The Pennsylvania State University
The Graduate School
The Harold and Inge Marcus Department of Industrial and Manufacturing Engineering

DETECTION & EVALUATION OF COMMUNITY STRUCTURES IN SOCIAL NETWORKS

A Thesis in
Industrial Engineering and Operations Research
by
Akshay Dattatraya Ghurye

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2011

The thesis of Akshay Dattatraya Ghurye was reviewed and approved* by the following:

Soundar R.T. Kumara
Allen E. Pearce/Allen M. Pearce Professor of Industrial Engineering
Thesis Advisor

Arvind Rangaswamy
Anchel Professor of Marketing

Paul Griffin
Professor of Industrial Engineering
Peter & Angela Dal Pezzo Department Head of Industrial Engineering

*Signatures are on file in the Graduate School.

Abstract

In today's world, social media networks capture interactions among people through comments on blogs, posts and feeds. The public availability of these networks has allowed researchers and businesses alike to delve more into these preferences so as to extract communities which clearly define their formation. In social networks, people tend to have more than one preference over different products which makes it difficult to put them in a single community. Although community detection has been well applied to social networks, not much work has been done in detecting overlapping communities within these networks. In this paper we describe an algorithm which applies a game theoretic approach to graph clustering to determine overlapping communities within complex networks and show how a parallel implementation of the algorithm can be used to detect communities in lesser time than its previous implementations. Further we run the algorithm on various social networks to detect overlapping communities and propose a method to analyze them once they are determined. Finally we conclude by providing impetus on the running time of this algorithm and expressing the need for faster algorithms to detect and analyze social media networks.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I thank my advisor Dr. Soundar Kumara for his continuous support during the Masters program. He has been a mentor, grooming me to scientific writing and presentations, and encouraging a quantitative approach to problem solving. I express my gratitude to my thesis reader Dr. Arvind Rangaswamy for helping me complete my thesis in time. I am grateful to my peers in the Laboratory of Intelligent Systems and Quality (LISQ) research group, especially Mr. Supreet Reddy Mandala (PhD Candidate), who patiently helped me formalize my thesis problem over repeated meetings. I am deeply indebted to my parents who were a source of motivation and strength thought my career. I am grateful to my friends for standing by me through the best and worst of times, and to the Almighty for aligning the turn of events in my good fortune.

Dedication

To my parents

Chapter 1

Introduction

1.1 Motivation

The proliferation of Internet and on-line marketing has changed the way companies market their products and services. The past paradigm of make and sell to a market segment has changed. Today, IT provides the universal access and reach to help market to a customer size of one. Customers have become an integral part of the company's marketing effort.

Over the years because of increasing competition, acquiring, retaining and supporting customers has become more challenging for businesses of all sizes. Companies these days are more concerned about the whats than the whos of customer base. In other words companies were focused on selling as many products and services as possible without much knowledge about their customer base. In the current age, marketing teams must plan and develop an increasing number of sophisticated campaigns, and deliver them through multiple mediums. Sales reps must follow-up on hundreds of new leads, while juggling existing sales cycles. Support staff must rapidly resolve a growing volume of customer problems and issues. And, management must oversee customer-facing operations across all departments, and ensure that all client interactions are handled in a responsive and professional manner.

Customer relationship management (CRM) systems has emerged as a way for businesses to streamline customer-related processes across functional areas, increase the efficiency and effectiveness of customer transactions at all levels, and

optimize service quality at each touch-point. Within the CRM world, there are many types of solutions, each having their own flavor, and each meeting different business needs. CRM is the infrastructure that enables the delineation of an increase in customer value, and the correct means by which to motivate valuable customers to remain loyal. At the same time companies are trying to leverage this customer satisfaction by trying to market products to these customers which relate to their wants/needs as opposed to using traditional marketing techniques. As a result companies are investing in various media through which they can gain more knowledge about their existing customers so as to be more specific in their marketing approaches.

With the advent of Facebook, Twitter, YouTube, people/users have started expressing their views and thoughts on various issues and products. These thoughts or views, which were not present previously are now readily available on-line and at the click of a button. As a result of this ubiquity , user rich content is easily available to companies which they can use to solidify their marketing campaigns and CRM approaches. Social Media Marketing is the current trend. Informally, Andreas Kaplan and Michael Haenlein [5] define social media as "a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0, and that allow the creation and exchange of user-generated content." Thus, social media marketing is a multifaceted, orchestrated marketing and advertising practice organizations follow to connect with their target markets. Various modes of social media marketing involve but are not restricted to advertising, personal selling, public relations, publicity, direct marketing and sales promotion [6]. Recently, social media has become a platform that is easily accessible to anyone with Internet access. Increased communication for organizations fosters brand awareness and often, improved customer service. Additionally, social media serves as a relatively inexpensive platform for organizations to implement marketing campaigns.

As the competition among firms intensifies, social media is gaining more and more importance and efforts are being made to utilize this data in an effective manner so as to achieve targets which were previously not feasible. Social networks form a part of social media which represent real life friendships or ties that exist amongst individuals and detecting extracting communities with similar be-

liefs inherent within such networks is a non trivial problem. This thesis aims at looking at various ways in which one can leverage a given social network to solve problems related to CRM, social media marketing and other aspects where user data is used such as supply chain management and retail analytics.

1.2 Problem Discussion

The emergence of smart-phones, tablets, and other portable electronics has allowed users to access the Internet more frequently than ever. With incoming technologies like 3G and 4G browsing speed has increased tremendously as opposed to the previous GPRS technology which was a major turn-down for many to even buy a smart phone. At the same time Facebook, Twitter and Flickr along with Wikipedia, Youtube has also grown tremendously. For example Figure 1.1 shows how the trend for conventional sites such as Yahoo, although constant, has lowered in comparison to that of social media sites, namely Facebook and Twitter. Just

Figure 1.1. The blue trendline is Facebook, yellow is YouTube and red is Yahoo. The rapidly growing popularity of YouTube and Facebook is characteristic of the way in which new products, technologies, or innovations rise to prominence, through feedback effects in the behavior of many individuals across a population. The plot depicts the number of Google queries for YouTube and Facebook over time. The image comes from the site Google Trends; by design, the units on the y-axis are suppressed in the output from this site.

to give a perspective on the emergence of social media we dig out a few statistics as noted by Kaplan in his research [5]. As of January 2009, the on-line social net-

working application Facebook registered more than 175 million active users which is only slightly less than the population of Brazil (190 million) and over twice the population of Germany (80 million). At the same time, every minute, 10 hours of content were uploaded to the video sharing platform YouTube. And, the image hosting site Flickr provided access to over 3 billion photographs, making the world-famous Louvre Museums collection of 300,000 objects seem tiny in comparison. According to Forrester Research, 75% of Internet surfers used Social Media in the second quarter of 2008 by joining social networks, reading blogs, or contributing reviews to shopping sites; this represents a significant rise from 56% in 2007.

The advent of Social Media has opened new possibilities in the field of social network analysis by making very large repositories of data available to researchers. Blog posts, news feeds and user preferences as well as phone calls, electronic communication via email, and scientific publication co-authorship records are now stored in a centralized, relatively easily-accessible locations. In addition, many social networking services and blog providers have emerged as important forums for individual expression and discourse. This allows businesses and researchers alike with rich and publicly observable data to use in the analysis of social interactions to aid either themselves or the community. The following subsection titled Social Media Tracking shows how leading companies are tracking the reviews about their companies from social media websites.

1.2.1 Social Media Tracking

The Gatorade Mission Control Center is a room that sits in the middle of the marketing department and could best be thought of as a war room for monitoring the brand in real-time across social media. This team tracks the discussions and blogs of their marketing campaigns & products which are released in the market. The monitor shown in Figure 1.2 is a visualization of tweets that are relevant to Gatorade; the company is tracking terms relating to its brand, including competitors, as well as its athletes and sports nutrition-related topics. Another monitor measures blog conversations across a variety of topics and shows how hot those conversations are across the blogosphere. The company also runs detailed senti-

ment analysis around key topics and product and campaign launches. The system has built the Social Media Tracking software through a joint collaboration with Radian 6 and IBM.

Figure 1.2. Gatorade Mission Control Center Dashboard

1.2.2 Social Media Software: *ListenLogic* and *Radian6*

ListenLogic and *Radian6* are Social Media Tracking software which track the web of information for blogs, posts, tweets and mentions about any given keyword or company product or a campaign. Some of the key notes which the software manages are listed below.

- Gain greater insight into the customer's attributes & perceptions about the brand name of a company.
- Identify threats and opportunities to manage reputation.
- Immediately flag, assign and engage customers into discussion and product buying.

Some functionalities offered by such software include tracking the number of mentions per day ; i.e. look into all possible social networking websites and try to

find all the places where the word is mentioned. They then provide visualizations which include both geographic (people/state) and demographic segmentations (age group/female) so that regional managers get a better view of the areas under their purview as opposed to the performance across the entire nation.

As Goldberg notes in [7], many social networks contain pairs of communities that overlap while not containing each other as a sub-community. Individuals often associate across many different social circles, such as those focused around the workplace, family unit, religious group, or social club. In this case, assuming the hierarchical social structure of the network would lead to missing important information about members attachment to the numerous social circles with which they concurrently interact. Another important issue is that of performing the above mentioned analysis in real-time. For example, as the number of users joining social media sites keep on increasing exponentially, analyzing these networks in real time is becoming all the more difficult. Thus in this thesis, we shall describe a method for detecting overlapping communities in social media networks and how they can be analyzed so to draw inferences from them. Also, we shall show an example of how these communities can be detected in real time by introducing concurrency in the method.

1.3 Methodology

As discussed in the previous section, there are three major issues with social media networks namely membership, scalability and evaluation. We shall be dealing with each of the three issues individually. Membership pertains to node membership in a network, namely either the community or set of communities of which the node is a member. In the literature (as will be further explained in chapter 3) most of the work pertaining to multiple node membership, that is, overlapping communities, has been done post 2009 [8]. Another analogy comes from the fact that Social Media has gained relevance starting post 2007. Keeping in mind the overlapping nature of social networks, we shall use a game theoretic approach to graph clustering method for community detection proposed by Mandala et al. [9]. The uniqueness of this method is that it can be used to detect overlapping as well as distinct communities. To test this algorithm we use a Facebook100 dataset used

by Traud et al.in [10, 11]. This dataset presents the social network of 100 Universities in the United States. Once these communities are detected, we propose a novel

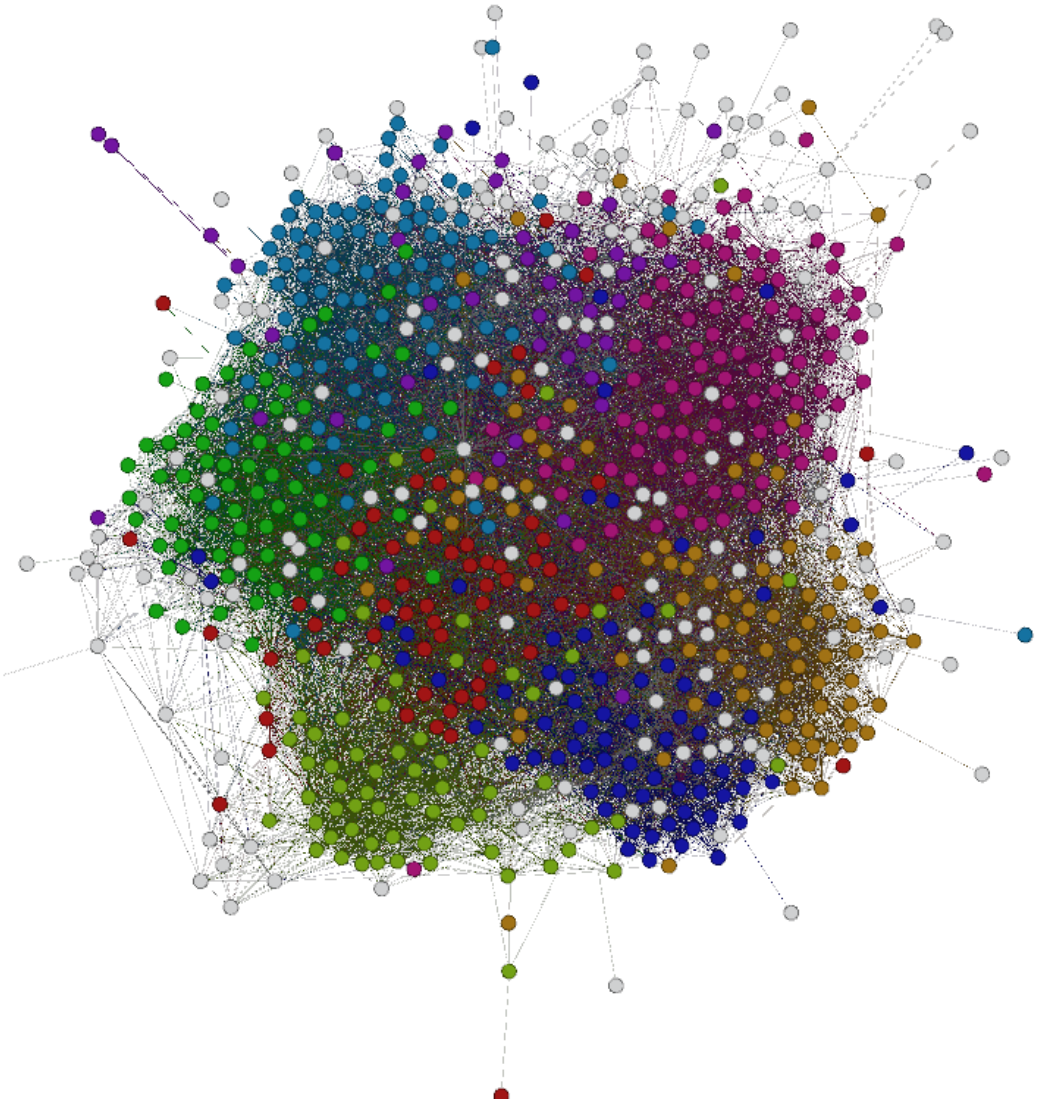


Figure 1.3. Caltech's Facebook network, part of the Facebook100 dataset

approach to evaluate these communities to find commonality of attributes between communities. This tackles the evaluation issue. Finally we tackle the scalability issue by successfully testing a fast implementation of the above mentioned algorithm using parallel programming. This shows how a parallel implementation of an algorithm can work wonders in terms of computational complexity as opposed to its sequential implementation.

1.4 Contributions

A mentioned in the previous section, we aim at tackling three main problems in community detection in social media namely membership, scalability and evaluation. We provide the following contributions tackling each of the above mentioned issues.

1. Provide a comprehensive literature review covering applications of community detection algorithms in social networks covering disjoint as well as overlapping communities.

2. Fast and efficient implementation of a community detection algorithm using parallel programming concepts.

3. Define an entropy-based method to evaluate the commonality of attributes within communities.

4. Provide topics for further research.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 gives an overview the emergence of network science, social media and how it can be used to analyze various inherent communities. It explains how network science has evolved over the years and has had an impact on social media by providing some perceivable real world networks. Finally, we give an overview about community detection and its use in social media and conclude the chapter by providing a formal definition to the thesis.

Chapter 3 is dedicated to background research work. It gives an overview of community detection without any specific application. It explains the need for overlapping community detection as opposed to disjoint communities and provides literature related to detecting overlapping communities in social networks. In chapter 4 we give the importance of finding overlapping communities in networks by giving a brief description of various algorithms dedicated to the same already present in the literature. We then explain in detail two algorithms which we shall

be using for community detection in our analysis. Concluding sections deal with computational aspects i.e. the running time of the algorithms and how it can be used using sophisticated techniques such as parallel programming.

