COMMUNITY DETECTION MODEL USING GENETIC ALGORITHM IN COMPLEX NETWORKS AND ITS APPLICATION IN REAL-LIFE NETWORKS

by

Mürsel Taşgın

B.Sc. in Computer Engineering, Middle East Technical University, 2002

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2005

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Haluk Bingöl for his invaluable guidance and helping during the preparation of this dissertation. I would like to mention his patience; giving me inspiration when I was stuck at dead-ends.

I would like to thank to the committee members Prof. Levent Akın and Assoc.Prof. Yağmur Denizhan for their contributions to this work.

I have very much benefited from the discussions with my dear friends in the Computer Engineering Department, among whom I would like to mention especially İsmail Güneş, Amaç Herdağdelen and Burak Çetin. I thank them all.

I would like to thank to all my teachers who guide me to improve myself with their great contributions.

I would like to dedicate this work to dear Zehra Özaydın and to my family.

ABSTRACT

COMMUNITY DETECTION MODEL USING GENETIC ALGORITHM IN COMPLEX NETWORKS AND ITS APPLICATION IN REAL-LIFE NETWORKS

This work is an endeavor towards analyzing complex networks. Mainly, a community detection algorithm based on genetic algorithm will be introduced, and detailed background will be developed.

Firstly, we introduce community detection methods in complex networks. A community in a complex network is a group of nodes that has more connectivity within and less connectivity with other communities. There are many community detection algorithms proposed so far, some of which performs very well, however most of them are not feasible in identifying communities in large complex networks, where many of the real life examples of the complex networks are large complex networks (e.g. www network, e-mail networks) due to time complexity of the algorithms. We introduce and apply a community detection algorithm on some real-life complex networks, like Zachary's Karate Club and the Enron e-mail network. Zachary's Karate Club network is a well-known network dataset. We collected data of Enron e-mail network and processed that data to form the Enron e-mail network.

We present a community detection algorithm that is based on the network modularity (Q) and is scalable to very large networks that has 100,000 nodes. We run our algorithm on known networks to assess the accuracy of our algorithm and then on Enron e-mail dataset as well to examine the scalability of our algorithm. Our algorithm gives optimal community structure in very short time and is scalable to very large networks.

ÖZET

GENETİK ALGORİTMA KULLANARAK KARMAŞIK AĞLARDA ALTTOPLULUK BULMA MODELİ VE BU MODELİN GERÇEK AĞLARDA UYGULANMASI

Bu çalışma karmaşık ağların incelenmesine yönelik bir çalışmadır. Temel olarak, bir ağdaki alttoplulukları genetik algoritma tabanlı bir algoritma ile bulmayı sağlayan bir algoritma tanıtılacak ve ilgili altyapı detaylı olarak geliştirilecektir.

Öncelikle karmaşık ağlarda alttopluluk yapısını ortaya çıkaran algoritmalar tanıtılacaktır. Bir karmaşık ağdaki alttopluluk yapısında, alttopluluk üyeleri kendi aralarında daha sıkı, diğer alttopluluklardaki üyelerle ise zayıf bağlıdır. Bugüne kadar alttopluluk yapısını ortaya çıkaran ve bazıları çok iyi sonuçlar veren birçok algoritma önerilmiştir. Ancak bu algoritmaların birçoğu çalışma süresi bakımından büyük ağlarda yetersiz kalmaktadır ki gerçek hayattaki birçok karmaşık ağ örneği büyük karmaşık ağdır (ör: www ağı, e-posta ağı). Karmaşık ağlarda alttopluluk bulan yeni bir algoritma geliştirip, bu algoritmayı gerçek hayattaki Zachary'nin Karate Kulübü ve Enron e-posta ağı gibi kompleks ağlarda uygulamaktayız. Zachary'nin Karate Kulübü ağı bilinen bir karmaşık ağ örneğidir. Enron e-posta ağını ise, verileri toplayarak ve bu verileri işleyerek oluşturmuş bulunmaktayız.

Ağ modülerliğini (*network modularity*) baz alan ve genetik algoritma kullanarak alttoplulukları belirleyen ve 100,000 üyeye sahip çok büyük karmaşık ağlarda çalışabilen, ölçeklenebilir bir algoritma sunmaktayız. Algoritmamızı, doğruluğunu görmek için alttopluluk yapısı bilinen ağlarda ve ölçeklenebilirliğini görmek için de Enron e-posta ağında çalıştırdık. Algoritmamız, makul bir süre içinde optimal alttopluluk yapıları ortaya çıkarmakta ve çok büyük ağlara ölçeklenebilmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

e_{ij}	Number of links connecting community *i* to community *j*
GA	Genetic Algorithm
GACD	A community detection algorithm based on genetic algorithm
n	Number of vertices in a network
P(k)	Degree distribution probability in complex networks
Q	Network Modularity in complex networks

1. INTRODUCTION

1.1. Motivation and Essentials

Complex networks have attracted increasing interest in recent years. A complex network is a representation of a system, where the members of the system, called vertices or nodes, are linked to each other via edges according to a relationship between the nodes. Complex networks have many characteristics that hide information about the birth of the network, its growth mechanism and the information flow within the networks. Some of the examples of complex networks are;

- *co-authorship networks*, where nodes are scientists and an edge is drawn between two nodes if they co-authored in same paper
- *the Internet network*, where nodes are routers and edges the connectivity between the routers
- *protein networks*, where nodes are proteins and edges represent the protein interactions between the nodes
- *friendship networks*, where nodes are people and edges represent the friendship between people.

Network representation of real life systems in terms of nodes and edges is based on graph theory and details of the network representation are given in **Appendix B** in details.

In a complex network, even if the network has many nodes and a large size, the average distance between any two nodes within the network is short. This property is called the small-world property. Another important feature of the complex networks is the right-skewed degree distribution of the network. Degree of a node is the total number of connections it has. If we plot the degree distribution of all the nodes in a complex network, we see that very few number of nodes have high degree and many of the nodes have less degree; which results a right-skewed plot for degree distribution in the network. Degree distribution tells us that there are a few numbers of nodes that have high connectivity with

other nodes. For example in the Internet network, some routers are heavily connected as they are the hubs in the network.

A third property that complex networks have in common is the community structure of the network. A community is a subgroup of nodes inside a complex network, where they have more connections with the nodes within the same community and less connection with the nodes that reside in other communities. For example, in friendship networks, the communities are formed by close friends, where all of them are connected to each other while some of them have connections to other people outside their friendship community.

Communities are the key elements of complex networks; for this reason identifying the community structure in complex networks attracts a great motivation in recent years. Communities and community structure of the complex networks are important, because it hides the information flow mechanism and growth model of a network. Most importantly, community structure reveals similar nodes in the network, a property which is crucial for protein networks, genetic networks, social networks, e-main networks and even for terrorist networks. As a result, community detection becomes an important topic in complex networks.

Many algorithms are proposed to identify communities that can be applied to raw data like e-mail messages among people, chat logs and telephone communications. The most famous community detection algorithm is Girvan-Newman (Girvan and Newman, 2002) algorithm that is proposed by Girvan and Mark Newman. Girvan-Newman algorithm, known as GN algorithm, is based on the betweenness centrality. It is a divisive algorithm and finds very accurate results in $O(n^3)$ time, where n is the number of nodes in network. On the other hand, there are some faster algorithms that are agglomerative and run in $O(n^2)$ time. Artificial intelligence methods are also used to detect communities in complex networks like Extremal Optimization (Duch and Arenas, 2005). However most of the algorithms are designed for small or mid-sized networks and are not suitable for very large networks. Newman proposed a faster algorithm for very large networks which is the only proposed algorithm for very large networks so far (Newman, 2004).

The problem in community detection is the time-complexity of the algorithms, which make the usage of these algorithms for very large networks impractical. Many real-life networks are very large, having thousands of nodes. However most of the proposed algorithms have very high time-complexity and are not suitable for identifying community structure in very large real-life networks. Another problem is that most of the algorithms produce a hierarchical structure, which is called *dendogram* and is explained in **Appendix B** in detail, that needs to be cut at some level to give community structure information. However, in order to cut the dendogram, one should know the number of the communities or some other priori information about the network, which is impossible for most of the networks where we do not know anything about the community structure. Some algorithms also need some prior knowledge about certain threshold values.

Our work proposes an artificial intelligence based optimization process, namely genetic algorithm and tries to optimize a network's global variable, network modularity (Newman and Girvan, 2004). Network modularity has become a key element in community detection, first proposed by Newman and Girvan and used by many algorithms. Genetic algorithm provides a good solution for community detection with its scalability for very large networks and with its mechanism that does not need any priori information like number of communities or some threshold values.

In our work we mainly present and analyze a community detection algorithm we have named GACD, which performs very well for very large networks and produces optimal solutions. The algorithm is iterative with O(n) time-complexity in the number of edges in the network. This linear time-complexity is central for our work because we intend its application on very large networks, like the Enron e-mail dataset.

In coming chapters we will introduce the algorithm formally and analyze its theoretical aspects. We have used the algorithm on networks which we know the actual community structure, like Zachary's Karate Club. We also performed runs on networks of different size to see the performance of the network in both small and in very large networks. We used Enron e-mail network as the very large network, where Enron e-mail network has 93 526 nodes and 344 264 edges.

Genetic algorithm is an artificial intelligence method and is based on evolutionary computation. It was first proposed by John Holland (Holland, 1975) and it was used as a useful tool and as a method to expand the understanding of adaptation in problem solving.

A genetic algorithm (GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states, rather than by modifying a single state. The analogy to natural selection is the same stochastic beam search, except that now we are dealing with sexual rather than asexual reproduction (Russell and Norvig, 2003). In genetic algorithms, the solution space is formed by a number of solution candidates; members. The number of members which form a population is parametric. For some number of iterations, members reproduce to have better solution candidates and pass those members to new generations.

In each iteration, reproduction of better members is carried out by some operations. First of all each solution member in the population is tested according to a function, called fitness function, and the result of fitness function, fitness value, determines the superiority of each member in terms of the real solution set. The members are sorted according to their fitness values and then genetic operations are held through the members. Cross-over between superior members and some random mutation on some members are carried out aiming the production of better solution members for the next generation.

The use of GACD has offered us the possibility to analyze community structure of networks from very small well known networks, Zachary Karate Club, to very large networks, Enron e-mail network, as it is a scalable algorithm. We hope the GACD algorithm and genetic algorithm in general prove to be useful tools in the future not only for community detection but also for different features of complex networks that need to be uncovered but have the difficulty of scalability.

1.2. Current Techniques

The most important community detection algorithms are Girvan-Newman (GN) that is based on edge betweenness, fast algorithm for detecting communities, Radicchi's algorithm, Wu-Huberman algorithm based on voltage drops, Reichardt-Bornholdt method based on Potts model and community detection using Extremal Optimization method.

a) The Girvan-Newman algorithm: The GN algorithm (Girvan and Newman, 2002) is based on removal of the edges according to edge betweenness (Appendix B). The edges are repeatedly removed from the network, starting from the node with highest betweenness centrality. The algorithm recalculates edge betweenness after removal of every edge and stops when there are no edges between vertices. This is a divisive algorithm and finds very accurate results in small networks.

b) Fast algorithm for detecting communities: This algorithm (Clauset *et al.*, 2004) is a hierarchical agglomerative approach based on the network modularity measure. Network modularity (Newman and Girvan, 2004) is a measure that determines the quality of splitting of a network into communities. The algorithm is a hierarchical agglomerative algorithm that starts with n communities of n vertices and merges the communities into bigger communities that yields the highest increase in network modularity. Each potential merge of two communities has a contribution of ΔQ value to network modularity Q, and the idea is to merge two communities that yield the highest increase in Q, that is the selection of the tuple with highest ΔQ value at each step. The algorithm produces a *dendogram* that is a hierarchical tree structure of the network; details of dendogram will be given in **Appendix B**. The algorithm is then modified so that it can perform on very large networks (Newman, 2004). It is the first algorithm proposed for community detection in very large networks.

c) Radicchi's algorithm: Radicchi's algorithm (Radicchi *et al*., 2004) is similar to GN algorithm; however it uses a different measure instead of edge betweenness. The measure is based on counting the short loops in the network, loops of length three, or triangles, in the simplest case. As in GN algorithm, this measure is recalculated after each removal, but

it is a local measure that can be calculated quickly, hence overall algorithm is faster than GN.

d) Wu-Huberman algorithm: The algorithm is based on voltage drops (Wu and Huberman, 2004). An electric circuit is formed by placing a unit resistor to each edge of the network and then applying a unit potential difference between two vertices chosen arbitrarily. The communities are identified by finding the largest voltage gap.

e) Reichardt-Bornholdt method: This algorithm assigns a spin state between 1 and q to each node at random (Reichardt and Bornholdt, 2004). The energy of the spin system is determined using a q-Potts Hamiltonian. A q-Potts Hamiltonian is a physics method, where the details is out of scope of this work and interested reader can get further details in (Reichardt and Bornholdt, 2004). The idea is that in the ground state of the system, communities are identified as groups with equal spin values. To get to the ground state, the system is allowed to evolve using a simple Monte-Carlo method with simulated annealing (Appendix B).

f) Community detection using Extremal Optimization: The algorithm is again a divisive method which is based on network modularity, Q (Duch and Arenas, 2005). Network is divided into two random communities, and each node in the communities has contribution value λ to network modularity Q. At each step the node with the lowest λ is transferred to other community. The transfer of nodes between communities continues until there is no increase in Q value of the network. At that time the network is divided into two communities and algorithm continues the same process recursively in the smaller communities.

The community detection algorithms mainly try to find communities by dividing the community into “reasonable” partitions or reversely merge nodes iteratively into same community. Other approaches try to optimize a global property of the network, which is in most cases the network modularity, Q. We choose to optimize the network modularity using genetic algorithm, because of the fact that it can be scaled to very large networks. We think that divisive or agglomerative approaches may not be suitable for large networks

all the time, because of intense computations during each iteration after a merge or a division.

The methods that try to optimize a global property of network are Reichardt-Bornholdt method and Extremal Optimization method. The latter method can also be accepted as a divisive method. The details of the community detection algorithms and comparison of these algorithms will be given in details in Chapter 3.

Genetic algorithm is an efficient algorithm that runs in O(n) time-complexity in number of edges. The fitness function is critical in the optimization process. We thought that the network modularity, Q, which is a global property of network, can be used as fitness evaluation criteria in genetic algorithm. We have found GA to be very valuable in providing a compact and clear framework which has a natural adeptness to our analysis purposes. In our opinion the concept of optimizing the whole network in terms of accurately specified communities provides reliable, fast and "optimum" results in both small and very large networks.

We believe that we also eliminate the problem of identifying the number of communities in the network. Many proposed algorithm provides hierarchical structure of the network, however the hierarchical structure gives information about communities only when the number of communities is known. Some algorithms even need the number of communities in the beginning. In our algorithm, the number of communities is the part of the optimization and is not need to be provided by user.

1.3. Main Contributions

The genetic algorithm is introduced in (Holland, 1975), as an optimization process model for complex problems where the result can be found by exhaustive search however exhaustive search is not practical because of time-complexity. Genetic algorithm uses genetic operations like ones in biology; i.e. cross-over, mutation. We believe that using GA as an optimization process for community detection in complex network may provide

beneficial for further research. To help facilitate this we develop a community detection algorithm based on GA and try to enumerate useful applications.

Possibly our most important contribution is the introduction of GA in community detection in complex networks. GA provides a rapid way to identify communities in a complex network very optimally and sometimes 100% accurately, a problem which needs a lot of time and computation power with other community detection algorithms in very large networks. Our algorithm provides very reasonable time-complexity which makes it feasible to use the algorithm for very large real-life complex networks.

Another contribution of our work is that our algorithm needs no priori knowledge about the complex network, like number of communities or some threshold values, where these values are needed in almost all of the community detection algorithms. Also, our work is the first effort for community detection in Enron e-mail network, which is made public in 2004 and contains e-mail connectivity of employees of Enron Corporation.

We provide a theoretical treatment of the algorithm (GACD), and cover topics such as convergence and accuracy. We also provide the use of GACD in very large networks.

GACD which is based on GA appeared to be a promising algorithm in our experiments on Zachary Karate Club network. GACD is desirable because it decomposes the network into correct communities without the priori knowledge of number of communities unlike many proposed algorithms, and it is also scalable to very large networks. We implemented and had the chance to test our algorithm in complex networks using different parameters.

1.4. Experimental Setup

For evaluating our work we have mainly worked on well known Zachary's Karate Club network, College Football network and later the Enron e-mail network, which had around 94 000 nodes and 344 000 edges.

We have written our own source code for reading network data and performing our GACD algorithm. We used Java as the programming language and Eclipse 3.0 as the application development environment. Our network reader program reads network files that are in Pajek (PAJ) format. Because of the scalability issues and analysis of Enron e-mail network, procedures and algorithms are optimized in terms of memory and CPU usage.

Enron e-mail dataset was stored in 512 000 text files in many recursive folders, when it was first made public. We coded a crawler program to scan all the e-mail files and extract "From", "To", "Cc" and "Bcc" parts from e-mails into a big database. After parsing all the text files into a single file, a complex network file is built where vertices are the e-mail addresses and the edges are the connections, where a connection occurs if a person sends e-mail to another person. The data is then converted to Pajek (PAJ) format.

Additionally, the comparison of our algorithm's results with the real community structure required coding of a mapping algorithm between different community configurations of the same network. We implemented a simple and useful algorithm for this purpose using Java language.

