COMMUNITY DETECTION USING AGENTS IN COMPLEX NETWORKS

by

İsmail Güneş

B.S., in Computer Engineering, İstanbul Technical University, 2003

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

2006

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Haluk Bingöl for his invaluable guidance and helping during the preparation of this thesis. 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 Assist.Prof. Tunga Güngör 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 Mürsel Taşkın, 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 Özlem Gemici and to my family.

ABSTRACT

COMMUNITY DETECTION USING AGENTS IN COMPLEX NETWORKS

The main purpose of this work is analyzing the complex networks. Mainly, a new community detection algorithm based on using software agents will be introduced. The technical background will be given in details and the performance of the algorithm for different networks will be examined.

First, the main concepts in complex networks and the information about usage of the software agents will be introduced. Then, the current methods finding the communities in complex networks will be presented. The community is defined to be the group of nodes which are densely connected within the group but rarely connected to the outside. There have been many algorithm proposed so far to detect the communities in complex networks. However, most of them have some weaknesses. For instance, some algorithms need some prior information about the network to find the communities such as number of communities. There are also some algorithms with higher complexity values. Yet, these algorithms are not feasible or applicable to the real world complex networks. Our algorithm proposes a new algorithm to detect the communities in complex networks and addresses these issues.

An algorithm is developed that utilizes software agents for gathering information about the network and uses network modularity to decide on the community among candidates. To test the accuracy of the algorithm, the networks with known community structure like Zachary Karate Club network are used. The large networks such as web sites network that we created by observing a proxy server are also used to test the complexity and scalability of the algorithm. Besides, the performance with a computer generated network is also presented. Finally, it can be said that the algorithm is able to reveal the nearly optimum communities with acceptable complexity.

YAZILIM AJANLARI KULLANARAK KARMAŞIK AĞLARDA ALTTOPLULUK BULMA

Bu çalışma karmaşık ağların incelenmesine yönelik bir çalışmadır. Temel olarak, bir ağdaki alttoplulukları yazılım ajanları kulanılarak bulmayı sağlayan yeni bir algoritma tanıtılacak, ilgili altyapı detaylı olarak verilecek ve çeşitli ağlar için algoritma performansı gözlenecektir.

Öncelikle, karmaşık ağlarda temel kavramlar tanıtılacak ve yazılım ajanlarının kullanımına ilişkin bilgiler verilecektir. Daha sonra, karmaşık ağlarda alttopluluk yapısını ortaya çıkaran mevcut algoritmalar tanıtılacaktır. Bir karmaşık ağda, kendi elemanları aralarında yoğun olarak bağlı olan, ancak diğer elemanlarla seyrek olarak bağlı olan alt gruplar, alttopluluk olarak tanımlanır. Bugüne kadar, alttopluluk yapısını ortaya çıkarmak için çok çeşitli algoritmalar sunulmuştur, ancak bu algoritmaların da bir takım eksiklikleri vardır. Örnek olarak, bu algoritmaların bir çoğu, doğru şekilde çalışabilmek için ağ hakkında bazı ön bilgilere ihtiyaç duyar ki, bu gerçek hayatta çok uygulanabilir bir yol değildir. Ayrıca, zaman karmaşıklığı yüksek olan bazı algoritmalar ise sadece küçük ağlara uygulanabilmektedir. Geliştirdiğimiz algoritma, karmaşık ağlarda alttopluluk bulmak için yeni bir yöntem önermekte ve bu sorunlara çözüm getirmektedir.

Algoritmamız, ağ hakkında bilgi edinmek için yazılım ajanlarını, altopluluklara karar vermek içinse ağ modülerliğini (*network modularity*) temel almaktadır. Algoritmayı test ederken, sonuçların doğruluğunu görebilmek için Zachary'nin Karate Kulübü ağı gibi alttopluluk yapısı bilenen ağlarla, karmaşıklığını ve ölçeklenebilirliğini test etmek içinse Reuters ağı gibi büyük ağlarla çalışıldı. Ayrıca, kendi oluşturduğumuz ve internet sitelerinin ziyaret edilme bilgilerinden oluşan internet ağı için de algoritmamızı test edildi. Bunun yaninda algoritma performansı, bilgisayar tarafından üretilen yapay ağlar için de test edildi. Sonuç olarak, algoritmanın kabul edilebilir zamanda, optimuma yakın alttoplulukları ortaya çıkardığını söyleyebiliriz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

C	Clustering Coefficient
deg(V)	Degree of a vertex
$\|E\|$	Number of edges in a network
G(V, E)	An undirected graph with vertices and edges
n	Number of nodes in a network, n = |V|
N_a	Number of agents used in the algorithm
N_m	Memory size of each agent
p(k)	Probability of a vertex having k neighbors
$P_E(j)$	Probability of an agent to move using an edge
Q	Network Modularity
Z_{out}	Number of intercommunity edges for a vertex in computer generated network

ACC	Agent Communication Channel
ACL	Agent Communication Language
AID	Agent Identifier
AMS	Agent Management System
DF	Directory Facilitator
EO	Extremal Optimization algorithm
FIPA	Foundation for Intelligent Physical Agents
GN	Girvan & Newman algorithm
JADE	Java Agent Development Environment
SACD	A community detection algorithm based on software agents

1. INTRODUCTION

1.1. Motivation and Essentials

In recent years, the use of complex networks has been more popular in several areas. A network is defined to be the system that has nodes as members and edges linking them according to a relationship. A complex network is considered complex on account of their topological features that do not exist in simple networks. The networks are represented by the graph theory. However, they cannot be modelled by classical random graphs, thus being more complex than these. Some examples of the complex networks are the World Wide Web (WWW), internet, actor collaboration network, metabolic networks, citation networks, ecological networks etc.

The most important features of the complex networks are;

- *Clustering,* where there are acquaintances in which every member knows every other member. This is measured by the *clustering coefficient.*
- *The small-world effect,* where the network has small average path length and higher clustering coefficient.
- *Degree distribution,* where many complex networks exhibit skewed degree distributions instead of Poisson distribution like random graphs. Usually, these degree distributions have a power-law tail.
- *Community structure,* where the groups of nodes have high density of edges within groups.

A community is a group of vertices within which connections are dense, but between which connections are sparser. For instance, in co-authorship network where the nodes are the authors of the papers and the edges represents the writing a paper collaboratively, communities are formed by the authors who work together frequently. The communities in

a network may reveal many hidden features of the network such as relationships between the members or some organizations among members of a subgroup in the network.

Detecting communities have appearing as a requirement by the increasing usage of huge networks as internet. The nodes belonging to the same community are more probable to have properties in common (Danon *et al*., 2005). For example, in the WWW, detecting communities has uncovered thematic clusters. In metabolic or biochemical networks, communities may represent the functional groups and detecting these groups may simplify the functional analysis. By detecting communities in the network, the information flow mechanism of the network may be revealed. Today, detecting communities is widely used in advertising, detecting terrorist organizations in the WWW etc. Hence, the community detection is an indispensable part of the complex network analysis.

The matter of detecting communities has been the subject of many issues in various disciplines. To avoid these problems, several methods have been proposed vary in terms of their needs and their approaches. The most well-known community detection algorithm is the algorithm of the Girvan and Newman (Girvan and Newman, 2002), which divides networks by removing edges iteratively. This algorithm is a divisive algorithm and based on the betweenness centrality. After the proposal of this main paper, many ideas are developed to decrease the computational effort. The Radicchi's algorithm (Radicchi *et al.*, 2004) tries to reduce the complexity of the GN algorithm by computing only local quantities. Besides, there are also some agglomerative methods which run faster. For the usage in very large networks a hierarchical agglomerative method (Clauset *et al.*, 2004) is proposed that uses greedy optimization for efficiently searching the candidates. At last, it may be said that there are several approaches on community detection to be used in different disciplines and they differ according to the accuracy and complexity they provide.

Although the community detection has been the topic of studies in different subjects, there are still some problems to be resolved. First, detecting communities is a very complex problem. It is similar with the *Graph Bipartitioning Problem (GBP)* that is NP complete problem. In the GBP, the graph is divided into two equally sized communities. However, there are usually more than two communities and the operation of detecting

these communities is costly. Hence, much of the community detection algorithms have high time complexity and require high computational effort. Moreover, the hierarchical structure of the communities is to be determined unlike GBP. The additional computations for finding community structure make algorithms more complex. Since much of the algorithms have higher complexity, they are not efficient for very large networks. Another problem of many community detection algorithms is the need for a priori knowledge about the network structure. For instance, some hierarchical algorithms provide *dendrogram* representation at the end of the algorithm and needs the number of communities to decide at which step the *dendrogram* will be divided. In real networks, the number of communities is generally unknown. These algorithms are not self-contained and they require some information about the network which usually we do not have.

In our algorithm, Smart Agents in Community Detection (SACD), an agent based approach is proposed to detect communities in the complex networks. The smart agents are used collaboratively to collect information about the network structure. These agents learn more while the program is running and make smart moves. The algorithm has two main phases; in the first phase, the agents traverse in the graph in a biased way and try to learn about network. After these exploration operations, the collected results are analyzed to decide on the communities of the network. The modularity value in the algorithm of Girvan and Newman (Newman and Girvan, 2004) is used for the analysis phase.

The main idea behind using agents is their behaviour of tending to stay in the community. A community is defined to be the densely connected subgroup in the network. Hence, there are more edges within the community than the edges from the community to the other communities. It is more probable for agents to stay in the community and our algorithm is inspired by this realistic behaviour of the agents. As a result, the agents helped us to learn the community structure in the networks.

In coming chapters, we will introduce the algorithm and analyze its aspects. We have used the algorithms on both the manually created datasets and the datasets from the real world like Zachary Karate Club dataset. We have also run the algorithm on both the small

networks and the large networks and analyzed their results. The test results with different parameters of the algorithm will be evaluated in the analysis section.

1.2. Main Contributions

Our algorithm is based on the agents and the agents are similar with threads. The thread like structure of the algorithm makes it possible to learn about network structure asynchronously and increase the performance. This structure of the algorithm also let us use program in the distributed environments. Running each agent in a different workstation will increase the efficiency of the algorithm.

Unlike most of the community detection algorithms, our algorithm needs no priori knowledge about the network. The number of communities, size of communities or any other threshold value may be needed in some algorithms. To avoid this, the SACD algorithm performs two main operations; first, the required information about the network is collected from the travelling of the agents and possible community candidates are found. Then, the network modularity value is used for deciding on the number of communities. Many algorithms providing dendrogram, needs further work like cutting the dendrogram at a level after the algorithm completion. But our algorithm needs no further work or any threshold value to decide on communities.

The contribution of the algorithm is the usage of agents in community detection. First, many generations of agents are used instead of only one and the results are improved by each generation. That is, an optimization is provided by the agents which are used after first generation agents. Second, the agents are placed in the network depending on the hit numbers of the nodes instead of placing randomly. The agents start to traverse from the most visited nodes. This strengthens the idea of exploring communities. Moreover, the agents move in a smart way that they use a global knowledge about the network for a biased move and they produce better results by the time. So, the algorithm has important additions to the usage of agents in detecting communities.

1.3. Organization of the Document

The rest of the document is organized as follows. Chapter 2 contains all the related literature survey and preliminaries offered. Three topics are explained; these are Complex Networks, Community Detection Concept and Agents. The Agent API JADE is also explained in details.

Chapter 3 formally introduces the community detection in complex networks and its uses. Some of the current methods are given in this chapter and their performance is compared. Chapter 4 explores the use of Agents as community detection method and gives the fundamentals of the algorithm of community detection using agents.

Chapter 5 contains all the experimental results we have obtained. The experimental setup part is also included in this chapter. The usage of the parameters in the algorithm is analyzed and the choice of these parameters is explored. Moreover, the accuracy is tested for different networks by comparing the provided results with the known community structures. Lastly, a performance test is made to estimate the computational and time costs which the algorithm has.

The conclusion and future work sections are in chapter 6. The discussions and directions are also included in this chapter.

2. RELATED RESEARCH AND OTHER PRELIMINARIES

In section 2.1., the Complex Networks are presented to clarify the main concepts of our work. In addition to related definitions, the types of the Complex Networks are listed. Besides, some basic properties of these networks are also explained.

The need for the community detection in Complex Networks is the main issue of section 2.2. We focus on the benefits behind community detection in this section. Then, the difficulties against detecting communities are mentioned.

Last, the agent mechanism is explained in details. Our algorithm is based on using software agents on community detection and the work flow of the agent API that we've used is clarified. Some code snippets are also included in the presentation of agents.

2.1. Complex Networks

2.1.1. Overview

A complex system is the one in which there are multiple interactions between many different components, constantly evolve and unfold over time (WIK). Some examples of complex systems are ants, nervous systems, cells, human beings etc. Many complex systems are represented as complex networks. Hence, the complex systems should be the first point while starting to study on complex networks.

There are many features that distinguish complex systems from others. First, the relationships in complex systems are nonlinear and the effect is usually not directly proportional to the cause. In addition, complex systems may consist of many other complex subsystems. Another distinguishing feature of complex systems is that determining the boundaries of the complex systems is hard. Complex systems are widespreadly studied in different disciplines. The physics, computer science, economics,

neuroscience, cell system, artificial systems and immune system are some of the areas that complex systems bring new solutions.

A network is defined as the interconnected system or configuration of components which are called nodes (vertices) and the edges connecting them. The neural networks, social networks, metabolic networks, scientific collaboration networks are the some examples of the networks (Newman, 2004). At this point, we can define the complex networks.

The “Complex Network” is known as the network that has non-trivial topological structure. There are many features like high clustering coefficient, assortativity and community structure which distinguish complex networks from others. Random graphs are used to be compared with complex networks. The properties of a random graph are summarized in section 2.1.2.

For instance, the Internet is assumed to be a complex network of routers and computers connected by physical or wireless connections, the social networks of the people connected with social relationships, WWW is the networks of web pages connected with hyperlinks and the cell is the network of chemicals connected by chemical connections (Albert and Barabasi, 2002). Several examples of complex networks may be given from different areas of life.

One of the frequently observed characteristics that the complex networks have is the scale free structure. Both the Internet and the protein networks have this structure and it is wondered how such different networks share the same topology. There are several different networks having the same properties. Hence, the laws behind the construction of these networks have been one of the main issues of complex network research in recent years. There are still many things to learn about complex networks.

2.1.2. Definitions

A graph is a set of vertices connected by edges. In an undirected graph, an edge from a point 1 to point 2 is considered to be the same thing as an edge from point 2 to point 1. For a directed graph, the two directions are assumed to be distinct arcs. In this work, we study undirected graphs. An undirected graph G is represented by $G(V, E)$ where V is the set of vertices and E is the set of edges.

i. Degree

The degree of a vertex is the number of edges incident to this edge. The degree of a vertex v is denoted by the $\deg(v)$. The degree is also known as connectivity. The in-degree of a vertex is the number of arcs arriving to the vertex and the out-degree of a vertex is the number of arcs leaving this edge. A vertex with zero degree is called isolated. A vertex with a degree of 1 is called leaf. Moreover, vertex with zero in-degree is called source while the vertex with zero out-degree is called sink.

For a graph, G = (V, E), the total number of degrees are given by,

$$\sum \deg(v) = 2\,|E| \tag{2.1}$$

ii. Shortest Path

The length of a path is defined to be the number of edges travelled during this path. There may be many paths between two vertices. The path with the shortest length is called the shortest path. The length of it is called the *distance* between the two vertices.

iii. Diameter

The diameter is the length of the longest shortest path in the graph. In other words, diameter is the longest distance between any two vertices. The diameter is used for measuring the size of the graph.

iv. Random Graph

A random graph is generated by some random process. It is obtained by starting with a set of vertices and adding edges between them at random. The random graph may have small average distance but it is not expected to show scale-free characteristic nor a community structure.

2.1.3. Types of Networks in the Real Life

The networks in the real life may be classified into four main categories: social networks, information networks, technological networks and the biological networks. The big part of these networks is examined in many papers and the characteristics are provided. These networks are given to show the usage of complex networks in various areas in our daily lives (Newman, 2004).

i. Social Networks

The social networks are the networks of people connected by some interactions or relationships. The main problems of the social network studies are the inaccuracy, small sample size and subjectivity. The data is usually collected by directly querying the people.

ii. Information Networks

The information networks are realized by using knowledge. Besides, this type of network structures holds information about its vertices. For instance, the citation network is constructed by using the citing knowledge of academic papers. A paper is connected to another if it cites other paper. Another example of the information networks is the web. The web pages are assumed to be connected if there is a hyperlink among them.

iii. Technological Networks

The technological networks are human made networks which are used for carrying some information. The electric power grid, delivery parcel companies, the telephone networks are the some examples of technological networks.

iv. Biological Networks

The biology is another area that complex networks are used. The metabolic units are assumed to be the vertices of the network and they are connected if there are any metabolic interactions between them. Moreover, neural networks, brain system and cell mechanism are other types of biological networks.