Chapter 5 is dedicated towards analyzing social media networks in general Facebook®networks of 100 U.S. universities consisting of students, faculty and staff as individuals. It mainly explains how the experiments are designed for our analysis and the data. The chapter concludes with a summary of results. Chapter 6 discusses the summary of results, contributions and topics for further research.

Chapter 2

Problem Discussion & Definition

Social scientists have been studying social networks to understand the underlying behavior behind formation of communities and preferential attachments amongst people. Social scientists define a community to be "A group of people who form relationships over time by interacting regularly around shared experiences, which are of interest to all of them for varying individual reasons" [12]. The emergence of social media from a technological perspective has allowed scientists from many disciplines to study these networks from different perspectives including the mathematical foundations behind the dynamics of social networks. In this chapter we shall briefly explain the need to use network science to analyze social networks and then describe the importance of detecting underlying communities in these networks. Finally building on the discussions from the previous sections we conclude this chapter by providing a concrete definition to the problem as discussed earlier.

2.1 Network Science

A network is defined as an interconnected system of things or people. Network Science is the study of systems mainly using their network structure or topology. Networks are represented by a graph of the form G(N,E) where N represents the number of nodes and E represents the number of edges or arcs in the network. Network science is a new and emerging scientific discipline that examines the interconnections among diverse physical or engineered networks, information networks,

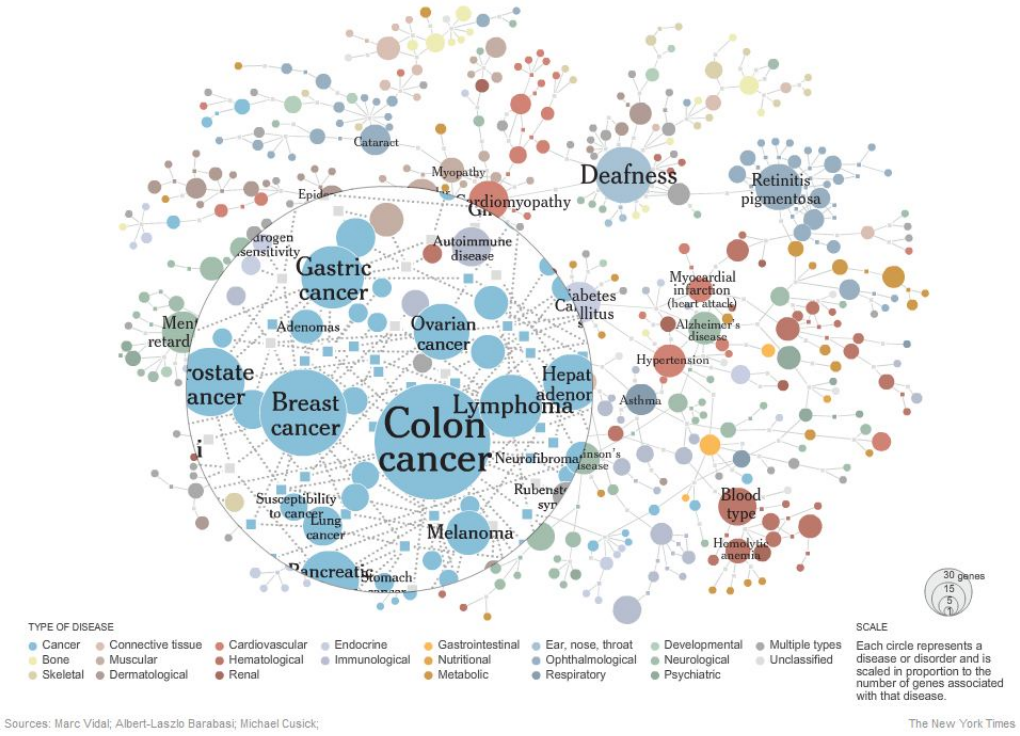


Figure 2.1. Researchers created a map linking different diseases, represented by circles, to the genes they have in common, represented by squares.

biological networks, cognitive and semantic networks, and social networks. This field of science seeks to discover the common principles, algorithms and tools that govern network behavior. The National Research Council defines Network Science as "the study of network representations of physical, biological, and social phenomena leading to predictive models of these phenomena."[1].Most of the complex structures observed in day to day life can be modeled as networks to study their properties and behavior. There are various types of networks in this world starting from a micro level cellular networks to a macro level social networks. Networks can be classified into two major categories; namely Natural Networks and Engineered Networks.

[1]www.wikipedia.org

2.2 Real World Networks

Real world networks are a representation of a system or a phenomenon which has occurred in nature and has evolved over a period of time without any artificial disturbance. These include networks such as:

1. **The Internet** It is a network of computers and routers connected by wired and wireless links. The study takes place namely at the router level and the autonomous level. The number of nodes at the router and domain level were 150000 in 2000 and 4000 in 1999[13].

2. **WWW** In this network a web-page represents a node and a hyper-link connecting two web pages is an edge. A recent study estimated the size to be 11.5 billion in January 2005[13]. With a large amount of data transferred and collected everyday by satellites and other sensors, retrieval from WWW is an interesting research topic. Further details can be found algorithms such as Page Rank [14] or those proposed by Kleinberg [15].

3. **Cellular Networks** Molecules of a cell are represented as nodes and biochemical interactions or regulatory relationships between the molecules are represented as edges [13, 16].

4. **Neural Networks** Nodes are represented as neurons and an existing chemical or electrical synapse represents an edge. Neural Network study is used to understand the functioning of the brain, mainly its capacity to store and process information [17].

5. **Scientific co-authorship** Scientists act as nodes and an edge exists between two scientists if they have worked on the same paper together. The *Erdos Number* project is used to study the co-authorship structures of successful scientists[18, 19].

6. **Movie actor collaborations** In this network, movie actors are nodes and edges represent the appearance of pairs of actors in the same movie. Research interests include the study of combination of actors that makes a successful movie [20].

Types	Examples
Blog	Wordpress, Blogspot, LiveJournal, BlogCatalog
Forum	Yahoo answers, Epinions
Media Sharing	Flickr, You Tube, Justin.tv, Ustream, Scribd
Microblogging	Twitter, foursqare, Google buzz
Social Networking	Facebook, MySpace, LinkedIn, Orkut, PatientsLikeMe
Social News	Digg, Reddit
Social Bookmarking	Del.icio.us, StumbleUpon, Diigo
Wikis	Wikipedia, Scholarpedia, ganfyd, AskDrWiki

Table 2.1. Various Forms of Social Media

2.3 Emergence of Social Media

Over the past decade we have witnessed the emergence of Web and social media, thus bringing people, devices and ideas closer in many new ways. Millions of users are playing, tagging, working, and socializing on-line, demonstrating new forms of collaboration, communication, and intelligence that were hardly imaginable just a short time ago. A number of web applications and social networking sites have been cropping up, drawing people together and empowering them with new forms of collaboration and communication. Traditionally, media such as TV, radio, movies, and newspapers had a selected number of broadcasters who used to decide upon which information to be produced and how it should be distributed to the masses. The majority of users are consumers who are separated from the production and broadcasting process. The communication pattern in the traditional media is one-way traffic, from a centralized producer to widespread consumers. A user of social media, however, can be both a consumer and a producer. With hundreds of millions of users active on various social media sites, everyone could be a media outlet. Table 2.1 shows a list of various social media websites that are currently present in the market. Although they are classified into different types, the underlying factors of freedom to broadcast information remain the same amongst all.

During the recent uprising in Egypt, the outrage, animosity and antagonism against the current regime of Hosni Mubarak was best followed real-time at #Cairo and #Egypt on Twitter. The influence of mass effervescence, the conviction that the protesters display and its myriad manifestations is translating into what is being seen and reported on the streets of Egypt. Thus, social media namely Twitter,

Rank	Site	Rank	Site
1	google.com	11	***blogger.com***
2	***facebook.com***	12	msn.com
3	yahoo.com	13	***myspace.com***
4	***youtube.com***	14	go.com
5	amazon.com	15	bing.com
6	***wikipedia.org***	16	aol.com
7	craigslist.org	17	***linkedin.com***
8	***twitter.com***	18	cnn.com
9	ebay.com	19	espn.go.com
10	live.com	20	***wordpress.com***

Table 2.2. Top 20 Websites in the US

Facebook and YouTube brought forth a revolution which had a consequential role to play in Egypt.The success of social media relies on the participation of users. More user interaction encourages more user participation, and vice versa. For example, Facebook claims to have more than 600 million active users[2] as of August, 2011. The user participation is a key element to the success of social media, and it has helped push eight social media sites to be among the top 20 websites as shown in Table 2.2(Internet traffic by Alexa on August 3, 2010). Users are connected though their interactions from which networks of users emerge. Novel opportunities arise for us to study human interaction and collective behavior on an unprecedented scale and many computational challenges ensue, urging the development of advanced computational techniques and algorithms.

2.4 Social Networks

A network constructed using the interactions between people is termed as a social network. Usually a social network consists of People as nodes and some form of previous interaction between them serves as an edge between two people in the network. This interaction could range from one person knowing another to two people sharing the same interest. More formally, a social network is a social structure made of nodes (individuals or organizations) and edges that connect

[2]http://www.facebook.com/press/info.php?statistics

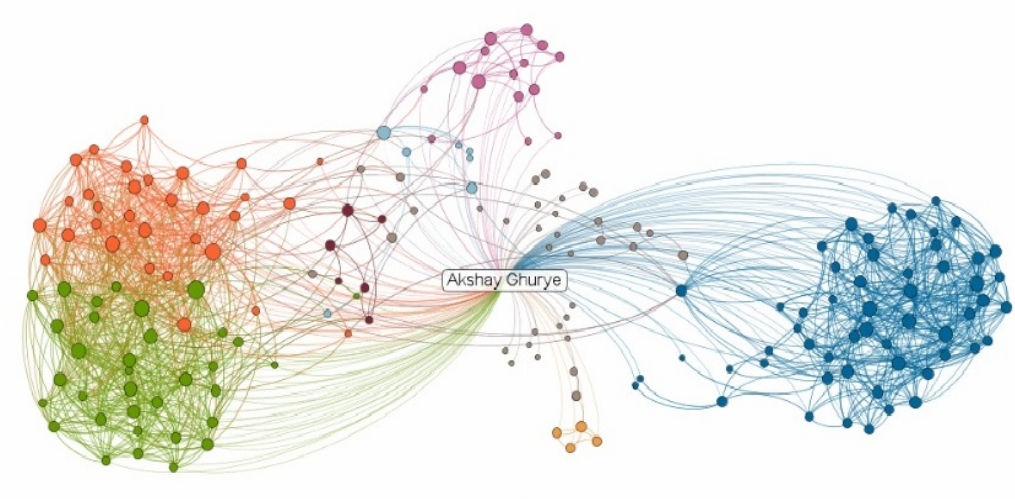


Figure 2.2. LinkedIn Social Network for Akshay Ghurye

nodes in various relationships like friendship and kinship. Thus in a generic sense, a social network can be constructed as based on the requirement specifications of the analysis. The network structure and topology changes from user to user based on what or how one wants to extract or utilize the social interactions within the network. Thus, we can say that Social Networks are a part or subset of a larger set of networks which form the Social Media. Millions of users are playing, working, and socializing on-line. This flood of data allows for an unprecedented large-scale social network analysis millions of actors or even more in one net- work. Examples include email communication networks [21], instant messaging networks [22], mobile call networks [23], friendship networks [24]. The next question that needs to be answered is how should one use this rich interaction data which captures users' sentiment which is otherwise hard to tap into? As these large-scale networks combined with unique characteristics of social media provides a unique opportunity for researchers and practitioners alike to extract meaningful insights which are otherwise not possible by using sophisticated techniques such as data mining and machine learning. We now some properties of social networks [3].

- ***Scalability*** Facebook itself has 600 million users with more than 10 billion connections between them. Thus, the sheer size of these networks impedes the use of existing network science techniques.

- ***Heterogeneity*** Multiple types of entities can be involved in one network. For many social bookmarking and media sharing sites, users, tags and content are intertwined with each other, leading to heterogeneous entities in one network. Analysis of these heterogeneous networks involving heterogeneous entities or interactions requires new concepts.

- ***Evolution*** People update their status, profile and express thoughts and views in a regular and a timely manner thus making these networks temporal. For example, in content sharing sites and blogosphere, people quickly lose their interest in most shared contents and blog posts. New users join in, new connections establish between existing members, and senior users become dormant or simply leave. Capturing these dynamics requires the network to behave in a temporal fashion thus calling for temporal network analysis [25].

- ***Collective Intelligence*** In social media, people tend to share their connections. The wisdom of crowds, in the form of tags, comments, reviews, and ratings, is often accessible. The meta information, in conjunction with user interactions, might be useful for many applications. It remains a challenge to effectively employ social connectivity information and collective intelligence to build social computing applications.

- ***Evaluation*** Privacy and lack of ground truth are a few frequently encountered problems for many social computing tasks, which further hinders some comparative study of different works. Thus evaluation of results following social network analysis is usually the most challenging part.

Social networks of various kinds demonstrate a strong community effect. That is, actors in a network tend to form closely-knit groups which interact more frequently with members within the group than those outside the group. Detecting cohesive groups in a social network (i.e., community detection) remains a core problem in social network analysis. The importance and necessity of the same is described in the following section and will be the main theme of this thesis.

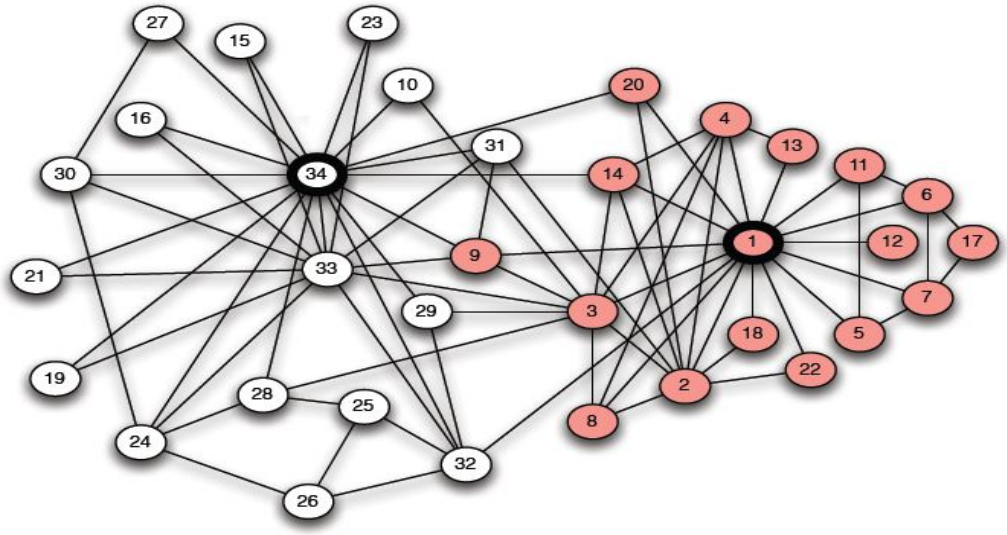


Figure 2.3. Zachary's Karate Club. Figure taken from [1]

2.5 Community Detection

Communities as introduced above are groups also called clusters, factions, cohesive subgroups or modules in different contexts of nodes or individuals who tend to interact with each other on a regular basis. Traditionally, "the founders of sociology claimed that the causes of social phenomena were to be found by studying groups rather than individuals" [26, 3]. The concept of community detection has been present in the network science literature for a long time. One of the oldest known example of community detection in social networks is that of Zachary's karate club [27]. On applying certain community detection algorithms, it was found that the group was divided into two factions where the central or highly influential people in the groups were the karate club master and his student. On evaluating the result with the ground truth it was revealed that there was a conflict between the master and his student which led the group to be split into two communities. Thus from the social network of friendships we can find clues to the latent schism that eventually splits the network into separate groups as shown in Figure 2.3.