1.5. Organization of the Document

The rest of the document is organized as follows. Chapter 2 contains all the related literature survey and preliminary offered. Three topics are treated; these are Complex Networks, Community Detection Concept, and Genetic Algorithm.

Chapter 3 formally introduces the community detection in complex networks and its uses. Chapter 4 explores the use of Genetic Algorithm as community detection method.

Chapter 5 is devoted to GACD and its application on Zachary's Karate Club network and College Football network. The algorithm GACD is presented here, and is examined closely. GACD is defined in this chapter.

Chapter 5 contains all the experimental results we have obtained. These include general analyses of Zachary's Karate Club network, College Football network and their community structures. The accuracy of GACD as a community detection algorithm is examined. Comparative plots are presented to investigate similarities in different community detection schemes. The success of the algorithm with different GA parameters and iterations are listed, and found to be very accurate to real community structure. The Enron e-mail network data is also analyzed with the algorithm to show that it is scalable to very large networks.

The conclusion and future work sections are in chapter 6. Appendix A is a brief treatment of the cluster similarity implementation created as background work and Appendix B contains the general concepts about complex networks and community detection in complex networks.

2. RELATED RESEARCH AND OTHER PRELIMINARIES

In this chapter we try to develop the foundation for three main topics of interest in our work, and present further the details of genetic algorithm background necessary for the rest of the text.

In section 2.1, we focus on Complex Networks whose features we want to uncover with our proposed algorithm in our work. A complete formal introduction is not necessary as it is available elsewhere (Strogatz, 2001), instead we try to develop a useful intuition and formally present only what is sufficient about the Complex Networks.

We conceive our work as an attempt to develop a useful tool for identifying community structure in Complex Networks. So in section 2.2, there is a brief survey of the Community Detection concept which lays the foundation for our motivation of implementing a community detection algorithm.

Ultimately, in this text we develop a fast, reliable and scalable community detection algorithm based on Genetic Algorithm. Thus, we present in section 2.3 an overview of Genetic Algorithm primarily focusing on the concept and the fields of usage of the algorithm for problem solving.

2.1. Complex Networks

A network is a set of items where each element is called a *node* or *vertex* connected to each other via *edges*. Scientists have been analyzing many networks including food web, WWW, electrical power grids, cellular and metabolic networks, co-authorship and citation networks (Strogatz, 2001). Why is network anatomy so important to characterize? Because the structure always affects the function. The structure of social network affects the spread of information and disease, topology of power grids affects the robustness and stability of power transmission.

The complex systems that form networks attain great interest. The complex networks are the structures that have many different properties from ordinary networks. They have structural complexity and their topology evolves over the time by newly added nodes or removed nodes. The links between nodes may have different directions and weights.

Traditionally the study of complex networks has been the territory of the graph theory (Albert and Barabasi, 2002). While the graph theory initially focused on regular graphs, since 1950s large scale networks with no apparent design principles have been described as random graphs. However complex networks should have a more sophisticated structure rather than random graphs. Internet or other complex networks cannot be formed in a random manner.

In (Albert and Barabasi, 2002), some of the features of complex networks are defined. Small-world property, the first one, is the fact that the nodes in a large network connect to each other via relatively short paths. The distance between two nodes is the number of edges along the shortest path connecting them. The most popular manifestation of small-world property is the Stanley Milgram's "*six degrees of separation*", who concluded that there was a path of acquaintances with a typical length of about six between most pairs of people in the United States.

The small-world property is not a property of only complex networks; random networks may have small-world property as well. The diameter of complex networks is also an identifier of small-world property (Dorogovtsev and Mendes, 2002). The average shortest-path length value is small even for very large networks. The maximal shortest path length over all the pairs of vertices between which a path exists is the maximal extent of the network, and it is the diameter of the network.

Another important feature of complex network is the clustering property, which is the base of our work. Complex networks have clusters, where the members within clusters know each other, like circle of friends. The tendency of clustering is quantified by clustering coefficient in (Watts and Strogatz, 1998). Clustering coefficient determines how well connected are the neighbours of a node with each other; i.e., if a node i has k_i

neighbours, the number of all possible connection among the neighbours is $k_i(k_i -1)/2$. The ratio between the actual number of edges E_i and the total number $k_i(k_i -1)/2$ gives the clustering coefficient of node i:

$$C_i = \frac{2E_i}{k_i(k_i - 1)} \tag{2.1}$$

The clustering coefficient of the whole network is the average of all individual C_i's. Clustering coefficient in complex networks is much larger than that in random networks.

The nodes in complex networks do not have the same degree distribution. Some nodes in the complex networks are more connected than other nodes. The degree distribution function *P(k)* gives the probability that a randomly selected node has exactly k edges. The degree distribution in complex networks has a power-law tail, which means that many nodes have small degrees while small number of nodes has high degree. Such networks are called scale free (Albert and Barabasi, 1999). The degree distribution function has a form;

$$P(k) \approx k^{-\gamma} \tag{2.2}$$

where γ is between 2 and 3 for most of the scale-free networks.

Generally, a network may contain disconnected parts. In undirected networks, if relative size of the largest connected cluster of vertices of a network (*the largest connected component*) approaches to a nonzero value when the network is grown to infinite size, the system is above the percolating threshold and this cluster is called the *giant connected component* of the network (Dorogovtsev and Mendes, 2002). The next largest cluster is small compared to the giant connected component for a large enough network.

Complex networks evolve over time; new nodes and edges are added, some nodes and edges are removed, or both at the same time. Newly added nodes show interesting behaviours in complex networks. Many network models before (Albert and Barabasi, 1999) assumed that the probability that two nodes are connected is independent of the nodes' degrees, i.e., new edges are placed randomly. Most real networks exhibit *preferential attachment* such that the likelihood of connecting to a node depends on the node's degree (Albert and Barabasi, 2002). A popular web page will more likely to include hyperlinks to popular documents with already high degrees. The probability **Π** that a new node will be connected to node *i* depends on the degree of k_i of node *i* such that

$$\Pi(k_i) = \frac{k_i}{\sum_j k_j} \tag{2.3}$$

where *j* is the total number of nodes.

2.2. Community Detection

A community is a collection of individuals who have more connectivity inside the community as a result of sharing same ideas, having similar features or locating in near neighbourhood. In sociological terms, a community in society is a group of people who have strong communication with each other, in terms of meetings, telephone calls, e-mail messages etc. In biology, a community is a group of living organisms who compose a life environment by locating together, forming a food-chain etc.

Most of the interactions in real life can be visualized as networks, where an individual is a node and an interaction between two nodes is an edge. The interaction can be a communication between nodes or having a common feature. A food-web network, e-mail network among people, the Internet network and collaboration network is some examples. Such kind of networks contains community structures, even if all the nodes seem to have connectivity with many or all the other nodes.

Detecting the community structure in the information networks allows one to mine information in a more efficient way, narrowing the exploration of a very large network into a small portion. Community structure identification can also reveal the informal organization structure in the network and the nature of information flow in the network.

The division of the individuals of a social network into communities is a fundamental aspect of a social system. In fact, the subgroups in social systems often have their own norms, orientations and subcultures, sometimes running counter to the official culture, an are the most important source of a person's identity (Scott, 2000). Because of this, the community detection or definition and identification of subgroups have become one of the main concerns of social network analysis. The first community detection algorithms were proposed in social network analysis.

Community detection is an important issue because the subgroups or communities are the basic elements in networks. They give information about the modular structure, growth mechanism and functioning of the network.

Community detection in biological networks like protein networks, gene networks will give valuable information in identifying the similar type objects. The raw data of a network will not give the community information at first look. A careful analysis is needed for identifying communities. Sometimes a node which seems to be connected to many nodes in a network can belong to only one community and one can not tell by just looking at its connections.

There have been many methodologies and algorithms to automate the process of community detection in social, biological, technological and other networks. As the size of the networks may be very large, there is a need to find an accurate and automated schema to uncover the community structure in real-life networks. Some applications also need the community structure information on-the-fly i.e., search engines, that makes the community detection more valuable.

In our work, we will mostly be dealing with complex networks and community detection in complex networks. In Chapter 3, community detection algorithms are introduced and explained in details.

2.3. Genetic Algorithm(GA)

In our work, we have proposed a community detection algorithm based on genetic algorithm. In this section we will introduce the necessary terminology, and the concepts on which we later on build our work.

This will allow us to depict an outline of the theory which should allow the reader to develop an understanding on the definition and usage of genetic algorithm in problem solving. The interested reader, who is willing to learn more about genetic algorithm is advised to consult the references (Holland, 1975) and (Russel and Norvig, 2003) for a comprehensive treatment of genetic algorithm. The reader should note that the algorithm is taken as the basic method and there are minor modifications in the implementation of GACD by keeping the analogy with the original algorithm.

2.3.1 Problem Solving using Genetic Algorithm

Genetic algorithm is first introduced in (Holland, 1975) by showing that evolutionary process in real life can be applied to solve a variety of problems in computer science. It is an artificial intelligence (AI) method for problem solving. Genetic algorithm (GA) tries to find solution to a problem by genetically reproducing a population of individuals (which are candidates of the best solution to the given problem) over a series of generations. During each reproduction of a new generation, the genetic algorithm applies some genetic principles like survival of the fittest, mutation and cross-over.

Genetic algorithm tries to optimize the solution set for a number of iterations, and at the end picks up the best available solution as the solution, which is the most optimized or often the exact solution to a problem. The algorithm has 4 basic functions:

- Initial population creation
- Fitness evaluation
- Cross-over
- Mutation

The solution space is defined and solution members are initialized once during initial population creation, and then the iterations start where the basic genetic functions take place at each iteration.

During each iteration the algorithm needs to improve solution sets further by the help of functions it has. For this purpose, inspired from the evolution of nature, the algorithm evaluates the goodness of each member in terms of providing the best solution and then performs cross-over operation between the better members. After cross-over some randomly selected members are mutated according to some parameters; to give chance to produce different genes that did not exist in the population. The iterations continue for a predefined number of times to reveal the most optimal solution.

2.3.2 Fundamentals of Genetic Algorithm

i. Initial Population Creation

Initial population creation is the first step of the genetic algorithm; the problem should be described in a representation scheme and the members of the possible solution set are expressed in a certain form. All the members who are candidates of solution form the *population*. The members of the *population* are like individuals in society; they have a chromosome structure, a quality measure and they reproduce by sexual reproduction with

other members in the society to give birth to genetically better members for next generations.

The user designs an artificial chromosome of a fixed size and finds a mapping between the solution set members in the search space and instances of the artificial chromosome.

ii. Fitness Evaluation

The algorithm needs a quality measure in order to evaluate the goodness of each member in the population in terms of finding the most desirable solution to a specific problem. For this purpose; genetic algorithm should define and try to maximize a fitness value, which assesses a member's ability and quality for problem solution. For the rectangle example, result of a function which favors the minimum placement area and minimum overlapping of rectangles will be a good fitness value, and this function will be taken as the fitness evaluation function for the algorithm.

Fitness evaluation is the hardest and the most crucial part of the algorithm implementation. Because the selection of best fitness function in order to reach desired solution is not straightforward for most of the real-world problem. The selection of the fitness function may provide an easy and fast convergence to solution or provides non-optimal solutions as well. Fitness function should contain the elements of a good and bad solution, so that it can favor or punish a solution member.

In each iteration, the members in the population are evaluated according to the fitness function and they are assigned a fitness value. Afterwards, the members of the population are sorted according to their fitness value. After fitness evaluation, the iteration continues with reproduction of next generation members via cross-over and mutation.

iii. Cross-over

In reproduction phase, new individuals who are different from both of their parents are produced via cross-over. The newly produced members are not completely different; as they are built from both of their parents' genes. In reproduction phase, the cross-over is performed on fixed length individuals, which are fixed length arrays. The children have the same first part as one of their parents, and after some selection point, they have the second part from the other parents. As a result the children that are produced for next generation will be individuals that are formed by the genes from both parents.

See Figure 2.1 for the details of cross-over. Here two members are selected according to their fitness value; their genes are cut at a random or predefined cross-over point. Then the cut parts are exchanged between the members and new childs are generated from both of the parents in the result.

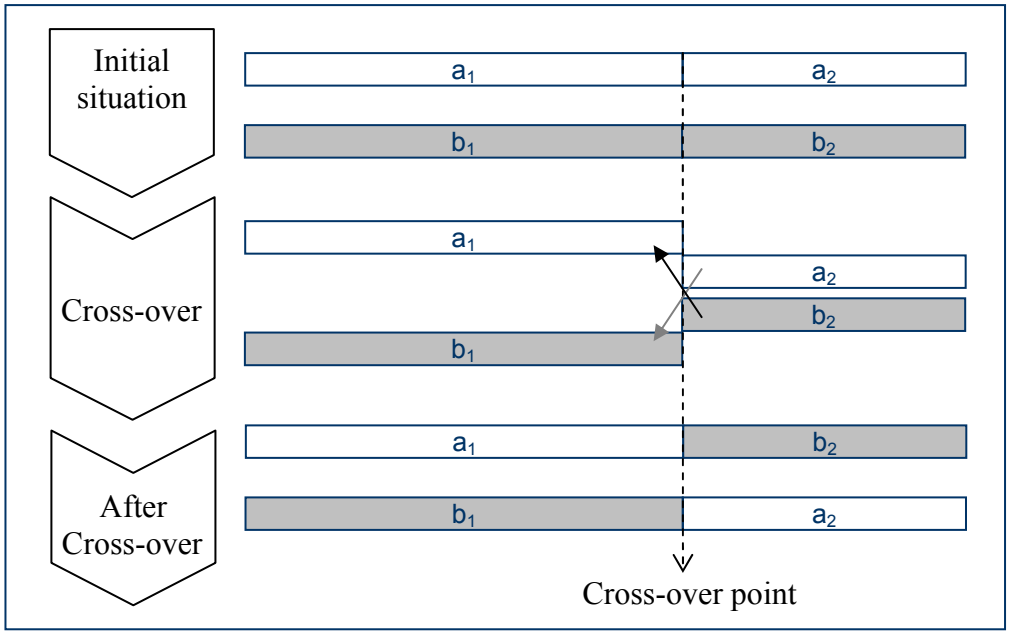

Figure 2.1. Cross-over operation

The rate of the cross-over and cross-over selection point are some of the predefined parameters in the genetic algorithm which need to be carefully fixed and can affect the performance of the algorithm. Some problem domains needs many number of cross-over operations to produce diversity while some needs only a necessary number of cross-overs where excess number of it will create a random population.

iv. Mutation

Each member in the population has an equal probability for mutation. In mutation phase, one or some number of members are selected and one single bit in their genes are mutated, XOR'ed in logical terms. By this operation the population can have members that did not exist before in it and this helps to solve stuck situations in problem solving.

The algorithm can include a mechanism for preserving the better genomes in iterations. Preserving of the better genomes guarantees that the algorithm never lose the member with highest fitness value. Since better genomes exchange genes in cross-over and mutation can happen to any member, there is always a possibility to have members with worse fitness value in the result of cross-over or mutation. To overcome this problem, before any genetic operation, some predefined number of members can be kept and the genetic operations can be done. If the genetic operations produce better results than the preserved ones, they are changed with the newer and better ones. Otherwise, the better members will keep alive for next generations.

The preserving of better genomes will improve the quality of population by keeping the best members alive who are generated in different iterations. The number of preserved members should be limited for not limiting the population into a narrow gene class. Preserving of better genes or members can be omitted in some implementations. In our work, we use the preserving of better genomes to improve the quality by keeping the best members from different iterations.

2.3.3 Performance of Genetic Algorithm

The genetic algorithm is a promising approach in problem solving when the search space is very large for an exhaustive search and an optimal solution is adorable. The power of the algorithm comes mainly from the cross-over operation, which tries to come together the best parts of each member to create better members. Like in natural evolution, the members in the society get genes from his/her mother and father and there is a probability that the children is better than both of his/her parents by taking good genes from both.

The genetic algorithm is very fast in converging the solution set to a narrow area in very short time. The best solution is not always guaranteed, but it does its best to find an optimized solution. Problem domains where a lot of cases affect the result are best playgrounds for the genetic algorithm, because it can set many parameters randomly and always improves the solution set by increasing the quality of solution members in each iteration.

In our work on this text we will implement a community detection method based on genetic algorithm, with some minor modification in genetic algorithm by preserving the main methodology of the algorithm.

3. COMMUNITY DETECTION IN COMPLEX NETWORKS

Complex Networks have many interesting features that attract attention to this field. Communities or distinct modules in complex networks are defined as subsets of nodes which are more densely linked, when compared to the rest of the network (Danon *et al.*, 2005). Community detection or community structure identification is useful in many networks, because community structure identification yields useful data about many complex networks such as protein networks, bacteria networks, social networks, WWW network and even terrorist networks. Revealing the community structure in a complex network is important to us for analyzing the complex networks.

Community structure in protein networks or in gene networks will improve biological research by providing which of the proteins or genes are alike according to the criteria that form the complex network. Virus spreading network in computers is also a complex network and community structure information of this network can help stop computer viruses by isolating healthy communities from infected ones. In recent years, terror has become an important issue for every country and uncovering the community structure of terrorist network from some intelligence dataset is helpful to identify and collapse these networks.

Community structure identification is critical for dynamically changing data as well. Data mining needs a kind of clustering information of data for retrieving and organizing huge amount of data. Search engines also try to classify and cluster the search results for ease of use. Vivisimo (VIV) is an example of such a clustering engine, which clusters the search results dynamically.