2.1.4. Properties of Complex Networks

After several researches on the complex systems it is seen that, the complex networks have some properties in common unlike random graphs. The most well known properties of the complex networks are explained in this section;

i. Clustering

The clustering of a complex network is measured by the *clustering coefficient*. It is introduced (Watts and Strogatz, 1998) to measure the small-world characteristic of the network. The clustering coefficient of a vertex is the fraction of edges with its neighbours and the number of edges that could possibly exist between them. The number of possible edges between vertex *i* and its neighbours is $k_i(k_i\text{-}1)/2$ in an undirected graph where, k_i is the number of neighbours of vertex *i*. Thus, the clustering coefficient of a vertex is given by;

$$C_i = \frac{2E_i}{k_i(k_i\text{-}1)} \tag{2.2}$$

where E_i shows the actual number of edges connecting vertex *i* with its neighbours. The clustering coefficient of a network is calculated as the average of the clustering coefficient for each vertex;

$$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} Ci \tag{2.3}$$

where, n is the number of nodes in the network. In a complex network, the clustering coefficient is expected to be higher than the clustering coefficient of a random network with the same number of vertices and edges (Newman, 2003).

ii. Small-World Property

Networks with small-world property are characterized by higher clustering coefficient and lower average shortest path length compared to the random graph with the same number of nodes and edges. The small world effect means that any vertex in a network may be reached from any other vertex with a small number of hops even though the network is large scaled. The small-world effect on a network also means that the spread of the information or any other unit will be fast (Newman, 2003). Moreover, the hop count on the Internet in a communication and the spreading speed of a disease between people are also the examples affected by the small-world effect.

iii. Scale-Free Property

In scale-free networks, a few nodes act as “highly connected hubs”, although most nodes are of low degree. (WIK) In 1999, Albert Barabasi investigated the connectedness of the Web. It was expected that the Web would show random connectivity that vertices have approximately equal degrees. However, it was seen that only a few nodes have higher degrees. This structure is called “power-law” and the networks having power-law structure is called scale-free. The most common model that explains the generation of the scale-free property of a network is the “rich get richer” model that is proposed by the Barabasi and Albert (Albert and Barabasi, 2002). A web page with higher number of in-links attracts more links than a webpage with lower degree has. This growth model is related with a model called “preferential attachment” that determines the node that a new node will connected to.

2.2. Community Structure

The most complex networks have vertices in a group structure that the vertices within the groups have higher density of edges and vertices between groups have low density of edges (Newman, 2003). This structure is called “Community structure” and it is not easy to distinguish these groups into communities because there is not a certain definition. The traditional way to detect communities is the “hierarchical clustering”. In this method, a “connection strength” value is assigned to all vertex pairs without any edges between them. Then, edges are added according to the decreasing strength value. Finally, a dendrogram is constructed at a time and the desired communities are provided. A dendrogram is a tree-like structure, where the leaves are individual nodes and it includes the hierarchical structure of a network by joining the nodes iteratively from bottom to up considering the hierarchy of the network. By detecting community structure, many underlying information may be explored about network. A sample network with three communities is shown in figure 2.1.

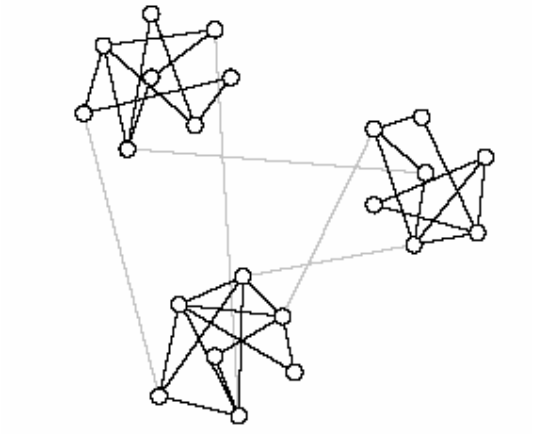


Figure 2.1. A sample network with community structure (Girvan and Newman, 2002)

There are several other definitions of community in the literature. As a general one, a community is defined to be the collection of living things that share an environment. In this work, we will use the community definition in complex networks. These communities

are distinguished by interaction in many ways. Also, the members of the communities have something in common. For instance, communities in a social network may represent real social groupings, perhaps by interest or background; communities in a citation network may represent related papers on a single topic; communities in a metabolic network may represent cycles and other functional groupings; communities in the Web may represent pages on related topics (Girvan and Newman, 2002). Another definition of a community is that the vertices in networks are often tightly-knit groups with a high density of within-group edges and a lower density of between-group edges (Newman, 2004).

In addition to many community definitions, there is also a mathematical formula to measure the level of community structure that networks have. This community definition is called *network modularity* and it is first proposed by the (Newman and Girvan, 2004)

$$Q = \sum_{i=1}^{k} (e_{ii} - a_i^2) \tag{2.4}$$

The network modularity is based on the idea that random networks do not have community structure (Danon *et al.*, 2005). We assume that we have a random network and random partition of this network into n_c partitions as the communities. Then, we define a $n_c \times n_c$ matrix e, where the elements e_{ij} denotes the fraction of edges starting in partition i and ending in partition j. The sum of a row or the sum of a column is defined to be $a_i = \sum_j e_{ij}$. If the network do not show community structure, it is possible to estimate the fraction of edges in a partition. It will be equal to the probability of an edge to start from a partition multiplied by the probability of an edge to end in the same partition. This is calculated to be $a_i \times a_i$. Besides, the real fraction of the edges in partition i is equal to the e_{ii}. Hence, the real fraction of edges and the expected fraction of edges in a random network is compared and the sum of this difference for each partitions is calculated as the network modularity. The community structure that has a modularity value within the interval of 0.3 and 0.7 is supposed to be a good one.

A community definition related with web is also included (Flake et al, 2000). They define a community to be a set of web pages that have more links to members of the community than to non-members. The web communities are identified according to this definition. Here, community membership is a function of both a web page's outbound hyperlinks as well as other hyperlinks on the web. So, these communities are natural in the sense that they are collectively organized by independently authored pages.

The community definition is measured in two ways by Radicchi et al. (Radicchi *et al.*, 2004). In a strong community, each node has more connections within the community than with the rest of the graph. This definition is hard to be satisfied and denotes a strong community structure. Another community definition is considered to be in a weak sense. In a weak community the sum of all degrees within a sub graph is larger than the sum of all degrees toward the rest of the network.

Although there are many studies in complex networks, there is no common community definition. The current definitions are divided into two main categories (Danon *et al.*, 2005). The first category is called self referring definitions. This definition type is related with basic community definition 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 in both directions. The shortest path between all the nodes in a clique is unity. The term n-clique is used if the length of this shortest path may be n. The community is defined only in reference to itself by the means of n. Another category of community definitions is called comparative definitions. In this type of definitions, the number of internal links is compared with the number of external links. This comparison is widely used by the algorithms assuming the notion that the edges of a community are denser than the edges to the outside of the community. This type of definitions is similar with the community definition of Radicchi. Although, the self referring definitions are useful for characterizing communities which are already known, they are not useful for identifying communities. In addition, finding cliques are costly. However, comparative definitions are useful for both finding and evaluating communities in complex networks.

2.3. Community Detection

There are many benefits of detecting communities. The nodes included in the same community are assumed to have similar properties. Using communities these similarities and the network structures may be explored. The most common instance about the usage of community detection is the World Wide Web. There are millions of web pages on the web and it is hard to determine which websites are common in subject and contains similar materials. The information about these clusters is valuable and may be used for different areas such as advertising, marketing etc. Moreover, communities may be used in neural or metabolic networks to divide the network into functional groups. This grouping may be used for uncovering the operations and the transactions in the network.

Community detection is a difficult problem. A similar problem is the Graph Bipartitioning Problem (GBP) which is NP complete (Danon *et al*., 2005). In GBP, graph is partitioned to two sets in such a way that the sizes are equal and interconnection of the sets is minimal. Community detection differs from GBP in the number of sets which can be more than two and the equal size condition is not required.

There are many difficulties while detecting communities in a network. First of all, the community has no absolute definition and some community definitions are given in section 2.1.4. Hence, it should be firstly determined that what the community means. According to this definition, the metric that is used for community detection may be selected. Besides, detecting communities in huge networks is another problem. The algorithms providing accurate results have higher computational costs. So, not only the accurate results are needed in community detection but also acceptable time complexities. Today, the fastest algorithm runs in linear time (Wu and Huberman, 2004) but this algorithm assumes that all communities are of similar sizes and needs some knowledge about number of communities.

2.4. Agents

2.4.1. Agent Definition

In computer science, the term agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its users (WIK). Another definition of an agent that is accepted by many researchers is the Shoham's; an agent is a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents or processes (Shoham, 1993). The agents are defined with their behaviors instead of the functions and attributes of other software entities. The agent concept has several different definitions but the most common properties of agents are listed below;

- Persistence: The agent code is not invoked on demand but it continuously works and decides when to perform some specific tasks.
- Autonomy: Agents do not need any human interaction for task selection, goal-directed behaviour or making decisions.
- Social Ability: The agents can communicate with other agents or some other components and they are able to coordinate.
- Reactivity: Agents are able to perceive some things in their environment and they are able to react them according to the context
- Mobility: Agents are able to migrate from one host environment to another.
- Adaptivity: Agents are able to learn and perform better with experience.

The most crucial abilities of a software agent are adaptation and learning. The adaptation means perceiving some changes in its environment and reacting to these changes by reconfiguring itself. By sensing and adapting, an agent gets more convenient to the conditions. The learning ability of an agent is gained through the trial-and-error process. While it's working period, an agent faces with failures or successes as a respond to its behaviours. After each response, it analyses the behaviours and make generalizations. After some working, the agent learns more and the number of successes gets more.

To define the agents accurately, it is needed to explain what the agents are not. First, an agent is different from programs because of its reaction to the environment, autonomy, goal-orientation and the persistence. Second, an agent is distinguished from an object because agents are more autonomous than objects and agents have flexible behaviors. Lastly, the agents are very different from expert systems although they are confused. The expert systems are not as smart and robust as the agents are. Moreover, the agents are able to messaging and they have social ability unlike expert systems. In addition, expert systems are not coupled to their environment and they are not designed to react to the environment.

2.4.2. Java Agent Development Framework (JADE)

i. JADE Overview

Java Agent Development Framework (JADE) is an agent framework implemented in Java and may be used only through Java (JAD). It handles the working and organization of multi-agent systems on different machines with different operating systems. The agents communicate in an efficient way in which queue of Agent Communication Language (ACL) messages are created and used by JADE. The features which are provided by the JADE while implementing multi-agent systems may be listed;

- A runtime environment in which the agents can survive and communicate through. An agent platform includes Agent Management System (AMS), the Directory Facilitator (DF) and the Agent Communication Channel (ACC).
- A library which includes many functions to be used for developing multi-agent systems.
- A graphical user interface and debugging tools to be used for monitoring, logging and managing agents.
- Distributed agent platform. Agents are java threads and each agent may run on different hosts.

- Provides parallel, concurrent and multiple working agents through behaviour activities.
- The peer-to-peer messaging between agents by ACL messaging framework.
- The registration of the agents to the AMS that is handled when the agents are started.
- The efficient naming service. The agents have a Globally unique identifier (GUID) form the platform automatically.

JADE is an open standard. It is an open source project generated by Telecom Italia LABoratory (TILAB), currently governed by an international board and used by several R&D projects. JADE allows interoperability through some predefined standards called Foundation for Intelligent Physical Agents (FIPA). It aims to encourage the developing of the agent-based applications. The originating point of the FIPA is providing the interoperability by serving internationally agreed specifications. This is handled by the collaboration of the companies, universities all over the world. The whole materials of the FIPA are public available (FIP).

ii. Containers & Platforms

A JADE runtime environment that contains many agents is called JADE Container. The active containers which run together are registered to a main container to form a Platform. That is, each Main Container means a different Platform. In a JADE running environment, each agent should have a unique name and they can communicate even that they are in the different platforms or containers.

iii. Agent Management System (AMS) & Directory Facilitator (DF)

The user defined agents are created by the developers and directed by giving behaviours. Besides, there are two types of special agents within the main container and they are started by the starting of the main agent. The aim of these agents is serving normal agents.

The AMS is the agent that is responsible for the control of the agent platform. It provides two different tasks; first, it gives unique names to the agents and guarantees that each name is given to one agent. Second, an agent may be created or killed by requesting it from AMS. The AMS have two services called white-page and life-cycle services which hold the Agent Identifiers (AID) and agent states respectively.

The DF is the agent that serves yellow-page service. Using yellow-page service, an agent can find any other agents. By using DF agent, agents can publish their services, find the services they desire and use them. This service is crucial in a multi-agent system for interoperability.

iv. The Agent Class

The agent is defined in various ways in different sources depending on their functions in the applications. According to the JADE documentation, “an agent is an autonomous and independent process that has an identity, possibly persistent, and that requires communication (e.g. collaboration or competition) with other agents in order to fulfil its tasks” (JAD).

The user defined agents are represented by the Agent class. To a programmer, an agent is not more than a Java class that extends this Agent class. That is, an agent is composed of the basic features of Agent class and some additional customized properties. An agent contains many tasks working concurrently and each task is called “Behaviour”. Agents serves by behaviours and each function means a different behaviour.

a. Agent Identifiers

All agents are identified by an “agent identifier” that is extended from jade.core.AID class and it is taken by getAID( ) method. An AID object includes a unique name plus an address and has a form of <nickname>@<platform-name>. The agent called Agent1 running on a platform called Platform1 has an agent identifier of “Agent1@Platform1”.

The platform name is only used when the agents needs to communicate with agents from different platforms.

b. Running and Terminating Agents

An agent is generated in a series of operations in some order; first, the agent constructor is executed. Then, the agent is given an agent identifier and registered with the AMS. After registration, the agent is passed to the ACTIVE state. Finally, the setup( ) is executed to implement main activities of the agent. In the setup phase, the behaviours of the agent are taken from queue and operated in an order.

v. The Behaviour Class

The behaviours are the tasks in which, the main activities of the agents are implemented. Each behaviour extends the jade.core.behaviours.Behaviour class and represents the separates tasks of an agent. In order to make an agent to do something, a behaviour is added to agent by calling the "addBehaviour( )" method of the agent class from setup or any other behaviour. Each behaviour has two main methods called action( ) and done( ). In the action method, the tasks to be run in the behaviour are included and functions in the action phase are operated when the behaviour is called. The done( ) method is used for determining whether or not a behaviour is completed or not.

a. Behaviour scheduling, execution

It is crucial to schedule the behaviours and the tasks in them to make them execute properly. The behaviours can work concurrently in an agent. The important point here is that, the scheduling of these behaviours is not pre-emptive but cooperative. That is, when an agent is executed the action method is called and it runs until the action method is returned. The main execution scenario of an agent is given in figure 2.2;

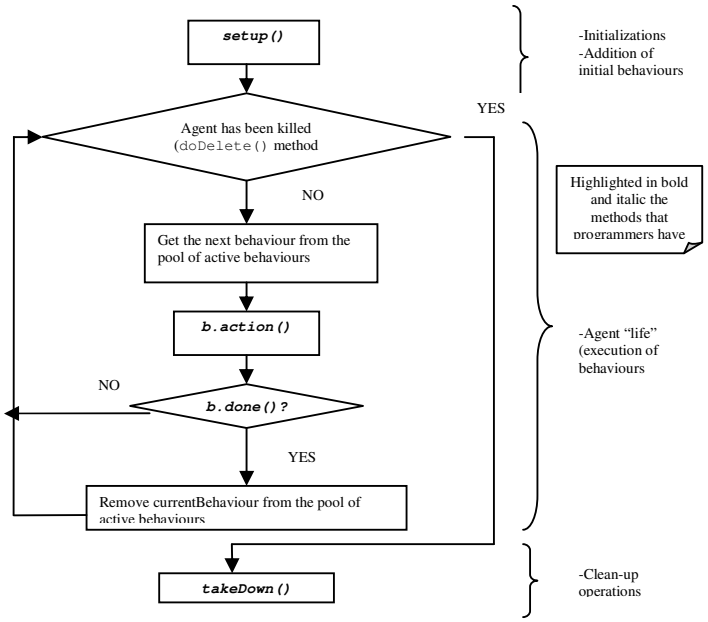


Figure 2.2. Main execution scenario of an agent (JAD)

b. One-shot, cyclic and generic behaviours

- One-shot Behaviours : The action( ) method executes only once and the behaviour is immediately completed.

- Cyclic Behaviours: The action( ) method executes the same operation each time it is called and the behaviour is never completed.

- Generic Behaviours: These behaviours holds some status and execute action( ) method depending on these status values. The operation is performed until a condition is satisfied.

- Waker and Ticker Behaviours : Using "TickerBehaviour", required behaviours are repeated in desired periods on tick events. Besides, by "WakerBehaviour" the behaviour is terminated on desired time.