Community detection is a very handy tool and is being used by a lot of companies to better understand their customers. For example, grouping customers with similar interests in social media renders efficient recommendations that ex-

pose customers to a wide range of relevant items to promote sales and improve revenues. Communities can also be used to visualize a large network, in terms of groups thus enabling problem solving at group level, instead of node level. In this thesis, we shall delve more into the concepts of community detection by looking into sophisticated ways of detecting inherent groups.

Thus by taking into perspective the concepts explained in this chapter, the following section formally defines the thesis.

2.6 Problem Definition

The emergence of social media has resulted in collection and storage of large amounts of data capturing people's sentiments, beliefs, attributes, actions and preferences. Due to the enormity of data, using mathematical tools for analysis involves extreme computational complexity thus making these tasks more difficult to perform. In this thesis we specifically consider the problem of community detection within social networks and propose a fast algorithm using game theoretic approach to graph clustering based on the one proposed by Mandala et al. [9]. In order to reduce the computational complexity we deploy parallel programming using threads in the above mentioned algorithm. Once the communities are detected we propose an entropy based approach to find commonalities amongst nodes in a given community and thus across communities. This approach in general can be extended to analyze any community once it has been detected provided the nodes have network independent attributes. Eventually we evaluate these two algorithms on Facebook social networks from 100 universities to determine the consistency among internal communities spanning across different universities.

Chapter 3

Literature Review

Community detection has proven to be valuable in a series of domains such as web search, biology, bibliometrics and because of the emergence of social media, its application in the field of social network analysis is gaining importance day by day. In this chapter, we shall first explain a few concepts which are present in the social network literature and then give a comprehensive summary of the literature currently present in the field of community detection applied to social media. In this thesis, we have used the words "social networks" and "social media networks" interchangeably. Social networks are a part of social media networks and can hence be abstracted from them.

3.1 Community Detection

Social media networks provide an elegant representation of social media data containing on-line objects as their vertices and the relationships/interactions among them as edges. The vertices of these networks can represent different types of objects such as user profiles, content items (e.g. blog posts, photos, videos) and even metadata items such as topics, categories, tags. The edges of a social media network can be different types such as simple, weighted and directed. Due to the abundance of related works and the variety of adopted perspectives, there is no unique and widely accepted definition of a community. Community definitions are formulated with reference to the network structure of the system under study and are commonly bound to some property either of some set of vertices or of the

entire network. Also, one must define a community with respect to the domain under observation. In this thesis, we have restricted ourselves to the domain of social media networks. Social media networks consist of millions of vertices and edgesand share some common patterns such as *scale-free distributions, the small world-effect, and strong community structure* [3]. At the most abstract level, given a network $G = (V, E)$ a social community can be defined as a subgraph $V_C \subseteq V$ of the network comprising of a set of entities that are associated with a common element of interest. These entities can be varied as a real-world person, topic, a place or an activity. For example, in a blogging network, the set of all bloggers articles, tags and comments related to a specific topic constitutes to a community. These communities can be *explicit* or *implicit* [28]. Explicit communities are created as a result of human decision and acquire members based on human consent. Examples of explicit social communities are $Facebook^{®}$ and $Flickr^{®}$ groups. Implicit communities on the other hand are assumed to exist in the system but need to be discovered. Implicit communities enable the study of emerging phenomenon within social systems and hence are studied more extensively than explicit communities. Social media networks include networks formed using blogs, tweets and feeds that users post on social media sites such as $Twitter^{®}$ whereas social networks are those formed by connecting user profiles on social networking sites such as $Facebook^{®}$. Although the literature review has been done pertaining to social media networks, in this thesis we shall focus on finding implicit communities present within social networks.

3.2 Definition of a Community

Over the years, researchers have tried to define the essence of a community based on the network structure but till date no concrete definition has been established for the same [9]. We shall now provide some of the globally adopted definitions. The most established notion of community-ness within a network is based on the principle that some sets of vertices are more densely connected to each other than to the rest of the network. Communities are classified according to the network structure as local (connected subset of vertices) or global (the entire network). In this thesis we shall summarize the various definitions as presented in [28] but more

detailed definitions are presented in [29]. Definitions of local communities emerged from social studies focusing on the concepts of subgroup cohesiveness and mutuality. Some of these definitions are cliques, n-cliques, n-clubs, n-clans, k-plexes, k-cores, LS and Lambda Sets as shown in [30, 31, 32]. Most of these definitions are strict in its own sense and hence have very limited application in social networks. Based on a node level, the internal degree and external degrees of subgraphs have been used to define a community. The internal degree of a node is the number of edges that connect it to vertices of the same community. The external degree is defined as the number of edges that connect it to the vertices outside the community. Other such local definitions include local and relative density [33], local modularity [34] and subgraph modularity [35]. Moving onto global definitions of communities, these relate to the communities within the network as a whole as opposed to a set of nodes in the network. At the outset, we want communities to be of the form that the number of edges between nodes in a given community are comparatively more than those between nodes of different communities. On these lines, calculating the actual number of inter-community edges is complex, normalized measures such as normalized cut [36] and conductance [37] have been used for quantifying the amount of separation between communities. The concept of *modularity* as explained by Newmann and Girvan in [38] relates to a global community structure as opposed to individual communities. It quantifies the extent to which a given partition of a network into communities deviated from a random network with the same degree distribution. Another global community definition is based on a similarity measure between nodes in a network. Examples of these include embedding nodes in n-dimensional Euclidean space and then use a distance measure such as Manhattan distance or Euclidean distance to cluster nodes which are closer to each other. One measure is to compute the Pearson correlation between the rows of adjacency matrix or random-walk based similarities [28, 39] another one being the overlap between the neighborhoods of two vertices given by structural equivalence [40]. The above mentioned methods relate to defining a community after they are found. An alternative means of defining a community is by considering some community formation process taking place on the network under study. This is given by the clique percolation method [41] which considers a k-clique template that adds on in a network thus forming a community consisting

of the union of all k-cliques that are adjacent to each other. Having given an overview of a community, we now proceed to review some community detection algorithms found in the literature.

3.3 Community Detection Algorithms

Community detection algorithms have been in existence since the foundations of Graph Theory. Traditionally algorithms used for graph partitioning were used for community detection mainly because of the underlying similarity in finding disconnected subgraphs. Community detection is different from graph partitioning mainly because neither the number of subgraphs nor their sizes are required as an input and the resulting communities may not be a partition (overlapping communities). On the other hand graph clustering is another aspect which draws similarity to community detection and has been used interchangeably with community detection with the only distinction being the number of communities need not be an input in community detection. The immense scale and evolving nature of social media makes it impossible apriori to estimate the number of communities in a social network. Although the number of community definitions are large, the number of methods for detecting communities are even larger. We now present a brief summary of existing community detection methods in the literature based on a classification schema as presented in [28].These are namely (a) cohesive subgraph discovery, (b) vertex clustering, (c) community quality optimization, (d) divisive, and (e) model-based. Although a large number of algorithms can be conveniently classified under each of these groups, only a few algorithms have been reviewed in this chapter.

1. **Cohesive Subgraph Discovery** As the name suggests, the communities detected by these methods must satisfy certain structural property constraints to be considered as valid communities. The structural properties a community should satisfy include k-cliques, weak/strong communities. The clique percolation method [41] falls under this class.

2. **Vertex Clustering** The types of methods in this class are related to traditional clustering methods such as k-means clustering and hierarchical ag-

glomerative clustering. Nodes in the network are assumed to be vectors in space where pair wise distances between vertices can be calculated. An example of this class is the spectral graph clustering method presented in [42] which uses the spectrum of the graph for mapping graph vertices to points in a low-dimensional space where the cluster structure is much simpler to evaluate.

3. **Community Quality Optimization** The methods in this class work on the concept of optimizing some graph based measure of community quality. The definition of modularity as described in [38] is an example of a graph based measure of community quality. Approximate modularity maximization algorithms such as extremal optimization [43], simulated annealing [44] and spectral optimization [45] are examples of community quality optimization algorithms.

4. **Divisive** These methods rely on identification and removal of network elements such as edges and vertices that are positioned between communities. Edge betweeness is a property which describes the importance of an edge (i.e. its importance in connecting two communities such that removing the edge will result in separate subgraphs) in the network. Divisive algorithms aim at removing edges with high betweeness values. The first algorithm was proposed by Girvan and Newman [46] which progressively removes the edges of a network based on edge betweeness measure until communities emerge as disconnected components of the graph.

5. **Model-based** This is a broad category of methods that either considers a dynamic process taking place on the network which reveals its communities or an underlying model of statistical nature that can generate the division of the network into communities. The label propagation algorithm by Raghavan et al., [19] generates communities by grouping together nodes with similar labels, thus extracting communities by allowing nodes to take on labels or join communities which most of its neighbors belong to.

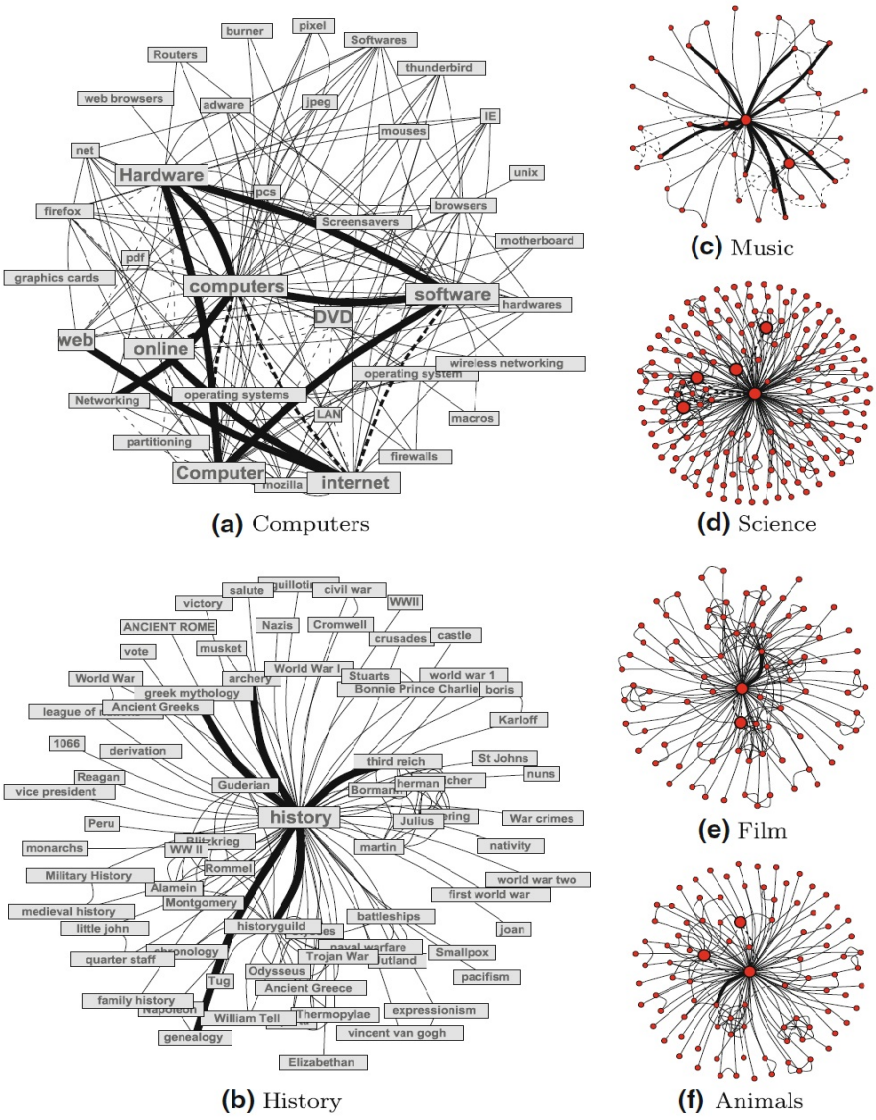


Figure 3.1. Examples of tag communities discovered by the local method of Papadopoulos et al. [2]. The presented communities were created using computers, history, music, science, film and animals as seed nodes.

3.4 Community Detection in Social Media

The large amount of tags attached to online content by users of Social Media applications creates the need for imposing organization on the flat tag spaces of collaborative tagging applications. This can be achieved by grouping tags based on the topic they are associated with. Recently Papadopoulos et al., [2] in their work defined an efficient community detection scheme that could discover the commu-

nity around a seed tag. Figure 3.1 presents several examples of tag communities discovered on a tag network created from LYCOS iQ community question answering application. The domain of personalized search and recommendation is growing day by day with search engines trying to provide as detailed and specific information as possible to a given user. More specifically, community detection can aid personalized search with clusters of tags acting as effective proxies of user's interests. Tsatsou et al [47] integrate the results of tag community detection in a personalized ad recommendation system and compared against conventional nearest-neighbour tag expansion schemes [28]. Events constitute an important unit of organization for social media content since a large part of user contributed content revolves around real-world events. Community detection has found applications in the detection and tracking of events from social text streams. The framework provided by Zhao et al. [48] incorporates textual, social and temporal aspects of blog feeds with the goal of tracking events. The N-cut graph partitioning method of Shi and Malik [36] is used twice in this framework: once to cluster a graph of blog posts connected by their textual similarity into topics, and at a second level, to cluster a graph of temporal activity profiles among users into communities that correspond to real world events. The heterogeneity present in network connections can hinder the success of collective inference. People can connect to their relatives, colleagues,college classmates, or some online friends. These relations play different roles in helping determine targeted behaviors. For instance, the *Facebook*® contacts of the first author in [3] can be seen in three key groups, as shown in Figure 3.2 friends at Arizona State University (ASU), undergraduate classmates at Fudan University, and some high-school friends in Sanzhong [3]. For example, it is reasonable to infer that his friends at ASU are presumably interested to know if Lei (first author) is watching an ASU football game, while his other friends at Fudan and Sanzhong are probably indifferent to his excitement. In other words, users can be involved in different relations, or a social network can consist of heterogeneous relations. Therefore, it is not appropriate to directly apply collective inference to this kind of networks as it does not differentiate these heterogeneous relations. It is recommended to differentiate relations for collective classification. In [10, 11] the authors study the structure of social networks of students by examining the graphs of *Facebook*® friendships at five American uni-

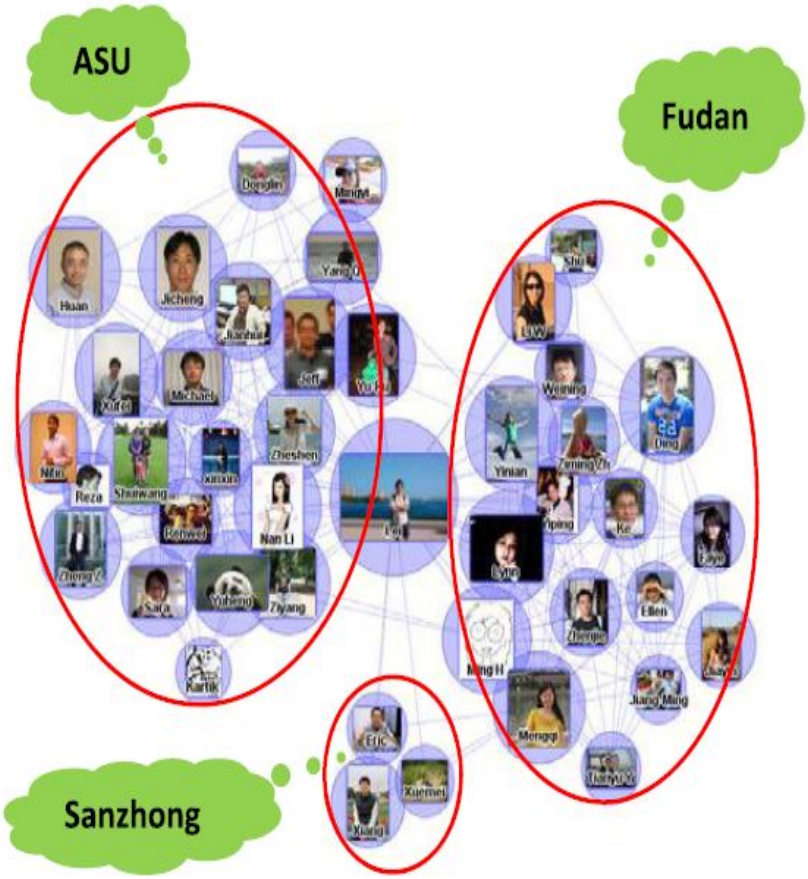


Figure 3.2. An example of a users contacts in *Facebook*®, involving three different relations: friends met at ASU, undergraduate classmates at Fudan University, and some high school friends at Sanzhong [3].

versities at a single point in time. They investigate each single-institution networks community structure and employ graphical and quantitative tools, including standardized pair-counting methods, to measure the correlations between the network communities and a set of self-identified user characteristics (residence, class year, major, and high school).Their study illustrates how to examine different instances of social networks constructed in similar environments and emphasizes on the fact that the array of social forces that combine to form communities, leads to comparative observations about online social lives that can be used to infer comparisons about offline social structures.