However community detection is not a straightforward method because of different characteristics of complex networks. Community detection in very large network is a tough problem as well, because of computational limitations and characteristics of very large networks.

In order to define and implement a successful community detection scheme, first we ensure the definition and the characteristics of a community in complex networks. The basic community definition is a *clique*, defined as a subgroup of a graph containing more than two nodes where all the nodes are connected to each other by means of links both directions (Danon *et al.,* 2005). The shortest path between any two nodes inside community is one. This is a strong self-referring definition and rarely fulfilled in real life networks. The *clique* definition is relaxed by *n-clique* where the shortest path between any two nodes is *n*.

A comparative approach defines a community according to its nodes' internal and external links. The nodes in a community should have more internal links than external links to other nodes. This definition is a strong definition and is relaxed in (Radicchi *et al.,* 2004) such that the sum of internal links of all nodes inside the community is more than the sum of the external links of the nodes.

Self-referring definitions are theoretically defining communities well; however they are very costly in detecting communities in complex networks. On the other hand, comparative approaches are better in detecting communities and give a sufficient measure about how good a network is partitioned into communities.

Community detection algorithms proposed so far use different approaches to undercover community structure. The earliest algorithms use link removal method to partition the network into communities, while some latter algorithms do hierarchical agglomerative method to merge individual nodes into meaningful communities. There are algorithms that tries to maximize a global value of network, mostly network modularity, and some algorithms use spectral analysis methods.

All the algorithms are based on the definition of the community. For example, GN Algorithm which is a link removal method is based on the fact that links inside the community is denser and the communities are tied to each other via some inter-community links. Algorithm iteratively tries to remove the right inter-community links to separate the communities from each other. Some agglomerative algorithms and optimization algorithms

are also based on the comparative definition of community. These algorithms take decisions according to network's global property, network modularity (Newman, 2004), which is calculated by number of internal and external links of the community. Information flow, voltage drop and random walk methods and other methods make use of the definition of community to perform a successful partitioning.

The main problem in community detection is that the number of communities is needed in some algorithms. Because they provide a hierarchical structure and the hierarchical structure does not give a community structure if it is cut at some level, which needs a priori knowledge of number of communities. However as the community detection process is to reveal an unknown structure, in most of the cases the actual community structure and the number of communities is unknown.

Another issue in community detection is the computational costs that make algorithms impractical to use in very large networks. The algorithms perform very well in small networks where memory and CPU resources needed are very few, however most of them cannot be scaled up to very large networks because of computational intensity in terms of memory and CPU.

Community detection algorithms proposed so far will be explained in details.

3.1. Girvan-Newman(GN) Algorithm

The Girvan-Newman algorithm which is also called as GN algorithm is based on removal of the edges according to edge betweenness. Betweenness is first proposed by Freeman (Freeman, 1977). *Betweenness centrality* of a vertex *i* is defined as the number of shortest paths between pairs of other vertices which run through *i*. It is a measure of the influence of a node over the flow of information between other nodes, especially in cases where information flow over a network primarily follows the shortest path. However in GN algorithm the edge betweenness is defined (Girvan and Newman, 2002) and used as the betweenness centrality measure to select which edge to remove where the remaining network will begin to split into different communities. *Edge betweenness* of an edge is the

number of shortest paths between pairs of vertices that run along it. If a network has communities that connect to each other via some inter-community edges, then all the shortest paths between nodes in different communities will pass through those inter-community edges that increase the edge betweenness of those inter-community edges.

GN algorithm starts with computation of each individual edge's edge betweenness centrality and then repeatedly removes the edges with highest edge betweenness. After removal of each edge, the edge betweenness of each edge is recalculated according to the new topology of the network. The removal of the edges continues until all the edges are removed in the network.

GN algorithm is a divisive algorithm, which is a link removal method, and finds hierarchical decomposition of network into individual nodes. The algorithm produces a hierarchical structure of network, which is called dendogram. A *dendogram* is a tree-like structure where individual leaves merge level by level and they merge into one big community at the top, details of a dendogram are given in Appendix B.

The algorithm performs very well in detecting community structure in network. It focuses on the boundaries of the communities by using the edge betweenness approach. However the computation time of the algorithm makes it unfeasible for using it in middle size or in large networks. It needs to recalculate edge betweenness of each individual node at each iteration because of the fact that edge betweenness of nodes may change as a result of removal of edges. There may be an improvement by recalculating only the effected nodes' edge betweenness values; however it will not change the worst case running time-complexity, which is $O(n^3)$. The time-complexity of the algorithm makes it impractical to scale it up for very large networks.

Another drawback of the algorithm is that it produces a hierarchical structure, dendogram. One should cut the dendogram at a level to reveal the communities, which is impractical for most of the cases where the actual community structure and number of communities are unknown.

3.2. Fast Algorithm for detecting community structure

Fast algorithm for detecting communities (Clauset *et al.,* 2004) is a hierarchical agglomerative method based on network modularity, *Q*. An agglomerative method in the beginning treats each individual node as a separate community and tries to merge communities one by one to until all the nodes are in one community.

The algorithm uses network modularity as a quality measure whether a partition is a meaningful community. The network modularity is a global property of a network and it is affected by each single community in the network. A partition that is not a real community will decrease the network modularity value of the network. The network modularity definition is explained in details in (Newman and Girvan, 2004) and (Clauset *et al.,* 2004). To give a brief explanation, the *network modularity* is the fraction of edges that fall within communities, minus the expected value of the same quantity if edges fall at random without regard for the community structure. The network modularity calculation can be done with the below formula:

$$Q=\sum_{i}(e_{ii}-a_i^2) \tag{3.1}$$

where i is the number of communities, e_{ii} is the fraction of edges to the total number of edges in the network that has both sides inside the community and a_i is the fraction of total number of edges that has one side inside the community to the total number of edges in the network. If a community has no within-community edges then the network modularity will be Q=0, while putting all the nodes into a single community will give Q=1. A good community structure will exhibit a network modularity between 0.3 and 0.7 (Newman and Girvan, 2004).

As the algorithm is an agglomerative method, it decides on which communities to merge at each iteration. The quality measure is, as mentioned, the network modularity. Starting with n communities, at each step, the join of communities that results in greatest increase (or smallest decrease) in Q, is selected. After merge operation is done, the ΔQ_{ij}

values for each potential join is calculated again and the iteration continues until all the communities are merged into one single community.

The algorithm uses an n x n adjacency matrix where n is the number of communities for storing e_{ij} and a_i values can be calculated from the matrix. In the beginning of the algorithm, all the elements of the matrix, which are e_{ij} values, are calculated once and then they are updated by adding rows and columns of corresponding communities who take place in join operation at each iteration. ΔQ_{ij} values are calculated at each iteration by using the n x n matrix.

The algorithm has $O(n^2)$ time-complexity which is better than GN algorithm. It produces very accurate results as well. However because of the n x n adjacency matrix data representation it is unfeasible for very large networks due to memory limitations. As in the beginning of the algorithm the number of communities is the number of nodes, the n x n matrix cannot be allocated for very large networks.

After the proposal of this algorithm; Newman, Clauset and Moore (Clauset *et al.,* 2004) proposed an improved version of this algorithm that is suitable for very large networks. The new algorithm overcomes the matrix representation's drawback by suitable data structures that are scalable. In the new algorithm, rather than maintaining the adjacency matrix and calculating ΔQ_{ij}, the ΔQ_{ij} are stored in new data structure. Since joining two communities with no edge between them can never produce an increase in Q, only the ΔQ_{ij} values need to be stored and matrix representation is waste of a lot of space in this situation. In the new algorithm, communities are stored in a binary tree and their adjacent communities are stored in individual max-heaps that are pointed by each community in the binary tree. There is also a global max-heap data structure which has the highest ΔQ_{ij} value in each iteration. The new data representation is promising in terms of scalability especially in terms of memory.

The algorithm again produces a dendogram. This algorithm however provides a method to cut the dendogram at some point. According to the algorithm, when the highest ΔQ_{ij} value starts to have negative values, it means that the further joins will not improve the community structure, so the merge operation should be cut. This kind of method is a good way, at least to provide a way to decide on where to cut the dendogram.

During each iteration, two communities are merged and only the ΔQ_{ij} values that are effected in the result of join are updated. Also the pointers to the merged communities are also updated in each iteration. By the efficiency of data structures the new algorithm has O(*nlogn*), which is superior to previous algorithm and GN. However the data structure used in the algorithm is still degrading the performance of the algorithm for very large networks. Also the problem of cutting the dendogram at a point is still a question here. Although the new algorithm provides a method to cut the dendogram, it is not ensured as the best cut when we do not know the real community structure.

3.3. Radicchi's algorithm

Radicchi and his friends (Radicchi *et al.*, 2004) investigated the definition of community and improved the definition in their work. The definition of a community in many contexts is as a partition where nodes are densely connected inside the community; however a quantitative definition is needed in order to ensure that a given partition is a good community. In their work, there are two definitions of community in terms of formulas.

A node *i* has a degree k_i, which contains both incoming links and outgoing links. In a community defined in strong sense contains nodes each of which has more connections within the community than with the rest of the graph. If we consider *V* as a community in a network *G*, the *V* is a *strong community* if the below condition holds:

$$k_i^{in}(V) > k_i^{out}(V) \quad , \quad \forall i \in V . \tag{3.2}$$

where $k_i^{in}(V)$ is the number of intra-community links that node *i* has in community *V* and $k_i^{out}(V)$ is the number of inter-community links that the node *i* has.

This definition in a *weak* sense can be expressed as if the sum of internal links is greater than the sum of outer links, then the community is a *weak community*, as explained below:

$$\sum_{i \in V} k_i^{in}(V) > \sum_{i \in V} k_i^{out}(V) \tag{3.3}$$

A community in strong sense is also a community in weak sense, whereas the converse is not true.

This definition is an improvement in the quality of all the partitions in the network. If a network is split into two parts randomly, one is very large and other is small, the large part almost fulfills the definition of community (Radicchi *et al.*, 2004). However the quality of the split is bad since it is a random partitioning. To deal with such problems, we should deal with quality of each individual split, instead of focusing only the overall picture.

Radicchi's algorithm has the similar methodology with GN algorithm. It tries to split the whole network to meaningful partitions by removing an edge in each iteration. Since GN algorithm is costly algorithm, Radicchi proposed a computationally better approach. The edge-clustering coefficient is defined in (Radicchi *et al*., 2004) as the number of triangles to which a given node belongs, divided by the number of possible triangles that might include it. The formal definition of *edge-clustering coefficient* is as follows:

$$C_{ij}^{(3)} = \frac{z_{ij}^{(3)} + 1}{\min[(k_i - 1),(k_j - 1)]} \tag{3.4}$$

where $z_{ij}^{(3)}$ is the number of triangles the edges take part and the divisor is the possible number of triangles that the edge can be included. The number of triangles is incremented by 1 to avoid the edge-clustering coefficient to be zero where edge has no triangles.

This quantity is useful in that the edges connecting different communities are included in a few or no triangles and will have small $C_{ij}^{(3)}$ values. However the edges inside a community will probably take part in many triangles.

Radicchi also proposed the algorithm for the weak definition of the community. The algorithm in weak sense uses the squares instead of triangles for calculating edge-clustering coefficients.

The algorithm starts dividing the network into partitions by removing the edge with the minimum edge-clustering coefficient at each iteration. Since the number of triangles is a local property, only the effected nodes need to be updated after removal of an edge. By this locality, the computation time is better than GN, although the methodology is same as GN.

The algorithm has $O(n^2)$ time-complexity and provides a good quantitative measure for the quality of community in a strong sense. However the triangle method may not perform well on different kind of networks where the connection density differs as the result of network topology.

3.4. Wu-Huberman's Algorithm

Wu and Huberman proposed an algorithm that behaves a network as an electric circuit and makes use of the voltage drops and Kirchoff's laws to uncover the community structure (Wu and Huberman, 2004).

In an earlier work (Newman and Girvan, 2004) the current flow technique was proposed and used first. Newman and Girvan defined current-flow betweenness for edges, which is the absolute value of the current along the edge summed over all source/sink pairs. It turned out to be the same as random walk betweenness but it leads to simpler derivation of the measure. However the computation time of this algorithm is very costly because of inverse matrix calculations; the overall cost is $O(n^4)$. The computation time-complexity of the algorithm makes it impossible to use it for large networks.

Wu and Huberman proposes a different method that discovers the community structure in $O(V+E)$, where V is the number of vertices (nodes) and E is the number of edges in the network. The algorithm needs a priori knowledge of two nodes that are in different communities for each splitting. The authors propose some methods to pick up two nodes, but it does not guarantee the perfect selection all the time.

Algorithm supposes that we already know that node A and B belong to different communities, which we call G_1 and G_2. We imagine each edge in the network as a resistor with the same resistance and we connect a battery between A and B so that they have fixed voltages, say 1 and 0. With these assumptions, the network has become an electric circuit with current flowing through each edge. By solving Kirchoff's equations, we can obtain the voltage value of each node, which should lie between 1 and 0. Then we set a threshold to classify nodes into G_1 and G_2 communities, say our threshold is 0.5, if a node has a voltage value over the threshold it belongs to G_1 and if it has a voltage below the threshold it belongs to G_2, or vice versa.

The algorithm performs well for the nodes that are between different communities. The classification of such nodes is a hard work because they have connections with nodes that belong to different communities. However in this algorithm, a node has a voltage value which is the average of its neighbours' voltages. As a result, if most of the neighbours have a voltage above threshold, then the voltage of the node is also above threshold and it places in the same community as its neighbours.

The algorithm proposes a physics approach and is successful if some priori knowledge is supplied, which is not possible in real-life networks. First, we need to know two nodes that lie in different communities to make a split according to those nodes. If we start with two nodes that are in the same community, the algorithm performs in a wrong way. The authors propose a solution to this problem by selecting two nodes that are very far from each other. However this operation needs the calculation of diameter of the network and doesn't yield a result in a dense network. The second information needed is the threshold value to make a good split. However there is no straightforward method to find a good threshold.

Wu and Huberman's algorithm seems very fast in terms of O(n) time complexity. However it needs a priori knowledge to perform well on partitioning a network into communities. Another drawback is the need for matrix calculations in Kirchoff's equations. The matrix representation always makes an algorithm impractical for the use in very large networks.

3.5. Reichardt-Bornholdt Algorithm

The algorithm proposed in (Reichardt and Bornholdt, 2004) is based on the definition of community structure; the nodes are densely linked inside and loosely connected to other nodes outside the community. Some complicated formulas are used to split the partitions and evaluate the quality of the communities.

The algorithm uses q-state Potts Hamiltonian and it forces the spins onto communities according to formula, which counter-balances the homogeneity of the spins and the quality of partitions. The algorithm is based on a physics approach, q-Potts Hamiltonian, and it implements the idea for placing the nodes into communities. The algorithm starts with q spins and each spin is mapped to a community. In the beginning, the system is given an initial temperature and the nodes start to move among the communities until the system cools down. The system is subsequently cooled down and ground state is reached in the end. This process is repeated for some number of times, and

we can evaluate the successful community structure by looking at the local maximas of the Hamiltonian.

The co-appearances of nodes in one community are represented in N x N matrix, where N is the number of communities. Well defined communities appear along the diagonal of the matrix.

The algorithm is a good example of optimization algorithms in community detection. It finds communities without the priori knowledge of number of communities or any further parameter. It is a scalable algorithm as well; however as all of the optimization algorithms, it is indeterminist. It is impractical for very large networks because of the N x N matrix data structure.

3.6. Community detection using Extremal Optimization

The network modularity (Newman and Girvan, 2004) is used by many community detection algorithms as the basic quality assessment criteria of partitions. The community detection algorithm using Extremal Optimization (Duch and Arenas, 2005) also uses the network modularity to evaluate the partitions whether they are of good community structure.

The algorithm tries to optimize the network modularity, Q, as the global variable by maximizing each individual node's contribution in Q. The contribution of each node in maximizing the global variable Q is defined as:

$$q_i = K_{r(i)} - k_i a_{r(i)} \tag{3.5}$$

where $K_{r(i)}$ is the number of links that a node i has that is inside community r, k_i is the degree of the node i and $a_{r(i)}$ is the fraction of links that have one or both vertices inside the community.

The algorithm is a recursive divisive algorithm and starts with one community, which is the whole network. In the beginning the algorithm splits the network into two random communities having the same number of nodes each. At each time step, the system self-organizes by moving the lower fitness from one partition to the other. After each movement the fitness of the nodes affected should be calculated. The movement between splits stops when a maximum value of Q is reached. At this point the links between are removed and the algorithm proceeds recursively with every resultant connected component. The process finishes when the modularity Q cannot be improved.

The algorithm is successful in using the artificial intelligence method and is deterministic. On the other hand, authors tried to avoid local maxima and selects node with lower fitness according to some probability distribution, which makes the algorithm indeterminist. Another issue of the algorithm is the decision of "where to stop". When network modularity Q cannot be improved at least for some number of iterations, the algorithm stops. Authors (Duch and Arenas, 2005) propose that a reasonable number of iterations are the number of nodes in network.

The algorithm has $O(n^2 ln(n))$ time-complexity after making performance improvements. It is a successful algorithm in many networks, however, is not scalable for very large networks because the recursive scheme and its time-complexity.