- Complex Behaviours : The simple agents are used together to form a complex behaviour. Some instances of these behaviours are FSMBehaviour, ParallelBehaviour and SequentialBehaviour etc.

vi. The Agent Communication Language (ACL) Message Class

The agent communication is one of the most important features which JADE provides to programmers. All agents can communicate with other agents to operate together. The JADE agent communication is based on the "Asynchronously Message Passing". Each agent has a mailbox that messages are exchanged through. When an agent wants to send a message, an ACLMessage class is created first. The attributes are filled with available values and the send( ) method of the agent class is called. The receiver agent is notified when the message is received, it calls receive( ) method to retrieve the content of the message and the operation is completed. The JADE asynchronous message passing concept is given in figure 2.3;

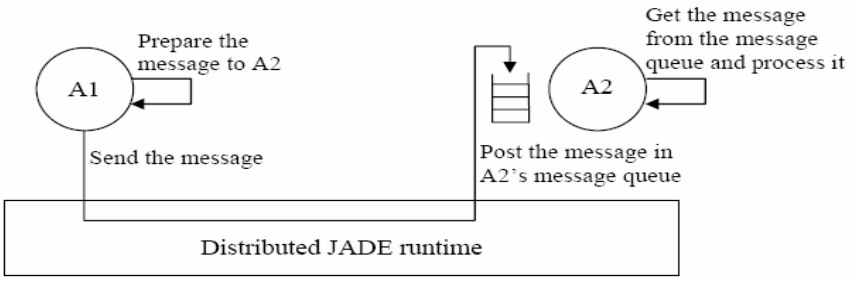


Figure 2.3. JADE asynchronous message passing concept (JAD)

a. The Language

The format of the messages exchanged by the agents is specified by the ACL defined by the FIPA. This definition provides the interoperability between the agents. The agent message format is composed of many attributes;

- The *sender* of the message
- The list of *receivers*

- The *performative* defines the sending objective of the agent. For instance, performative is INFORM, if the sender agent is willing to make the receiver agent know something. The other performatives are REQUEST, QUERY_IF, PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL etc.
- The *content* is the information included in the message.
- The content *language* is used for defining the syntax of the content. It ensures both sides to know content syntax.
- The *ontology* that describes the set of symbols used in the message content. The ontology may be thought as structures or templates to be sent.
- The *protocol* is the protocol that message belongs to.
- The *reply-to* feature holds the recipient of the message reply

3. COMMUNITY DETECTION IN COMPLEX NETWORKS

Detecting communities is not a straight forward process because of different characteristics of the complex networks. There have been many algorithms proposed so far to detect communities. They all use different approaches to provide community structure. These algorithms trying to find the community structure in complex networks are given in details below;

3.1. Girvan-Newman Algorithm

The Girvan-Newman algorithm is proposed by the M.E.J. Newman and M. Girvan in 2004 and it is based on the divisive methods that begin with the whole network and proceed to divide it into smaller clusters (Newman and Girvan, 2004). This process can be represented by a dendrogram. A metric called "betweenness" is used in the algorithm. Unlike other divisive methods, the edges with highest betweenness are found instead of the edges connecting least similar pairs. An edge is considered to have high betweenness if the edge connects high number of vertices. The "betweenness" term is first proposed by the Freeman in 1997 (Freeman, 1977). The algorithm defines three types of different betweenness metrics.

The algorithm is developed for the networks with single type of vertex, single type of undirected, unweighted edges but it can be generalized to include more complicated networks. Mainly, the algorithm has two important distinguishing features compared to other community detection algorithms;

The first feature distinguishing the algorithm is that it is not an agglomerative algorithm but a divisive one. There are also other divisive algorithms but the algorithm differs from them by removing edges with higher betweenness instead of removing edges with lower similarities. This is the distinctive feature of the algorithm.

There are many ways to measure the betweenness but the main idea is common. Given a set of paths, the number of paths that passing through an edge between two vertices is used to identify the betweenness of this edge. There are some measures based on this idea;

- The simplest example to define the betweenness is based on the shortest (geodesic) path. Shortest paths for every pair of every vertex are determined. Every edge is weighted by the number of shortest paths passing through it. This way of measure is proposed by Anthonisse (Anthonisse, 1971) for the first time but the word "edge betweenness" of the Freeman is used because measure of the Anthonisse is the generalized case of the Freeman's. This simplest way of measuring edge betweenness is called "shortest-path betweenness".

- Using shortest-path-betweenness may provide good results but not only the shortest paths are used in a network but also possible longer ones. For this reason, the random walks are used for finding edge betweenness and this measure is called "random-walk betweenness". The random walks from each edge to the other edges are simulated and the number of transitions through edges is calculated. The matrix methods are used for finding results easily.

- The last measure of calculating betweenness is based on the elementary circuit theory. The edges may be considered to be unit resistances, a vertex is a unit current source and another vertex is a unit current sink. The current among these two vertices will flow along shortest paths, but some will flow along longer ones inversely proportional to their length. The magnitude of the current flow is used as the betweenness value between each vertex pair. This measure is called "current-flow" betweenness.

After calculating edge betweenness according to one of these measures, the edge with the highest betweenness is removed because this edge is thought to be between different communities. Then, the betweenness values are calculated again for each edge and edge removal process is repeated.

The second feature distinguishing the algorithm is the way of adding the recalculation step. In a standard divisive clustering algorithm based on edge betweenness, the edge betweenness values for each edge are calculated and then the edges are removed with descending betweenness order. However, after removing an edge, the previously calculated betweenness values of the edges do not show the true values anymore because the structure of the network is changed. This is an undesired value and may cause some wrong results unless it is fixed. The algorithm solves this problem by recalculating edge betweenness values after each edge removal. By doing so, the betweenness values are always kept accurate and consistent. Although this solution increases computational effort, it is needed for correct results.

The algorithm works well for simple, intuitive networks and networks with known community structures. Its results are satisfactory for not only the programmatically created networks but also the real data networks. The biggest disadvantage of the algorithm is that it needs high computational effort. Hence, it is not efficient for the huge networks because of the complexity of $O(n^3)$.

3.2. Community structure in very large networks

The fast algorithm for detecting very large networks (Clauset *et al.*, 2004) is a hierarchical agglomerative method that uses greedy optimization. The hierarchical agglomerative method is a community detection way that aims to join vertices into successively larger clusters, using a measure of similarity. Using these optimizations, the algorithm runs faster and may be used in huge networks.

As in the case of GN, the modularity Q is defined to be the measure of division of a network. A division with higher modularity is assumed to be good if there is higher number of edges within communities after division operation. The modularity values higher than 0.3 are considered to have acceptable community structure. To find the division with the highest modularity value, the candidates should be searched. While searching some optimization techniques are used. This algorithm is inspired by Newman's

(Newman, 2004) and adds some optimizations. The fast algorithm of Newman starts with the communities with exactly one vertex. These nodes are repeatedly joined in a way that the joined nodes results with the highest increase in the modularity. This operation continues until only one community is obtained and it is represented as a tree whose leaves are the vertices of the network and the inner nodes denotes the join operations. At this point a *dendrogram* is obtained and communities may be detected at different levels of this dendrogram.

In the simple implementation of the idea that is issued in the Newman's includes the matrix operations using all row column pairs and it is not efficient in the sparse matrixes which are the primary interests. The algorithm proposed by the Clauset et al. avoids these unneeded operations and increase speed and memory efficiency.

The algorithm may be summarized as follows; first the pairs with maximum modularity changes are found and they are joined. The modularity values are updated and this operation is repeated until one community remains. Using the data structures, the update operations are made quickly.

The algorithm provides similar results as Newman's (Newman, 2004) algorithm does, but faster than Newman's. Hence, the algorithm may be used in very large networks. The data from www.amazon.com are analyzed in August 2003 as an instance to huge networks with 409 687 nodes.

The proposed algorithm greedily optimizes the modularity value and detects communities using changes in modularity. The time complexity of the algorithm is $O(md \log n)$ for a network with n vertices, m edges and d depth of dendrogram. This is significantly faster than the similar algorithms and gives us the possibility of detecting communities in very large networks.

3.3. Radicchi's Algorithm

(Radicchi *et al.*, 2004) proposes a different algorithm to resolve two main issues;

- While using dendrogram, the additional information about network is needed to decide whether the community structure is reliable or not. Because, some edges may be crucial for community structure and these should be known. The Wilkinson, Huberman (Wilkinson and Huberman, 2004) and the Girvan, Newman (Girvan and Newman, 2002) addresses this issue. The GN algorithm proposes the modularity that is used in many algorithms to define the strength of the community structure but it is not assumed to be exactly an objective discrimination.
- The "edge-betweenness" that is used in Girvan, Newman requires big amount of computational effort and this problem is also addresses in this algorithm.

The algorithm aims to decide on real community structure. First, two community definitions are discussed and then an alternative algorithm is proposed that gives similar results with GN algorithm but performs faster.

The solution of the first problem is provided by finding a definition to decide whether subgraphs are communities or not. The two formulas defining this community structure are considered. Firstly, the degree k_i of a node is a crucial quantity and it is considered to define a formula. The community definition of a subgraph *V'* in strong sense is given below;

$$k_i^{in}(V') > k_i^{out} \;, \forall\, i \in V' \tag{3.1}$$

The community definition in weak sense is given below;

$$\sum_{i \in V} k_i^{in}(V') > \sum_{i \in V} k_i^{out}(V') \tag{3.2}$$

where V' is a subgraph to be tested for being a community, $k_i^{in}(V')$ is the number of connections of node i within the community and $k_i^{out}(V')$ is the number of connections of node i to the rest of the graph. Here we can say that, it is important for every node to have more edges within the community than the edges into other communities.

Secondly, it is claimed that there should be at least two communities after edge removal operation. Because, if there are one big community and another small one, the smaller one can not fulfill the definition of the community and the cut is assumed to be unsuccessful. To address this issue, the self-contained version of the GN algorithm is improved. First, a community definition is selected. The edge betweenness values of all edges are calculated and the edges with higher ones are removed. If the cut splits the graph it is checked whether at least two sub graphs fulfill the community definition. If so, the cut operation is performed. This operation is repeated until no edges are left in the network.

As an innovation, Radicchi et al. proposes an algorithm that has no computational effort as much as the GN has while computing edge betweenness for all edges. The Radicchi algorithm addresses this issue by using a divisive algorithm that computes only local values and performs faster. The "edge-clustering coefficient" is defined instead of the node-clustering coefficient as the fraction of the number of triangles including the given edge to the number of triangles that may include the given edge. For the edge connecting node i to the node j, the edge-clustering coefficient is defined as;

$$C_{i,j}^{(3)} = \frac{z_{i,j}^{(3)} + 1}{\min\left[(k_i - 1), (k_j - 1)\right]} \tag{3.3}$$

where $z_{i,j}^{(3)}$ is the number of triangles including that edge, $\min[(k_i - 1), (k_j - 1)]$ is the maximum number of triangles which may include that edge and k_i is the degree of vertex i. The main idea behind this formula is that the edges connecting different communities are included in fewer numbers of triangles or not included in any triangles. So, bridge edges which connecting communities have lower $C_{i,j}^{(3)}$ values while edges in clusters have higher $C_{i,j}^{(3)}$ values because an edge in a cluster is included in many triangles.

3.4. Community Detection Using Extremal Optimization

The modularity value Q that is proposed by the Newman and Girvan (Newman and Girvan, 2004) is widely used in community detection methods. It is assumed that larger Q means more community structure the network has. However, it seems to be an NP-hard problem to optimize the modularity value because there are several variations of node partitions (Duch and Arenas, 2005). To optimize community with smaller search scope, the genetic algorithms and simulated annealing have been used. In this algorithm (Duch and Arenas, 2005), a divisive algorithm using a heuristic search on the Extremal Optimization (EO) is used. This EO algorithm is firstly proposed by the Boettcher and Percus (Boettcher and Percus, 2001) and based on optimizing a global variable by developing local variables. The EO algorithm performs better and more efficient than Simulated Annealing and Genetic Algorithms with acceptable accuracy.

$$Q = \sum_{r} \left(e_{rr} - a_{r}^{2} \right) \tag{3.4}$$

The algorithm aims to optimize Q value. Here, the local variables in the EO should be associated with the addition of each node into the equation above. A new formula appears at this point.

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

Where k_i is the degree of the node i, $a_{r(i)}$ is the fraction of edges that have at least one vertex in the community and $\kappa_{r(i)}$ is the number of edges from node i to the nodes included in the community r that also includes the node i. Thus, it can be said that, $Q = \frac{1}{2L} \sum_{i} q_i$ where L denotes the number of edges in the network. By tuning the local variable q_i using the degree of the node, we can provide the contribution of each node into the modularity definition. Hence, the $\frac{q_i}{k_i}$ is used as the characteristic “fitness” of the node i in the EO

algorithm. After these definitions, the heuristic search to optimize the modularity value can be summarized as follows;

- First, the graph is divided into two partitions both having the same number of nodes. In these partitions, connected components are assumed to be initial communities.
- By each time step, the nodes with the lowest fitness values are moved into another partition and the graph organizes itself. After each movement, the fitness values are recalculated.
- This iteration continues until the modularity value reaches a maximum. Then, the edges between both partitions are removed and the same operation is made for each connected components until modularity value can not be improved.

In the EO algorithm, the node with the lowest fitness value is selected and moved into other partition. However, there are some disadvantages about this choice. The result of the algorithm highly depends on the initial partitioning and related fitness value. Besides, the local maximas may prevent finding the optimum value. To solve these issues, the nodes are ranked according to the fitness values and a probabilistic distribution is used instead of selecting the node with the worst fitness value. Thus, the effects of the initial choice are decreased and the selecting the local maximas are avoided.

The EO algorithm provides maximum modularity values which are more than the modularity values of other known algorithms for different graphs. Hence, the algorithm explores accurate community structures. While the EO algorithm provides better results, it has a $O(n^2 \ln(n))$ time-complexity that is not efficient for large networks.

3.5. Other Methods

In addition to these main algorithms on community detection, there are also some other methods. These methods will be used to be compared with our algorithm in section 4.3.

- Wu & Huberman (WH): The algorithm (Wu and Huberman, 2004) is based on the notions of voltage drops across networks and avoids edge cutting. It focuses on communities not on their hierarchical structures. Algorithm does not need to find all big communities before finding the small ones. That causes the algorithm to have a significant speed (O(n)). It can be used to locate the community to which one specific node belongs. However, to detect the communities, successive iterations of the method is required. Other drawbacks of the algorithm are the complex matrix calculations and the need for prior information if a correct identification is desired.
- Newman (NF): The fast algorithm of Newman (Newman, 2004) starts with the communities with exactly one vertex. These nodes are repeatedly joined in a way that the joined nodes results with the highest increase in the modularity. This operation continues until only one community is obtained and it is represented as a *dendrogram* and communities may be detected at different levels of this dendrogram.
- Donetti & Munoz (DM/DMN): This algorithm (Donetti and Munoz, 2004) combines spectral techniques, cluster analysis, and modularity concept. The performance of the algorithm is improved by using normalized Laplacian matrix to project network nodes into eigenvector space. The hierarchical clustering techniques are used to generate dendrogram and the network modularity is tried to be maximized. This is similar with the algorithm of Capocci et. al, but this algorithm do not take into account weights unlike Capocci algorithm. The usage of network modularity and the hierarchical clustering are the other differences of this algorithm.
- Capocci et al. (CSCC): The algorithm (Capocci et al., 2004) is based on the spectral methods and takes into account weights and link orientation. Spectral methods are the ways of analyzing adjacency matrix using Laplacian matrices, eigenvectors etc. Unlike other spectral methods, it is not based on iterative bisection. It combines spectral analysis with correlation measurements.
- Reichardt & Bornholdt (RB): The algorithm (Reichardt and Bornholdt, 2004) aims to find overlapping (fuzzy) communities. It is based on the q-state Potts model. An approximation is made through Monte Carlo optimization. No prior

knowledge of the number of communities is needed. The algorithm is nondeterministic and non-hierarchical.

- Fortunato et al. (FLM): A new centrality measure called information centrality is introduced in the algorithm (Fortunato et al., 2004). It is based on the efficient propagation of information over the network. The classification is realized using centrality measure. Then, edges are removed and centralities of edges are recalculated. It is the variation of GN algorithm that changes betweenness with information centrality but the performance of this algorithm is poor ($O(n^4)$).
- Bagrow & Bollt (BB): The algorithm (Bagrow & Bollt, 2005) is a local one that detects the communities without requiring information about the entire network. It is suitable especially if the information about only one community is required. Although the method is simple and flexible, its complexity is high ($O(n^3)$).
- Zhou & Lipowsky (EM): The algorithm (Zhou and Lipowsky, 2004) is based on the biased network Brownian motion. A measure for the degree of proximity among vertices is described. This proximity integrates both the local and global structural knowledge of a network. Its advantages are that it can be applicable to large networks and weighted networks. Yet, it includes matrix inversion and has a high complexity of $O(n^3)$
- Young et al : This is an agent-based approach that agents travel in the network randomly once and keep a list of travelled nodes. After this exploring phase, the voting phase is performed to determine the communities. The algorithm has also an optional clean-up phase. This operation is repeated until a desired number of communities are reached. The voting cut-off and the number of desired communities are drawbacks of the algorithm.

.

4. COMMUNITY DETECTION USING AGENTS

4.1. Use of Agents in Community Detection

As mentioned in the previous chapter, community detection is performed in many different approaches. In addition to these algorithms, we will propose an algorithm that makes use of agents. First, the agents are used for exploring the network and revealing its structure. Then, the modularity value Q is used as the measure for community structures.

Although, the software agents are rarely used in community detection, they are appropriate for network analysis issues. The main reason behind the idea of using agents in the community detection is their collaboration. The software agents function together and work in a coordinative way. That is, lots of agents are used in the network and the global structure of a network may be revealed. Besides, the easy messaging mechanism of the agents may be used while detecting communities in a network.

The community concept in a network has no absolute definition and has no strict boundaries. Our algorithm has also some stochastic issues and we claim that the agents are appropriate for using in stochastic operations. Lastly, agents are also convenient for larger networks. Using the asynchronously working agents, the consumed time and effort may be reduced. Different than other approaches, this method allows distributed computing. That is, the algorithm is able to work on many computers in parallel. This is a big advantage considering the large size of networks.

