathematics Series, *Springer* 2008.

[2] Merris R.; Graph Theory, Wiley-Interscience Series in Discrete Mathematics and Optimization, *John Wiley & Sons, Inc.* 2001.

[3] Diestel R.; Graph Theory- Graduate Texts in Mathematics Series, *Springer* 2000.

[4] Page L., Brin S.; The anatomy of a large-scale hypertextual web search engine, *Proceedings of the Seventh International Web Conference (WWW 98)*, 1998.

[5] Page L., Brin S.; The PageRank Citation Ranking: Bringing Order to the Web, 1998.

[6] Broder A. Z., Lempel R, Maghoul F, Pederson J; Efficient PageRank approximation via graph aggregation, *Springer Science & Business Media, Inc.* 2006.

[7] Cormen T., Leiserson C., Rivest R., Stein C.; Introduction to algorithms, *The MIT Press / McGraw Hill* (2003).

[8] Haveliwala T. H., Kanvar S. D.; The second eigenvalue of the Google matrix, *Stanford University,* 2003.

[9] Chung F.; Spectral Graph Theory, *American Mathematical Society 2nd edition,* 1997.

[10] Spielman D.; Combinatorial Scientific Computing; *Chapman & Hall/CRC Computational Science,* 2012.

[11] Hanneman, Robert A. and Riddle M.; Introduction to Social Network methods. Riverside, CA: *University of California,* 2005.

[12] Kleinberg J. M.; Authoritative Source in a Hyperlinked Environment, *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms,* 1998.

[13] Brandes U.; A faster Algorithm for Betweenness Centrality, *Journal of Mathematical Sociology 25(2):163-177,* 2001.

[14] Newman M. E. J.; A measure of betweenness centrality based on random walks. *Social Networks 27, 39–54* (2005).

[15] Newman M. E. J.; The structure and function of complex networks. *SIAM Review 45(2), 167–256* (2003).

[16] Newman M. E. J.; Detecting community structure in networks. *The European physical journal B, Condensed matter physics 38(2), 321–330* (2004).

[17] Newman M. E. J.; Finding community structure in networks using the eigenvectors of matrices. *Physical Review E 74(036104)* (2006).

[18] Newman M. E. J.; Girvan, M.: Finding and evaluating community structure in networks. *Physical Review E 69(026113)* (2004).

[19] Newman M. E. J.; Fast algorithm for detecting community structure in networks. *Physical Review E 69(066133)* (2004).

[20] Sibson R.; SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method, *The Computer Journal (British Computer Society)* ***16*** *(1): 30–34.* 1973.

[21] Defays D.; An Efficient Algorithm for a Complete Link Method, *The Computer Journal (British Computer Society)* ***20*** *(4): 364–366.* 1977.

[22] Schaeffer S. E.; Survey: Graph Clustering, http://dx.doi.org/10.1016/j.cosrev.2007.05.001.

[23] Fortunato S., Barthelemy M.; Resolution Limit in Community Detection, *Proc. Nat. Acad. Sci., Vol. 104, pp. 36-41*, 2007, www.pnas.org/cgi/doi/10.1073/pnas.0605965104.

[24] Arenas A., Fernandez A., Gomez S.; Analysis of the Structure of Complex Networks at Different Resolution Levels, *New Journal of Physics 10(2008) 053039.*

[25] MacQueen J.; Some Methods for Classification and Analysis of Multivariate Observations, *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. pp. 281–297,* (1967).

[26] Zhao Q., Hautamäki V., Fränti P.; Knee Point Detection in BIC for Detecting the Number of Clusters. ACIVS 2008: 664-673.

[27] Hu X., Xu L.; Inverstigating on Several Model Selection Criteria for Determining the Number of Cluster, *Neutral Information Processing-Letter & Reviews, Vol. 4, No. 1*, 2004.

[28] Feng Z., Xu X., Yuruk N., Schweiger T. A. J.; A novel similarity-based modularity function for graph partitioning, *DaWak'07*, pages 385–396, 2007.

[29] Huang J., Sun H., Han J., Deng H., Sun Y., Liu Y.; SHRINK: A Structural Algorithm for Detecting Hierarchical Communities in Networks, *ACM 978-1-4503-0099,* 2010.

[30] Goldberg D. E., Holland J. H., Booker L B; Classifier Systems and Genetic Algorithms, *Artificial Intelligence 40, pp. 235-282*, 1989.

[31] Goldberg D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[32] Gordon V. S., Whitley D.; Serial and Parallel Genetic Algorithms as Function Optimizers, Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 177-183, 1993.

[33] Lipczak M., Milios E.; Agglomerative Genetic Algorithm for Clustering in Social Networks, *ACM 978-1-60558-325,* 2009.

[34] Lorena L. A. N., Furtado J C; Clustering Genetic Algorithm for Clustering Problems, *Evolutionary Computation 9(3), MIT press* 2001.

[35] Lin H. J., Yan F. W., Kao Y. T.; An Efficient GA-based Clustering Technique, *Tamkang Journal of Science and Engineering, Vol. 8, No. 2, pp. 113-122* (2005).

[36] Chen D.; A Novel Clustering Algorithm for Graphs, International Conference on Artificial Intelligence and Computational Intelligence, *IEEE Computer Society,* 2009.

[37] Chiou Y. C., Lan L. W.; Theory and Methodology- Genetic Clustering Algorithms, *European Journal of Operational Research 135 pp. 413-427*, (2001).

[38] www.aia.es

[39] www.hemolia.eu

CRM
CENTRE DE RECERCA MATEMÀTICA

Campus de Bellaterra, Edifici C
08193 Bellaterra, Spain
Tel.: +34 93 581 10 81
Fax: +34 93 581 22 02
crm@crm.cat
www.crm.cat