3.5 Overlapping Community Detection in Social Networks

We now discuss some background work which is done in detecting overlapping communities in social networks. In [49] the authors propose an algorithm to find overlapping communities by aggregating the community perspectives of friendship groups derived from egonets. One plus point about their algorithm is that once the overlapping communities are found, one can also identify key nodes which bind the communities together. Nguyen et al. [50] propose a DOCA algorithm which is very imited parameter dependent and only requires local knowledge about the network topology. The communities that it detects is deterministic i.e. there is no fuzziness involved in the overlap. They demonstrate the performance on a few social networks and their results suggest that overlapped nodes tend to be active users who participate in multiple communities at the same time. A similar algorithm is proposed by Padrol-Sureda et al. in [51] to find overlapping communities faster. Pizzuti in [52] proposes a genetic algorithm which optimizes a fitness function to identify densely connected groups of nodes by employing it on the line graph corresponding to the network. Goldberg et al. take a different perspective at community detection in their paper [53]. They gather a set of axioms from previously published literature to define overlapping communities and modify a previously existing algorithm Iterative Scan [54] to satisfy these constraints. Moving away from a social network perspective, Sawardecker et al. in [55] apply three group identification methods namely modularity maximization, k-clique percolation and modularity-landscape surveying to a set of network ensembles. They find that modularity landscape surveying method detects the heterogeneity in node memberships better than the other two. A game theoretic framework for overlapping communities is provided by Chen et al. in [56]. Using the concept of risk, reward and utility, nodes act as players and try to maximize their own utility function by either joining or leaving communities untill an equilibrium is reached. Psorakis et al. [57] use a Bayesian nonnegative matrix factorization (NMF) model to detect overlapping communities. Recently, a comprehensive survey done by Xie et al. [58] presents a list of most community detection algorithms present in the literature. They also propose a framework to evaluate an algorithm's ability to assess the

extent of detection. That is once communities are formed, they assess percentage of nodes which belong to multiple communities and how many communities does each node belong to. Their results on various networks suggest that less than 30% of nodes are present in overlap and most of them belong to 2 or maximum 3 communities. Finally Gregory proposes a two-phase method for detecting overlapping communities [59]. In the first phase the network is transformed by splitting vertices and then using a disjoint community detection algorithm already present in literature to detect communities. This allows algorithms which provide hard clustering to be used to find overlapping communities within networks. In chapter 4 we will focus on the fast model based community detection by Mandala et al [9]. Most of the discussion in social networks need to be taken in real time, thus necessitating a faster real-time algorithm.

Chapter 4

Methodology

Community detection in social networks is gaining importance day by day. Having described various algorithms and methods present in the literature in the previous chapter, this chapter goes on to extensively describe another algorithm which can be used to find overlapping clusters. We first explain the importance and uniqueness of the algorithm followed by the algorithm and finally the computational complexity aspects of the same.

4.1 Introduction

As explained in the previous chapters, we have seen the gaining importance of social media analytics and how community detection can be used to extract previously unknown details existing within network interactions. All of the algorithms described in the previous chapter were used to detect strict clusters i.e. a particular node can belong to only one community or cluster depending on the algorithm being used. But in today's social networks, people tend to belong to various groups simultaneously. For example, in a friendship network between various individuals, an individual may belong to more than one community simultaneously namely, his college network, his school network and may be his work place network. Thus overlapping communities allows us to see nodes which act as bridges between two or more communities. There have been several algorithms to detect overlapping clusters. One of the most popular technique for detecting overlapping clusters is the Clique Percolation Method (CPM)[41]. Their algorithm starts by enumerating

all k-cliques which are cliques of size k. Then, a cluster is defined as a collection of adjacent k-cliques. Any two k-cliques are adjacent if they share $k-1$ vertices. Li et al [60] propose a two-phase algorithm to cluster named entities - people, organizations, etc. The first phase is similar to CPM (with $k=3$). In the second phase, any remaining cliques are combined based on the content. Another interesting approach based on seed expansions is proposed by Wei et al [61]. The main idea is to compute initial non-overlapping clusters using existing spectral modularity optimization techniques. Then each of seed clusters are expanded in a locally optimal sense leading to overlapping clusters. Although both these methods do give us overlapping communities, they are computationally taxing. For our analysis we and use the algorithm proposed by Mandala et al. [9] which utilizes a game theoretic approach to graph clustering where nodes act as players and the algorithm converges to a Nash Equilibria. The reason we choose this algorithm is because it is computationally more efficient and can be altered to find overlapping as well as strict communities. We now discuss the algorithm in detail.

4.2 Game-Theoretic Algorithms

The game theoretic approach to graph clustering algorithm presented by Mandala et al. [9] has two forms namely Sequential Best Response (SEBR) and Simultaneous Best Response (SMBR). The general outline of both the algorithms is as follows. One provides network parameters as the input. The nodes of the network act as players and try to compute their best response by calculating their utility based on a certain cost and reward. The reward is based on joining a particular community by taking up the community label and the cost is based on how many players from the entire network have joined the community. The reward function is given by

$$r_v(l; a_{-v}) = \sum_{w \in N_v} |l \cap a_w| - \rho \sum_{w \in V - v} |l \cap a_w| \tag{4.1}$$

The first term in the summation equals the vertex taking up a neighbouring label and the second term maintains the cluster density as the vertex adds labels. The

cost function $c_v(a_v)$ is given as follows

$$c_v(a_v) = \frac{1}{2}\lambda_v |a_v|(|a_v| - 1)$$

which is the total number of vertices which have the corresponding candidate labels. The utility of vertex v, U_v, is obtained by combining reward and cost as shown in Equation 4.2.

$$U_v(a_v; a_{-v}) = \sum_{l \in a_v} r_v(l; a_{-v}) - c_v(a_v) \tag{4.2}$$

4.2.1 Sequential Best Response (SEBR)

As mentioned we first assign all the nodes with unique labels. Then we start with the node labeled 1 and calculate the best response for node 1. If the node changes its label based on the best response then we say that a node has updated its label else it keeps its original label. Once node 1 has updated its labels based on its utility, node 2 calculates its best response taking into consideration label 1's updated label. After all the labels compute their best response and new labels, we calculate the number of nodes that have updated their label from their original label. If the number of updates is more than zero then we recompute the best responses for all the nodes based on the new labels. We compute this process till no more updates are possible. The pseudo code for the algorithm is described in Algorithm 1

Algorithm 1 SEBR: Sequential Best Response [9]

Input: $G = (V, E)$, ρ, λ_v, *vertex ordering rule*
Output: $a_v^* \; \forall v \in V$
set $a_v = \{l_v\} \quad \forall v \in V$
All vertices are ordered according to *vertex ordering rule*
loop
  Each ordered vertex computes its best response a_v^{BR} and updates its action $a_v \leftarrow a_v^{BR}$
  If there are no updates then Exit the loop
end loop
return $a_v^* = a_v$

4.2.2 Best Response Calculation

The best response here relates to a node taking up a label which gives the best reward. The node keeps on accepting labels till the utility function no longer provides a positive return. The best response is a set of labels for a given node is a set of labels which give a positive reward. The algorithm converges when no more best responses are possible for all the nodes or alternatively no node will give up his label. Algorithm {Algo:OBR details the steps required to compute the best response, where the input to the algorithm is a node, a set of potential labels that it can take along with a few network parameters the details of which are listed in [9]. The node keeps on adding nodes till the utility is positive.

Algorithm 2 Optimal Best Response [9]

Input: a_{-v}, ρ, λ_v
Output: a_v^{BR}
Compute $r_v(l; a_{-v})\ \forall l \in L$
Sort labels in the descending order of rewards using any deterministic tie breaking rule
Set $a_v^{BR} = \phi$
for $i = 1 : |L|$ **do**
  if $r_v(l_i; a_{-v}) < \lambda_v(i-1)$ **then**
    break
  end if
  $a_v^{BR} \leftarrow a_v^{BR} \cup \{l_i\}$
end for

4.2.3 Simultaneous Best Response (SMBR)

Similar to SEBR algorithm, we first assign all the nodes with unique labels. Then we start with the node labeled 1 and calculate the best response for node 1. Node 2 calculates its best response without taking into consideration label 1's updated label and so on and so forth. That is, every node's best response is independent of the other nodes current round's best response. Similarly, each node need not update its label. We continue this till the number of nodes that update their labels is zero. The pseudo code for the algorithm is described in Algorithm 3.

Algorithm 3 SMBR: Simultaneous Best Response [9]

Input: $G = (V, E)$, $\rho, \lambda_v, \epsilon$
Output: $a_v^* \; \forall v \in V$
set $\mathtt{T} = \lfloor \frac{1}{1-\epsilon} \rfloor$
set $a_v = \{l_v\} \quad \forall v \in V$
loop
  Set V_{active} (*active vertices*) by randomly selecting vertices from V with probability $1 - \epsilon$
  All active vertices compute their best responses a_v^{BR} simultaneously (*this operation can be parallelized*)
  All active vertices update their actions $a_v \leftarrow a_v^{BR}$
  if no updates for $\mathtt{T}$ consecutive rounds and $\mathbf{a} \in S_{NE}$ **then**
    Exit the loop
  end if
end loop
return $a_v^* = a_v$

4.3 Parallel Programming

In today's world, taking into consideration the amount of time people spend on the Internet, the amount of data being collected every second is enormous. Social networking website, $Facebook^{®}$ has 600 million registered users and almost one third of those are active on a daily basis. Thus, in-order to perform analysis on this graph of 600 million nodes and almost 10 billion edges, it would take years for sophisticated algorithms to analyze it. Another example would be $Youtube^{®}$ which has daily uploads of almost 20 GB per minute, which results in a large number of videos. Thus building a tagging graph from these would produce a graph of almost a billion nodes and few billion edges. Thus, given the scale of these social media networks, one cannot depend on a single computer to perform network based analysis. We would therefore require parallelized algorithms. In this section we explain the basic parallel programming framework and how it can be applied to the above mentioned algorithms so as to reduce the overall execution time as well as address scalability.

4.3.1 Introduction

Many scientific computations require a considerable amount of computing time. This computing time can be reduced by distributing a problem over several processors. Multiprocessor computers used to be quite expensive, and not everybody had access to them. Since 2005, x86-compatible CPUs designed for desktop computers are available with two cores, which essentially makes them dual processor systems. This cheap extra computing power has to be used efficiently, which requires parallel programming. Parallel programming is the simultaneous use of multiple resources to solve a computational problem. While parallel programming is much more feasible, it isn't necessarily any easier. Writing complex and correct multi-threaded programs is hard. There are many details of concurrency - thread management, data consistency, and synchronization to name a few - that must be considered while writing a parallel code. Programming mistakes can lead to subtle and difficult-to-find errors. Because there is no "one-size-fits-all" solution to parallelism, at times one might find himself focusing more on developing a concurrency framework and less on solving the original problem.

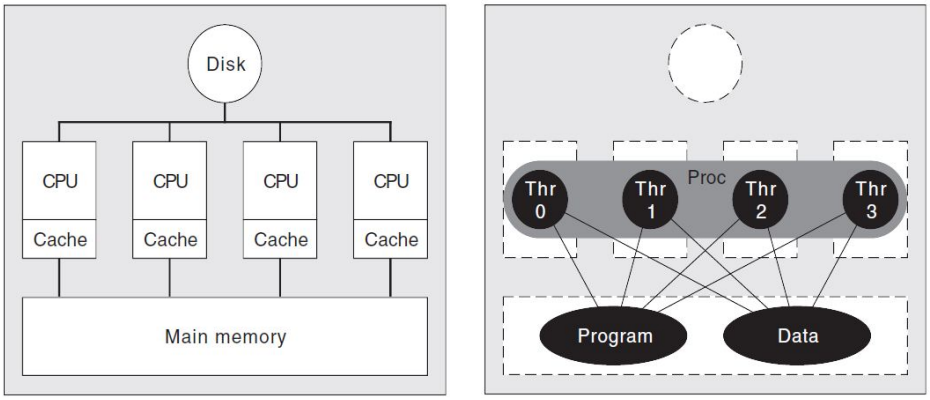
4.3.2 SEBR v/s SMBR

The Sequential Best Response (SEBR) algorithm takes each vertex from the vertex list and calculates the best response for that particular vertex. After calculating the best response for a given vertex i , the vertex i updates its action based on the optimal best response. Now when we shall calculate the best response for vertex $i+1$, the updated label of vertex i will be taken into account. That is the vertices sequentially update their actions once their corresponding best response strategy is calculated. On the other hand the Simultaneous Best Response (SMBR) algorithm works in a slightly different manner. As opposed to SEBR, after calculating the best response for a given vertex i, the vertex i stores its optimal best response but does not update its actions. Thus, while calculating the best response for node $i+1$, the non-updated action of vertex $i+1$ will be taken into account. Once all the best response strategies are calculated for a given game, all the vertices simultaneously update their actions based on their stored optimal best responses. In SMBR, we can see that calculation of the best response for each vertex will be independent of

the other vertex calculation. The property of independent calculations allows us to parallelize this part of the algorithm. Also, one can see from the analysis presented in [1] that the number of Best Response calculations will be more in SMBR than in SEBR thus the computational time, although marginally, is more for SMBR than SEBR. But as the number of vertices increases linearly the computational time will also increase. Hence, by parallelizing the above mentioned part of the algorithm we can decrease the total computational time of the SMBR algorithm.