4.2. Previous Usage of Agents in Community Detection

One of a few works on using agents in community detection is the paper proposed by Young et al (Young *et al.*, 2005). Our algorithm is mainly inspired by this paper and develops the idea that is proposed there. In this paper, an agent-based approach that many agents are initially located in the network randomly is presented. These agents traverse

randomly in the graph once and keep a list of the nodes. After this exploring phase, the voting phase is performed to determine the communities. The main idea is that, the agent will spend the most of its random walk within the community because there are more edges to stay within the community than the edges to leave the community. While performing voting phase, the number of agents which have traversed both the nodes u and the node v is calculated. This number is called A. The number of ants that traversed one of u or v is also calculated and this value is called B. Then, the nodes u and v are merged if $\frac{A}{B} > c$. Here, c denotes the voting cut-off that is used to determine whether two nodes belong to the same community or not. The algorithm has also an optional clean-up phase. A found community is merged with another if they have more shared edges than the community has with others. This operation is repeated until a desired number of communities are reached. The voting cut-off and the number of desired communities are two crucial parameters of the algorithm. The need for this prior information is a drawback of the algorithm. In addition, the agents may explore only some parts of the graph and there may be some unvisited parts in the graph, because they are located randomly in the graph. Lastly, the effect of randomness may cause undesired results and it should be decreased using some biases.

In our algorithm, there are many improvements to the previously mentioned algorithm. First of all, our agents are not randomly located nor do not traverse randomly except the first generation. That is, the agents we used are more intelligent than the agents used in this algorithm. Second, there is only one generation of agents in the mentioned algorithm but there are many generations in our work and all of the moves from agents of a generation are collected and used for next generations. Moreover, our algorithm provides the global knowledge about all parts of the graph unlike the paper does randomly. Lastly, our algorithm removes the need for some parameters such as voting cut-off or the number of desired communities. In this chapter, we propose a new community detection algorithm based on using agents for revealing network structure.

4.3. Smart Agents in Community Detection (SACD) Algorithm

In this part, we will introduce a new algorithm in finding community structure using smart agents in complex networks. In the algorithm, the agents of a certain number of generations are used and they collect more information about the network structure by travelling in the network in each generation. We call them “smart agents” because agents of following generations make moves using the information collected by the former generation agents. There are also some improvements in the implementation of the algorithm to increase the accuracy of the results. Besides, our algorithm is an indeterminist one. That is, the same input may not give the same results for different runs of algorithm.

4.3.1. Fundamentals of SACD

The Smart Agents in Community Detection (SACD) algorithm is based on using agents in detecting communities in complex networks. There are two main phases in the algorithm. The first phase is the collective exploration and it is handled by the smart agents. In this phase, agents traverse in the graph in a biased way and collect information about current network structure. In the second phase, this collected information about network structure is analyzed and the community structure of the network is found using the modularity.

The SACD algorithm needs no a priori knowledge unlike many community detection algorithms. Other algorithm may need the number of communities, the size of each community or some threshold values. However, the SACD algorithm works without any structure information about network and learns the data about network structure in the exploration phase using smart agents. The algorithm works for networks with undirected edges. That is, edges should be used instead of arcs.

There are many algorithms like (Girvan and Newman, 2002) which provide a dendrogram at the end of algorithm and use this dendrogram to determine the communities. Such algorithms also need a threshold value to decide on communities and

different thresholds produces different community structures as the result. But in SACD algorithm, the user needs no further process after running the algorithm. Because, the definite community structure is provided and there is no need for the threshold value.

The main idea behind using agents in community detection is the tendency of agents to stay in the community rather than leaving the community. A community in the network is defined to be a densely connected group of vertices. Hence, there are more edges with both starting and ending vertices within community than the edges with vertices outside the community. So, it is more probable for an agent to stay in the community because there are more ways for an agent to stay within the community than there are to leave it. To clarify this idea, consider the gas molecules in the room A in figure 4.1. They move around the room. It is probable for the molecules to stay in the room rather than to enter to the room B. Here, the communities in a network may be thought as the rooms and the connections between rooms are similar to the bridge areas in the network.

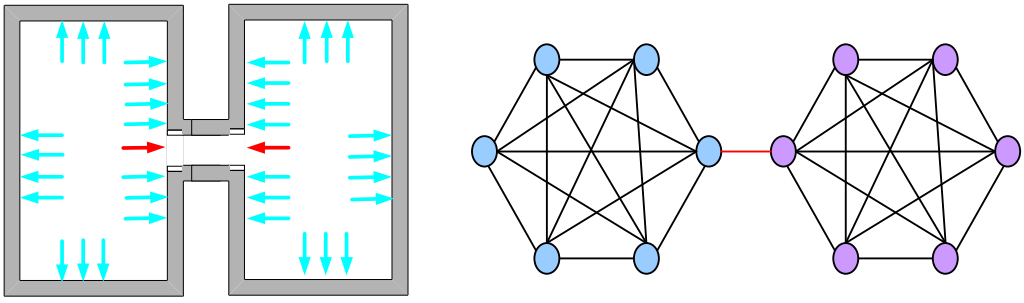


Figure 4.1. Main idea behind algorithm

4.3.2. Exploration Phase

The exploration phase is the first part of the algorithm and the network structure is explored in this part. The information about the community structure of the network is obtained by the moves of agents in this phase. The exploration phase is the most crucial and characterizing part of the algorithm. Hence, the exploration phase should be handled

accurately to make analysis phase work well. The information about network structure which will be used in the analysis phase is kept in a matrix structure after this phase.

The exploration phase is handled cooperatively by the master agent and the slave agents. These agents use messages to provide efficient data collection. The details of the operations performed in the master agent and the slave agents will be mentioned in the next subsections;

i. Master Agent Program

Among two types of agents, there may be several instances of slave agents while there is only one instance of the master (coordinator) agent. The master agent is used for administrative purposes in the algorithm. A lot of agents with the type of "slave agents" collects data and send the data to the master agent. The master agent collects these feedbacks from slave agents and uses them to update some variables in the algorithm. The learning of slave agents is realized by these update operations.

The first step in the SACD algorithm is getting the network data. The network data are read from a file and kept in an adjacency matrix to be used in the later stages of the algorithm. Then, the neighbours of all nodes are determined and stored. The agents will use this neighbourhood information to make moves towards neighbours from a vertex.

The master agent describes itself and registers to the yellow pages to be able to communicate with slave agents. Then, a cyclic behaviour is added to the master agent to make it work iteratively like a service. The master agent should work continuously for listening to the slave agents and meet their needs.

After all agents starts to operate, the slave agents collect information about network structure by travelling in the network. Each slave agent has a memory of a certain size and this memory holds the travelled nodes by the agent. The slave agents send these feedbacks (visited nodes) to the master agent. This feedback information is carried in a message. This message holds the node numbers which are traversed by an agent. The message content is

parsed and the traversed nodes are obtained by the master agent. At this point, the *weight* matrix should be mentioned. The *weight* matrix is an n × n matrix that initially consists of zeros. When a feedback message is received by the master agent, the weight matrix is updated like figure 4.2;

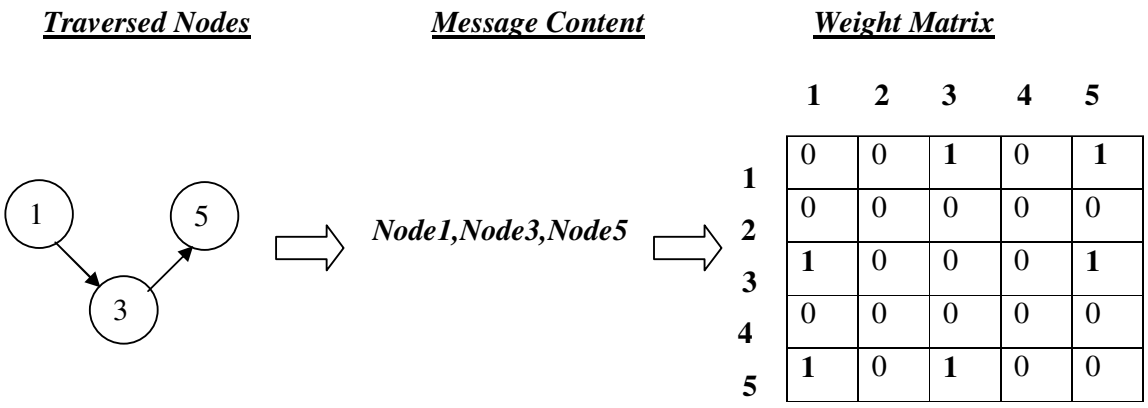


Figure 4.2. The update of weight matrix for an agent after a feedback

The master agent updates the weight matrix using the feedbacks (visited nodes) from slave agents. The weights between the nodes which are traversed by the same agent are incremented by 1, because the nodes which are traversed by the same agent are assumed to be *probably near*. Incrementing the weight between two probably near nodes in the weight matrix, increase the probability of these two nodes to be in the same community.

After updating the weight matrix, the slave agents of next generation start traversing and make their moves according to the new weight matrix. Before making a move, the weights of the edges with neighbours are taken from weight matrix. Then, these weights are considered and the move is made proportional to these weights. If an edge has higher weight, then the nodes connected by this edge are said to be probably near. The slave agents tend to move towards probably near neighbours. Therefore, slave agents move in a biased way instead of moving randomly and tend to stay in the community. We will see in the experimental results that the agents explore the network structure more in every

generation by using the weights and the accuracy of the provided community structure increases with the increasing number of generations.

ii. Slave Agents Program

Unlike master agent, there may be several instances of agents of type *slave agents*. These agents' functions are the exploration of the network and sending feedbacks about network structure to the master agent. All of these agents are administrated by the master agent and the synchronization between these agents is provided by the master agent. Moreover, these agents are also initiated and terminated by the master agent.

In first generation, all *slave agents* behave completely random. That is, they are located in the network randomly and make their first traverse operation randomly. In following generations, the global information about the network will be taken from the master agent (using weight matrix) and they will start to make smart moves and capture the network structure more efficiently. The behaviour of first generation agents and next generation agents are shown in Table 4.1.

These agents travel in the graph and collect information about the network. Each agent has its own *memory*. During travelling in the graph, the traversed nodes are stored in the memory. After traversing certain number of nodes, the nodes stored in this memory is sent to the master agent.

In the first generation, the agent makes a certain number moves. For each move, a neighbour of the current node is selected randomly and the agent moves to this node. Each travelled node is also added to the node history in order not to repeat the same nodes and not to enter in a loop. Before each move, it is checked whether the selected node is in the node history or not. If it is in the history, select another node as well as it is possible. This behaviour also and increases the probability of exploring more parts of the network. Besides, this will prevent staying in the area that between the communities (bridge areas). So, the agents tend to enter into a community rather than get stuck in a bridge area.

After first generation agents complete their moves, the travelled nodes should be sent to the master agent to give information about the network structure. These nodes will be evaluated in the master agent and will be used for updating the weight matrix.

In the following generations, the agents are not located randomly in the graph. The most visited nodes (hub nodes) are assigned to be the next starting points. The reason behind assigning the hub nodes as starting points is the better exploration of communities. Starting from these nodes will strengthen the behaviour of staying in the community and formation of the community structure. However, starting all agents from the nodes which have been crossed most may prevent discovering all parts of the graph. Because, if some parts of the graph are not used as starting points, it would be difficult for them to participate in the competition and they will never be visited. Hence, we also assign some least visited nodes as starting points and add them to the competition with other nodes. By assigning these least visited nodes as starting points, we give them a chance to be a hub and we have a possibility of discovering all parts of the graph.

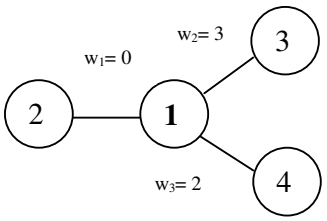
Table 4.1. The behaviours of agents of different generations

Generation	**Start**	**Move**
First generation	Randomly	Randomly
Following generations	From most visited nodes From least visited nodes	Biased (using weight matrix)

After deciding nodes to start, the next generation agents are ready to traverse in the graph in a smart way. While making these moves, the *weight matrix* is considered by the next generation agents. The neighbours of a node with higher weights are found and the more chance is given to these nodes for travelling. The slave agents tend to prefer moving to these nodes. To give the edges with 0 weights a chance to be used, the weight in the formula is incremented by 1. The probability of using an edge for a move is calculated like below;

$$P_E(j) = \frac{(1 + w_j)}{\sum(1 + w_i)} \tag{4.1}$$

where, w_j denotes the weight value of edge between a node and its j^{th} neighbor and $P_E(j)$ denotes the probability of moving to the j^{th} neighbor. A biased move as explained is shown in figure 4.3;



Move	Probability
1 – 2	(1+0)/8 = 1/8
1 – 3	(1+3)/8 = 4/8
1 – 4	(1+2)/8 = 3/8

Figure 4.3. A biased move of an agent

The lastly visited nodes are not traversed again as well as it is possible also in the next generation agents. . After making moves and filling the memory of agents, the agents sends these visited nodes to the master agent. This information will be evaluated in the master agent and will be used by the next generation agents.

4.3.3. Analysis Phase

After the exploration phase is handled by the slave agents, we have the weight matrix that holds the weight values between all vertex pairs. An edge with lower weight is probably a bridge, because the weight between a node and other nodes in the same community is incremented while agent is travelling in the community. It is said that an agent tends to stay in the community and the weight of the inner community edges will be higher. So, the weight of the bridges will be lower. The edges with the lower weight values are removed from the graph to detect communities.

i. Calculating the number of connected components

After removing the edge with the lowest weight from the graph, it is checked whether a new community is formed or not. To check this, an algorithm called "Flood Fill" (FLO) that gives the number of connected components in a graph is used. The basic idea here is finding the node which has not been assigned to a component yet and finding the component that contains it. The algorithm has $n+m$ time complexity where, n is the number of vertices and m is the number of edges. The execution of the "breadth-first flood fill" algorithm is explained in details in Figure 4.4;

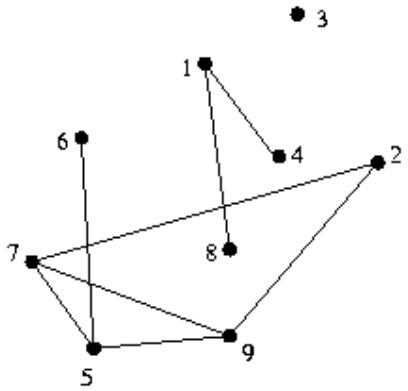


Figure 4.4. An example graph to be analyzed (FLO)

At first, no nodes are assigned to a component. It is started with vertex 1 and the component number of it is set to -2.

Table 4.2. The first phase of Flood-Fill algorithm (FLO)

Node	Component
1	-2

Then, the nodes are checked and it is found that the node 1 is assigned to component -2 and it is assigned to the first component, component 1. It is marked as visited and its neighbour node 4 is assigned to the component -2.

Table 4.3. The second phase of Flood-Fill algorithm (FLO)

Node	Component
1	1
4	-2

At next loop, it is seen that node 4 has been assigned to the component -2, it is assigned to the component 1 and its neighbour node 8 is assigned to the component -2.

Table 4.4. The third phase of Flood-Fill algorithm (FLO)

Node	Component
1	1
4	1
8	-2

After that, node 8 is processed and its component is set to the component 1.

Table 4.5. The fourth phase of Flood-Fill algorithm (FLO)

Node	Component
1	1
4	1
8	1

It is seen that, there is no nodes that have not been assigned yet and the component 1 is complete. The node 2 is found next and its component is assigned to the -2.

Table 4.6. The fifth phase of Flood-Fill algorithm (FLO)

Node	Component
1	1
2	-2

4	1
8	1

The node 2 is the first unvisited node and its component number is set to the new component 2. The neighbours of the node 2 are set to the component -2.

Table 4.7. The sixth phase of Flood-Fill algorithm (FLO)

Node	**Component**
1	1
2	-2
4	1
7	-2
8	1
9	-2

Next, node 7 and node 9 are also processed and assigned to component 2. Then, their neighbour node 5 is processed and assigned to the same community. Other nodes are also processed and the connected components are detected as in Table 4.2.

Table 4.8. The full-fill algorithm progress result for finding connected components (FLO)

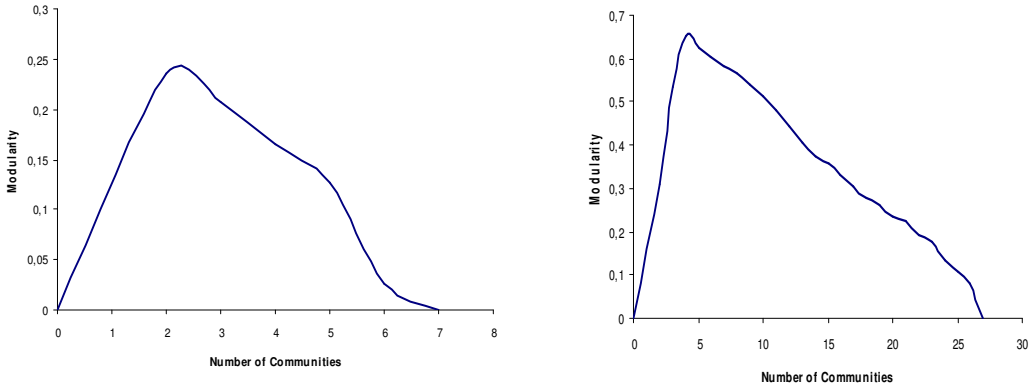
Node	**Component**
1	1
2	2
3	3
4	1
5	2
6	2
7	2
8	1
9	2

After these iterations, we are able to find the number of connected components after each edge removal. Now, we should decide when we stop the edge removal process.

ii. Deciding on the number of communities

After removing an edge from the graph, it is checked whether a new connected component is formed or not. If it is, the number of connected components is changed. Then, we consider the current connected components as the communities and calculate the *network modularity* value related with this community structure as mentioned in chapter 2.2.