4. COMMUNITY DETECTION USING GENETIC ALGORITM

4.1. Use of Genetic Algorithm in community detection

As has been suggested in the previous chapter, community detection analysis is proposed to be done in various ways. For the task of finding accurate community structure, we will present genetic algorithm optimization model. The model focuses on maximizing the network's global variable, network modularity Q, which is the overall rating of quality of partitioning in terms of good community divisions. The model is meant to reveal the community structure of a complex network that reflects the actual community division of the network, thus resulting in an effective way of community detection in complex networks. We essentially build our algorithm on the network modularity (Newman and Girvan, 2004) optimization, using the genetic algorithm as optimization algorithm.

Genetic algorithm provides optimal solutions to problems where the solution space is very large and it is computationally very expensive to do exhaustive search on the solution space. For the community detection task, we believe in the power of genetic algorithm in that it is converging to the solution very rapidly. The cross-over process in genetic algorithm creates better solution members in each iteration, which guides the search into a narrower search space.

Genetic algorithm is also superior for the fact that it has O(n) time-complexity, which provides the scalability of it for very large complex networks. The operations and the data structures used in the algorithm makes it practical for use in very large networks.

Unlike many other algorithms, GA does not need any priori knowledge about the number of communities or any kind of threshold values in community detection process. It dynamically decides on the number of communities, by looking at which community partitioning approaches the maximum network modularity value. If a community partitioning is not a good one, it will automatically have a bad fitness value and will be disregarded by GA.

In our work, we find an efficient and easy way of problem representation scheme for community detection by genetic algorithm. The fitness function is also suitable for assessing the quality of each candidate solution.

In this chapter we propose GACD (Genetic Algorithm in Community Detection), a new community detection algorithm based on genetic algorithm, which we build on the concept of maximizing network modularity (Q). This maximizing of the network modularity is firstly presented in (Newman and Girvan, 2004), and the idea is to merge two communities with the highest gain in network modularity. There have been artificial intelligence methods to optimize the network modularity as a global variable, like extremal optimization, simulated annealing etc., which are explained in Chapter 3.

4.2. GACD Algorithm: Genetic Algorithm in Community Detection

In this part we will introduce our new algorithm in finding the community structure in complex networks. In algorithm implementation, we made some modifications in genetic algorithm processes and add some techniques to improve the quality of algorithm results.

Our algorithm is an indeterminist approach, where each run can produce different solutions. As genetic algorithm is an optimization process, our algorithm finds the optimal solution. From the experiments on some real networks, our algorithm found 100% correct solutions as well. We will be giving the results of our algorithm in Chapter 5 in details.

In the subsections below, we will introduce the details of algorithm. We also give brief information about the implementation of the algorithm.

4.2.1 Fundamentals of GACD

GACD algorithm is based on an optimization process of network modularity for a predefined number of iterations. At the end of the iteration process, the algorithm promises to find the optimum community partition. The nodes in the network are assigned a unique community identifier, named as *commID*, and the *commID'*s of nodes change through the optimization according to the re-placement of nodes into different communities.

The algorithm does not need any priori input other than the network data like the number of communities or any threshold value. The network data needs to be in Pajek (PAJ) format where the nodes (vertices) are defined first and then the edges between the nodes are given. Algorithm treats the connections between the nodes as undirected edges, not as arcs even if they are provided as arcs.

GACD will, in the result, assign each node a unique commID and does not consider the overlapping communities which are discussed in some recent documents (Palla *et al.,* 2005) and (Pissard and Assadi, 2005). As it gives the resultant community identifiers (*commIDs*) of each node in the result, the user does not need to deal with any further process to get the community structure like cutting a dendogram at some point.

GACD is a strong algorithm in finding good community partitions without stucking into big giant connected components. Most of the algorithms, because of their hierarchical methods, are cumulating the nodes around a giant connected component, even if those nodes form standalone subgroups apart from the giant component.

4.2.2 Data representation

The first step in genetic algorithm is to define a representation scheme of the problem and its solution set. In GACD, we store the network data in an array of structures where the node ID, neighbour count and the identifiers of the neighbour nodes are stored.

This information is read once from Pajek formatted input dataset and will kept unchanged for the rest of the algorithm.

A data structure, called chromosome, is defined as a member in solution set. A chromosome contains an integer array, which has the length of number of nodes in the network. The integer array in the chromosome contains the community identifier of the corresponding node. The chromosome also has a floating point variable "fitness value", which stores the result of fitness function of that particular chromosome. See Figure 4.1 for the details of the chromosome data structure for a network of *n* nodes. Here the genome[] array contains the community identifier for each node and the array has size of *n*, which is the number of nodes in the network.

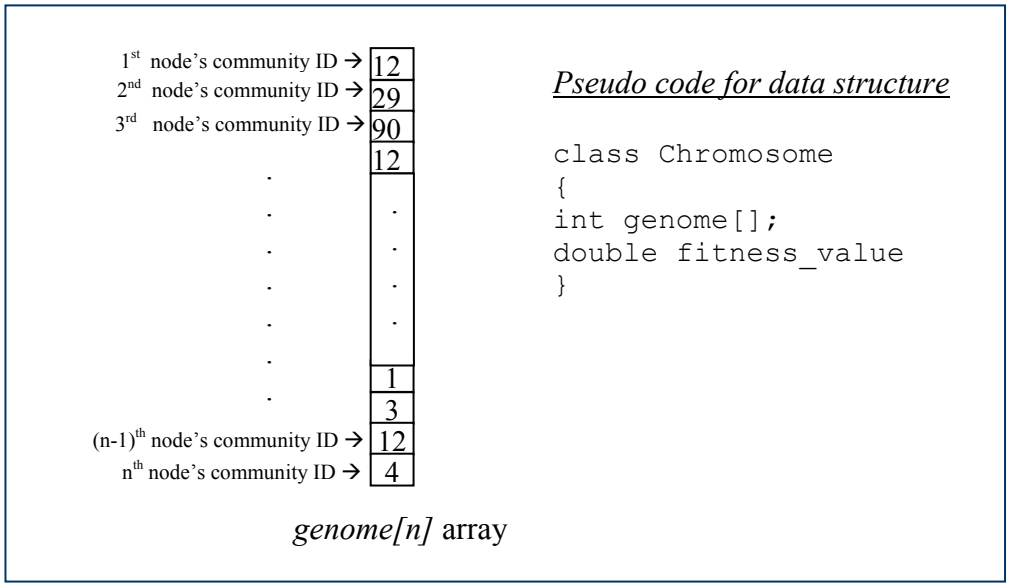

Figure 4.1. Chromosome data structure in GACD

There should be a number of members in the population so that we can perform reproduction for next generations. The population members, each of them is a chromosome, is stored as an array of chromosomes. The size of the population is predefined and we decided on the population size due to memory limitations. On the other hand population size should not be so small, because genetic algorithm needs various members and their genes to create as much variation and diversity as it can do for creating better members for next generations.

4.2.3 Initial population creation

The chromosomes, which are the members of population are created and initialized once in the beginning of the algorithm. For the purpose of diversity, all the community IDs of every member in the population is initialized to a random number. The random number is limited with the number of nodes in the population, namely *n*. Theoretically, in the worst case, every node will fall into separate community and there will be *n* communities in the network.

After random initialization, we saw that algorithm needs a kind of bias during the initial population creation, because it takes a lot of time to improve the random community partitions into meaningful communities. The initial population creation is a crucial phase in genetic algorithm, because by a careful creation of the initial population, the algorithm can perform very rapidly and much of the computation time can be saved.

In our algorithm, we give perform a bias for community IDs during the initial population creation. Our bias is based on the fact that, if two nodes are supposed to be in the same community, then they must be connected directly or via some hopes. To implement this idea, we randomly select nodes and assign its community ID to all of its neighbours. This bias has a potential to create one giant connected component, but the selection of the nodes in the random order decreases this probability and creates very successful initial populations.

4.2.4 Fitness evaluation function

GACD algorithm aims to maximize the network modularity value by selecting the best community partition in the network. For this purpose, we use the network modularity as fitness value of each chromosome in the algorithm. Network modularity, as defined in the previous chapters, is calculated using the below formula:

$$Q=\sum_{i} (e_{ii} - a_i^2) \tag{4.1}$$

where i is the number of communities, e_{ii} is the fraction of edges to the total number of edges in the network that has both sides inside the community and a_i is the fraction of total number of edges that has at least one side inside the community to the total number of edges in the network.

The e_{ii} values are evaluated by summing up the internal links of communities and then dividing it by the total number of edges in the network. For this purpose, we have an array, namely $e[]$, that stores the e_{ii} values. We iteratively parse the nodes and their neighbours. If a node whose community ID *C1* has a neighbour that has also community ID *C1*, then we increment $e[C1]$ array element by one. At the end of the iteration, we divide all the $e[]$ element by the total number of edges, in order to get the fractions.

After finding the internal edge counts (fractions), we need a_i values, which are the total number of edges that has at least one edge inside the community. This is sum internal links which are already calculated, plus the nodes that has connections with other nodes outside the community.

During the iteration to calculate e_{ii} values, we also get the number of communities in the network. For the number of community counts, we iteratively apply the network modularity summation and in the result we obtain the network modularity value as the fitness value of the given community partition configuration of a chromosome member.

Fitness evaluation is a straightforward phase in the algorithm and it is solely based on the network modularity. One can improve the power of fitness function by adding some more quality measure to the fitness function.

The chromosomes in the population are then sorted in decreasing order according to their fitness values. After sorting some predefined numbers of chromosomes that are on the top of list is kept by replication, by the purpose of preserving of better genes, which will be described in details in section 4.2.6. After this operation, the chromosomes are ready for genetic operations for reproducing new members for the next generation.

4.2.5 Genetic operations

There are two basic genetic operations in GACD. The cross-over is the most powerful element of the algorithm in creating better solutions from genetically good parents. Then some randomly selected members are mutated to create diversity in the population.

i. Cross-over

In GACD, cross-over is modified completely for the needs of the algorithm. The cross-over in the definition in genetic algorithm is performed by exchanging of genes between chromosomes at a selection point. Because of the data representation scheme and the relational structure of the problem domain, the ordinary cross-over does not yield good results in creating better members in result. We will explain the causes in details.

First of all we assign a community ID to each node, and nodes that have the same community ID are thought to be in the same community. Suppose that for two of our distinct solutions, i.e. chromosome *C1* and chromosome *C2*, two nodes are in the same community. Say node 1 and 2 has community ID 76 in *C1*, which means they belong to the same community. And say node 1 and 2 has community ID 5 in *C2*, which again means they belong to the same community. However since 76 is different than 5, we cannot say that community 76 for *C1* is identical to community 5 for *C2* by just looking at their values, even if they contain the same nodes as community members.

For this reason, the simple cross-over operation will divide the communities into smaller splits, because the community identifiers used to name the same community in different solution members can be different. To overcome this problem, we perform the cross-over by transferring the communities onto the destination chromosome, i.e., we select a number of communities to transfer, then transfer the nodes belonging to those communities to the destination chromosome instead of splitting the chromosomes at some point and exchange the parts. In order to overcome the problem of dividing communities into smaller communities, the cross-over operation is performed one way; i.e., the

destination chromosome does not transfer any communities to the source chromosome. We chose to use one way cross-over, because during the transfer of communities to destination chromosome, it is guaranteed that the transferred communities will live in the destination chromosome. And also the source chromosome preserves its community structure and continues to survive. It is because of relational structure of nodes with each other in terms of forming good communities; no individual node can behave freely because they form the communities together. On the other hand, if cross-over is two-way then the community IDs of the nodes which are affected during the transfer of community from source to destination, should be passed to source. In this situation, a totally random collection of community IDs will be passed to the source chromosome, because the selection is not performed at a certain point but by selecting some number of communities from the source chromosome.

See Figure 4.2 for potential cross-over problems in GACD. These two solutions *C1* and *C2* are selected according to their fitness value. Here the community ID 1 is selected in *C1* for cross-over. The community IDs of the nodes in *C1* are transferred to the nodes in *C2*. After cross-over the community ID 1 in *C2* has more members than community ID 1 in *C1*, because there was already a node that has community ID 1 in *C2*.

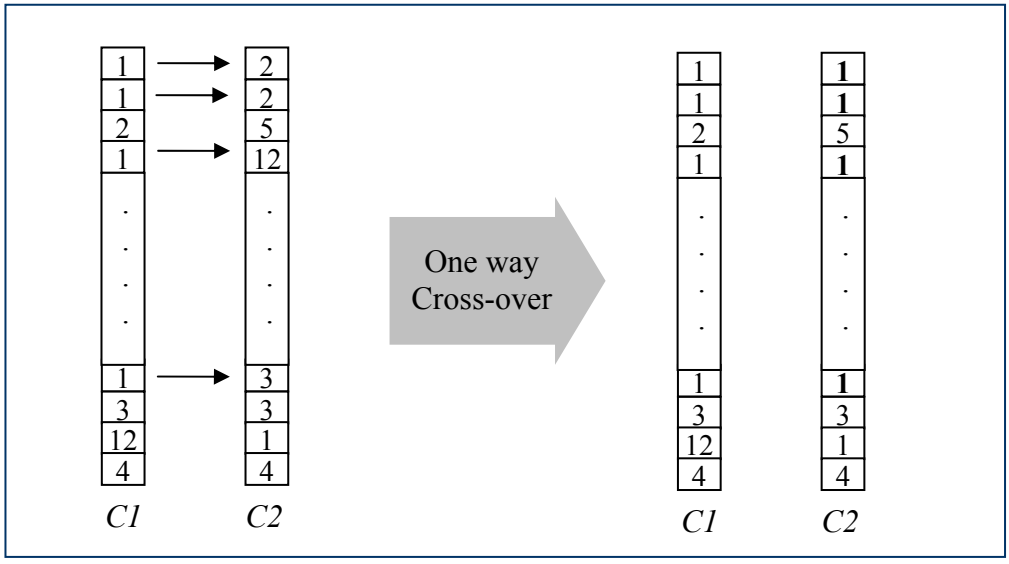

Figure 4.2. One way cross-over in GACD

By one way cross-over, the cross-overs will always produce larger communities, or at worse preserve their members, and this situation improves the quality of solutions. Another advantage of the one-way cross-over is that, we are sure that at least one of the members in the cross-over is preserved from the defects of this operation, because we do not change the source member. Cross-over may or may not produce better genes, but since it occurs on the destination chromosome, its negative effect is limited.

As we have a relational structure in community detection, we can not treat each single node as independent solution element. Community structure is relational feature of network, where each individual node has a role in it. The quality of the solution will improve by considering the relative positioning of nodes.

We also practiced the two-way cross-over and the classical cross-over. However both of them produce random distribution of nodes into communities, which results worse community partitions in the network. We will give the experimental results of both methods in our experimental work section. Two-way cross-over has the potential problem that it can worsen the quality of both of the chromosomes that take place in cross-over operation. When transferring community IDs from source chromosome to destination, the community IDs of the destination should also be transferred to source. However the nodes that are known to be in the same community in source chromosome may be in different communities in the destination, and this condition will split successfully built communities into different partitions. See Figure 4.3 for details of two-way cross-over process. Here we transfer the community IDs of the nodes selected in source chromosome, and the corresponding community IDs in the destination chromosome are then transferred from destination chromosome to the source chromosome.

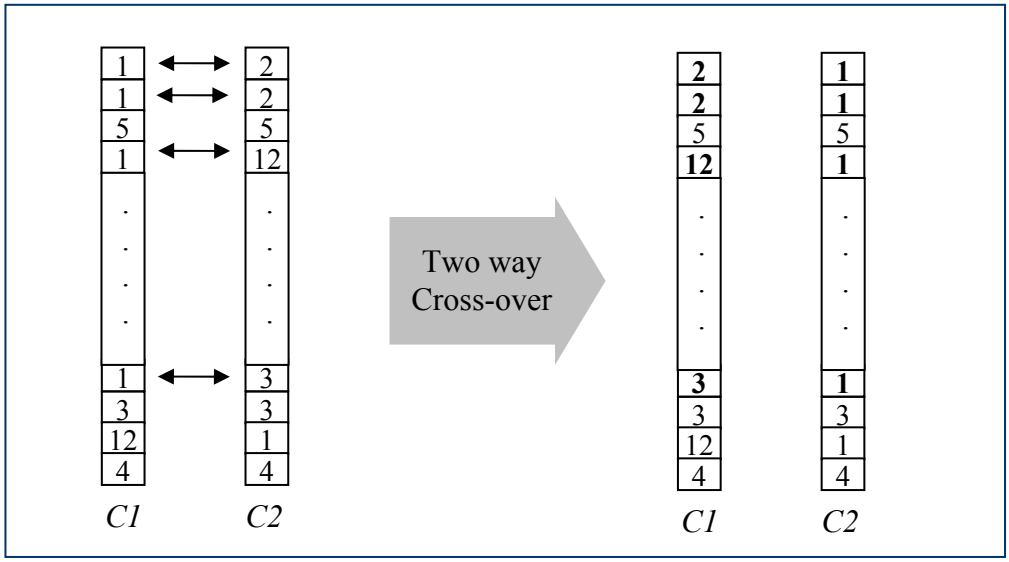

Figure 4.3. Two way cross-over in GACD

Note that, in the example in Figure 4.3, because of the two-way cross-over, some of the nodes in the source chromosome split into different communities although they were supposed to be in the same community in the beginning. There are 3 members in *C1* with community ID 1, and after the cross-over, *C1* there are 2 members in *C1* with community ID 2 as the community ID changed from 1 to 2 by the result of cross-over. One of the members of the community is taken apart from the community by assigning it community ID 12, instead of 2.