4.3.3 Parallelization of SMBR

As explained above the computation of best response for each vertex in SMBR is independent of that of other vertices in the network. Thus we can see that this part can be parallelized and run on distributed cores or systems to reduce the computational time on the system. Since we have coded the algorithm in Java for efficient computation, we exploit Java parallelization as opposed to other well known approaches such as MPI, OpenMPI, CUDA,etc. There are multiple ways of performing Java parallelization, namely Fork/Join: Divide-and-Conquer, Pervasive DataRush: Dataflow Programming, Terracotta: JVM Clustering and Hadoop: Distributed MapReduce. Below we present a naive approach for parallelizing the SMBR using threads. A thread of execution is the smallest unit of processing that can be scheduled by an operating system. Usually a thread exists in a process and a process can have multiple threads. Usually threads are used to split a process into parts such that each part is executed separately. Every process runs on a processor. A processor can have multiple cores. Cores are the central processing units of a computer. If a computer has multiple cores then as many threads can be run simultaneously on each core such that the total time is reduced. These threads need to be created/invoked and certain libraries can invoke these threads. Threading is very easy if there is a single motherboard with multiple cores on it but when one uses shared memory and cores from different motherboards then it becomes more difficult to handle the synchronization between these threads. Usually this is done using a framework called message parsing interface. Since, implementation of MPI requires exquisite programming skills as well as knowledge about parallel programming we will not be discussing its implementation in this



(a) (b)

Figure 4.1. (a) SMP Parallel Computer (b) SMP Parallel Program. Figures taken from [4]

thesis.

4.3.4 Parallel Computing Framework

In general, parallel programming can be performed on any of the following settings

1. **Shared Memory Parallel Computers(SMP)** relate to a single CPU with multiple cores and a single memory shared by all the processors as shown in Figure 4.1(a) These are particularly useful when there is a lot of mutual communication between the nodes. A parallel program running on SMP computer consists of one process with multiple threads, one thread executing on each processor. The process's program and data reside in the shared main memory. Because all threads are in the same process, all threads are part of the same address space, so all threads access the same program and data. Each thread performs its portion of the computation and stores its results in the shared data structures. If the threads need to communicate or coordinate with each other, they do so by reading and writing vales in the shared data structures [4]. This is shown in Figure 4.1(b). Suited for small scale programs because as the number of threads increase, the memory requirements increase exponentially.

2. **Cluster Parallel Computers** relate to multiple CPUs with a single core and an independent memory per CPU. These are particularly useful when

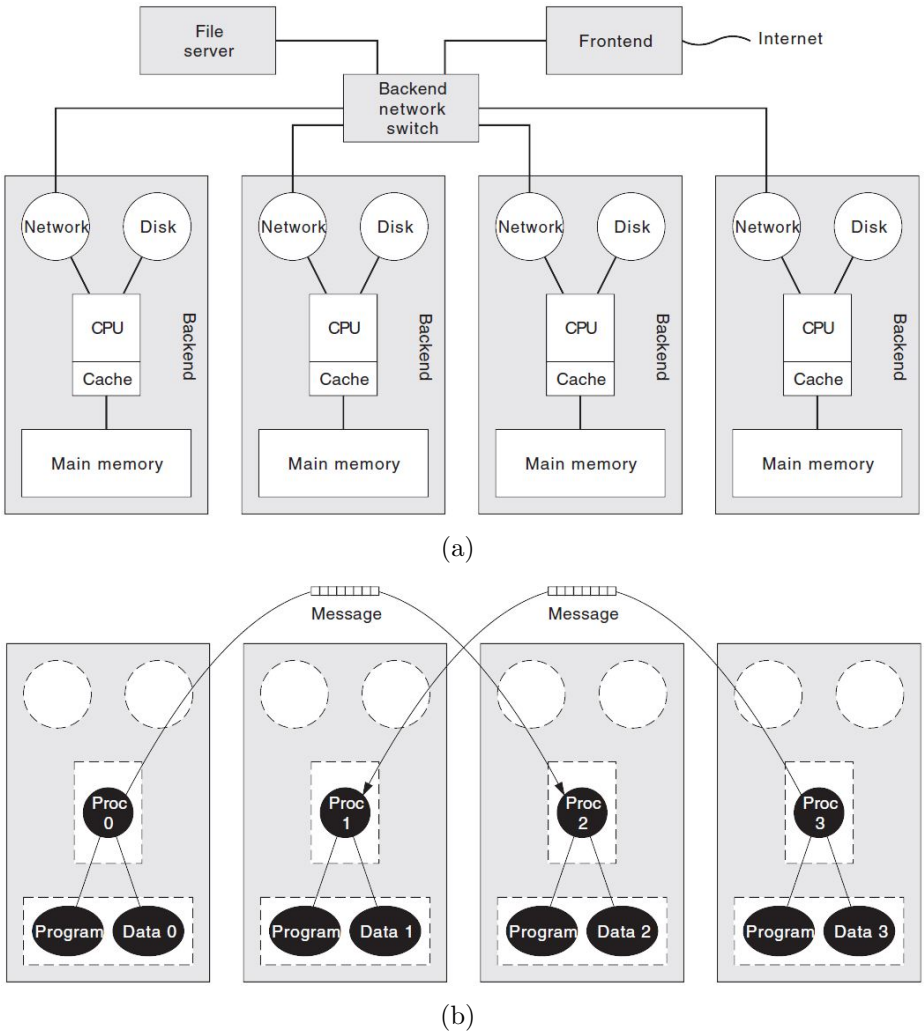


Figure 4.2. (a) Cluster Parallel Computer (b) Cluster Parallel Program. Figures taken from [4]

there is not much communication between nodes during computation. Usually a cluster parallel computer consists of multiple interconnected processor nodes as shown in Figure 4.2(a). There are several back-end processors that carry out parallel computations. There is also typically a separate front-end processor; users log into the front-end to compile and run their programs. There may be a shared file server. Each back-end has its own CPU, a cache, main memory and peripherals, such as a local disk drive. Each processor is also connected to the others through a dedicated high-speed back-end network. Unlike an SMP parallel computer, there is no global shared memory;

each back-end can access only its local memory. Thus, a cluster computer is said to have a distributed memory. A parallel program on a cluster parallel computer consists of multiple processes, one process executing on each back-end processor. Each process has its own, seperate address space. All processes execute the same program, a copy of which resides in each process's address space in each back-end's main memory [4]. This frame work uses a concept called Message Parsing where messages are passed to communicate between independent CPUs as shown in Figure 4.2(b). They are mostly useful for large scale programs with minimal communication between nodes. Although initially the running time decreases, as the number of nodes increase, the overhead memory and computation time required for message passing between the nodes also increases exponentially.

3. **Hybrid Parallel Computers** refer to a combination of the SMP and cluster parallel computers which include multiple CPUs with many cores and each CPU has multiple shared memory. This is shown in Figure 4.3(a). In general, a hybrid parallel computer is a cluster in which each back-end processor is an SMP machine. It has both shared and distributed memory. A hybrid parallel computer is programmed using a combination of cluster and SMP parallel programming techniques. Like a cluster parallel computer, each back-end runs a seperate process with its own address space. Each process executes a copy of the same program and each process has a portion of the data [4]. The usual program runs in the way it is shown in Figure 4.3(b). These computers are usually used for solving very large problems and the only drawback is that they cost a lot of money.

4.4 Computational Complexity

To parallelize SMBR, we need to have either a cluster parallel computer or hybrid parallel computer because, the best response calculation for each node is independent of other nodes in the network. We now explain the reduction in running time for the SMBR algorithm when running in parallel as opposed to its sequential implementation. In order to analyze the computational complexity of both the

algorithms, we need to know (a) complexity of computing a best response and (b) number of best responses computed. The former can be analyzed easily as there are three main operations in Algorithm 2. Firstly, note that only labels adopted by neighbors need to be considered as all other labels will carry negative reward. In order to compute the reward, we need to total the number of neighboring vertices and total the number of vertices who have chosen a given label. The former requires $O(|N_v|)$ operations while the latter is constant time if we maintain and update an label frequency distribution. Secondly, sorting can also be done in linear time, $O(|N_v|)$. Thirdly, $O(o_v)$ computations are required to choose the right number of labels. Therefore, the total complexity of vertex v to compute a best response is $O(2|N_v|+o_v)$. On an average, BR computation requires $O(\bar{d}+\bar{o})$ where $\bar{d}, \bar{o}$ are the average degree and overlap of the graph respectively.

We find that the number of best responses required by SMBR is always greater than that for SEBR. Moreover, the difference increases as the network size grows. Our conjecture is that the number of best responses required vary as $O(|V|log|V|)$ and $O(|V|log|V|^2)$ for SEBR and SMBR respectively. It follows that the overall complexity is nearly linear in number of edges as $\bar{d}|V| = 2|E|$. Thus, we see that the total number of best response calculations vary as $O(|V|log|V|)$ and $O(|V|log|V|^2)$ for SEBR and SMBR respectively. If we implement the SMBR algorithm in parallel then the best response calculations would reduce to $O(log|V|^2)$ that is assuming $O(|V|)$ threads are running simultaneously. This implies $|V|$ computations per iterations are performed simultaneously as opposed to being computed sequentially as before. The computational aspect of community detection will be demonstrated using an example in the next chapter.

4.5 Measuring Parallelization

A program's problem size, N, is the number of computations the program performs to solve that problem. The particular problem's algorithmic structure and input dataset determines how the problem size is measured. The problem size also relates to the computational complexity of the problem at hand. In general, the problem size is defined so that the amount of computation is proportional to N. A program's running time, T is the amount of time the program takes to compute

the answer to a problem. Many factors influence T. The computer's hardware characteristics– such as CPU clock speed, memory speed, caches and so on–affect T; the faster the computer, the shorter the running time. T depends on the problem size; the larger the problem, the longer the running time. Furthermore, the algorithm used to solve the problem determines how quickly T increases as N increases; an $O(NlogN)$ algorithm's running time will not grow nearly as quickly as $O(N^2)$ algorithm's. T also depends on how the program is implemented; the same algorithm can sometimes run faster when coded differently. T depends on the number of processors K; adding more processors reduces the running time. Thus according to [4] we use the notation $T(N, K)$ to emphasize that the running time is a function of the problem size and the number of processors. We use the terminology $T_{seq}(N, K)$ to represent a sequential implementation of the code and $T_{par}(N, K)$. A program's speed S, is the rate at which program runs can be done. Speed is the reciprocal of running time is given by

$$S(N, K) = \frac{1}{T(N, K)} \tag{4.3}$$

where T is measured in seconds per program run, S is measured in program runs per second. A programs *speedup* is the speed of the parallel version running on K processors relative to the speed of the sequential version running on one processor for a given problem size N.

$$Speedup(N, K) = \frac{S_{par}(N, K)}{S_{seq}(N, 1)} \tag{4.4}$$

Thus on substituting Equation 4.3 in equation 4.4 we get the following equation.

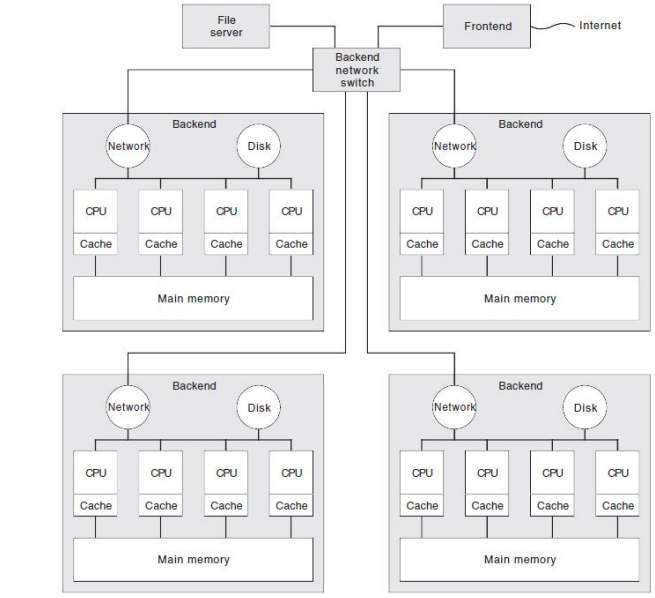
$$Speedup(N, K) = \frac{\frac{1}{T_{par}(N,K)}}{\frac{1}{T_{seq}(N,1)}} = \frac{T_{seq}(N, 1)}{T_{par}(N, K)} \tag{4.5}$$

Ideally a parallel program should run 2 times faster on 2 processors and 4 times faster on 4 processors. Thus the *speedup* should equal K. This is the ideal case which in reality may not happen. This is measured by a metric known as *efficiency*

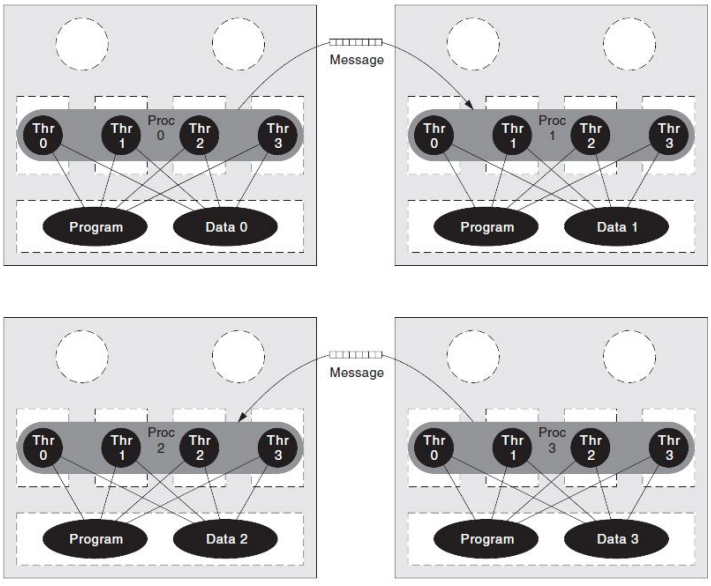
which captures how well the program runs as compared to an ideal case.

$$Efficiency(N, K) = \frac{Speedup(N, K)}{K} \tag{4.6}$$

We will be using speedup and efficiency to check how well our program runs when implemented in parallel.



(a)



(b)

Figure 4.3. (a) Hybrid Parallel Computer (b) Hybrid Parallel Program. Figures taken from [4]

Chapter 5

Analysis

In the previous chapters we have developed a basic understanding of community detection methods and how it has been applied in social media applications. We also described the algorithm which we shall be using for our analysis. In this chapter we focus on implementation of the method to analyze social networks from various universities. We start off by explaining the experiments and analysis that we shall perform followed by the data description. We then implement the algorithm on the data and this chapter concludes with the.

5.1 Experimental Design

Though a number of methods have been described in the literature for identifying communities within a network, not many deal with analyzing the information once the communities have been discovered. Usually every node in a network has a set of attributes which describe its uniqueness. For example, a *Facebook*® profile of an individual usually represents his characteristics outside the network. So once a community is detected inorder to learn more about the nodes in the community one should look into the attributes of all the nodes in the network. Ideally, the way networks are formed are decided by the way these individual communities shape up within the networks. Thus, analyzing these communities is as important as their detection. We present a unique way of analyzing the pattern within a community once they are formed. Further, overlapping communities tend to give a different perspective as opposed to disjoint communities. Thus analyzing them

becomes all the more difficult. In the literature, a lot of work has been done in understanding mixing patterns within disjoint communities. Whereas in the case of overlapping structures some work has been done in comparing the community structure to the ground truth but no work has been done in finding ways to look at these communities when the ground truth is unavailable. As in most real world social networks, ground truth is rarely available thus making the task of community detection a data mining task as opposed to a machine learning one. The lack of ground truth makes analyzing these communities more interesting because the analysis from one algorithm may lead to communities which are different than those found using another. As a result the conclusions drawn though relavant are slightly different than the other ones. Below we propose a way to analyze these overlapping communities so as to draw relevant and potentially useful conlcusions.