Calculating the modularity value for each number of connected components will increase the computational cost, because there may be a large number of vertices in the network and the number of candidate communities (connected components) may be as much as the number of vertices. Hence, edge removal process should be ended at some point. To decide on this point we check the local maximum points in Figure 4.5, including the modularity values for different number of communities.



a) Computer generated network b) Zachary karate club network

Figure 4.5. The modularity values by the increasing number of communities after edge removals. a) The change in the modularity value for computer generated network. b) The change in the modularity value for Zachary karate club network.

Figure 4.5, shows the modularity value during edge removal process. As the edges are removed, the number of candidate communities (connected components) increases. By the increasing number of connected components, the modularity value changes. From the figure 4.5.a, it is clear that there is only one local maximum in the graph for computer generated network (Appendix A). So, we can terminate the edge removal process when we detected the local maximum in the graph. That is, the algorithm is completed when the modularity value reaches the maximum value. It is the same for Zachary karate club network as it is seen from figure 4.5.b. In general, the graph has only one local maximum but there may be the second local maximums for some networks.

4.3.4. Parameters in the SACD algorithm

Due to the fact that our algorithm is a self contained one, there are only a few parameters to be used. However, the usage of these parameters is important and they should be chosen carefully. Although using these parameters with their default values may provide acceptable results, determining different values for each dataset will produce much better results. The choice of these parameters depends on the size of the network to be explored. These parameters may be optimized experimentally.

i. Number of Agents

This parameter specifies the number of slave agents to be used for exploring the network structure. Increasing the number of agent will also increase the possibility of exploring different parts of the network. Hence, using more agents provides more accurate results. However, using a lot of agents in small networks will cause a confusion and complexity. To avoid unnecessary time and computational costs the number of agents should be chosen according to the size of the network. The number of agents for different network sizes is given in figure 5.7.

ii.Size of Agent Memories

In the algorithm, each slave agent has a memory to keep the travelled nodes in the network. The size of these memories may be tuned before the algorithm runs. The small size of memories will prevent the better exploration of the graph and only small parts of the network may be travelled. However, the large size of memories will cause leaving the current community and misidentification of the communities. Hence, this parameter should also be chosen depending on the number of nodes in the network. A small memory size should be used for small networks while a large memory size is needed for the large networks. The suitable memory values for different nodes are given in figure 5.7.

iii. Program Termination

The exploration of the network makes it easy to identify the communities. But, the program should be terminated when it is decided that the network is explored enough. To decide on this issue we define a termination condition for the algorithm. We want any node to be visited at least *n* times by the agents, where n is the number of nodes. Therefore, the network is explored depending on its size.

4.3.5. The Performance of the Algorithm

i.Time complexity

The algorithm has $O(n^2)$ time complexity where n is the number of vertices in the network. The main operations in the algorithm are forming an adjacency matrix, making agents travel in the network, forming a weight matrix using information from agents, finding the number of connected components in the network, calculating network modularity. Calculating network modularity is a simple process and not a time consuming one. The operation that takes more time and determines the time complexity is matrix operations. The matrix is an $n \times n$ matrix where n is the number of nodes in the network. To operate on this matrix, two nested for loops are used. Then, the time complexity will be $O(n^2)$.

The time consumed during the run time of the algorithm may be decreased using agents in a distributed environment. Each agent is located on different machines and the algorithm may be performed in less time period. Using a sparse matrix will also decrease the time complexity of the algorithm.

Our algorithm with $O(n^2)$ complexity provides acceptable complexity that is better than many of the current methods. Moreover, the algorithms have better complexity values than ours can not find the correct community structure as well as our algorithm could or they need some priori information about the network. It is shown in section 4.3.

ii.Accuracy

Our algorithm provides optimum community partitions in complex networks. To test the accuracy of the algorithm, many real world networks with known community structures are used. Minimum of 90% accuracy is obtained for these networks. Accuracy is defined to be the fraction of correctly identified nodes.

To increase the accuracy, the exploration phase of the algorithm is iterated several times and the network structure is learned more at each iteration. The main reason behind higher accuracy is the learning of the network by the agents.

The algorithm is tested with many networks of different types and sizes. The algorithm provided high accuracy values for both the real world networks and the computer generated networks. The accuracy tests are explained in details in experimental results section.

iii.Scalability

The algorithm is tested not only with small networks but also larger networks to measure the scalability. We've tested our network with Reuters network (Ozgur and Bingol, 2004) with 2245 nodes and Web Site network with 1669 nodes. These networks do not have known community structure. So, we could only perform some scalability tests

with these large networks. The time and memory consumed for these networks are included in the section 5.12.

Although the usage of matrix structure to keep the adjacency information is considered to restrict the algorithm with some size of networks, the algorithm works fine for networks with 10 000 nodes. Tests with larger networks may be performed as a future work.

iv.Resource consumption (Memory, CPU)

The first memory allocation in the algorithm is used for getting the network information into the memory. The neighbourhood information is kept in an adjacency matrix. After that, the agents make biased moves among these neighbour nodes. These operations in the adjacency matrix take the biggest part of the memory and consumption. An operation in this two dimensional array takes $O(n^2)$ complexity as it is mentioned. Hence, the memory and CPU consumption depends on the number of nodes in a network.

Another memory consuming operation is finding the number of connected components. This operation requires analyzing all network and deciding on the number of separate components. The flood fill algorithm is implemented to find the number of connected components. Our implementation of flood-fill algorithm is called breadth-first scanning and requires no extra space by some additions to the algorithm. The running time of this algorithm is $O(n \times 2)$ where n is the number of nodes in the network.

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 SACD algorithm never exceeded the 200MB RAM boundary. The memory allocation details in different networks will be given in experiments section. We tested our runs on a PC with Intel Pentium-4 2.4 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.

4.4. Comparing SACD with Other Algorithms

The community detection methods are distinguished by their approach and application. They are useful in wide spread areas of interests and the right method should be selected. However, there is no common metric for measuring their efficiency and it is difficult to compare them. To compare the performance of our algorithm with other methods', we assume the fraction of nodes correctly identified as the common metric. The computer generated network with 128 nodes divided into 4 communities of 32 nodes each is used as the common dataset because the performances of many algorithms with this network is known. In this network, the nodes have a total degree of 16 and the number of edges from each node to other communities is tunable. The fraction of nodes correctly identified is compared for each algorithm according to increasing values of z_{out}. The figure 4.6 shows the accuracy of the algorithms for different values of z_{out}.

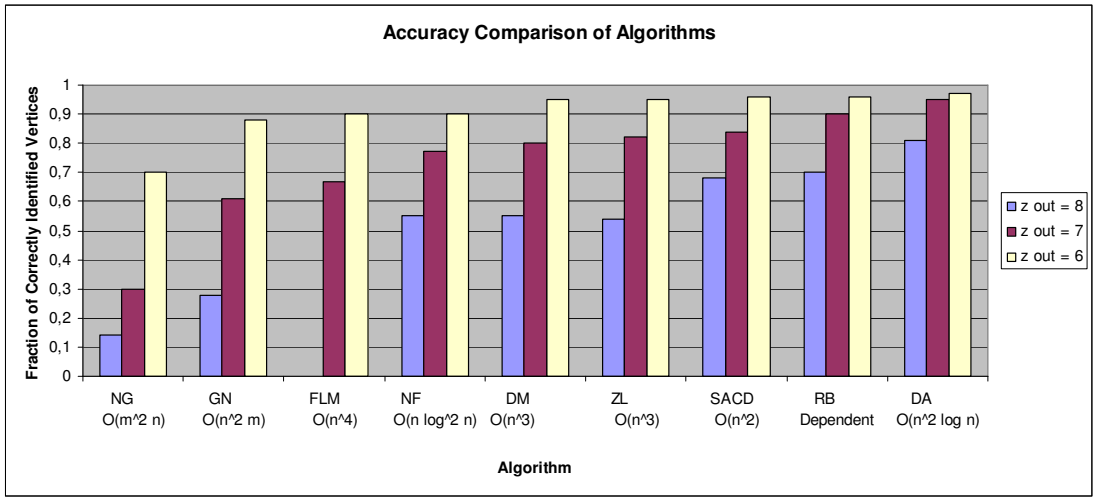


Figure 4.6. The fraction of correctly identified nodes at different values of z_{out} for current algorithms (Danon et al., 2005).

First of all, the methods to be compared with our algorithm are explained briefly. It is seen from the figure 4.6 (Danon *et al*., 2005) that, most of the methods are successful to find the real community structure with the z_{out} value 6. However, with the z_{out} value 8, the accuracy starts to decrease because the network tends to random network. Among all

methods, which are explained in chapter 3, our algorithm performs better than most of the algorithms for different values of z_{out}. There are only two algorithms; RB and DA able to identify more vertices accurately than ours could do. However, these two algorithms have a higher complexity value than our algorithm has ($O(n^2)$). Hence, it can be claimed that, our algorithm provides highest accuracy among the algorithms with the same complexity. There are no algorithms giving better results with the same or less complexity value.

Not only the accuracy is needed to evaluate the wellness of a method, but also the complexity of a method is needed. Usually, there is a trade off between the accuracy and the computational effort. Hence, the choice of the right method depends on the type of application. For instance, in a small network, the speed is not so important and the most accurate method may be selected. But in a huge network, the computational cost is getting more important and the accuracy comes after. To compare the complexity of our algorithm with other methods, the computational performances of the methods for *n* nodes, *m* links and the <k> average degree are listed below (Danon *et al*., 2005);

Table 4.9. Complexities of current methods (Danon et al., 2005)

Authors	Label	Complexity
Newman & Girvan	NG	$O(m^2n)$
Girvan & Newman	GN	$O(n^2m)$
Fortunato et al	FLM	$O(n^4)$
Radicchi et al	RCCLP	$O(n^2)$
Newman	NF	$O(n \log^2 n)$
Donetti & Munoz	DM/DMN	$O(n^3)$
Eckman & Moses	EM	$O(m<k^2>)$
Zhou & Lipowski	ZL	$O(n^3)$
Bagrow & Bollt	BB	$O(n^3)$
Duch & Arenas	DA	$O(n^2 \log n)$
Capocci et al	CSCC	$O(n^2)$
Wu & Huberman	WH	$O(n + m)$
Palla et al	PK	$O(\exp(n))$

Reichardt & Bornholdt	RB	Parameter dependent
Guimera et al	SA	Parameter dependent
Our algorithm	**SACD**	**$O(n^2)$**

It is seen from Table 4.3, that the fastest algorithm is Wu & Huberman (Wu and Huberman, 2004) and it runs linear in time but it needs a priori knowledge about network and it assumes that all communities are of similar size. The fastest method exploring the unknown community structure is the Newman's (Newman, 2004). However, it does not guarantee the most accurate results. Hence, all algorithms have some advantages as their disadvantages and all of them may be applicable to different networks. Our algorithm provides better complexity value than most of the current algorithms. The only method gives the results in definitely shorter time than ours is the Wu & Hubermans's but it's drawback is that it needs a priori information about the network and it makes it not applicable to the real world networks. Finally, it can be said that, our algorithm gives nearly optimum accuracy without any information about the network in satisfying complexity. The algorithms providing better complexity values than our algorithm need information about network to run or do not find the most accurate community structure.

5. ANALYSIS OF EXPERIMENTAL RESULTS

5.1. Overview of Results

We have tried to assess the accuracy and performance issues of our algorithm. To measure the accuracy a comparative analysis is made between the community structure that the algorithm provides and the known community structure of the network. We have also examined the parameters which are used in the algorithm to find the best parameters for different networks. Besides, the performance of the algorithm is tested and the performance of the algorithm is compared with current methods.

5.2. Zachary's Karate Club Network

The Zachary Karate Club data (Zachary, 1977) is collected by Wayne Zachary from the karate club of a university. This dataset is widely used in community detection algorithms. In 1977, a karate club is observed for collecting data for two years. First, the karate club was consisting of 34 members. Then, the instructor left the club because of a disagreement. The half of the members of the karate club has left the club with the instructor. After that, a network of friendship is constructed among all members of both clubs by Wayne Zachary. This friendship network includes 34 nodes and 78 edges. The community detection algorithms aim to detect the structure of these two clubs.

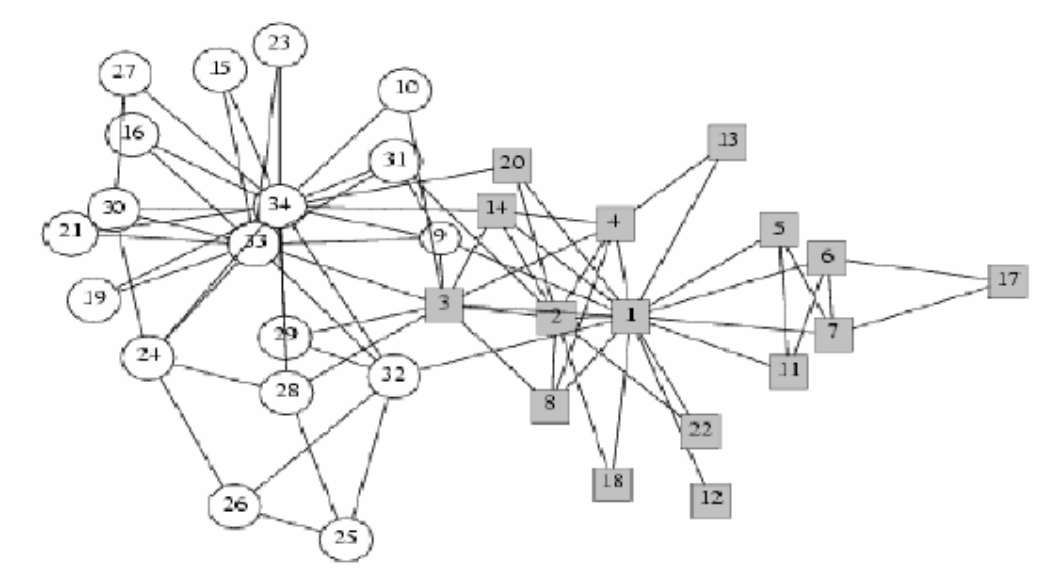


Figure 5.1. The Zachary Karate Club Network (NET)

5.3. College Football Network

The College football data is collected using the football matches in USA during year 2000. The nodes represent the teams in the conferences and there is an edge between two teams if they played a match at least once. Teams are allowed matches with teams from other conferences but the probability of making matches within conference is higher. Teams play an average of about seven intra-conference games and four inter-conference games. The community detection we want to find is the conference structure.

The college football network dataset contains 93 teams as nodes and 452 matches as edges. There are 10 conferences and these conferences show the real community structure of the network. That is, there are 10 communities to detect. Clubs.

5.4. Political Books Network

The political books data is produced by Valdis Krebs (POL) in 2003 using the book buying information on some retailers. The information like “Customers who bought this

book also bought:" on the web pages is used to form the network. The nodes denote the political books bought and there is an edge between two nodes if they were bought together at a major retailer on the web. The books are divided into two different categories depending on the idea they contain. These two categories should be found as the community structure. This network of political books has 49 nodes and 292 edges.

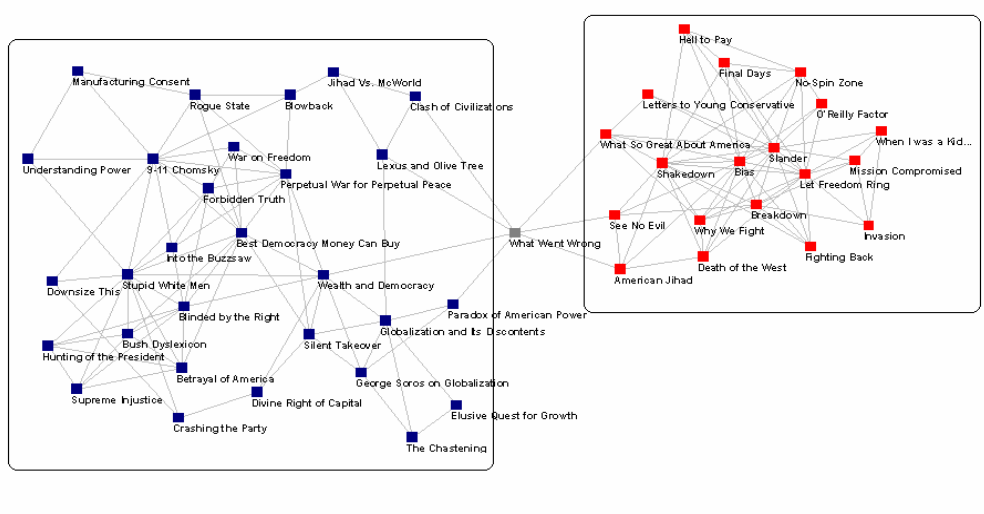


Figure 5.2. The Political Books network (POL)

5.5. Reuters Network

The Reuters news data is produced by Özgür, A. (Ozgur and Bingol, 2004) in 2004 by examining the news which appeared in the Reuters news. The network is constructed by the people in these news'. The nodes represent the people who have appeared in the Reuters news and there is an edge between two nodes if these two people have appeared in the same news. However, there is no known community structure and we have to find the communities and comment on these findings. The Reuters News network includes 2245 nodes and 3633 edges.

5.6. Company Employees Network

The Employees data is created by collecting information from the employees of a company as a part of this work. Each employee is asked for the colleagues which he has good relations. Every employee gave the names of at least 1 up to 10 people. Then, a network is formed in a way that each edge represents employees and there is an edge between two nodes if these two people have good relations. Using this network, the project groups are tried to be detected as the communities. The network has 169 nodes and 415 edges.