One may argue that the cross-over operation is supposed to create diversity and this kind of changes should occur to create diversity. However if we were performing cross-over by selecting a point and exchanging genes after that point, then only the genes around the nodes will be affected very much. Because of the relational nature of cross-over in GACD algorithm, each transfer of community to other gene has the potential of creating random network, if the destination chromosome transfers back the the community IDs of nodes that are changed by cross-over.

ii. Mutation

In GACD, we use the conventional mutation operation in the chromosomes. We select a predefined number of chromosomes and apply mutation operation on these chromosomes.

Mutation is done on the community IDs that are stored in the integer array in each chromosome. During mutation, a node is selected randomly and its community ID is converted to its binary representation. Then a randomly selected bit of binary representation of the number is altered such that if the bit is 0 it is converted to 1 and if it is 1 then converted to 0. See Figure 4.4 for the details of mutation operation.

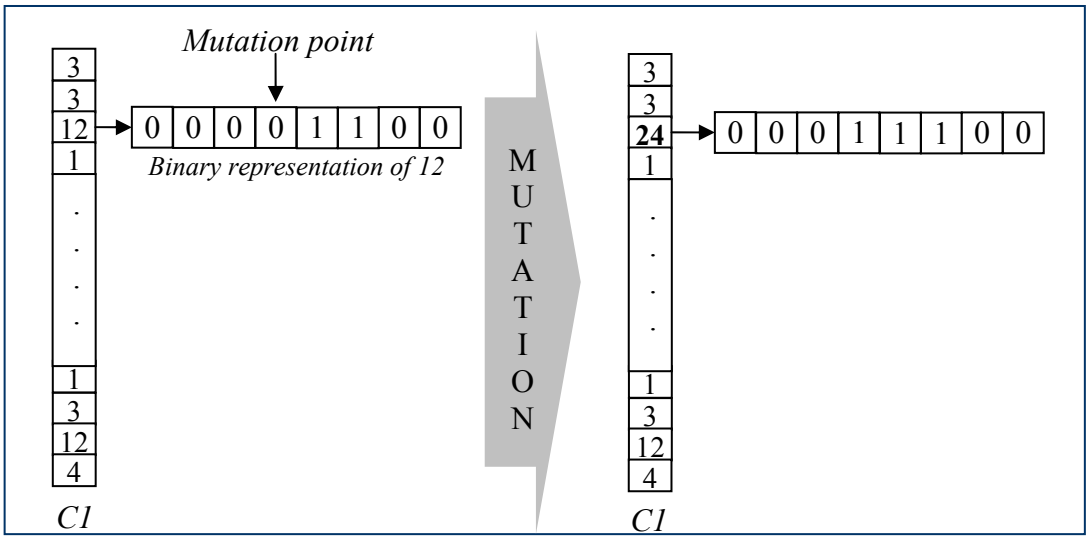

Figure 4.4. Mutation in GACD

Here we select the 3rd node randomly and then convert the community ID of the node into the binary representation. Then we select a random bit in the binary representation and then convert to its inverse by a simple XOR operation. In the result of the mutation, the node will have a community ID that did not exist in the chromosome before.

During the batch run of our algorithm on different network data, we experienced that the mutation is a powerful operator to create diversity in the population. Mutation decreases the probabilities of local maximas, where the algorithm get stucks by assuming

that it is around the solution. The experiment results will be given in the experiment section.

4.2.6 Preserving of the better genomes

In genetic algorithm, as the algorithm runs for a number of iterations, some solution members or chromosomes may have very good fitness functions. However during the cross-over or mutation process, these chromosomes can change in a way that they can have a poorer fitness value in the result. There is no guarantee in genetic algorithm that the genetic operations always improve the overall quality.

Another issue is that, chromosomes with high fitness values can occur in different iterations and we will not be able to perform cross-over between the best chromosomes that we had in different iterations. For example, we have *C1* in the 10^{th} iteration which is the best until now. Then we have *C2* in the 24^{th} iteration which is as good as *C1*. However if since we go for 14 iterations since 10^{th} iteration, we have lost the *C1* chromosome and we lost the chance to perform cross-over between *C1* and *C2*.

To overcome this problem, preserving of the better genomes method is used in genetic algorithm. The idea is to keep the best chromosomes that come out from each iteration through the whole genetic algorithm process. For this purpose, we reserved a number of places at the top of the population, and keep those chromosomes live unless some better chromosomes are reproduced in iterations.

We check all the chromosomes in each iteration, whether they are better than our preserved ones. If they are better, then they become the members of preserved genes.

Preserving of the better genes method provide us a memory in the algorithm. It turns the evolution process into a cumulative approach, where any contribution during the iterations is not wasted.

4.2.7 Clean-up process

Apart from the genetic algorithm, in GACD, we define and implement a clean-up process that checks and reorganizes the community structure of some randomly selected nodes at predefined intervals. The idea of the process is that a node and its neighbours should belong to the same community with a predefined probability.

There may be some nodes as a by-product of genetic algorithm, where they and their neighbours all belong to different communities. We think that the variance of community IDs among the neighbours of a node should be lower than the predefined threshold. We define a value called "Community ID Variance" and formulate this value as below:

Definition 4.1 (Community ID Variance) *Given a node n with k neighbours in a network, we define the Community ID Variance of the node as:*

$$CV(n) = \frac{\sum_{k} f_k}{k} \quad where \quad f_j = \begin{cases} 1 \; if \; \forall j \; neighbour(n)=j \; and \; commID(n) \neq commID(j) \\ 0 \; otherwise \end{cases} \tag{4.2}$$

Proposition 4.1

Given a node with k neighbours, it Community ID Variance, CV(n) should be lower than a threshold value to provide a good community structure.

We proved our proposition by the definition of community. A community should contain more internal links than external links with outside communities. For this reason, a node's neighbours should belong to some limited number of communities and if it is inside a community, the neighbours should mostly be inside the same community with it. Otherwise, the community partition does not reflect the real community structure of the network.

In some cases a node can place among many different communities, then the node is supposed to be a “hub” node. In this situation it is hard to tell the community ID the node belongs to.

We propose the clean-up process to improve the quality of community partitions and to provide a mechanism to identify the community IDs of so-called “hub” nodes in the network. Our clean-up process also provides a mechanism for replacing totally misplaced nodes, where it resides in a community where it does not have a community.

Clean-up process iteratively searches through the nodes and check its community ID variance value against a predefined threshold. If it is lower than the threshold, then the node is connected to the highest connected component in the its neighbourhood, i.e., the community ID with the highest number of members among neighbours will be the community ID for the node.

In order to find the community ID with the highest number of members among neighbours, we used a data structure that keeps track of each community ID and its occurrence in the neighbours of the node. After we looked at all the neighbours of the node, we assign the community ID with highest number of occurrence as the new community ID of all the neighbours of our node and the node itself.

We tried the performance of the clean-up process through some batches and experienced about 2% improvement in the accuracy of our GACD algorithm. We will be providing the details of batch runs in the experiment section.

4.2.8 Parameters in the GACD algorithm

Genetic algorithm is powerful when its parameters are tuned carefully. We implemented our GACD in a parametric way so that we will be able to tune it for best performance. For this purpose, we define some number of parameter variables, most of which are directly related to genetic algorithm. There are also some parameters that are specific to our GACD algorithm.

In this section we will give the names and explanations of all the parameters in the GACD algorithm. We also introduce the effects of these parameters in details here. We will give the result of changes of these parameters in the experiments chapter.

As a general overview, all the parameters in GACD algorithm are listed in **Table 4.1**. Here the parameter names, parameter data types and the possible values that a parameter can take are presented.

Table 4.1. Parameters in GACD Algorithm

parameter name	type	value range
Population Size	Integer	between 50-200
Iteration Count	Integer	between 100-500
Randomization rate during initial population creation	Floating point	between 0.0-1.0
Cross-over Rate	Floating point	between 0.0-1.0
Cross-over Count	Floating point	between 0.0-1.0
Clean-up	Floating point	0 or 1
Community ID Variance Threshold	Floating point	between 0.0-1.0
Clean-up Cycle	Floating point	between 0.0-1.0
Clean-up rate	Floating point	between 0.0-1.0

The parameters details are given below:

i.Population Size

This parameter specifies the number of chromosomes in the population, on of which is a candidate for solution of the problem. This number should be large enough to create diversity in the genetic algorithm and in GACD. We found that the ideal population size should be between 50-200. We generally used 100 as our population size. Increasing the population size needs more memory resources, as the number of chromosomes kept memory will increase with the increase of number of members in the population.

Besides the memory consumption, increasing the population size improves the quality of the GACD in that if we have more candidates in the populations, we will have more number of choices for solution. This also helps us to increase the diversity in the population. Because more members in the community means, more number of genes in the population. As the number of genes increase in the population, the probability of having a rich variety of genes in the population also increases.

However the population size should be kept limited at a precise value. From some point on, increasing the population size does not yield much in improvement of the quality of population, and also introduces memory and CPU overhead.

In our work, we kept the community count 100 for most of the runs and increased it sometimes to 200. We figured out the population size as a critical parameter in GACD.

ii.Iteration Count

Iteration count is the number of iterations in genetic algorithm. Our GACD algorithm runs iteratively during the interval that is defined by this parameter. Iteration count is the most important parameter in GACD, because it is directly related with the improvement of populations. If we do not continue genetic algorithm operations for a required number of times, even if our algorithm is very good, the candidate solution could not converge to the desired solution.

For this reason, iteration count should be large enough to optimize the solution, or sometimes to find the exact result. Having a very large iteration count only increases the algorithm's running time. So assigning a larger number is always preferable to giving a small number to iteration count parameter.

In our work, we tried the iteration counts between 200 and 500. After some number of batch runs, we decided that a number between these values (200-500) yields good results and increasing this value does not put on the performance of the algorithm.

iii. Randomization rate during initial population creation

During initial population creation, we need to give a bias to initialization such that nodes transfer their community IDs to their neighbours iteratively. This bias, on the other hand, may cause to create giant connected components according to structure of the network. To overcome this problem, after the initialization we select a number of nodes randomly so that they re-transfers their community IDs to their neighbours. As a result of this operation, we get rid of the possibility of the giant connected component in the initialization phase.

Another advantage of this operation is that it creates different chromosomes in the initialization phase, which creates diversity and variance in the network.

iv.Cross-over Rate

Another critical parameter, may be the second most important one, is the cross-over rate in GACD. As cross-over is the most critical function of genetic algorithm in terms of creating better chromosomes, we should define a precise number of cross-overs.

This parameter is specified as a rate of the population. The floating point value of this parameter is multiplied by the population size, and the result is taken as the number of chromosomes or members in the population that take part in cross-over process.

According to our experiments, having a small number of cross-overs slows down the convergence of algorithm to the solution. If we give large rate of cross-overs, we increase the probability of creating more number of chromosomes that are better than their parents.

On the other hand, having a high number of cross-over, i.e., more than the half of population, then it turns out that cross-over creates poorer members for next generations. This because of the fact that, cross-over operations are performed between members that are selected according to their fitness values. The first cross-over takes place between the first and second member at the top of the population. So, if the rate of cross-overs increase,

i.e., more than half of the population, then the chromosomes that has very low fitness values have a chance for cross-over, and their children become even poorer in terms of fitness value.

We experienced that, a precise rate for cross-over is between 0.3 and 0.4 in GACD. We tested these results and see that it produced successful results.

v.Cross-over Count

This parameter is specific for only GACD and not occurs in genetic algorithm. As cross-over scheme is different in GACD than the one in conventional genetic algorithm, we should define a number of cross-overs instead of the cross-over selection point.

This parameter defines the number of cross-overs in GACD. It is multiplied by the number of nodes in the network and the result is the number of random nodes in the source chromosome, which are to transfer their community IDs to the destination chromosome.

For a number of iteration that is defined by this parameter, we randomly select a node, look at its community ID and then iteratively transfer the community IDs of all the nodes that belong to this particular community ID.

In our work, we found that a value between 0.3 and 0.4 is appropriate for this parameter.

vi.Clean-up

We defined the clean-up process in details in above sections. This parameter specifies whether the algorithm will do clean-up process or not. The parameter value can be 0, which means “do not perform clean-up”; and 1, which means “perform clean-up process with the specified parameters”.

The clean-up process is made as an optional feature in GACD, to see whether it increase the performance of algorithm. We actually see that it increases the quality of community partitions and set this parameter to 1 as default value.

vii.Community ID Variance Threshold

Community ID Variance is a variable that is defined in our work, in previous section. In clean-up process, the Community ID Variance of nodes are calculated and they are compared with the Community ID Variance Threshold, which specifies whether the nodes will need further operation or not.

If the Community ID Variance of a node is below the threshold, clean-up process does not do anything for that node. However if it is above the threshold, the most favorite community ID is found among the neighbours and it is assigned to the node as the new community ID.

In our work, we found that this threshold should be between 0.1 and 0.2.

viii.Clean-up Cycle

This parameter is a floating point parameter and it specifies at which points of the iteration the clean-up process should take place. This parameter is multiplied by the iteration count parameter and the result is taken as the iteration interval, where we perform clean-up process at the beginning of each interval.

Clean-up process in each iteration is a costly process and it will also decrease diversity if it occurs in each iteration.

We need a kind of correction process, after a number of iteration. The purpose of clean-up process is to replace some misplaced nodes into more correct communities so that a kind of correction process is held.

Doing clean-up during each iteration increases the computation time, because it randomly selects some nodes and performs some iterative functions on that node. So performing this operation at the end of some interval improves the overall running time, while it increases the quality of the solution.

ix.Clean-up Rate

This parameter defines the number of nodes that takes part in clean-up process. This parameter is a rate parameter and is multiplied by the number of nodes in the network. The result is the number of nodes that we perform clean-up process on.

Performing clean-up process on all the nodes will create giant components in the network, which will decrease the community partition in the network. However performing the process on some number of randomly selected nodes increase the quality of community partitions by eliminating the wrongly misplaced nodes in the communities.

In our tests, we found that a rate between 0.1 and 0.2 creates good results in clean-up process.

4.2.9 The performance of the algorithm

i.Time complexity

The algorithm has O(n) time complexity which is in the order of the number of edges in the network. The main operations in the algorithm are initial population creation, fitness evaluation, cross-over, mutation and clean-up process. Initial population creation is performed once in the beginning of GACD and all the other operations are performed for the number of iteration count.

Each operation in an iteration is has at worse $O(n)$ time-complexity. Most of these operations have less computation time, since they are performed on a limited number of nodes.

The most critical function, which increases the time-complexity of the genetic algorithm at most is the fitness evaluation function. It needs to traverse all the edges to compute network modularity. The sorting of the chromosomes takes O(*nlogn*) but the *n* value is the number of chromosomes, which is fixed and not related with the number of edges or number of vertices. For this reason, this computation time is taken as a constant value and it does not affect the overall complexity.

The GACD algorithm has $O(n)$ time-complexity. Even if we have a high number of iteration counts, it is not proportional to n and it does not affect the overall time-complexity of the algorithm.

ii.Accuracy

GACD is a fast algorithm that finds optimal community partitions in complex networks. To test the accuracy of our algorithm, we test GACD in real life complex networks where the real community structure is known and we found very successful community divisions. For example, for the famous Zachary's Karate Club example, GACD produced results that match with the real community structure between 97 %-100%.

The accuracy of the algorithm comes from the theoretic background of the algorithm. The quality of the community partitions is based on the network modularity that is the network's global property. It tries very different combinations to reach an ideal community partition through its process. Cross-over operation in GACD is modified according to the needs of complex networks. Cross-over in GACD produces better results than conventional cross-over in genetic algorithm.

To increase the accuracy, we also dealt with the minor disorders in the community partition. We defined and implemented a clean-up process, which eliminates the misplacement of nodes into wrong communities. Clean-up process seeks and eliminates small disorders in the community partition and this helps us to find communities that suit the strong definition of community.

iii.Scalability

GACD has O(n) time-complexity, which means that it can be applied to very large networks. Obviously, as the algorithm has a reasonable time-complexity, GACD eliminates the handicap of many community detection algorithms; the use of algorithm in very large networks.

The data structures used in the implementation algorithm, the calculations and the resource usage of the algorithm in terms of memory and CPU also approves the scalability of the algorithm. Some algorithms, even if they have reasonable time-complexity, have some square matrix calculations, which make it impractical for the use of it in very large networks. The memory size of square matrix of size n where n is very large, does not allow us to allocate it in current computers. For example, if we have 100,000 nodes, then a square matrix with size 100,000 will need 100,000 x 100,000 matrix slots, which we can not allocate in memory.

GACD has also advantage in that it uses simple array structure, which provides easy access and update of data. On the other hand, implementation schemes like heap, binary-tree has a drawback of long access times and very time-consuming reorganization processes. When the network is very large, the performance of these data structures get even worse.

We tried GACD for very large networks that have about 95,000 nodes and about 344,000 edges. We will give you the details of the runs in the experiments section.

iv.Resource consumption (Memory, CPU)

The main memory allocation in community detection is for the real network data that is read from Pajek formatted input dataset. No matter which algorithm we use, we must keep the network data in the memory. Other than that, GACD algorithm stores its population in the memory. The size of the memory allocation depends on the size of the

population only. The members of the population are simple data structures, whose main elements are integer arrays of size *n*, where *n* are the number of nodes in the network.

Some minor local data structures are also needed to store temporary values like the variables used in network modularity calculations or clean-up process. Algorithm does not need very complicated data structures and simple fixed-sized arrays are mainly used.