5.1.1 Detecting Overlapping Communities

Inorder to find the overlapping communities we use the SEBR algorithm described in chapter 4. The input to the algorithm is a graph in edge list format along with a few parameters and the output is a set of communities with each community consisting of a set of nodes along with their properties. The input is in text format and the output is also in text format with each community being a seperate text file. This output format allows us to easily analyze individual communities after they have been detected so that we can derive useful conclusions from the same. For all our experiments the parameters of density and loss in fitness of a node by belonging to an additional cluster are set as follows $\rho = 0.1, \lambda_v = 1.8$. ρ refers to the cluster density and is set to one because most real world networks exhibit this density [9]. The value of $lambda_v$ refers to the cost a node has to pay to join a cluster. The reason for choosing this value is detailed in [9]. We also compute and store other details such as running time, number of communities, number of best responses and the number of rounds until the algorithm converges.

5.1.2 Parallel Implementation

As described in section 4.5 the SMBR algorithm can be parallelized on a multicore processor so as to improve the overall computation time. In this experiment we

write a code to run the SMBR algorithm in parallel with other algorithms and compare the results in section 5.4. For the parallel implementation we use a Java library called Parallel Java [4] which enables us to implement the code on multiple cores depending on the type of system that we use. We compare the running time of the SMBR algorithm when run in parallel and otherwise and see the apparent difference in implementing these algorithms alternatively.

5.1.3 Analyzing Communities

As discussed earlier, inorder to derive inference from the communities, we need to analyze these communities individually. We develop a novel way to anaylze the communities. The objective is to find the attribute that is most common or best describes the behaviour of the community. Let x_i be a node in a derived community. Then $A_i = A_i^1, A_i^2, A_i^3, \ldots, A_i^k$ represents the attribute set for a node x_i where A_i^k represents an attribute k of node x_i. Once we have the attribute set for all the nodes in the community we shall compute the uncertainty that each of these attributes bring to the community. In order to do that we use the concept of entropy as explained by Shannon in his ground breaking paper titled "A Mathematical Theory of Communication" [62]. The formula for entropy is given as:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

where K is a positive constant and p_i in the probability of a label i occuring in the dataset. In our case the above equation can be generalized for each attribue A^i to compute its uncertainty.

$$H(A^j) = -K \sum_{i=1}^{n} p(A^j = i) \log p(A^j = i)$$

where $H(A^j)$ is the uncertainty within attribute A^j for a given community set. Once we compute the uncertainty for all the attributes within a community set, we rank them in the order of increasing entropy. Since entropy refers to the uncertainty, higher the value more the uncertainty hence the attribute is hard to guess. In order to rank them, we give a score of n among n attributes to the one that

ranked the highest and 1 to the lowest. Thus if there are n attributes, all of them get ranked accordingly. So if there are m communities we get m such rankings from all the communities. Let S_m be the number of nodes in a given community. Thus, we now do a weighted average for each attribute across all communities.

$$Avg(A^k) = \frac{\sum_{i=1}^{m} Rank(A_i^k) S_i}{\sum_{i=1}^{m} S_i}$$

where $Avg(A^k)$ is the weighted average score of a given attribute and takes a value between 0 and 1. After getting the scores for all the attributes, we again rank them and find the one that is most highly ranked. That attribute is the most dominant one averaged across all communities. Results are presented in Section 5.3.

5.2 Data Description

We used the Facebook100 dataset which contains the facebook details of students from 100 US Universities as of September 2005 [10, 11]. The dataset consists of two files namely the edge set and the node set. The edge set is represented in the form of an adjacency matrix amongst the people in a particular university. The node set on the other hand represents the detials about each person in the university. The node set is represented in the form as shown below:
<node_id><second_major><gender><major_index><year><dorm><high_school><student_fac>

All the student attributes are annonymized; hence it is difficult to decrypt them but some of them attributes can be decrypted such as

- **<gender>:** {*1=male;2=female*}
- **<student_fac>:** {*1=undergratuate_student;2=graduate_student*}

The data has been presented in .mat file or MATLAB files. We have converted those files to simple text files representing an edge list and a node list. The graph details of a sample set of universities is presented in Appendix A. Now that we have discussed the experimental design and the data, we shall show the results of our experiments. First we shall discuss the results for individual communities in

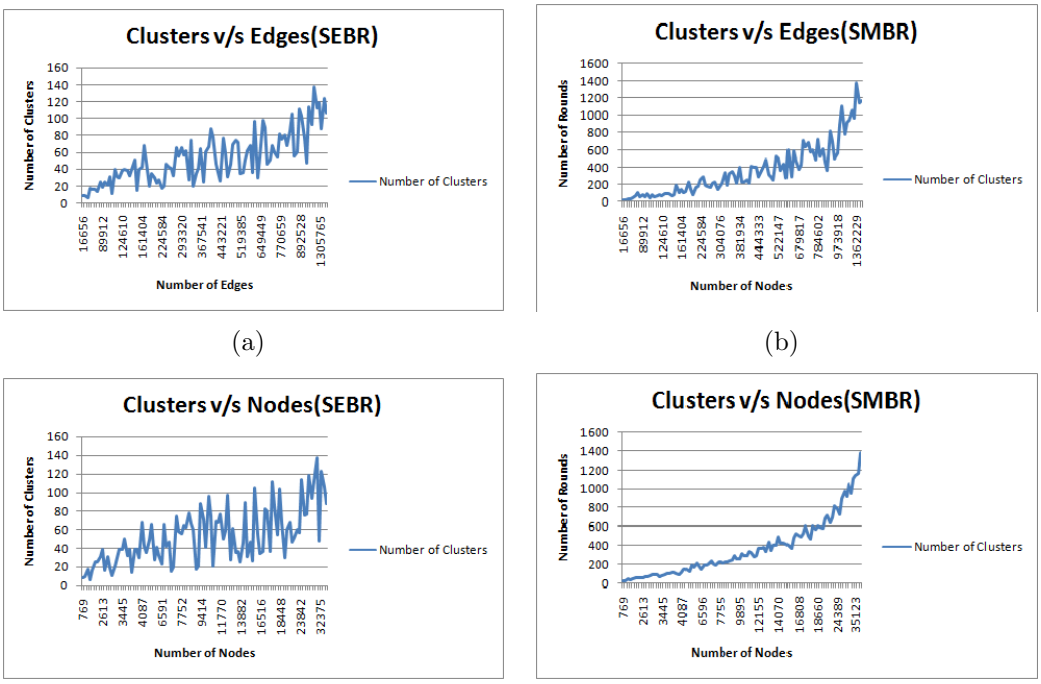


Figure 5.1. (a) The variation in the number of communities detected using SEBR Algorithm v/s number of edges (b) The variation in the number of communities detected using SMBR Algorithm v/s number of edges (c) The variation in the number of communities detected using SEBR Algorithm v/s number of nodes (d) The variation in the number of communities detected using SMBR Algorithm v/s number of nodes.

section 5.3. Then, we compare the running time and computational complexity for the SMBR algorithm vs SEBR algorithm in section 5.4.

5.3 Clustering Analysis

Having described the experimental setup, we ran the algorithm mentioned in chapter 4 over the datasets as described in section 5.2. The number of nodes that we ran the algorithm on ranged from 769 for Caltech to 41554 for Pennsylvania State University. While the number of edges ranged from for 16556 of Caltech to 1590655 of University of Texas. Note that we have used two algorithms for our analysis. One is Sequential Best Response (SEBR) and the other one is Simultaneous Best Response (SMBR). We plot the graph of number of rounds where the players play a game v/s the number of nodes and edges. These are shown in the Figures 5.1(a)

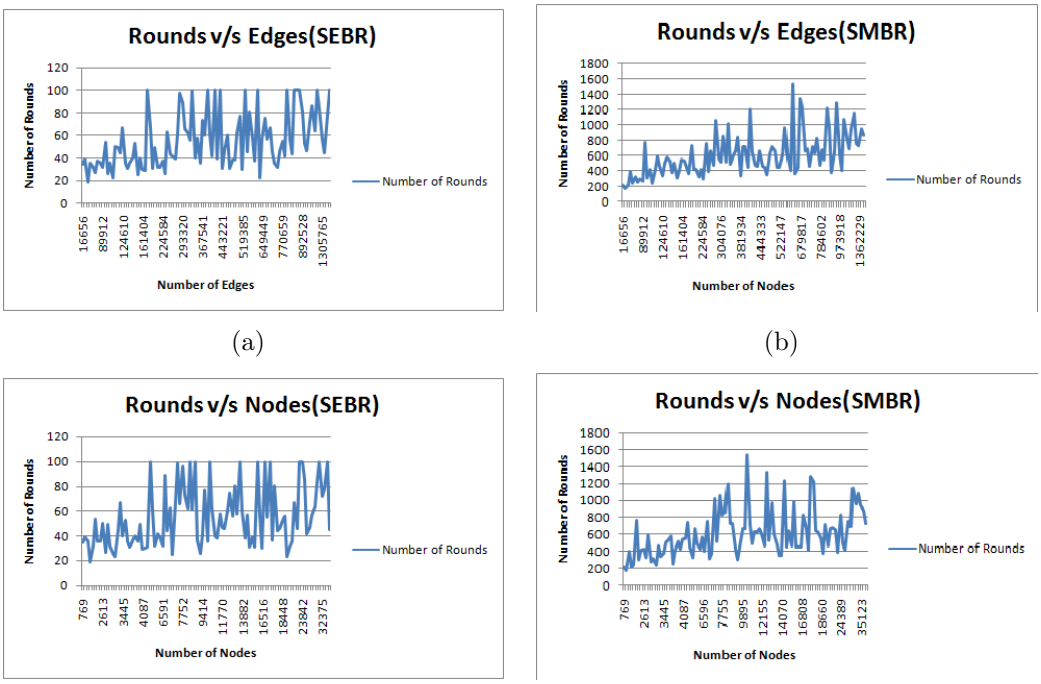


Figure 5.2. (a) The variation in the number of best responses computed using SEBR Algorithm v/s number of edges (b) The variation in the number of best responses detected using SMBR Algorithm v/s number of edges (c) The variation in the number of best responses detected using SEBR Algorithm v/s number of nodes (d) The variation in the number of best responses detected using SMBR Algorithm v/s number of nodes

and 5.1(b) and Figures 5.1(c) and 5.1(d). From these figures we can see that the number of clusters tend to increase with the increase in number of edges both in the case of SMBR as as well as SEBR algorithm. This suggests that the choice of the algorithm does not impact the formation of the clusters. Keeping this in mind we shall now see the variaion in number of Rounds (Best response calculations) done by both the algorithms. This is shown in Figures 5.2(a) and 5.2(b) and Figures 5.2(c) and 5.2(d). One thing that can be seen is the number of best response calculations for SMBR is much higher than SEBR. This suggests that SMBR requries more computations to reach Nash Equilibria becasue the updates take place globally as opposed to locally. Thus for nodes to update their labels, they have to wait till all the other nodes have finished updating their labels. This increases the overall number of rounds but there is not enough increase in the computation time as compared to SEBR algorithm. The reason for this being, we

only allow a fraction of players to participate in a game of nodes every time it is played. For these experiments, we found that the number of rounds increase with the percentage of players playing the game. In this case we allow 10% to play the game in every round in the SMBR algorithm. We shall explain more about computational complexity in the next section.Next we analyze the mixing patterns in the clusters found above.

5.3.1 Ranking Attributes

Mixing patterns relate to how attributes of nodes are distributed across communities/clusters. In our case a node relates to a person in a University and they have characteristics as described in section 5.2. Keeping these in mind, once the communities are detected, we use the analysis method which we described in section 5.1.3. Once the communities are detected for a given network, we compute $H(A^j)$ for each attribute in a given community. We then rank the attributes based on increasing order of $H(A^j)$ and then assign a score of 7 (Since there are 7 attributes, we give the highest score to the lowest $H(A^j)$) for the 1^{st} ranked attribute and 1 to the last ranked attribute. Then we do a weighted average for all attributes, across all ranks in a community to compute $Avg(A^k)$. Thus, we first aggregate the results over all the communities in a given university and then we aggregate over all the universities. The $Avg(A^k)$ values for all the attributes indicate the percentage of time a given attribute was the most common attribute within the communities. That is, it gives the attribute which was most certain across most communities detected in the network.

Since mixing patterns of all the universities are difficult to show for all the universities, in this case we just present an aggregate over all the universities. This is shown in Figure 5.3.1. From this we can see that the Designation of a person in the university was most common attribute in 25% of all the communities, this is quite intuitive taking into consideration the ratio of the number of students to the number of faculty and staff combined in any university. The next attribute in the list is the Gender of the person. It was seen that the number of males compared to females was quite high in the university, thus in a community, the number of males or females does not vary much. With this we mean that since the

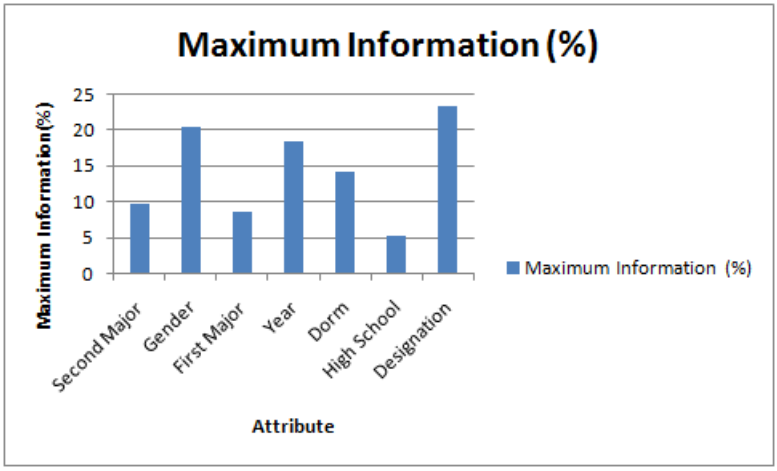


Figure 5.3. Percentage of times a given attribute accounted for least uncertainty within a given community.

network represents a friendship network, a community with many females would have less males and vice versa because of the way friends are made. Next the year plays a major role in creating friends and in general students tend to be friends with those students who entered in the same year. Also, after the year in which they entered, the Dorm in which they reside also accounts for the next level in friendship. Eventually, the first major and second major tends to be reason for mixing between close communities and finally the high school from where they have come. The high school of a student in a university accounts for the least certainty because, there are very few communities which have people from the same high school going to the same university and since we are looking for students from all the universities across the U.S., people tend to disperse to other places.