5.7. Web Sites Network

The Web Sites Network data is formed by observing the log files of a proxy server in a university. The collection of the Web sites network data is part of our work. The web sites visiting data is gathered from the proxy server in a time period of a day time. 10 Megabytes of log file that contains individual IP's and the web sites they have visited were analyzed. A parser program is written to obtain the data required for constructing the network. An URL represents a node in the graph and there is an edge between two nodes if these two URL's are visited by the same person. There is 1669 nodes and 76506 edges in the network.

5.8. Computer Generated Network

The Computer Generated Network data is created programmatically using randomness. This network is used by many algorithms for testing their ability of detecting communities in complex networks. The network includes 128 vertices divided into four communities of 32 vertices each. Besides, each vertex has a total degree of 16 and the number of intercommunity edges from a vertex is tuneable. The number of intercommunity edges from a vertex is denoted with z_{out}. The performances of current methods were measured for some values of z_{out}. We will run our algorithm for different z_{out} values and perform accuracy tests for increasing values of z_{out}.

5.9. The Experimental Setup

To identify the communities in a network, we have effectively built a community detection algorithm that may be applicable to the networks of different types and sizes. We have built our algorithm using Java programming language in JBuilder 9 application development environment.

We have used JADE (Java Agent Development Framework) API to include agents in the algorithm. JADE is an agent framework implemented in Java and may be used only through Java. It handles the working and organization of multi-agent systems on different machines with different operating systems.

5.10. Choice of the Parameters in SACD

The choice of parameters is crucial in SACD algorithm. The choice of parameters and the effects of the parameters to the accuracy are examined. First, the algorithm is tested for different values of agent population size and memory size of each agent. Then, the optimum values are found using these tests. After that, it is checked whether there is a convergence on the iteration count for the exploration phase or not using the optimum values found.

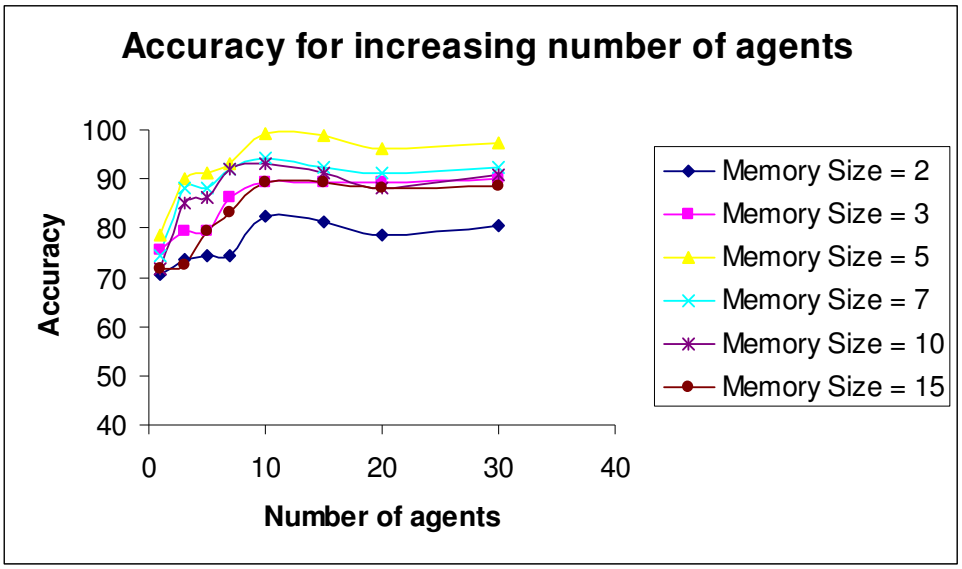
Each test with different memory size and number of agents is made 10 times. Then, the average value of these tests is calculated and assumed to be the final result of experience.

i. Number of Agents and Memory Size Tests

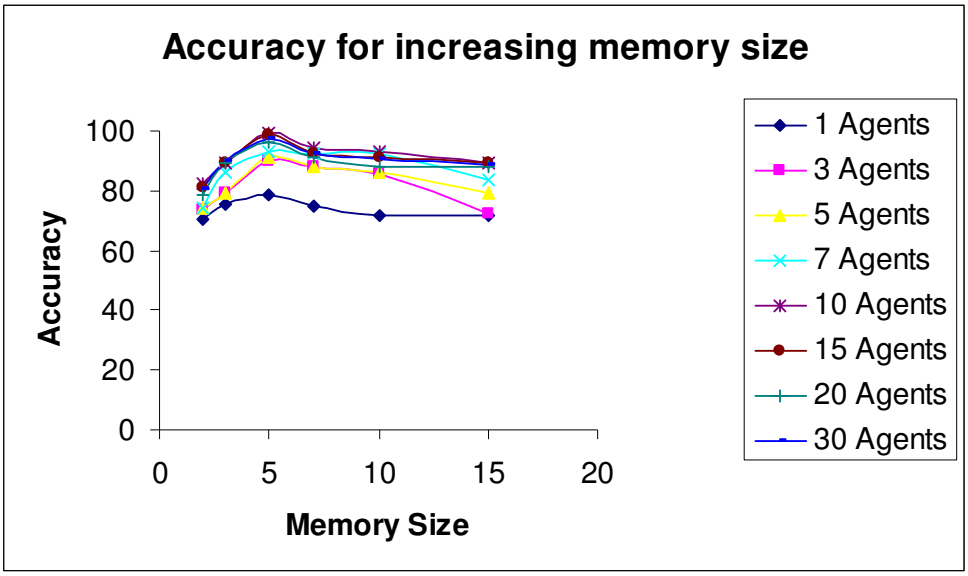
We tested different parameters of SACD which are required for both the operation of agents and the algorithm. The tests are related with the number of agents, the size of the

memory used by the agents, the number of agent generations during the exploration phase and the accuracy related with different values of these parameters.

The first test is about the effect of the number of agents to the community detection accuracy. The fraction of correctly classified nodes to all nodes is calculated for different number of agents using different memory sizes for each agent. The tests are made for 1, 3, 5, 7, 10, 15, 20 and 30 agents to find the optimum number of agents to be used for Zachary Karate Club data. The tests about the number of agents are all made with different values of memory sizes. Besides, the exploration phase is continued until each node is visited n times where n is the number of nodes.



a) Accuracy values by the increasing number of agents for different memory sizes for Zachary Karate Club data



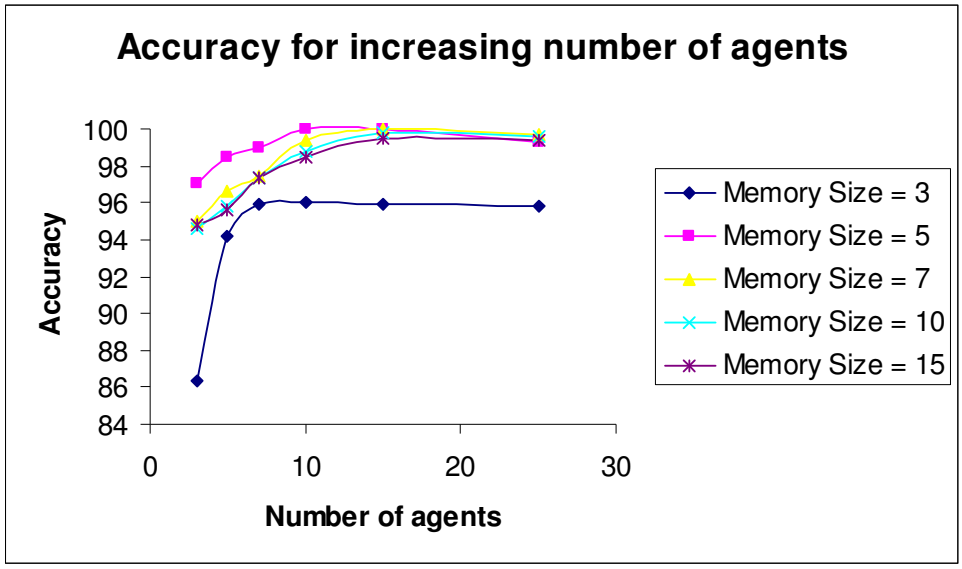
b) Accuracy values by the increasing memory sizes for different number of agents for Zachary Karate Club data

Figure 5.3. Accuracy values by the different number of agents for different memory sizes for Zachary Karate Club data

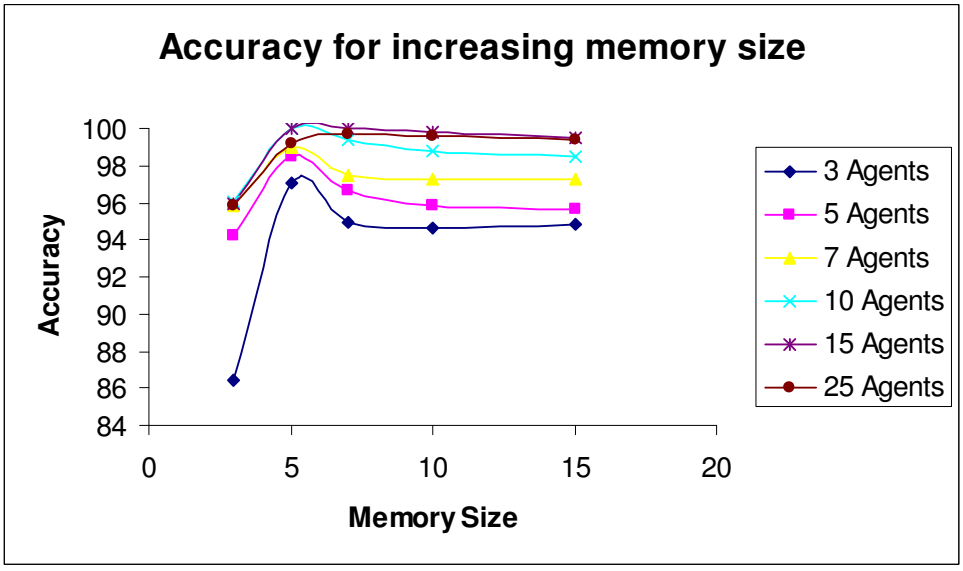
Here are the graphs (Figure 5.3) related to the accuracy values provided for different number of agents each of them having different memory sizes. It is obviously seen that, increase in number of agents improves the accuracy of the algorithm, however further increase does not yield any further improvement. There are some optimum values for agent number and more than this value will not result with better results. It is seen that, the best results are obtained with 7-10 agents. Less number of agents is not enough for gathering required information about the network structure. However, more than these numbers of agents have no definite improvement on the accuracy. It is also clear that, using memory size of 2 or using only one agent gives the worst accuracy values.

Moreover, the effect of the memory size is also seen from the graph. The least accurate results are provided for memory size 2 and memory size 15. Small memory results in the lack of exploring phase. A travel consists of only two nodes do not give enough information about communities. However, big memory is also not appropriate for using in the algorithm. Bigger memory may force an agent to cross into different communities and results in misidentification of communities. The optimum memory sizes

seem to be 3 or 5. Hence, number of 7-10 agents and memory sizes of 3-5 are considered to be best parameters to be used for the Zachary's Karate Club data. In addition, we can find the optimum parameter set for other networks as well.



a) Accuracy values by the increasing number of agents for different memory sizes for Political Books data



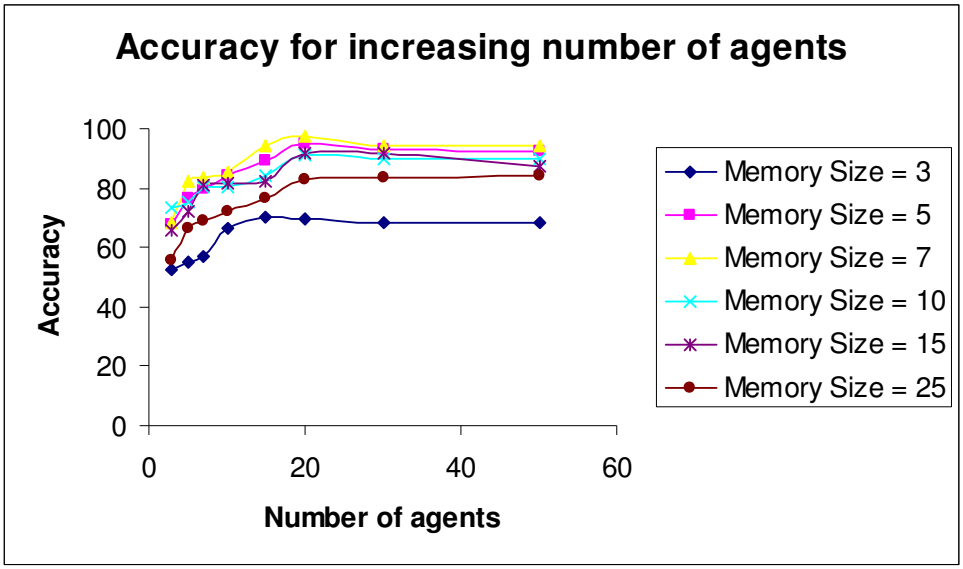
b) Accuracy values by the increasing memory sizes for different number of agents for Political Books data

Figure 5.4. Accuracy values by the different number of agents and memory size for Political Books data

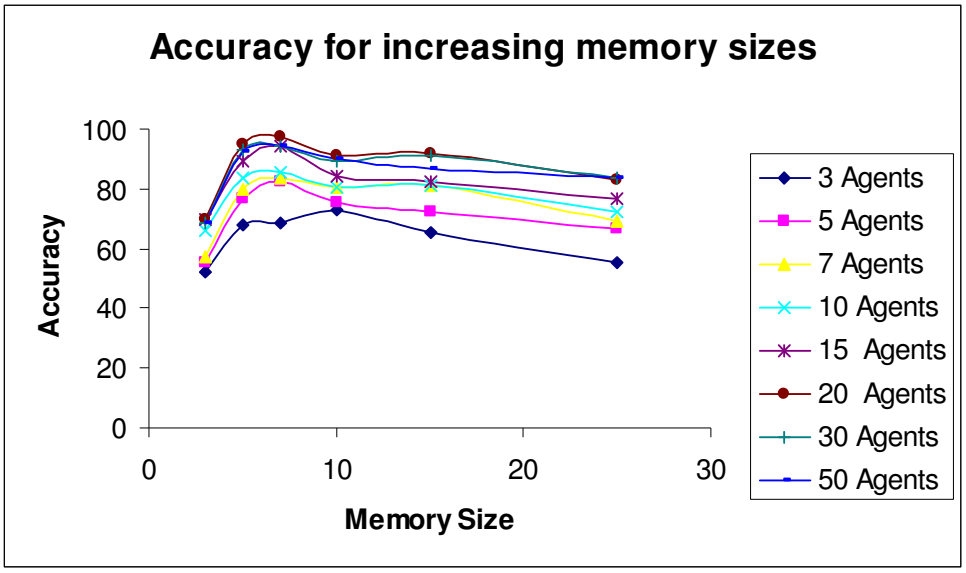
The graphs in figure 5.4 are related to the accuracy values provided for different number of agents and different memory sizes for Political books data. As it may be seen from the graphs for Zachary Karate Club data, the accuracy values do not increase for more than an optimum number of agents. For Political Books data, 10 agents is enough for exploring the network and more than 10 agents are almost useless.

The sizes of agent memories have also significant effect on the accuracy values. It is seen that, the best accuracy values are obtained for memory size 5. The experiences with memory size 7, 10 and 15 also gave sufficient results, because the communities are large relatively to the network size. Hence, the memory size giving the best accuracy value is related with also the size of communities to be found.