We tested our algorithm on a PC with 512 MB memory. However even for the very large network, the program that contains the network reader and GACD algorithm never exceeded the 300MB RAM boundary. The memory allocation details in different networks will be given in experiments section.

GACD uses CPU mainly for the cross-over process. Cross-over is done by searching through the nodes iteratively, so it is a linear time process. The most CPU-consuming process is the sorting of the chromosomes according to their fitness value. We used the Java's native array sorting function for this purpose, and it produces very fast results.

We tested our batch runs on a PC with Intel Pentium-4 3.0 GHz processor and the operating system is Microsoft Windows XP with SP2. The elapsed times of algorithm runs for different sized networks will be given in experiments section in detail.

4.3. Comparing GACD with other algorithms

GACD is fast and accurate community detection algorithm that certainly scales to very large complex networks. The network modularity is used as the quality assessment of community partitions in the algorithm like most of the algorithms proposed so far. Fast community detection algorithm (Newman, 2004), algorithm for very large networks (Clauset *et al*., 2004), community detection using extremal optimization (Duch and Arenas, 2005) are the examples of algorithms that uses network modularity as the quality criteria of algorithm.

While most of the proposed algorithms appear to be successful, they all are not supposed to be fast and scalable to very large networks. The only algorithm that is scalable to very large networks so far is the algorithm for very large networks (Clauset *et al.,* 2004). The other algorithms all perform well on small networks but are not suitable for large ones.

We implemented the algorithm for very large networks to compare the performance of GACD according to it. We saw that the algorithm is very slow compared to GACD; GACD performed 40 to 50 times faster than the algorithm for very large networks.

Another advantage of the GACD is that it does not need any priori knowledge like the number of communities. However the algorithm for very large networks needs a post process to reveal the community structure from the resultant dendogram. The algorithm proposed a way to do that but it is not guaranteed to be the best split all the time, as the algorithm does not calculate the global network modularity at all merging operation. The algorithm needs more memory and CPU resources than GACD uses too.

GACD is based on the strong definition community and avoids giant components of the networks, where the algorithm for very large networks has a tendency to create giant connected components. The algorithm tries to find the best merge operation according to the yield of merge to the network modularity. However in some cases, merging of a node to a component often increases the delta modularity value of the component and the component tends to collect many other nodes or components to stick them to it. On the other hand, GACD tries to find the best community configuration, by looking at the total configuration of the communities, and then it does not ignore the small components in the network by reorganizing nodes in the clean-up process.

The only advantage of the algorithm for very large networks over GACD is its deterministic structure. As GACD is an optimization algorithm, it is indeterminist and it promises to find the optimal solution, and as a result it can produce different solutions in different runs.

The two algorithms are run on Zachary's Karate Club data, which has the real community structure as well. The algorithm for very large networks placed a node in wrong community. GACD, as well, placed that particular node in wrong community for most of the runs, but it sometimes found 100% correct solution, which is superior to the algorithm for very large networks.

GACD has advantages on the algorithm with extremal optimization in terms of the time-complexity and resource usage. The algorithm with extremal optimization needs very much CPU time because of its recursive nature and its overall time-complexity. Recursive process needs also very large memory space, and it will be impractical in some cases to calculate recursive process in very large networks due to stack limitations in computers.

5. ANALYSIS OF EXPERIMENTAL RESULTS

5.1. Overview of Results

We have tried to assess the accuracy and performance of our introduced algorithm, examine its results and find out the best parameters of the GACD algorithm. We have the opportunity to measure the accuracy of our algorithm, by comparative analysis between what the algorithm produces and the real community structure of well known networks. We have also examined the scalability of the algorithm, which is one of the powerful capabilities of our algorithm, by running it for a very large network, namely Enron e-mail network.

The main problem to tackle is the proof of our algorithm in terms of accuracy. As we can not be sure about the accuracy of another algorithm's results, we had to try our algorithm on networks, where the real community structure is known. We tried our algorithm on well-known Zachary's Karate Club network and College Football network.

Another issue is that we need to device a method to automate the comparison of our algorithm's output with the real network data. As it is not a big issue to compare the network structure in Zachary's Karate Club data for once, it becomes a real burden to compare the results for many batch runs of the algorithm. This process is a real mass in larger networks as well. For this purpose, we devised a simple mapping algorithm and coded in Java such that it tries to map communities in two different community partition configuration of the same network, where the communities that have the most common nodes are assumed to be the communities that map to each other.

Then we tried many of the parameter combinations of GACD, and the accuracy of the algorithm for these values is noted. We chose to decide on the most critical parameters of GACD, and after finding optimal values of those parameters we dealt with other parameters.

We also played with the number of population and iteration count parameters. We experienced that both of these numbers are very crucial and should not be kept small. However after a point, increasing these numbers does not improve the performance of GACD any further.

We used one-way cross-over in GACD. However to see the success of two-way cross-over and the conventional cross-over, we implemented both of these methods and tried our algorithm with these methods. We saw that conventional cross-over creates randomness and does not improve the solutions. Two-way cross-over, which is not as worse as conventional cross-over, worsens the performance of the algorithm by a small percentage. For this reason we kept on selection of one-way cross-over.

Mutation, which is another critical genetic operation, is also tested. We remove the mutation from the algorithm and tested the effects of mutation. We saw that mutation is very critical in creating diversity and variance in the population. Without mutation, the algorithm gets stuck around some local maxima values during some of the runs.

Our correction effort on replacing the misplaced nodes is also tested. Clean-up process, which is supposed to eliminate the wrongly place nodes into more accurate communities is tested in some batch runs. We sometimes totally dismissed clean-up process and saw that the results get worse. We also tried different parameter combinations of the clean-up process and tried to find the combination that yields the best results.

To attain an understanding on the character of the network we deal with, we have examined the characteristics of the networks we observed such as, degree distributions, average distances, and diameter and provided the real community structure where available.

We also introduce the network structure for Enron e-mail network. As we do not have the actual community structure of the network and as it is a very large network, we tried to find the accuracy of our algorithm in Enron e-mail network, by sampling the results

of our algorithm and comparing it manually by investigating our knowledge about the Enron Corporation.

We present comparative analysis between the results of GACD according to different parameter combinations by the help of graphics and plots.

In our experiments we have observed that the accuracy of the algorithm is between 97-100 % in the networks we already know the real community structure. In this section, we present the accuracy of GACD with different parameter combinations. We also give the performance of GACD in details. We exhibit the running time of algorithm in networks with different sizes, i.e., for very large network like Enron e-mail network. We also show the resource usage of the algorithm in different networks and by using different parameter combinations.

5.2. Overview of Data: Zachary's Karate Club Network

We have run our GACD algorithm on Zachary's Karate Club dataset, which is popular as a community dataset for many community detection algorithms. Zachary's Karate Club data is drawn from the result of Zachary's (Zachary, 1977) "karate club" study.

In this study, Zachary observed 34 members of a karate club over a period of two years. During the course of the study, a disagreement developed between the administer of the club and the club's instructor, which is ultimately resulted in the instructor's leaving and starting a new club, taking about a half of the original club's members with him (Girvan and Newman, 2002). Zachary constructed a network of friendships between members of the club, using a variety of measures to estimate the strength of ties between individuals.

In our work, we used an unweighted version of this network dataset, which is the same as the ones used in GN algorithm or fast community detection algorithm. The network has 34 vertices (nodes) and 78 edges. There are two communities in the network. Some of the characteristic values of the network are listed in Table 5.1.

Table 5.1. Zachary's Karate Club network properties

Name of the value	Value
num. of vertices	34
num. of (undirected) edges	78
average path length	2
diameter	5
power-law exponent	1.23
num. of communities	2

5.3. Overview of Data: College Football Network

College football network is composed by the college football matches in USA, for Division I during 2000. The nodes in the network are the college football teams and there is a link between two teams if they played a match. The real community structure is the conferences that each team belongs to. The teams tend to play more matches with teams that are in the same conference and play less inter-conference matches. On the average 6 intra-conference matches and 3 inter-conference matches played by each single team.

The actual data is gathered from web, however all the teams did not satisfied the rule that number of intra-conference matches should be more than inter-conference matches. For this reason, the teams that play more inter-conference matches are removed from the network because of they do not exhibit a good community structure in data. The resultant dataset contains 93 teams and 452 matches among these teams. There are 10 conferences and these conferences show the actual community structure of the network.

The algorithm is run on this network and produced 93% accurate community structure in this network. GACD found the 100% correct community structure during some runs as well. The fast community detection algorithm is also run on this network, however it produced 78% accurate community structure, even if it needed the number of communities.

5.4. Overview of Data: Enron E-main Network

Enron was an energy firm in US and had many employees. It had business mainly in energy sector but had joint ventures from telecom to financial services. It collapsed as a result of some misleading investment and auditing frauds. After the investigations, mailboxes of 150 important employees were made public for academic purposes. We preprocessed about 512,000 text files to form a complex network dataset, where nodes are the e-mail addresses of the people and the edges are the e-mail connectivity between people.

Enron e-mail dataset was available at the end of 2004, and some recent research has been done on this dataset, in terms of complex networks. There has not been a community detection experiment that ran on this network yet, however, since the company's organization and some illegal actions are known, the real community structure can be formed for this large network.

Some of the important values of the Enron e-mail network are listed in Table 5.2.

Table 5.2. Enron e-mail network properties

Name of the value	Value
num. of vertices	93,526
num. of (undirected) edges	344,264
average path length	4.67
diameter	14
power-law exponent	1.51

5.5. The Experimental Setup

To facilitate a comprehensive analysis of the community structure, we have effectively built a community detection algorithm that can handle batch runs and can compare its results with actual data. We have built our algorithm using Java programming language in Eclipse 3.0 application development environment (ECL). We ran our code in Eclipse environment as well.

Our implementation enabled us to run very long batches by coding parameters in a text file that our code parses and runs according to those parameters. This eliminated big amount of manuel process which we handled during the early phase of our work, like running the algorithm with different parameters by stopping and starting it every time and starting the comparison process for each result. It also enabled documenting the correct parameter combination of GACD with the corresponding results. During manuel process, we did not track which parameter set produced which results.

We have used our own network reader module to read Pajek formatted data. We have tried to use JUNG (JUN) libraries, however, because of the performance issues for very large networks, we decided that our implementation is much faster than the one provided in JUNG.

We had spent a lot of effort to build the Enron e-mail network data. We obtained the Enron e-mail data from the web (URL1) in a raw format, which consists of 512,000 text files that contain e-mail messages. We implemented a text parser that works like a crawler. The parser searches through the folders of e-mail dataset and it parses the “from”, “to”, “cc” and “bcc” parts from e-mail files. It then merges all these information to a single text file.

After collecting e-mail connectivity information into a single file, we compose the Enron e-mail network by making a connection (*edge*) between two people, if they had an e-mail communication. We eliminate some of the noisy data from the network, where the e-mail addresses are hidden or anonymous. We build the network dataset in Pajek format.

Also, as we detail in Appendix A, we have constructed a community mapping algorithm that is used in assessing the result of our algorithm with the real community information of a network.

5.6. Choice of the Parameters in GACD

The choice of parameters is an important issue for the application of any of the algorithm.

Firstly, we have tried to find the best combination of genetic algorithm. These parameters are “Population Size”, “Iteration Count”, “Randomization rate in Initial population creation”, “Cross-over rate”, “Cross-over count” and “Mutation Rate”. The performance of the GACD improves significantly by the selection of best combination of these variables.

Afterwards, we tried to analyze the performance of cross-over operation in GACD. For this purpose, by keeping all the parameters fixed and as optimized as possible, we have run tests with one-way cross-over, two-way cross-over and the conventional cross-over in genetic algorithm. We noted the accuracy of the results with each selection.

Finally we tested the performance of our clean-up process, which was devised for making the community partition more accurate by applying the strong community definition. We tested clean-up process with different parameters.

We have tested the algorithm with Zachary’s Karate Club and compared the results’ accuracy with the real community structure of the network. We decided that we should run the algorithm for a number of time with a given parameter set in order to get an average result of the algorithm. As our algorithm is indeterminist, we must calculate the average accuracy value of the algorithm. For this purpose, we select the run count as 10 and calculated the average accuracy value of the result of these runs for each parameter set.

i.Parameter test results

We tested different parameters of GACD that are related solely with the genetic algorithm methods. We turned of the clean-up process for now and we will discuss the effect of clean-up process in the proceeding test results.

The first test is performed to check the effect of randomization rate during initialization phase. This rate is multiplied by the population size and some number of transfers of community IDs occur between randomly selected communities in chromosomes. We tested this operation with different values and the results are listed in **Table 5.3**. We set the genetic algorithm parameters to the optimal values, turned-off the clean-up process and changed only the tested parameter.

Table 5.3. Test results for randomization rate during initial population creation

Randomization rate during initialization	Accuracy percentage
0.0	92,65%
0.1	90,88%
0.2	92,94%
0.3	93,82%
0.4	95,29%
0.5	94,12%
0.6	95,00%
0.7	91,76%
0.8	95,00%
0.9	90,00%
1.0	89,71%

The test results show us that we can not omit the random initialization during initial population creation. The algorithm performed best with setting the parameter to 0.4, as it can be from the graphical representation of the results in Figure 5.1.

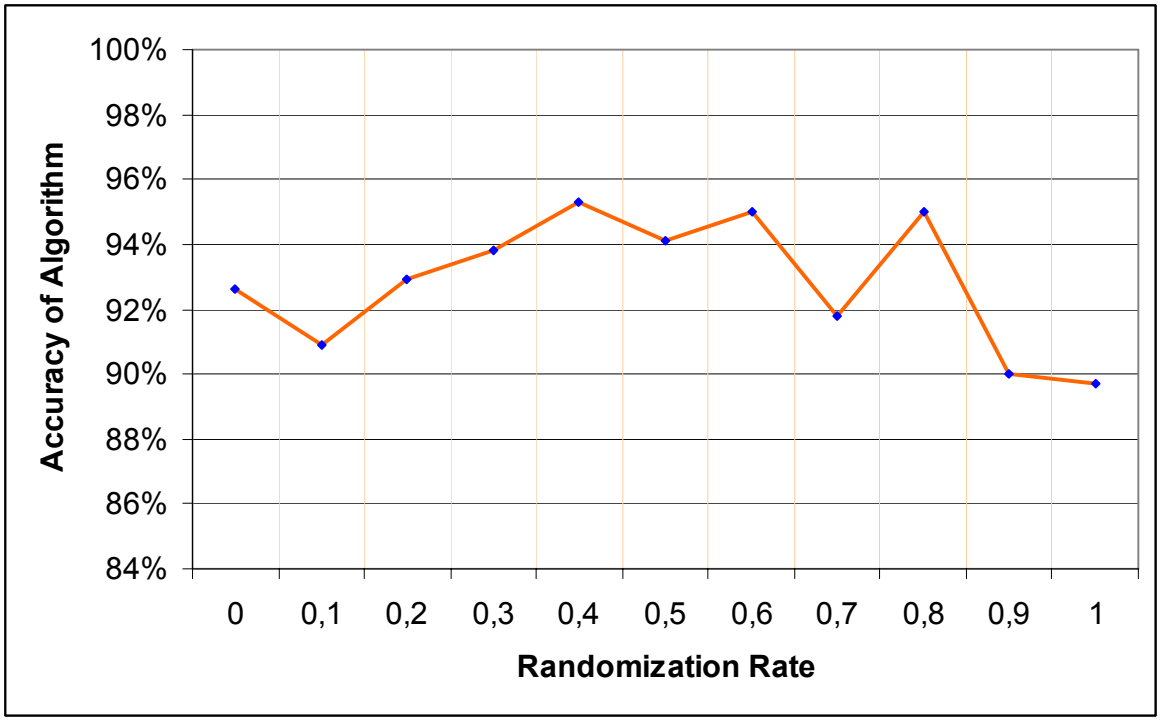

Figure 5.1. Effect of Randomization Rate on the Accuracy of GACD during Initialization

We will use the values of 0.4 for this parameter in the proceeding testing phases.

We tested the genetic algorithm parameters as our second test. During this test we tried many different combinations of the parameters. Like the first test, we turned-off the clean-up process, set the randomization rate of initialization to 0.4. However we ran the algorithm for 50 times with each parameter set on Zachary's Karate Club dataset, to find the most appropriate results by increasing the number of result samples. The test results are listed in Table 5.4. Here we ran our tests on populations with size 100 and the iteration count is 250 for genetic algorithm.

Table 5.4. Test results for diffetent genetic algorithm parameters of GACD

Cross-over rate	Mutation rate	Cross-over count	Accuracy %
0,1	0,1	0,1	79,47%
0,2	0,1	0,1	75,00%
0,3	0,1	0,1	74,88%
0,4	0,1	0,1	74,12%
0,5	0,1	0,1	69,88%
0,1	0,2	0,1	88,12%
0,1	0,3	0,1	87,71%
0,1	0,4	0,1	91,71%
0,1	0,5	0,1	89,76%
0,2	0,1	0,1	76,94%
0,2	0,2	0,1	86,47%
0,2	0,3	0,1	82,29%
0,2	0,4	0,1	89,76%
0,2	0,5	0,1	88,82%
0,3	0,2	0,1	84,18%
0,3	0,3	0,1	84,82%
0,3	0,4	0,1	85,12%
0,3	0,5	0,1	89,76%
0,4	0,2	0,1	77,76%
0,4	0,3	0,1	80,59%
0,4	0,4	0,1	82,12%
0,4	0,5	0,1	84,94%
0,1	0,1	0,2	82,29%
0,1	0,1	0,3	83,71%
0,1	0,1	0,4	85,41%
0,1	0,1	0,5	83,18%
0,2	0,2	0,2	89,53%
0,2	0,2	0,3	90,00%
0,2	0,2	0,4	91,88%
0,2	0,2	0,5	91,94%
0,3	0,3	0,2	89,65%
0,3	0,3	0,3	89,41%
0,3	**0,4**	**0,3**	**92,18%**
0,4	0,3	0,4	89,41%
0,4	0,4	0,4	90,00%
0,5	0,5	0,5	91,94%

From the experiments, we saw that the algorithm's accuracy is highest for the parameter combination (0.3-0.4-0.3). It is obviously seen that, increasing only one parameter does not improve the performance very much. The highest performance increase can be reached by a good combination of these three parameters. So we will use these values constant for further tests with other parameter variances.