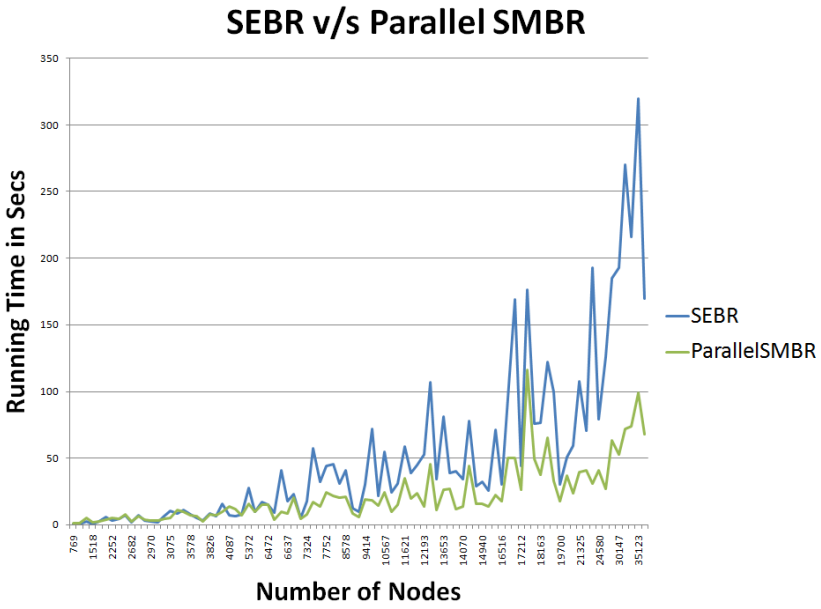
In the following section we shall discuss the running time of a sequential implementation of the SEBR algorithm against the parallel implementation of SMBR algorithm explained in chapter 4.

5.4 Comparing Running Times

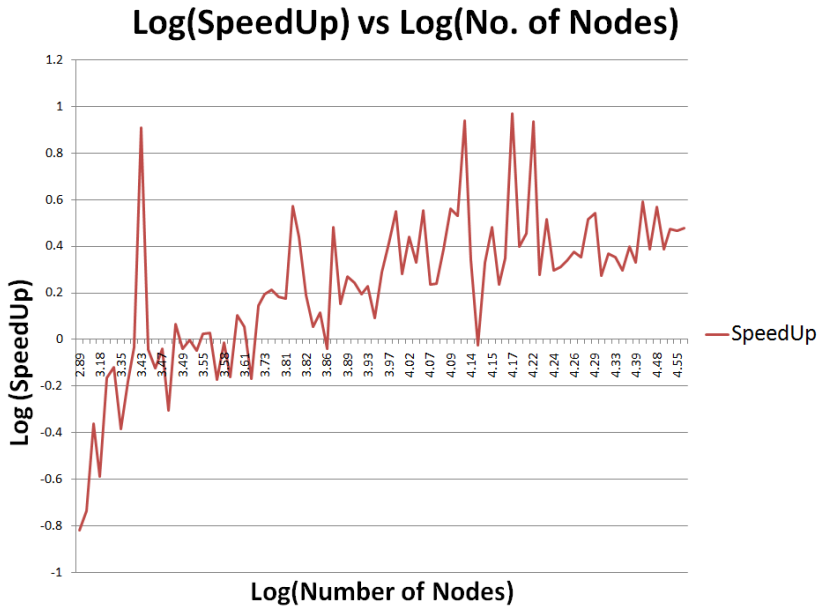
We have discussed the need for faster implementation of algorithms to take into consideration the amount of social media data that is being collected in real time in Chapter 4 and section 5.1.2. Having explained parallelization of algorithms in

Chapter 4, we now go to show an actual implementation of the same. SMBR algorithm can be parallelized using any parallel programming technique. We use the Parallel Java library to parallelize the SMBR algorithm (ParallelSMBR). We ran the parallelized algorithm ; lets call it ParallelSMBR and compared the running time against SEBR algorithm (See Figure 5.4(a)). As we see that the number of communities found and the running time of both the algorithms are the same, a parallel implementation of the same will definitely be much faster than a sequential implementation. In this experiment we compare the running time of SEBR and ParallelSMBR algorithms to the number of nodes and number of edges of the networks. This is shown in Figure 5.4(a). From this we see that initially there is not much difference between the running times of both the algorithms when the number of nodes are less. This is explained by the fact that more time is spent in creating the threads than actual computation. But as the number of nodes increase, the difference in running time also increases exponentially. Thus, we can see that parallelization may be computationally expensive for less number of nodes but as the number of nodes increase exponentially, the running time also increases exponentially. This is shown by plotting the speedup against the number of nodes in a log-log plot as seen in Figure 5.6(a). This clearly depicts that there is not speed up initially buy as the number of nodes increase, the speedup also increases exponentially. This can be explained by the fact that for small networks (those with number of nodes <5000), more time is spent in creating and executing threads as opposed to bigger networks (number of nodes >5000) where there is a tradeoff between number of threads created and running time. In comparison, for SEBR algorithm when run on multiple cores and a single core does not affect the running time and the speedup which can be seen from the Figure 5.4.

We mentioned in Chapter 4 about two metrics to measure the effect of parallelization of an algorithm. They were *Speedup* and $Efficiency$. We measured the two metrics by running the 100 networks in 3 different settings. A single core machine, a 4-core machine and an 8-core machine. Figure 5.6(a) shows the vairation in *Speedup* and Figure 5.6(b). As we can see the average speedup is 2.21 for 4-core processor and 2.73 for 8-cores where as the ideal speedup should have been 4 and 8. This is explained by the fact that the overall speedup depends on a lot of factors which are usually beyond our control. These are explained in chapter 4.



(a)



(b)

Figure 5.4. (a)This Log-Log plot compares the running time with the number of nodes in the network (b)This Log-Log plot shows the variation in speedup v/s number of nodes in the network for SMBR Algorithm

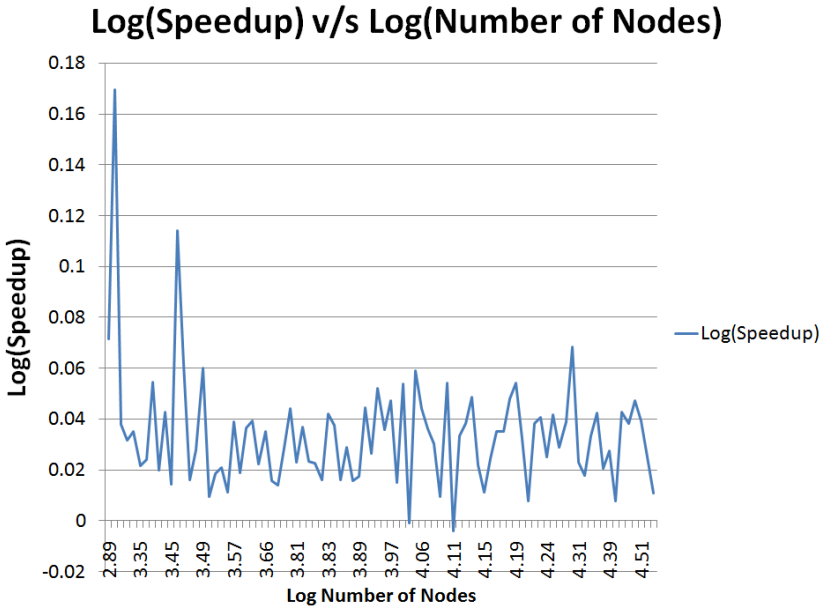
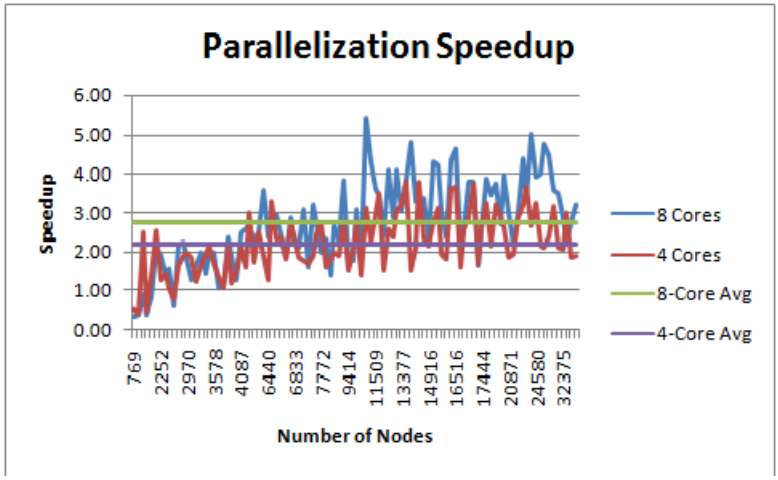


Figure 5.5. This Log-Log plot shows the variation in speedup v/s number of nodes in the network for SEBR Algorithm

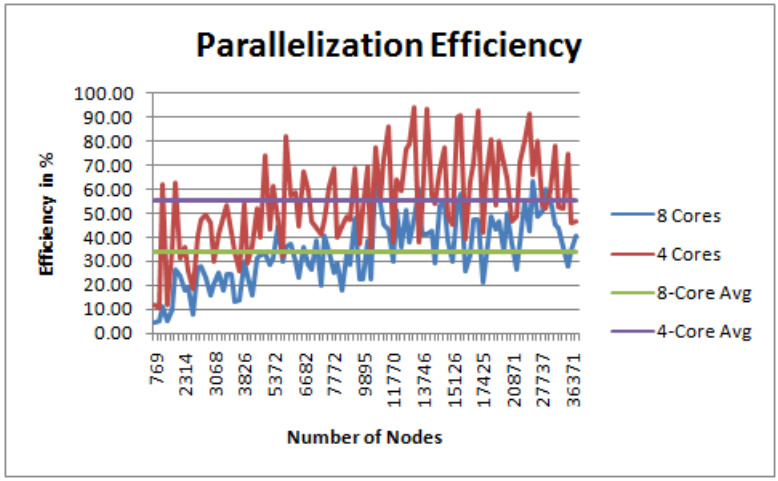
Similarly you can see from Figure 5.6(b) that the average efficiency reduces as the number of cores increased. Although the efficiency decreases, the overall running time is reduced nonetheless.

5.5 Results

This chapter mainly dealt with implementing the methods which we described in the previous chapters. We first explained our experimental setup and further went on to propose a framework to analyze overlapping communities with nodes having network independent attributes. We first used the SEBR and SMBR community detection algorithms explained in chapter 4. Once the communities were detected we used our proposed method in section 5.1.3 method to find the most common attribute in each community. On analyzing these communities from all the universities, we found that the order of significance of attributes inherent within these communities is {Designation, Gender, Year, Dorm, Second Major, First Major, High School} as explained in section 5.3.1. These results were explained in more detail in section 5.3. Finally we showed how parallelizing an alogrithm can reduce



(a)



(b)

Figure 5.6. (a)The speedup is plotted against the number of nodes for each core. (b) The efficiency is plotted against the number of nodesn for each core.

the running time tremendously. We found that initially the running time increases but as the number of nodes increase exponentially, the running time decreases exponentially. In the next chapter we will explain the significance of these results and their practical applications.

Chapter 6

Conclusions

Chapter 2, motivated the problem of community detection in social media and then defined the scope of the thesis. In Chapter 3 we gave a comprehensive review about various methods and definitions aiding community detection in social networks including both overlapping and disjoint communities. Chapter 4 delved into explaining an algorithm proposed by Mandala et al. [9] and then showing how one can parallelize it for improved efficiency and performance. Chapter 5 dealt more into running the algorithm on a dataset of *Facebook*® social networks and then developing a method to analyze communities once they have been detected. We also computed the *speedup* and efficiency of the algorithm. In this chapter we summarize our findings from the previous chapters by providing inferences and topics for further research.

6.1 Contributions

The main purpose of this thesis was to show a successful implementation of a community detection algorithm on online social networks. We approached the problem by first reviewing the literature in detecting disjoint and overlapping communities in networks. Through this research we realized that most of the work is done post 2009 which means more and more researchers are trying to develop methods to delve more into social networks. This also helped us reach to the conclusion that detecting overlapping communities as opposed to disjoint communities makes more sense in online social networks because people tend to be associated with more

than one community. Thus, it is our suggestion that those interested in working on the problem of community detection in social networks should work from an overlapping community detection perspective as opposed to disjoint communities. Overlapping communities helps us to identify nodes in the network which act as bridges between adjoining communities. These nodes of high influence hold the key to transition from one community to another. Targeting these nodes usually enables better information flow in networks. Thus our first contribution from this thesis is:

- Provide a comprehensive literature review covering applications of community detection algorithms in social networks covering disjoint as well as overlapping communities. Detecting overlapping communities in online social networks will capture more information in terms of user behaviour as opposed to detecting disjoint communities.

Next, based on the size of online social networks, there is an increasing demand for local density-based methods and iterative approximation schemes as opposed to the currently established global optimization algorithms. Keeping this in mind we selected a game theoretic approach based algorithm proposed by [9] to detect overlapping communities. The novelty of this algorithm is the fact that by changing the network parameters, we can detect overlapping as well as disjoint communities. Also, due to the volume and dynamics of online social networks, communities need to be detected in near linear time. With numerous blogs, feeds and posts increasing day by day, faster and efficient implementation of algorithms is needed as opposed to slow and optimal ones. On these lines our second contribution is:

- Fast and efficient implementation of a community detection algorithm using parallel programming concepts.

In most of the community detection methods reviewed in the literature, we found that after communities are detected not much work has been done in evaluating these communities. In this thesis, we proposed an entropy based method to rank the attributes on their relevance of occurence in the communities. This leads to our final conclusion:

- Define an entropy based method to evaluate the commonality of attributes within communities.

6.2 Summary

We now provide a brief summary containing the scope and flow of the thesis. We started off by explaining how companies are trying to retain customers by providing products and services which tender to an individual customer rather than a group of customers that is mass customization to a customer size of 1 [63]. We followed up by explaining how companies are using social media tracking softwares to shape their products and improve their services to cope up with the current competition and to better serve the consumers (Chapter 1). Based on this introduction we went on to motivate and finally define our problem statement. First by explaining the basic concepts of complex networks and then by describing a few real world networks we laid the ground work for our analysis. We then explained the emergence of social media networks and what impact it has had in recent times. Using these two concepts we defined our problem statement elaborating the detection and evaluation of communities in social networks. The definition focused on implementation of a fast community detection algorithm using parallel programming and proposing a method to evaluate communities once they are found (Chapter 2). Before going into the details of our algorithm, we first had to review the available literature focusing on community detection, overlapping communities and community detection in social media. While reviewing the available literature, we found that most of the research in the field of detecting overlapping communities is done post 2009. Also, most of the work is applied to detecting overlapping communities in social networks than in any other field. This strengthens our problem statement is within the current scope of research. Based on that we selected a game theory based algorithm developed by Mandala et al. [9]. The reason for this choice of algorithm is because nodes tend to behave like players and try to form communities, just like humans try to make friends with other humans and form communities in the real world. (Chapter 3). After selecting the algorithm, we explain how the algorithm works using a pseudo code and explained the two versions of the same, namely, Simultaneous Best Response (SMBR) and Sequential Best Response (SEBR). We then explained the parallel programming framework required for the fast implementation of the algorithm (Chapter 4). Following the explanation of the theory behind the SEBR and SMBR algorithm, we

explain the design of our experiments. We then propose an entropy based method to evaluate communities once they have been found, i.e. find the attribute which best describes a given community. Then we tested our parallelized algorithm on *Facebook*® social networks from 100 universities across the United States. From our analysis we found that there is no speed up when the parallelized algorithm is run on smaller networks, but as the network size increases exponentially, the running time also decreases exponentially. After analyzing the detected communities, we realized that most of the communities had the "Designation" of the person as the dominant attribute followed by "Gender", "Graduation Year" and "Dorms" in which they live (Chapter 5). We conclude by explaining our contributions from the thesis followed by this summary and finally topics for further research (Chapter 6).