Lastly, it is clear that the communities are easier to distinguish because higher accuracy values are obtained compared to the results from other networks. The 100% accuracy values are received as output from many experiences.



a) Accuracy values by the increasing number of agents for different memory sizes for College Football League data

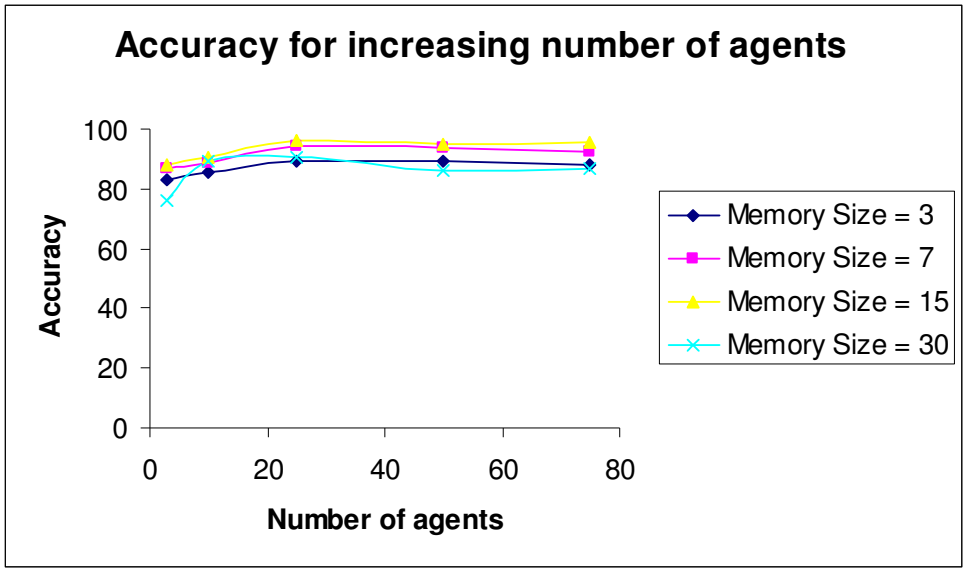


b) Accuracy values by the increasing memory sizes for different number of agents for College Football League data

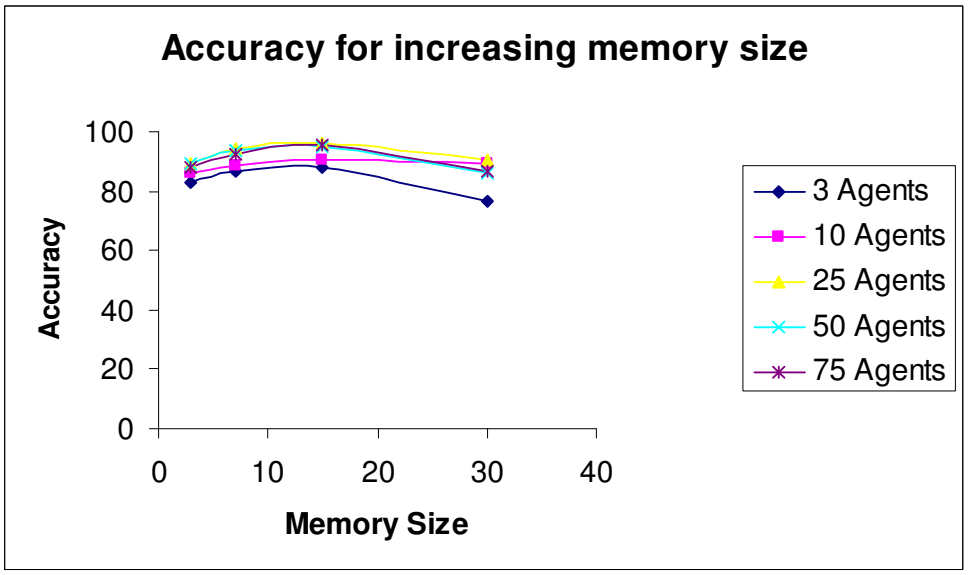
Figure 5.5. Accuracy values by the different number of agents for different memory sizes for College Football League data

Above are the graphs (Figure 5.5) related to the accuracy values of different parameter sets for College Football data. It is seen that, using more than 20 agents brings no improvement in the accuracy. There is even a little bit decrease in the accuracy for more than 20 agents. So, the optimum number of agents is 20 for College Football League data.

The test results show that the best results are provided for the memory size of 7. Besides, the accuracy values seem to be lower than the other networks', because the communities are not definite and they may be overlapped. So, it is hard to detect the real community structure in College Football data compared to the other networks.



a) Accuracy values by the increasing number of agents for different memory sizes for Employee data

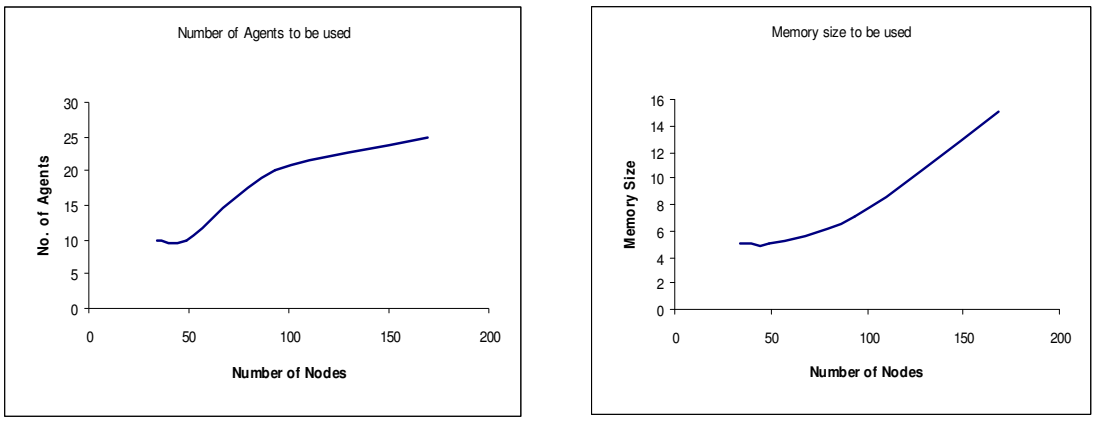


b) Accuracy values by the increasing memory sizes for different number of agents for for Employee data

Figure 5.6. Accuracy values by the different number of agents for different memory sizes for Employee data

As it is seen from the Figure 5.6, the optimum number of agents is 25 and the optimum size of memory is 15 for the Employee data. The parameter values are higher for this data because the network has more nodes and it is more difficult to detect the communities.

To use our algorithm for a network with unknown community structure, the parameters should be selected according to the size of the network. The graphs above should be considered before using the algorithm. To give an idea about the relation between the number of nodes in the network and the parameters to be used, the figure is given;



a) Number of agents to be used b) Memory size to be used

Figure 5.7. Number of agents and memory size of each agent to be used in the networks with unknown structure according to the number of nodes in the network. a) Number of agents to be utilized. b) Memory size to select considering number of nodes in the network.

ii. Iteration Count Tests

Iteration count denotes the number of generation of agents used in the algorithm. The exploration phase is terminated automatically using a termination condition. The exploration phase is completed if each node in the network is visited n times, where n is the number of nodes. We want to check whether we terminate the exploration phase before the convergence or not. The convergence means the point that further iterations do not

bring any improvements to the accuracy. In the figure 5.8, the improvement of the accuracy with increasing generations (iterations) is shown;

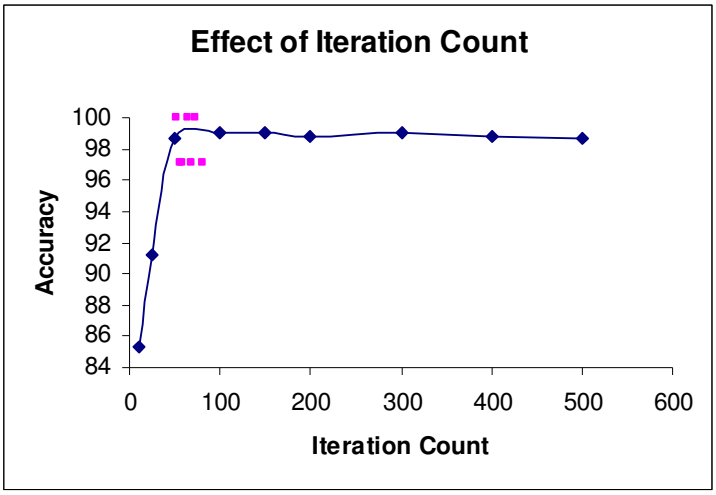


Figure 5.8. Effect of iteration count to the accuracy for Zachary Karate Club data

We've tested the algorithm with 10, 25, 50, 100, 150, 200, 300, 400 and 500 fixed iterations. As it is seen from the Figure 5.8, the accuracy is being improved with the increasing iteration count until a convergence limit. When this limit is reached, there is no further improvement in the accuracy. Here, a minimum iteration count of 50 is required for the exploration phase to correctly identify the communities in the network. The dots around the line show the number of used generations (iterations) and related accuracy values after our algorithm is completed with "termination condition" (not with fixed iteration) for different runs. Here, the dots denote that, the algorithm is terminated after convergence of the iteration count and network structure is explored enough. Hence, the termination control of the algorithm is suitable. The increasing accuracy by the increasing iteration count shows also the optimization made at each generation of agents.

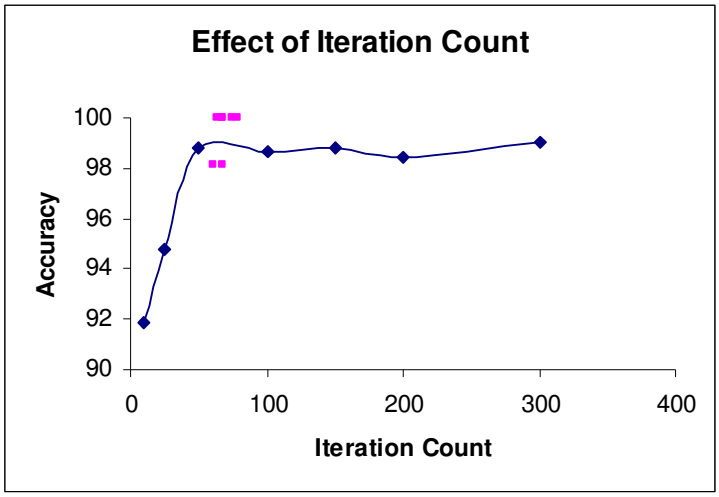


Figure 5.9. Effect of iteration count to the accuracy for Political Books data

As it is seen from the Figure 5.9, the accuracy is being improved with the increasing iteration count until a convergence limit for Political Books data, too. The convergence is again 50. The dots on the graph show that, the network is explored enough when the termination condition is satisfied.

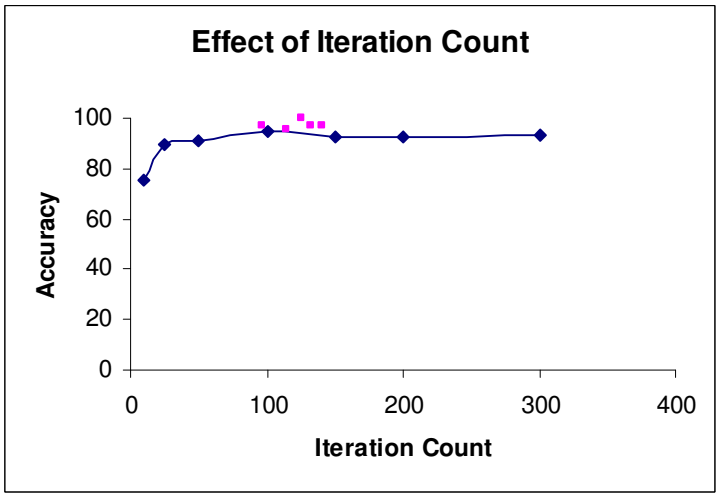


Figure 5.10. Effect of iteration count to the accuracy for College Football data

Here, it is seen from Figure 5.10 that the upper limit for the iteration count is 100. Iterations of more than this upper limit will not bring any improvements to the result. It is

clear that, the runs represented by dots managed to explore the network structure because the iteration count of these runs exceeds the convergence point.

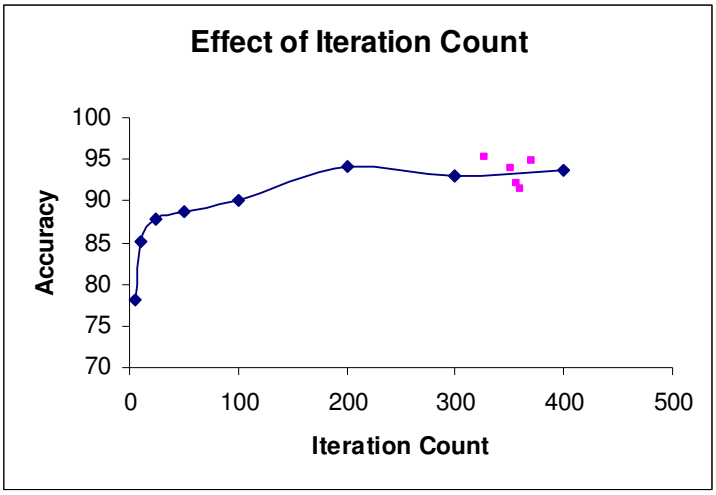


Figure 5.11. Effect of iteration count to the accuracy for Employee data

The iteration count required to explore the Employee network is more than the iteration counts for other networks (Figure 5.11), because it is hard to identify the communities of this network. It is seen that 200 iterations are required for exploration and further increase do not bring any improvement in accuracy value. Dots around the line mean that algorithm runs more than the required number of iterations needed for providing adequate accuracy.

5.11. Community Structure Identification Tests in Different Networks Using SACD

After deciding on some parameters of the algorithm and making some performance tests, we ran our algorithm on different networks which have known community structures to be compared with the results from our algorithm. We ran our algorithm for Zachary' Karate Club data, College Football data, Political Books data, Company Employees data and the computer generated network data. Then, the accuracy results are compared with the known community structures.

5.11.1. Zachary's Karate Club Analysis

The details of the Zachary Karate Club data were given in previous sections. The network has two communities to be identified. We ran our algorithm with some parameters near optimum values to test the accuracy of the results.

Table 5.1. Accuracy test for Zachary's Karate Club Data

Memory / Agent	2	3	5	7	10	15
1	70,58%	75,5%	78,44%	74,53%	71,56%	71,57%
3	73,52%	79,42%	90,2%	88,24%	85,26%	72,55%
5	74,5%	79,5%	91,22%	88,27%	86,27%	79,42%
7	74,52%	86,27%	93,14%	92,15%	92,14%	83,34%
10	82,41%	89,26%	**99,12%**	94,15%	93,24%	89,3%
15	81,37%	89,22%	99,01%	92,24%	91,18%	89,22%
20	78,52%	89,21%	96,18%	91,24%	88,21%	88,24%
30	80,69%	90,01%	97,26%	92,43%	90,82%	88,61%

As it can be seen from the Table 5.1, the algorithm performs well for Zachary's Karate Club data with different parameter sets. It provides from 99% accuracy with optimum parameter set. This means that only 1 node is misclassified sometimes.

5.11.2. College Football Network

The College football data consists of 10 communities which represents conferences in a football league. There are 93 nodes and 452 edges in the network.

Table 5.2. Accuracy test for College Football Data

Memory / Agent	3	5	7	10	15	25
3	52,33%	67,7%	68,37%	73,12%	65,6%	55,38%
5	55,2%	76,76%	82,43%	75,27%	72,41%	66,67%
7	56,99%	79,91%	83,65%	80,64%	81%	68,9%
10	66,31%	83,88%	85,72%	80,66%	81,36%	72,4%
15	70,03%	89,03%	94,1%	84,23%	82,44%	76,89%
20	69,52%	95,18%	**97,18%**	91,04%	91,68%	83,12%
30	68,67%	93,18%	94,32%	89,61%	91,48%	83,34%
50	68,52%	92,48%	94,4%	89,64%	87,1%	83,88%

As it is seen from Table 5.2, the number of agents required for the algorithm is more than the number of agents in the previous tests, because this network is bigger than the others. However, 20 agents are enough for the exploration of the network and any further increase in the agent population do not increase the percentage of the accuracy. Thus, 20 agents with memory sizes of 7 are the optimum values for the Collegue Football network.

5.11.3. Political Books Network

The Political Books Data is a network data which includes of 49 nodes and 292 edges. This network consists of 3 communities and one of these communities is only a nodes. The table below shows the results of the algorithm for different parameter pairs;

Table 5.3. Accuracy test for Political Books Data

Memory / Agent	3	5	7	10	15
3	86,39%	97,07%	94,96%	94,64%	94,81%
5	94,23%	98,46%	96,68%	95,82%	95,64%
7	95,89%	98,96%	97,47%	97,31%	97,31%
10	96,04%	**100,00%**	99,41%	98,81%	98,44%
15	95,97%	**100,00%**	**100,00%**	99,78%	99,46%
25	95,86%	99,24%	99,66%	99,56%	99,42%

The Table 5.3. shows that the algorithm identifies almost all nodes correctly. Maximum of 1 node is not classified correctly for some parameters. This node is the node among two communities.

5.11.4. Employees Network

The Employees network includes the nodes representing employees of a company and the edges as the relationship between the employees. There are 19 communities of different sizes to detect. The accuracy values of Employees network is shown in Figure 5.4

Table 5.4. Accuracy test for Employee Data

Memory / Agent	3	7	15	30
3	82,83%	86,88%	88,17%	76,34%
10	85,82%	88,79%	90,54%	89,35%
25	89,36%	94,09%	**96,47%**	90,55%
50	89,34%	93,56%	94,68%	86,11%
75	88,17%	92,32%	95,86%	86,99%

5.11.5. Computer Generated Network

The computer generated network is used to compare the performance of our algorithm with other methods'. The network has 4 communities each of them with 32 nodes. The accuracy values are calculated for different values of average intercommunity edges. That is, if the average number of intercommunity is equal to 6, every node in the network has 6 edges to the nodes from other communities. The accuracy graph by the increasing number of intercommunity edges is given in figure 5.12;

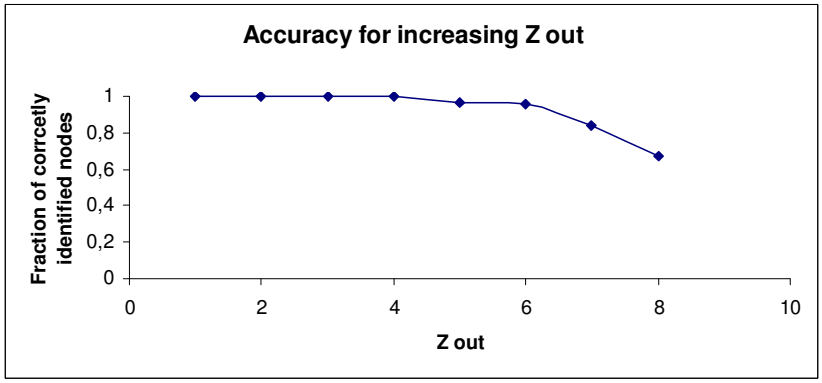


Figure 5.12. Effect of intercommunity edges to the accuracy

It is clear that, our algorithm identifies all nodes correctly for $z_{out} < 5$. For $z_{out} = 5$ or $z_{out} = 6$, only 5-10 nodes are misidentified among 128 nodes with a percentage of 96%. However, for z_{out} values bigger than 6, the algorithm tends to misidentify nodes and the accuracy decreases to 84% for $z_{out} = 7$.