Now we are ready to test effect of the population size and iteration count. By keeping other variables constant, we play with the population size and the iteration count of the algorithm on Zachary's Karate Club data. The results are listed in Table 5.5.

Table 5.5. Effects of population size and iteration count in GACD

Cross-over rate	Mutation rate	Accuracy %
50	50	67,94%
100	50	74,24%
200	50	77,94%
300	50	76,88%
50	100	78,41%
100	100	82,12%
200	100	87,47%
300	100	85,94%
50	150	83,29%
100	150	85,12%
200	150	89,94%
300	150	89,71%
50	250	89,29%
100	250	93,12%
200	250	93,94%
300	250	94,71%
50	500	90,65%
100	500	93,35%
200	500	96,53%
300	500	97,24%

It is obviously seen that increasing the population size and the iteration count improves the algorithm's accuracy, because increasing the population size increases the variance in the population which leads to create better chromosomes for the next generations. Increasing iteration count also increases the life-time of the evolution process and for this reason it again increases the accuracy of the algorithm. We can see the effects of both parameters from the plot exhibited in **Figure 5.2**.

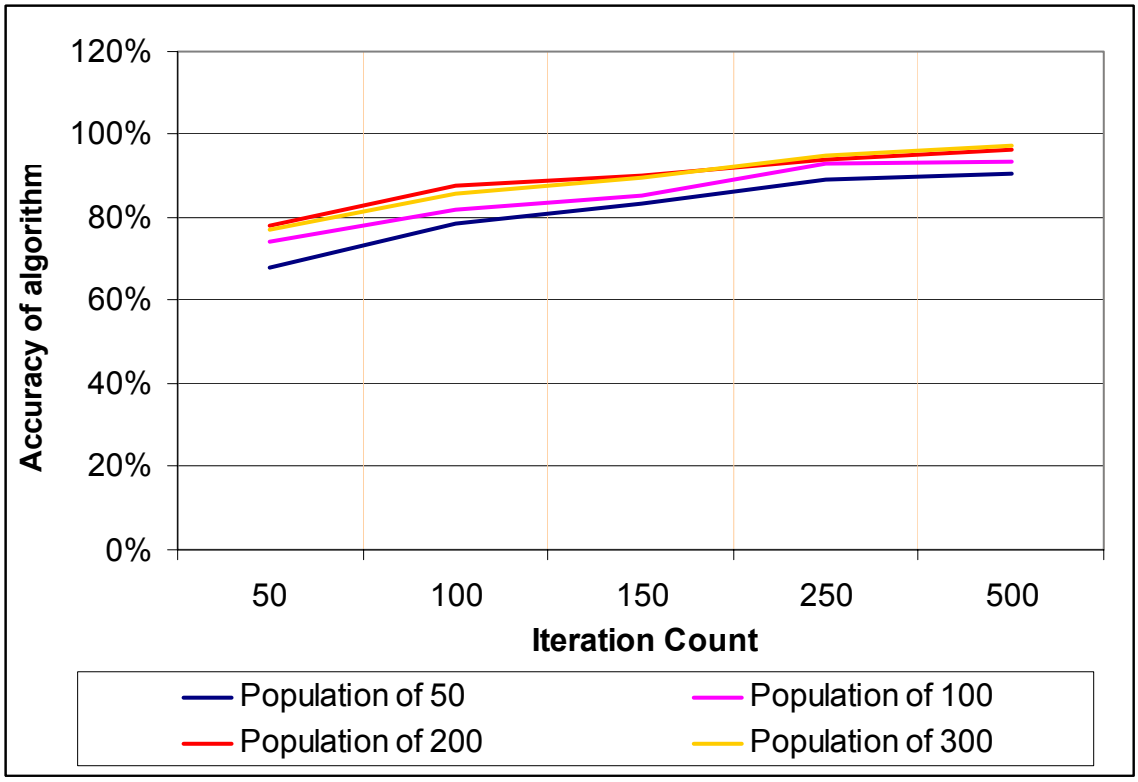

Figure 5.2. Effect of population size and iteration count

From the plot in Figure 5.2 we can easily see that increase in both of the variables improves the accuracy of the algorithm, however further increase in these variables does not yield further improvement, especially in the population size. From the data in Table 5.5. we see that, increasing population size from 200 to 300(for iteration count 500) improves the algorithm only by 1% while increasing the iteration count from 250 to 500(for population size of 200) improves the algorithm by 3%.

We concluded that a population size of 200 and iteration count of 250 are optimal values for GACD. Iteration count can be set to 500 for very large networks as well.

ii.Cross-over test results

We set the genetic algorithm parameters to fixed values, which are the optimal values found in the previous tests. We turned-off the clean-up process of GACD and tried to analyze the accuracy of one-way, two-way and conventional cross-over performances. The results of test are listed in Table 5.6.

Table 5.6. Performance of different cross-over schemens

Cross-over method	Accuracy %
One-way cross-over	94,65%
Two-way cross-over	61,94%
Conventional cross-over	92,82%

From the test results, it is obvious that one-way cross-over scheme which is used in GACD is the best scheme among them. Two-way cross over scheme decreased the accuracy of the algorithm very much, because it creates a random partition structure in the result. For the proceeding tests, we continue to use one-way cross-over scheme.

iii.Clean-up test results

We tested the clean-up process and compared it with the results that omit the clean-up operation. We ran our test with fixed parameters of the genetic algorithm, which were 0.3-0.4-0.3 combination. However to see the correlation of the clean-up with genetic algorithm parameters, we tried the clean-up process with 0.4-0.3-0.4 combination. Surprisingly, when we perform the clean-up process in GACD, the 0.4-0.3-0.4 combination produced very close results to the 0.3-0.4-0.3 combination.

Clean-up is proved to improve the algorithm's accuracy up to 4% which will be shown in the results of the batch runs. We also tested the runs with smaller population sizes and iteration counts to see if the clean-up process can make the performance degradation seamless. We observed that even if we decrease the population size and iteration count down to 50 %, by the improvement of the clean-up process, the overall accuracy of the algorithm decreased by a small percentage or stays the same for some runs. The results of the batch runs of algorithm with clean-up process are listed in

Table 5.7.

Table 5.7. Clean-up process results with different parameters

Community ID Variance Threshold	Clean-up Rate	Clean-up Cycle	Accuracy %
0,5	**0,1**	**0,1**	**93,59%**
0,1	0,1	0,1	97,24%
0,2	0,1	0,1	97,82%
0,3	0,1	0,1	97,29%
0,4	0,1	0,1	97,76%
0,5	**0,1**	**0,1**	**97,76%**
0,2	0,2	0,1	97,76%
0,2	0,3	0,1	96,88%
0,2	0,4	0,1	96,47%
0,2	0,5	0,1	95,12%
0,3	0,2	0,1	97,12%
0,3	0,3	0,1	97,00%
0,3	0,4	0,1	96,35%
0,3	0,5	0,1	95,71%
0,4	0,2	0,1	96,65%
0,4	0,3	0,1	97,18%
0,4	0,4	0,1	96,59%
0,3	0,3	0,3	97,47%
0,2	0,3	0,2	97,24%
0,4	0,3	0,4	97,82%
0,2	0,1	0,5	97,94%
0,2	**0,3**	**0,5**	**98,29%**

The first record in the table is the result of a run without clean-up process. The run with the same parameters and with the clean-up process produced 4% better accurate result. From the results of the runs, the best parameter combination of the clean-up is 0.2-0.3-0.5 combination and the second one is 0.2-0.1-0.5 combination. The result of tests with decreased number of iteration counts and population size is listed in Table 5.8.

Table 5.8. Decreasing iteration count and population size with clean-up process

Population Size	Iteration Count	Community ID Variance Threshold	Clean-up Rate	Clean-up Cycle	Accuracy %
200	250	0,2	0,3	0,2	97,24%
200	250	0,4	0,3	0,4	97,82%
200	250	0,2	0,1	0,5	97,94%
200	250	0,2	0,3	0,5	98,29%
200	200	0,2	0,3	0,2	97,00%
200	200	0,4	0,3	0,4	97,12%
200	200	0,2	0,1	0,5	97,06%

200	200	0,2	0,3	0,5	97,53%
100	250	0,2	0,3	0,2	95,65%
100	250	0,4	0,3	0,4	97,12%
100	250	0,2	0,1	0,5	98,24%
100	250	0,2	0,3	0,5	96,53%

From the test results, we can conclude that decreasing the population size by 50% does not a produce 2% accuracy loss. On the other hand the decrease in the iteration count by 20% does a very little effect on the accuracy, which is %0.24.

To sum up, the clean-up process improves the algorithm's accuracy significantly. It also helps us to decrease the number of iterations and population size, which will help us to decrease the resource usage in terms of memory and CPU and will decrease the elapsed time of the overall GACD algorithm.

5.7. Performance Tests

Once the choice of appropriate parameters is done, the algorithm is tested in terms of its performance and resource consumption.

We have constructed testing mechanism, which measures the memory allocation of the algorithm, the elapsed times in different networks. We tested our algorithm on a PC with Intel Pentium4 3.0 GHz computer and the operating system is Windows XP with SP2. We ran our java code in Eclipse 3.0 environment.

The memory allocation and elapsed times for different networks are listed in Table 5.9. Here we ran the algorithm with population size of 100 and with the iteration count of 250.

Table 5.9. Memory allocation sizes an elapsed times of GACD

Name of the network	Number of vertices	Number of edges	Total memory allocation in Kbytes	Elapsed time in seconds
Sampson Monastery Network	18	41	36,592	0.385
Zachary's Karate Club	34	78	36,900	0.459

Write Network	56	97	37,100	0.759
Smyth Network	286	554	37,748	2.780
E-coli Network	418	519	37,888	3.681
ENRON E-mail Network	93526	344264	136,204	1547.118

Note that these networks are used for measuring the algorithm's performance in terms of memory and elapsed time. For this reason, we do not go into details of the community structure of these networks. It is also important to note that, our GACD application uses a baseline memory of 19,996 K, which should be subtracted from the total memory allocation, if one needs the sole memory allocation of the GACD algorithm.

From the performance test results, we will show the relation between the number of nodes and the elapsed times with a plot, which will give us information about the time-complexity of the algorithm. See Figure 5.3. for the correlation of elapsed time with the number of edges.

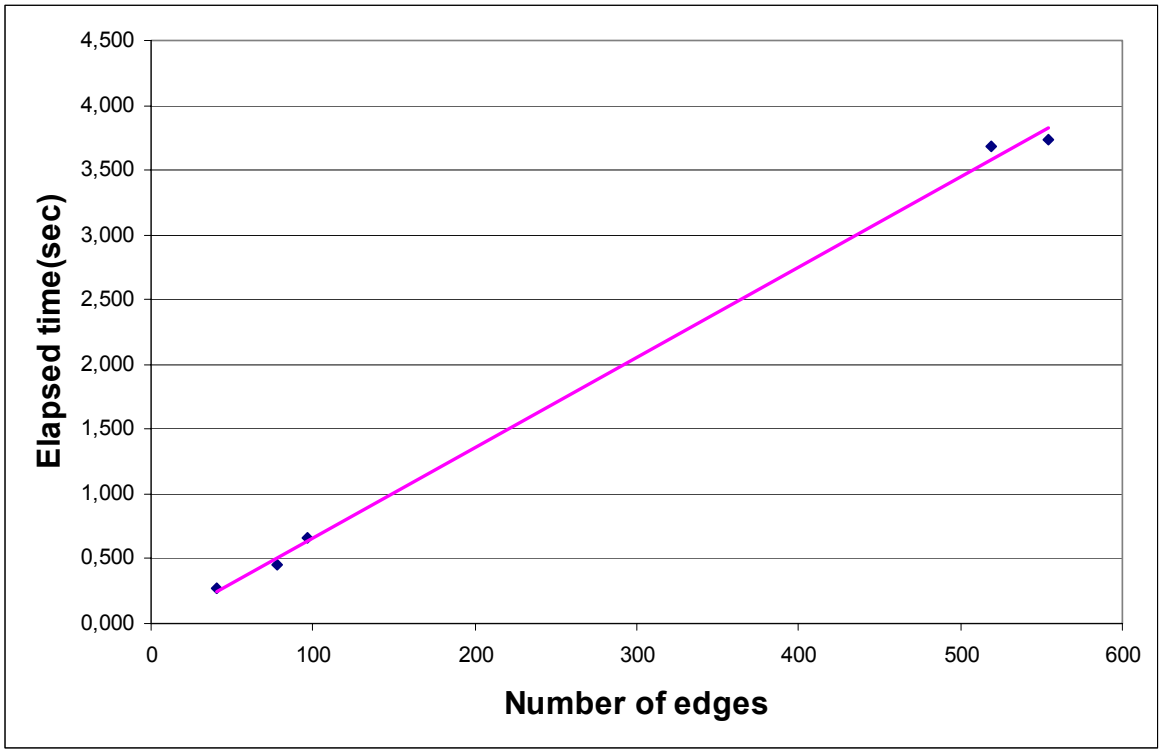

Figure 5.3. Elapsed times of GACD with different number of edges

The results prove the linear time-complexity O(*n*) where n is the number of edges in the network. The time-complexity is more obvious if we give the following correlation of elapsed time and network characteristics in Table 5.10.

Table 5.10. O(*n*) time-complexity of GACD

Name of the network	**V** *(Number of vertices)*	**E** *(Number of edges)*	**T** *(Elapsed time in seconds)*	**T/E**
Sampson Monastery Network	18	41	0,27500	0,004661
Zachary's Karate Club	34	78	0,45900	0,004098
Write Network	56	97	0,65900	0,004307
E-coli Network	418	519	3,68100	0,003928
Smyth Network	286	554	3,74000	0,004452
ENRON E-mail Network	93526	344264	1547.118	0,003534

It is obvious from the table that the algorithm has O(*n*) time-complexity even in very large complex networks. The algorithm actually has operations that take place in linear time-complexity.

We also compared the time-complexity of our GACD algorithm in very large networks with another algorithm; namely "the algorithm for very large networks" (Clauset *et al.,* 2004). We used Enron e-mail network dataset which has 93 526 nodes and 344 264 edges. "The algorithm for very large networks" completed its run in 23 hours. On the other hand GACD algorithm took only 25 minutes to complete its run. From these results, we concluded that our algorithm runs up to 50 times faster than "the algorithm for very large networks".

As we do not know the actual community structure of the Enron e-mail network, we could not compute the accuracy of our algorithm and the results of "the algorithm for very large networks" as well. We evaluated the accuracy of our algorithm only on the networks whose real community structures are known.

5.8. Community Structure Identification Tests in Different Networks using GACD

During the previous tests, we tried to find the best parameter set and tried to prove the performance of the methods we used in the implementation of GACD. We ran our algorithm on different networks, to evaluate the time-complexity and resource usage of the algorithm. As we need a baseline to assess the accuracy of our algorithm, we applied GACD algorithm on networks whose real community structures are known. We firstly applied our algorithm on well-known Zachary's Karate Club network.

5.8.1 Zachary's Karate Club Analysis

We gave the details of Zachary's Karate Club data in previous sections. The network has two communities and the community structure is visualized in Figure 5.4. We have drawn the network layout with Pajek software and the community information was already known from (Zachary, 1975).

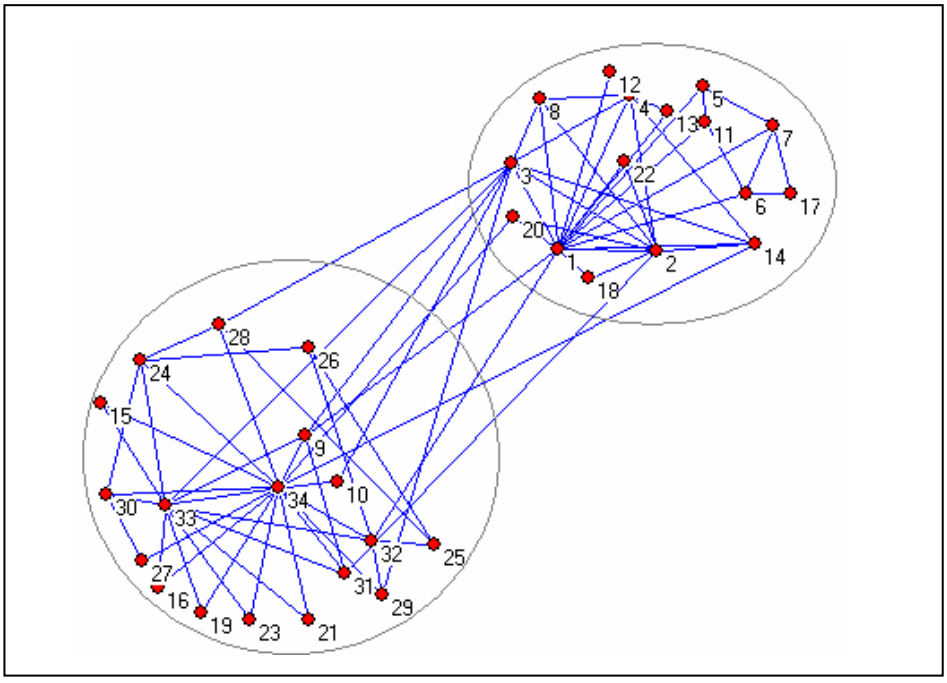

Figure 5.4. The Real Community Structure of the Zachary's Karate Club network

GACD finds the community partitioning 100% accurate for 33 nodes. It sometimes calculated the community of node 10 wrongly, which was always classified wrongly by the fast algorithm of Newman (Newman, 2004). In some runs, GACD finds the correct community classification for all the 34 nodes in the network, which increases the accuracy value of algorithm from 97% to 100% in this network.