5.12. Performance Tests

After performing parameter test, we are able to know which parameter sets are optimums for which networks. Then some convergence tests are made to check whether a convergence exist about iteration count or not. At this point we will run our algorithm for some larger networks without known community structure.

Due to the reason that the Reuters and Web Site Networks have no known community structure, we can only perform some performance tests and make some logical inferences according to the found communities.

First, the communities are identified for Reuters network. The members of ministries are found to be in the same communities as expected. Also, some US government members are included in the same communities. The white house members are also represented by a different community. The members of Soviet, Turkish and Polish governments are also other communities. The ministers of the Middle East countries are assumed to be the members of the same community.

The web pages similar in content are detected using our algorithm for web sites network. For instance, some of the newspapers, web sites related with sports, education and magazine are identified in the same communities.

Almost all of these findings are logical and several inferences may be obtained using these community structures. However, we are not able to asses the accuracy of the algorithm for the Reuters Network. Yet, we are able to measure the allocated memory and time value for the algorithm. They are shown in Table 5.5.

Table 5.5. Elapsed time and allocated memory for different networks

Name of the network	**V** *(Number of Vertices)*	**E** *(Number of Edges)*	**T** *(Elapsed time in seconds)*	**M** *(Allocated memory in KB)*
Zachary Karate Club Network	34	78	3,62	19,952
Political Books Network	49	292	4,87	20,648
College Football Network	93	452	5,02	20,927
Computer Generated Network	128	1024	8,33	22,212
Employees Network	169	415	11,69	24,128
Reuters Network	877	1242	201,12	54,108
Web Site Network	1669	76506	934,08	142,475

6. DISCUSSION AND CONCLUSION

6.1. A Review of Work Done

In this work we have introduced a new community detection algorithm using smart software agents. SACD provides fast and accurate identification of communities.

First, we've studied the main parts of the complex networks and we have given some definitions required to understand the work done. Then, the research about the complex networks of different types and their properties are made and the findings are included in the work. As a part of the research, the definitions and motivation for the community detection are also presented.

A detailed study about the agents is also given in the work. Besides, a general explanation on agents and an extensive one about JADE agents are parts of the research we've made. The complete functions and mechanisms of agents we've used are explained in details. Some code snippets about JADE agents are even included in this part.

The community detection methods proposed so far were analyzed in details and their main contributions are given. After analyzing them one by one, their performances were compared in terms of complexity and accuracy. This comparison makes it possible to asses our algorithm. We've also written a program to create a computer generated network that is used in current methods. This network is used as common criteria to compare the performances of the algorithms.

We've proposed an agent based algorithm for detecting communities in complex networks. The algorithm consists of two main phases. The first phase is the exploration phase and handled by the agents. The information about the network is collected in this part by making agents traverse in the network. The main idea behind collecting network information is explained in details. Then, the second phase is the analyze phase that includes the revealing of the communities using the collected information in former phase.

The edge removal method is used to identify communities in this phase. An algorithm called “full-fill” is implemented to check whether a new community is formed or not after a removal operation. The network modularity value is also used for deciding on the candidate communities.

We have presented a wide scope of experimental results. First, the effects of the parameters used in the algorithm are studied in details. Some graphs required to understand the selection of the parameters are given. Then, the community structure identification tests were made to measure the accuracy provided by the algorithm. Some well known networks are used for these tests. We have also created a network by making a questionnaire among 169 people. The performance tests are also included by using some larger networks with unknown community structure.

6.2. Discussion and Directions for Theoretical Aspects

There are only a few algorithms which use agents in detecting communities in complex networks. Our algorithm is one of these algorithms and provides fast and accurate results. It uses software agents to collect information about the network structure. The main idea is the tendency of an agent to stay in the community while travelling in the network. An agent with random move has higher probability of staying within the community than the probability of leaving the community. However, our algorithm makes agents move in a smart way instead of random moves. The agents use the gathered network structure and make biased moves. Hence, the algorithm has superiority to other community detection algorithms using agents.

Our algorithm is an iterative one. At each iteration, the network structure is learned more and the results are optimized. So, more accurate results are provided by the increasing number of iterations until a convergence and an optimization is realized. Another advantage of the algorithm is that the agents utilized in our algorithm have a global knowledge of the network unlike the agents used in other methods. The feedback information from many agents from different parts of the network are received and

evaluated. So, global knowledge of a network is obtained and reasonable moves are made by the agents.

Moreover, our algorithm removes the need for a priori knowledge about the network. Many of the community detection algorithms need some knowledge about the community structure such as number of communities, community sizes, threshold value etc. However, it is impossible to know about these parameters in real life. Our algorithm needs no priori information and the required information is collected by the smart agents during the exploration phase. This is another improvement of our algorithm.

To improve the algorithm, some parameters in the algorithm may be removed as a future work. The algorithm can be fully self-contained if the parameters of the algorithm are learned by the time. The optimum number of agents to be utilized in the algorithm and the memory size of each agent are the parameters of the algorithm and they can be learned using some learning methods by assessing the results for different values of these parameters. The Genetic Algorithm may be used for choosing the optimum parameter set.

Community structure identification has been one of the most popular research areas in recent years due to its applicability to the wide scale of disciplines. There are several areas that community detection is utilized. By detecting communities in a network, groups of nodes which are probably in a relationship are explored. For instance, if we assume a network that the nodes are the websites and there is an edge between two nodes if some numbers of people visit both sites. By detecting communities in this network, the group of websites which are visited by the same group of people is found. This group of people visiting the same sites may be the target customers of a product. So, this community information may be useful for an advertisement company. The advertisers can reach more target customers by giving ads to only small number of websites. As another example of the use of community detection, the protein networks may be thought. The community structure in protein networks reveals the information of the similar proteins according to the relationship that form the complex network. Moreover, virus spreading networks may be analyzed in this way. The healthy nodes are detected to protect them form infected nodes. In recent years, detecting communities are considered to be used in detecting some

terrorist actions on the Web by exploring some patterns as the communities. Hence, it can be claimed that, detecting communities has a wide scope of use and has also important potential for further uses.

6.3. Discussion and Directions for Experimental Results and Methodology

We have used some networks with known community structures for tests to asses the accuracy values provided by our algorithm. The first network is the Zachary Karate Club network and it is the most widely used network for community detection tests. Because of its wide use, it is suitable to be used in comparing the results of the algorithms. Like some other successful methods, our algorithm correctly identifies the 95-100% of the nodes with suitable parameters. Zachary Karate Club network is the not only network used for accuracy tests. We've also used the Political Books network that is easily identified with almost 100% accuracy. The communities are joined with a few edges and easily detected.

A bigger network used in the tests is the College Football network that consists of 93 nodes. The network includes 10 communities of different sizes. This network is useful to see the performance of the algorithm for the communities of different sizes. The algorithm also provides over 90% accuracy for this network. The last network with known community structure is the Employee network that is created by us by preparing a questionnaire and asking people the relationship with others. Collecting data from people was one of the most difficult parts of the work. This network has 169 nodes and over 90% accuracy is obtained for this network, too. This is a social network and it is useful to reflect the performance of our algorithm with a social network.

As a last test, we've made a test with a computer generated network to be able to compare the performance with other known algorithms. First, we've written a program that creates a tuneable network with 128 nodes. The performances of current methods with this network are known and the performance of our algorithm was also calculated for different z_{out} values. The results are comparably well for computer generated network. Almost 100% accuracy is obtained for z_{out} values lower than 6, while the accuracy values tend to decrease after 7 and some nodes are misidentified.

Our algorithm has two main parameters; number of agents and the memory size of each agent. To see the effect of these parameters, we've made some parameter tests with different values of these parameters. First, the program is run several times for different value pairs of number of agents and memory size and the accuracy values are recorded. Then, the graph of accuracy values by increasing number of agents for different memory sizes is drawn and it is seen that there is a convergence in number of agents. That is, increasing the number of agents more than an upper limit does not bring any improvement to the accuracy. Using the same test results, we've also drawn the graph of accuracy by the increasing memory sizes for different number of agents. At this point, it is realized that the best accuracy is obtained for a different memory size for each network and the lower and bigger memory sizes cause misidentification of the nodes. It can be claimed that, the memory size and number of agents to be utilized is proportional to the size of network. If the network is larger, more agents with bigger memory sizes are needed to get the best results.

Not only networks with 100-200 nodes are used for tests but also larger ones. The larger networks do not have known community structure but we've used them for scalability purposes and to be able to compare with other algorithms. The first network used for scalability tests is Reuters Network with 2245 vertices and the other is Web Sites network with 1669 nodes. The scalability of the algorithm is tested using these networks.

It should be noted that, every different test is made 10 times and the average of the accuracy values from these tests is assumed to be the final result of this test. Because of the stochastic structure of the algorithm, the same tests may not give the same result. So, using the average value is the right way and decreases the effect of randomness.

Although we've tested the algorithm with networks of different types and sizes, more network datasets should be used for testing both accuracy and scalability. The performance of the algorithm is not clear for a huge network with more than 10 000 nodes.

The complexity of the algorithm is found as $O(n^2)$. This complexity value is better than most of the current algorithm detecting communities in complex networks. Using simple network modularity also keeps complexity small. However, the complexity may be decreased using some methods such as utilizing a sparse matrix. Because most parts of the matrices used in the algorithm is 0 and using a sparse matrix will bring many improvements on the complexity. The use of sparse matrix can be implemented as a further work.

APPENDIX A. TUNABLE COMPUTER GENERATED NETWORK

In addition to the known networks like Zachary Karate Club, we also use a computer generated network to be able to compare our results with current methods'. Using this computer generated network we can asses the success of the algorithm for different types of networks because we can change the level of distinctness of the communities by only tuning a parameter. Moreover, most of the current algorithms were tested with this computer generated network and the test results with this network will be a common metric for all algorithms.

We've generated a graph with 128 vertices, divided into four communities of 32 vertices each. Each vertex has z_{in} edges to vertices in the same community and z_{out} edges to vertices in other communities. The total degree is $z_{in} + z_{out} = 16$. By changing the value of z_{out}, the community structure of the network may be adjusted. If we add 16 edges to all vertices in order, the required conditions will not satisfied, because every node should have z_{out} edges to the vertices in other communities and (16 - z_{out}) edges within the community. The network should be created in a balanced way. So, we add edges among vertices in a controlled way. To add inner community edges, the 2 edges with lowest inner degrees within the community are selected and they are joined. The inner degrees of these two vertices are also increased by one. Then, this process is iterated until each vertex in the community has an inner degree of (16 - z_{out}). This operation is made for each 4 communities and inner connections are made.

After adding inner connections, the outer edges are joined. In the same way, 2 nodes with lowest intercommunity degrees are selected and they are joined if they are in different communities and if they have not already been joined. A connectivity matrix is kept for this control. This process is iterated until each vertex has an outer degree of z_{out}. The selection of the node to be joined should be made totally random without any bias to prevent different communities to appear. The tests with these networks show the capability of our algorithm identifying the communities when the communities are fuzzier inside the network.

APPENDIX B. NETWORKS CREATED AS A PART OF THIS WORK

In this section, the networks which we've created for testing with our algorithm are presented. First, Web Site Network is clarified and it is explained how we've created this network. Then, the process to form the Employee Network is mentioned.

B.1. Creating Web Site Network

The first network we've created is the Web Site Network. The network is created using the log file of a proxy server in a university. The proxy server is observed during a time period and the log file summarizing the data files passing through the server is examined. There are thousands of entries in the log file and each entry includes individual IP's, the web sites people have visited, protocol numbers, port numbers, file names etc. We've written a parser program to obtain the required information.

Our program analyzes the log file and extracts the individual IP's and the names of web sites from this log file. Then the IP's and names of the web sites visited by these IP's are kept in a database. Using this information, the network is constructed. The web sites are represented by the nodes in the network and there is an edge between two sites if 2 web sites are visited by the same person.

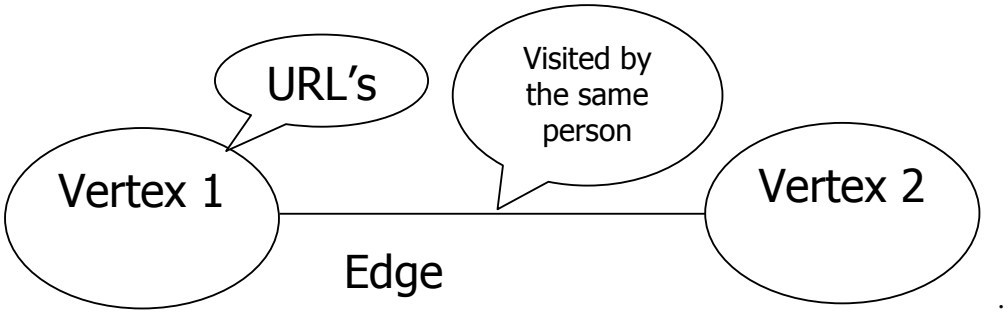


Figure B.1. The relation behind the web site network

There are 1669 vertices and 76506 edges in the network. The network is a small world network. The most connected nodes in the network are "login.yahoo.com", www.google.com.tr and surprisingly "ad.doubleclick.net". Some other characteristics of the network are given below;

Table B.1. Web sites network properties

Property	Value
num. of vertices	1669
num. of (undirected) edges	76506
average path length	2.13
Diameter	4
Clustering Coefficient	0.042 (0.03 for random network)

B.2. Creating Employees Network

The second network we've created for this work is called Employees network. The network data is collected by questionnairing the employees in a company. The employees are asked for the people who, they get on well with. Each employee is free to give the names of 1 – 10 colleagues. This information is used for revealing the community structure of the employee network. The communities are assumed to be the project groups. The project groups are determined by the relationships among people. That is, hiring someone may be the result of a recommendation of a group member or someone may be transferred to another group that includes the people who are in good relationship with him. So, the relationship between employees is used to identify project groups.

The employee network has 169 nodes as employees and 19 communities representing the project groups. There are 415 edges in the network and existence of edge denotes the good relationship among two nodes representing employees. That is, two employees are joined if at least one of them said that they have a good relationship.

REFERENCES

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

Anthonisse, J.M., (1971). Technical Report Bn. In *Stichtinh Mathemtisch Centrum*, 9:71.

Bagrow, J.P., Bollt, E.M., (2005). Local method for detecting communities. In *Physical Review E*, 72:046108.

Boettcher, S., Percus, A.G., (2001). Optimization with extremal dynamics. In *Physical Review Letters*, 86:5211-5214.

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

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

Danon, L., Duch, J., Arenas, A., Diaz-Guilera, A., (2005). Comparing community structure identification. In *Journal of Staistical Mechanics (2005),* P09008.

Donetti, L., Munoz, M.A., (2004). Detecting network communities: a new systematic and powerful algorithm. In *Journal of Staistical Mechanics (2005),* P10012.

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

(FIP) The foundation for intelligent physical agents http://www.fipa.org/

Flake, G.W., Lawrence, S., Giles, C.L., (2000). Efficient identification of web communities. In *Proc. 6th international conference on knowledge discovery and data mining, Boston, Massachussets, United States (2000),* p.150-160.

(FLO) Flood-Fill Algorithm http://www.comp.nus.edu.sg/~stevenha

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

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

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

(JAD) JADE Java Agent Development Framework http://jade.tilab.com

(NET) Gallery of network images http://www-personal.umich.edu/~mejn/networks

Newman, M.E.J., (2003). The structure and function of complex networks. In *SIAM Review*, 45:167-256.

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

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

Ozgur, A., Bingol, H., (2004). Social network of co-occurrence in news articles. LNCS 3280: 688-695.

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

(POL) Network dataset on political books http://www.orgnet.com/leftright.html

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, J., Bornholdt, S., (2004). Detecting fuzzy communities in complex networks with a Potts model. In *Physical Review Letter*, 93:218701.

Rosvall, M., (2003). Complex Networks and Dynamics of an Information Network. In Sweden. M.S. thesis, University of Umea, Umea.

Shoham, Y., (1993). Agent oriented programming. In *Artificial Intelligence* 60(1):51-92

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

(WIK) Online encyclopedia http://www.wikipedia.org

Wilkinson, D.M., Huberman, B.A., (2004). A method for finding communities of related genes. In *Proc Natl Acad Sci*, 101:5241-5248.

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

Young, M., Sager, J., Csardi, G., (2005). An agent-based algorithm for detecting community structure in networks. In *Proceedings of National Academy of Science in USA,* 101:2658-2663.

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

Zhou, H., Lipowsky, R., (2004). Network Brownian motion: A new method to measure vertex-vertex proximity and to identify communities and sub communities. In *LNCS*, 3038:1062-1069.

REFERENCES NOT CITED

Bradshaw, J., (1997). Software agents. AAAI Press/The MIT Press.

Castellano, C., Cecconi, F., Loreto, V., Parisi, D., Radicchi, F., (2004). Self-contained algorithms to detect communities in networks. In *European Physical Journal*, 38:311-318

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

Iamnitchi, A., Ripeanu, M., Foster, I. (2004). Small-world file-sharing communities. In *Infocom 2004*, Hong Kong, March 2004.

Newman, M.E.J. (2001). Who is the best connected scientist? A study of scientific co-authorship networks. In *Physical Review*, 64:016131-016132.

Newman, M.E.J. (2003). A measure of betweenness centrality based on random walks. In *Social Networks*, 27:39-54.

Shoham, Y. (1997). An Overview of Agent-Oriented Programming. In: J. M. Bradshaw, ed. Software Agents. AAAI Press/The MIT Press, p. 271 - 290.

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