6. DISCUSSION AND CONCLUSION

6.1. A Review of Work Done

In this work we have introduced and analyzed a new community detection algorithm we named GACD for analyzing community partitions in complex networks using genetic algorithm. GACD provides a fast and accurate approach for revealing the community structure in complex networks.

Community detection is crucial in understanding the fundamental structure of a network, its growth mechanism and the information flow in the network. Various properties can be obtained from the analysis on community structure of a complex network in food web, protein networks, e-mail networks etc. It also provides the grouping of similar objects in a network, which is important for biological and computer networks.

We have used the genetic algorithm as our base and used the network modularity to assess the quality of the partitions we form in terms of good community divisions. Genetic algorithm provides us a very efficient time-complexity for the overall algorithm that makes the algorithm scalable to very large networks.

We modified the operations of genetic algorithm for our needs and tuned the algorithm with the best parameter combination, according to test results we ran. We have tested our algorithm's accuracy in real-life networks, where the real community structure is known. We also tested the algorithm's elapsed time performance in very large networks.

We have presented the community detection concept in details and given the details of known community detection algorithms proposed so far. We have analyzed these algorithms according to different measures like time-complexity, resource usage and scalability. We even implemented some of them, which are out of the scope of this work, to see the performance of the algorithm in very large networks.

We have presented various experimental results. We ran GACD with various parameter sets on Zachary's Karate Club, to select the best parameter combination of these parameters. We also analyzed the benefit of modifications we made in algorithm by aiming to increase the overall accuracy. We found that the algorithm in the final version, which is used in the experiments, is the best tuned algorithm for accurate community detection.

Our study reveals that the most crucial concept in community detection process is the definition of community in theory. Most algorithms are biased to produce giant components while partitioning a network into communities. However giant connected components decrease the quality of community partitions and do not give much information about the real community structure inside itself. We used the strong definition of the community as the base concept and implemented GACD accordingly.

We introduce and use a measure we call Community ID Variance (CV), in our clean-up process, to increase the accuracy of the algorithm by eliminating the misplaced nodes into wrong communities. We analyzed and found the best threshold value for this measure in our experiments as well.

We have presented the accuracy of our algorithms in details for different networks in previous section.

6.2. Discussion and Directions for Theoretical Aspects

We hope our work to stimulate interest in evolutionary process for community detection in complex networks. The GACD algorithm provides a good example of this, and what we have presented in this work is an entry model of genetic algorithm of the possible further uses. In our work, we tried to optimize a network's global variable, which already existed in some other community detection algorithms, but we used genetic algorithm that provides us very fast and scalable algorithm different than other optimization processes.

We believe, genetic algorithm based approaches may have an important potential in finding community structure in complex networks. We do our part in this work by introducing the first usage of genetic algorithm in community detection in complex networks, which make it possible to apply a community detection algorithm to very large networks with very reasonable elapsed time.

In this work we have mainly focused on a fast and accurate community detection scheme, which is scalable to very large networks. However, it is well possible to improve it further by adding more operations that eliminates the defects of randomness and by improving the fitness function.

The strong community definition presents a good handling of partition of nodes into communities in a correct way. The clean-up procedure, which is based on strong definition of community, has experimentally proven to be useful in increasing the accuracy of our algorithm, and decreases the resource consumption, in terms of memory and CPU, by improving the convergence of the overall algorithm.

For an effective genetic algorithm implementation, the fitness function is one of the foremost issues to be improved. In this sense, some of the other important features of the network and community division metrics should be included in the fitness evaluation function, this remains as an important future work to improve the accuracy and convergence of the algorithm.

The algorithm can be improved further by increasing the accuracy. We haven't got the chance to see the accuracy of our algorithm in very large networks, because of the lack of real-life network data that is very large and contains the real community structure. However a sampling technique can be used to assess the accuracy of the algorithm.

We have used very different combination of parameter values for our experimentations. The results provided us the best combination of the parameters that resulted the most accurate results. For the further experiments, we set those values fixed and analyzed different aspects of our algorithm.

6.3. Discussion and Directions for Experimental Results and Methodology

We have used the Zachary's Karate Club network as our main data for accuracy testing. On the other hand, we have also used very different networks for performance experiments of the algorithm. Firstly, we believe that a real-life network that has the real community structure forms the best baseline to compare our algorithm's results. We could also compare the results of our algorithm with other known algorithms. However, we are not sure about the accuracy of those algorithms and it is documented in the definitions of the algorithms that none of them produces the 100% result for any network. We also tried to analyze the scalability of our algorithm for very large networks. We used the Enron e-mail network, which has 93 526 nodes, and experienced that our algorithm produced the community structure of the network in a reasonable time when compared to "the algorithm for very large networks" (Clauset *et al*., 2004).

Yet ultimately the goal of our algorithm is to be applicable to all sorts of the complex network. Community detection algorithms certainly have a great importance in biological, social and computer networks, so we believe that application and assessment of our algorithm on different networks is an important future work for our algorithm.

The experiment results also approved the selection of the network modularity as the fitness value in the algorithm to be successful. However, as we mentioned earlier, the convergence of the algorithm can be improved further, by adding more valuable parameters to this value. Network modularity, which is the basic quality assessment value for many other algorithms, exhibits a good community partition by its global nature. Since it is a global variable, the easy calculation of this value does not increase the overall time-complexity of the algorithm. On the other hand, using this variable weakens the community definition. To overcome this problem, we defined and implemented the clean-up process, which strengthens the community definition and improves the accuracy of the algorithm significantly.

An interesting result in our experiments is that we successfully proved the time-complexity of our algorithm. The proposed O(n) time-complexity exactly matched with the results of the runs in very large networks. This time-complexity of the algorithm, that does not get worse for very large networks, promises the usage of GACD in larger networks than Enron e-mail network.

While we have proposed some operations to decrease the resource usage and elapsed time, it remains a desirable future work to find further improvements on the convergence of the algorithm. Such an improvement will provide very quick response times that makes the algorithm more attractive in community detection. Quick response times make it more practical to use GACD in areas where dynamic community detection is desirable with a huge amount of data, like clustering the web search results or stopping the spreading of a virus in computer networks.

APPENDIX A. A MAPPING METHOD FOR COMPARING COMMUNITIES

In this appendix we will present a brief overview of a simple procedure used to map different community partitions for a network that is used to automatically compare our algorithm's accuracy by comparing its results with the actual community partition.

GACD algorithm gives the results as two-dimensional array, such that the first element is the node identifier and the second is the community identifier of that node. We have the real community partition of a network, as a dendogram structure so that we can cut the dendogram according to the community count GACD finds in its result. We needed a method to make a comparison between these two partitions, to assess the quality of our algorithm. See Figure A.1. for the details of the mapping algorithm. Here, we cut the dendogram at a point such that the subtrees of the dendogram will give the same number of communities as the GACD produces. After that we start to find a mapping between the GACD communities and the real communities.

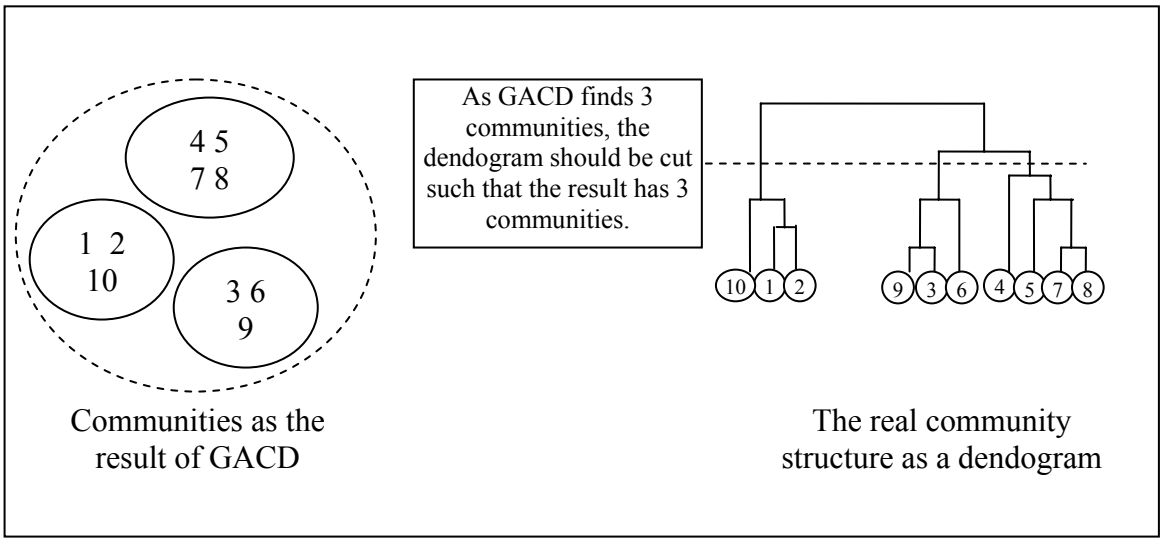

Figure A.1. Cutting of the real community structure's dendogram

For the first step, we have to find a mapping between these two sets, which tells us which community in the first configuration maps to which community in the second configuration. We defined and implemented a method that finds maps the communities with the highest resemblance into each other. Here the resemblance metric is the number of nodes that are common in both communities.

We defined a data structure to find the highest common resemblance between communities in different configuration. Then we implemented our method. The algorithm looks at the nodes' community IDs of two different configurations iteratively. For the first node in the network, the community ID from the first configuration and the community ID from the second configuration are inserted as a tuple, and its rating is set to 1. If algorithm tries to insert these tuples again, algorithm simply rejects insertion, it increases the rating of the tuple instead. By this method we iteratively search through all of the nodes in the community and insert every possible matching of the communities with their corresponding rating to our database.

After the search and insertion is done, we extract the tuple with highest rating and map the community ID taken from first configuration to the community ID taken from the second configuration. If we map a community ID from first configuration to a community ID from second configuration, then we remove all the tuples that include one of these community IDs from our database.

. Then we calculate the ratio of correctly classified nodes. The accuracy metric we used for our algorithm is the ratio of number of correctly placed nodes over the total number of nodes.

APPENDIX B. COMMON CONCEPTS IN COMPLEX NETWORKS AND COMMUNITY DETECTION

In this section we will give information about common concepts of complex networks and community detection.

Network: From the definition in graph theory, a network is the collection of members which are called vertices V (or nodes) and edges E those connect vertices according to some relationship. The edges (or links) can be directed, i.e., the relation between two vertices is one directional; or it can be undirected, i.e., the relation between two vertices is symmetric.

Edge betweenness: It is the number of shortest paths between pairs of vertices that run along a specific edge. Vertices are connected to each other via edges, and there are shortest paths between each vertice pairs. For a given edge, the number of shortest paths, that contains that particular edge on its way, is the edge betweenness value of that particular edge.

Dendogram: A dendogram is a tree-like structure, where the leaves are individual nodes and it contains the hierarchical structure of a network by combining the nodes iteratively from bottom to up according to the hierarchy of the network. The first connection is done between nodes, but further connections in upper levels can be done between a node and a previously connected component or between two previously connected components. See Figure B.2 for an example of dendogram.

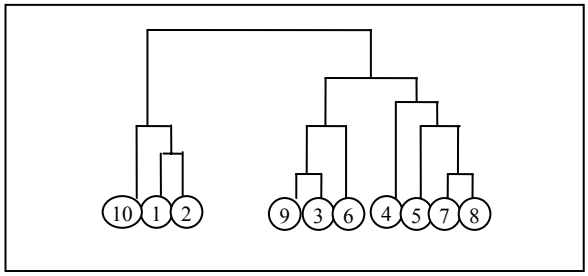

Figure B.2. An example of dendogram

REFERENCES

(Albert and Barabasi,1999) Albert, R., Barabasi, A.-L., (1999). The diameter of the world-wide web. In *Nature*, 401:130.

(Albert and Barabasi, 2002) Albert, R., Barabasi, A.-L., (2002). Statistical mechanics of complex networks. In *Reviews in Modern Physics*, 74:47-97.

(Clauset *et al.*, 2004) Clauset, A., Newman , M.E.J., Moore, C., (2004). Finding community structure in very large networks. In *Physical Review E*, 70:061111.

(Danon *et al*., 2005) Danon, L., Duch, J., Arenas, A., Diaz-Guilera, A., (2005). Community structure identification. Pre-print arXiv:cond-mat/0505245.

(Dorogovtsev and Mendes, 2002) Dorogovtsev, S.N., Mendes, J.F.F., (2002). Evolution of networks. In *Advances in Physics*, 51:1079-1187.

(Duch and Arenas, 2005) Duch, J., Arenas, A., (2005). Community detection in complex networks using extremal optimization. Pre-print cond-mat/0501368.

(ECL) Eclipse Java framework http://www.eclipse.org

(Freeman, 1977) Freeman, L., (1977). A set of measures of centrality based upon betweenness. In *Sociometry* 40:35-41.

(Girvan and Newman, 2002) Girvan, M., Newman, M.E.J., (2002). Community structure in social and biological networks. In *Proceedings of National Academy of Science*, 99:7821-7826.

(Holland, 1975) Holland, J.H., (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press*, Ann Arbor, Michigan.

(JUN) JUNG framework http://jung.sourceforge.net

(Newman, 2004) Newman, M.E.J., (2004). Fast algorithm for detecting community structure in networks. In *Physical Review E*, 69:066133.

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

(PAJ) Pajek network analysis tool available at http://vlado.fmf.uni-lj.si

(Radicchi *et al.*, 2004) Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D., (2004). Defining and identifying communities in networks. In *Proceedings of National Academy of Science in USA,* 101:2658-2663.

(Reichardt and Bornholdt, 2004) Reichardt, J., Bornholdt, S., (2004) Detecting fuzzy community structure in complex networks with a q-state potts model. In *Physical Review Letters*, 93:218701.

(Russell and Norvig, 2003) Russell, S., Norvig, P. (2003). Local Search and Optimization Problems. In Russell, S., Norvig, P., editors, *Artificial Intelligence, A Modern Approach*, pages 116-117,132-133, New Jersey, USA. Pearson Education Inc & Prentice Hall.

(Palla *et al*., 2005) Palla, G., Derenyi, I., Farkas, I., Vicsek, T., (2005). Uncovering the overlapping community structure of complex networks in nature and society. Pre-print arXiv:physics/0506133.

(Pissard and Assadi, 2005) Pissard, N., Assadi, H., (2005). Detecting overlapping communities in linear time with P&A algorithm. Pre-print ccsd-00009164.

(Scott, 2000) Scott, J., (2000). *Social Network Analysis: A Handbook*, 2nd edition. Sage, London.

(Strogatz, 2001) Strogatz, S.H., (2001). Exploring complex networks. In *Nature* 410:268-276

(URL1) Enron e-mail dataset available at http://www.cs.cmu.edu/~enron/

(VIV) Vivisimo clustered search engine http://www.vivisimo.com

(Watts and Strogatz, 1998) Watts, D.J., Strogatz, S.H., (1998). Collective dynamics of 'small-world' networks. In *Nature*, 393:440-442.

(Wu and Huberman, 2004) Wu, F., Huberman, B.A., (2004). Finding communities in linear time: A physics approach. In *European Physical Journal B*, 38:331-338.

(Zachary, 1977) Zachary, W.W., (1977). An information flow model for conflict and fission in small groups. In *Journal of Anthropological Research*, 33:452-473.

REFERENCES NOT CITED

Capocci, A., Servedio, V.D.P., Caldrelli, G., Colaiori, F., (2004). Detecting communities in large networks. Pre-print arXiv:cond-mat/0402499.

Castellano, C., Cecconi, F., Loreto, V., Parisi, D., Radicchi, F., (2004). Self-contained algorithms to detect communities in networks. In *European Physical Journal B*, 38:311-319

Clauset, A., (2005). Finding local community structure in networks. Pre-print arXiv:physics/0503036.

Drineas, P., Krishnamoorthy, M.S., Sofka, M.D., Yener B. Studying E-mail Graphs for Intelligence Monitoring and Analysis in the Absence of Semantic Information

Exploring enron, UC Berkeley http://jheer.org/enron

Fortunato, S., Latora, V., Marchiori, M., (2004). Method to find community structures based on information centrality. In *Physical Review E*, 70:056104.

Gfeller, D., Chappelier, J.C., Rios, P.D.L., (2005). Finding instabilities in the community structure of complex networks. Pre-print arXiv:cond-mat/0503593.

Muff, S., Rao, F., Calflisch, A., (2005). Local modularity measure for network clusterizations. Pre-print arXiv:cond-mat/0503252.