6.3 Future Work

In this section, we provide topics for future research based on our findings and contributions. From our analysis we found that there are many papers published in detecting overlapping communities in social networks but none of them focus on their evaluation once they are detected. Only a few methods exist in the literature, namely Normalized Mutual Information (NMI) and Omega index but the existence of ground truth is required for using them. Since there is a lack of ground truth incase of social networks, other methods need to be developed which provides estimates based on expectations rather than ground truth. Further, since the scale of these online social networks is enormous, community detection algorithms should consider this fact and try to develop local schemas which focus on parts of the network as opposed to the entire network itself. A major problem with online social network data is the missing data problem. Usually people either estimate the missing data using some local methods or simply neglect them from their analysis. Since communities tend to find similarity between people in a group, estimating missing data after communities have been formed is another topic of further research.

Appendix A

University Details

A.1 Sample Dataset

This appendix lists the names of all universities networks used in our analysis from the Facebook100 dataset released by Traud et al, [10, 11]. This data has been downloaded from [64].

Table A.1. University Details

University	Number	Edges	Vertices	Max Degree
Harvard	1	824617	15126	1183
Columbia	2	444333	11770	3375
Stanford	3	568330	11621	1172
Yale	4	405450	8578	2517
Cornell	5	790777	18660	3156
Dartmouth	6	304076	7694	948
UPenn	7	686501	14916	1602
MIT	8	251252	6440	708
NYU	9	715715	21679	2315
BU	10	637528	19700	1819
Brown	11	384526	8600	1075
Princeton	12	293320	6596	628
Berkeley	13	852444	22937	3434
Duke	14	506442	9895	1887
Georgetown	15	425638	9414	1235
UVA	16	789321	17196	3182
BC	17	486967	11509	1377
Tufts	18	249728	6682	827
Northeastern	19	381934	13882	968
UIllinios	20	1264428	30809	4632
UF	21	1465660	35123	8246
Wellesley	22	94899	2970	746

Appendix B

Algorithm Codes

This chaper lists all the codes that have been written for community detection and their analysis.

B.1 Main Class

```
1  import java.io.*;
   import java.io.BufferedReader;
   import java.io.BufferedWriter;
   import java.io.FileNotFoundException;
   import java.io.FileReader;
6  import java.io.FileWriter;
   import java.io.IOException;
   import java.util.ArrayList;
   import java.util.StringTokenizer;

11 public class MainClass {

    /**
     * @param args
     */
16  public static int Processors = 8;
    public static void main(String[] args)throws FileNotFoundException,
        IOException,Exception {

     String[][] numbers = new String[101][6];
```

```
    BufferedReader bufRdr = new BufferedReader(new FileReader("univ.csv
        "));
    String line = null;
    int row = 0;
    int col = 0;
    //read each line of text file
    while(((line = bufRdr.readLine()) != null))
    {
     StringTokenizer st = new StringTokenizer(line,",");
     while (st.hasMoreTokens())
     {
     //get next token and store it in the array
      numbers[row][col] = st.nextToken();
      col++;
     }
     col = 0;
     row++;
    }

    //close the file
    bufRdr.close();
    FileWriter fstream = new FileWriter("Summary.txt");
        BufferedWriter out1 = new BufferedWriter(fstream);
    out1.write("Name Nodes Edges Rounds Clusters Time");
    out1.newLine();
    ArrayList<Integer> Elements = new ArrayList<Integer>();
    //int[] myArray = new int[]{7,11,27,32,39,64,65,69,70,88,89,100};
    //int[] myArray = new int[]{100};
    int[] myArray = new int[]{Integer.parseInt(args[0])};
    Elements = initArrayList(myArray);
    for(int i = 1; i<=100;i++){
    // TODO Auto-generated method stub
    if (Elements.contains(i+1) != true){
     continue;
    }
    System.out.print(i);
    String filename = numbers[i][1];
    int current_nodes = Integer.parseInt(numbers[i][3]);
    int current_edges = Integer.parseInt(numbers[i][2]);
    //System.out.println("File Name "+filename);
```

```
    //System.out.println();
    out1.write(filename+" ");
61  out1.write(current_nodes+" ");
    out1.write(current_edges+" ");
    String strDirectory = "//gpfs//work//adg181//UniversityDataSets//"+
        filename+"//";
    SimultaneousGame cg=new SimultaneousGame(0.1,1.8,0.10,current_nodes
        ,strDirectory+filename+".txt");
    //ClusteringGame cg=new ClusteringGame(0.1,1.8,current_nodes,
        strDirectory+filename+".txt");
66  double time=System.currentTimeMillis();
    //File F1 = new File(strDirectory+"Clusters");
    //boolean fail = F1.delete();
    //System.out.println(fail);
    File F2 = new File(strDirectory+"Clusters");
71  boolean success = F2.mkdir();
    //System.out.println(success);
    cg.start();
    time=System.currentTimeMillis()-time;
    out1.write((time/1000)+" ");
76  System.out.println(time/1000);
    String[] node_list = new String[current_nodes+1];
    String filename1 = "//gpfs//work//adg181//UniversityDataSets//"+
        filename+"//"+filename;
     BufferedReader rd = new BufferedReader(new FileReader(filename1
         + ".nodelist"));
     //BufferedReader rd = new BufferedReader(new FileReader(
         filename1 + "1.nodelist"));
81  int node_list_counter = 0;
    while (true){
          line = rd.readLine();
          if(line==null){
               break;
86        }
     node_list[node_list_counter] = line;
     node_list_counter++;
    }
    cg.write_clusters(strDirectory,filename,node_list,20,out1);
91  out1.newLine();}
    out1.close();
```

```
 }

 public static ArrayList<Integer> initArrayList(int[] a) {
96 ArrayList<Integer> list = new ArrayList<Integer>();
 for (int i : a){
  list.add(i);
 } //close for loop
 return list;
101 } //close initArrayList method

}
```

Listing B.1. Main Class Java Code

B.2 SEBR Algorithm Class

```
 import java.io.BufferedReader;
2 import java.io.BufferedWriter;
 import java.io.FileReader;
 import java.io.FileWriter;
 import java.util.Iterator;
 import java.io.IOException;
7 import java.io.FileNotFoundException;

 public class ClusteringGame {

  Vertex[] vertices;
12  Cluster[] clusters;
  int rounds=0;
  public ClusteringGame(double rho, double lambda, int num_vertices,
     String fname){
   Vertex.global_label_count=new int[num_vertices];
   Vertex.updated_global_label_count=new int[num_vertices];
17   Vertex.num_vertices=num_vertices;
   Vertex.rho=rho;
   Vertex.lambda=lambda;
   Vertex.num_updates=1;

22   vertices = new Vertex[num_vertices];
```

```
  clusters = new Cluster[num_vertices];

  for(int i=0;i<num_vertices;i++){
   vertices[i] = new Vertex(i);
  }
  read_graph(fname);
 }

 public void read_graph(String fname){
  BufferedReader in;
  String str;String[] split_str;
  int i,j,num_edges=0;
  try{
   in = new BufferedReader(new FileReader(fname));
      while ((str = in.readLine()) != null) {
       split_str=str.split(" ");
       i=Integer.parseInt(split_str[0]);
       j=Integer.parseInt(split_str[1]);
   if(vertices[i-1].neighbors.contains(vertices[j-1])||vertices[j-1].
       neighbors.contains(vertices[i-1]))
        {
         continue;
        }
   else{
       vertices[i-1].add_neighbor(vertices[j-1]);
       vertices[j-1].add_neighbor(vertices[i-1]);
       num_edges++;}
      }
      in.close();
  }
  catch(Exception e){
  System.out.println(e);}
//  System.out.println("Number of Vertices : "+Vertex.num_vertices);
//  System.out.println("Number of Edges : "+num_edges);
 }

 public void start(){
  while(Vertex.num_updates>0){
   Vertex.num_updates=0;
   for(int i=0;i<Vertex.num_vertices;i++){
```

```
62        vertices[i].compute_best_response();
      }
      rounds++;
//    System.out.println("End of round "+rounds);
    }
67  //display();
    populate_clusters();
    //System.out.println("Total Number of rounds:"+rounds);
  }

72  public void populate_clusters(){
    Iterator it;
    int tmp_label;
    for(int i=0;i<Vertex.num_vertices;i++){
      it=vertices[i].labels.iterator();
77    while(it.hasNext()){
        tmp_label=(Integer)it.next();
        if(clusters[tmp_label]==null){
          clusters[tmp_label]=new Cluster();
        }
82      clusters[tmp_label].members.add(vertices[i]);
      }
    }
  }

87  public void display(){
    System.out.println("Clusters:");
    for(int i=0;i<Vertex.num_vertices;i++){
      vertices[i].display();
    }
92
  }

  public void display_clusters(int thresh){
    for(int i=0;i<Vertex.num_vertices;i++){
97    if (clusters[i]!=null && clusters[i].members.size()>=thresh){
        clusters[i].display();
      }
    }
```

```
102  }
     public void write_clusters(String strDirectory, String filename,
         String[] node_list,int thresh,BufferedWriter out1) throws
         FileNotFoundException, IOException{
      String filename1 = strDirectory+"//Clusters//SEDetails";
      FileWriter fstream = new FileWriter(filename1 +".txt");
          BufferedWriter out = new BufferedWriter(fstream);
107   int cluster_node_count = 0;
      int number_of_clusters = 0;
      for(int i=0;i<Vertex.num_vertices;i++){
       if (clusters[i]!=null && clusters[i].members.size()>=thresh){
        cluster_node_count = clusters[i].write(i+1,filename,strDirectory,
            node_list);
112     out.write((i+1)+" "+cluster_node_count);
        out.newLine();
        number_of_clusters++;
       }
       cluster_node_count = 0;
117   }
      out1.write(rounds+" ");
      out1.write(number_of_clusters+" ");
      out.close();
     }
122  }
```

Listing B.2. SEBR Algorithm Java Code

B.3 SMBR Algorithm Class

```
     import java.io.BufferedReader;
     import java.io.BufferedWriter;
  3  import java.io.FileReader;
     import java.util.*;
     import java.io.FileWriter;
     import java.util.Iterator;
     import java.io.IOException;
  8  import java.io.FileNotFoundException;
     public class SimultaneousGame {
```

```
Vertex[] vertices;
Cluster[] clusters;
int rounds = 0;

public SimultaneousGame(double rho, double lambda,double probability
    , int num_vertices,String fname){
 Vertex.global_label_count=new int[num_vertices];
 Vertex.updated_global_label_count = new int[num_vertices];
 Vertex.num_vertices=num_vertices;
 Vertex.rho=rho;
 Vertex.lambda=lambda;
 Vertex.num_updates=1;
 Vertex.probability = probability;

 vertices = new Vertex[num_vertices];
 clusters = new Cluster[num_vertices];

 for(int i=0;i<num_vertices;i++){
  vertices[i] = new Vertex(i);
  vertices[i].old_labels.add(i);
 }
 read_graph(fname);
}

public void read_graph(String fname){
 BufferedReader in;
 String str;String[] split_str;
 int i,j,num_edges=0;
 try{
  in = new BufferedReader(new FileReader(fname));
      while ((str = in.readLine()) != null) {
       split_str=str.split(" ");
       i=Integer.parseInt(split_str[0]);
       j=Integer.parseInt(split_str[1]);
       vertices[i-1].add_neighbor(vertices[j-1]);
       vertices[j-1].add_neighbor(vertices[i-1]);
       num_edges++;
      }
      in.close();
 }
```

```
 catch(Exception e){}
 System.out.println("Number of Vertices : "+Vertex.num_vertices);
 System.out.println("Number of Edges : "+num_edges);
}

public void start() throws Exception{
    Random generator = new Random();
    while(Vertex.num_updates>0){
     final ArrayList<Integer> playing_nodes = new ArrayList<Integer
         >();
  Vertex.num_updates=0;
  for(int i=0;i<Vertex.num_vertices;i++){
   float randomIndex = generator.nextFloat();
   if(randomIndex<Vertex.probability){
    playing_nodes.add(i);
   }}
  for(int i=0;i<playing_nodes.size();i++){
   vertices[playing_nodes.get(i)].simultaneous_best_response();
  }
  Vertex.global_label_count = Vertex.updated_global_label_count;
  for(int i=0;i<playing_nodes.size();i++){
   vertices[playing_nodes.get(i)].old_labels.clear();
   vertices[playing_nodes.get(i)].old_labels.addAll(vertices[
       playing_nodes.get(i)].labels);
  }
  rounds++;
  //System.out.println("End of round "+rounds);
  //System.out.println("Number of Updates "+Vertex.num_updates);
 }
 //display();
 populate_clusters();
}

public void populate_clusters(){
 Iterator it;
 int tmp_label;
 for(int i=0;i<Vertex.num_vertices;i++){
  it=vertices[i].labels.iterator();
  while(it.hasNext()){
   tmp_label=(Integer)it.next();
```

```
88      if(clusters[tmp_label]==null){
         clusters[tmp_label]=new Cluster();
        }
        clusters[tmp_label].members.add(vertices[i]);
       }
93    }
    }

    public void display(){
     System.out.println("Clusters:");
98    for(int i=0;i<Vertex.num_vertices;i++){
       vertices[i].display();
     }

    }
103
    public void display_clusters(int thresh){
     for(int i=0;i<Vertex.num_vertices;i++){
      if (clusters[i]!=null && clusters[i].members.size()>=thresh){
       clusters[i].display();
108   }
     }

    }

113
    public void write_clusters(String strDirectory, String filename,
        String[] node_list,int thresh,BufferedWriter out1) throws
        FileNotFoundException, IOException{
     String filename1 = strDirectory+"//Clusters//SMDetails";
     FileWriter fstream = new FileWriter(filename1 +".txt");
         BufferedWriter out = new BufferedWriter(fstream);
118   int cluster_node_count = 0;
     int number_of_clusters = 0;
     for(int i=0;i<Vertex.num_vertices;i++){
      if (clusters[i]!=null && clusters[i].members.size()>=thresh){
       cluster_node_count = clusters[i].write(i+1,filename,strDirectory,
           node_list);
123    out.write((i+1)+" "+cluster_node_count);
       out.newLine();
```

```
      number_of_clusters++;
     }
     cluster_node_count = 0;
128  }
   out1.write(rounds+" ");
   out1.write(number_of_clusters+" ");
   out.close();
 }
133 }
```

Listing B.3. SMBR Algorithm Java Code

B.4 ParallelSMBR Algorithm Class

```
   import java.io.BufferedReader;
 2 import java.io.BufferedWriter;
   import java.io.FileReader;
   import java.util.*;
   import java.io.FileWriter;
   import java.util.Iterator;
 7 import java.io.IOException;
   import java.io.FileNotFoundException;
   import edu.rit.pj.IntegerForLoop;
   import edu.rit.pj.ParallelRegion;
   import edu.rit.pj.ParallelTeam;
12 public class SimultaneousGame {

   Vertex[] vertices;
   Cluster[] clusters;
   int rounds = 0;
17

   public SimultaneousGame(double rho, double lambda,double probability
       , int num_vertices,String fname){
    Vertex.global_label_count=new int[num_vertices];
    Vertex.updated_global_label_count = new int[num_vertices];
    Vertex.num_vertices=num_vertices;
22  Vertex.rho=rho;
    Vertex.lambda=lambda;
    Vertex.num_updates=1;
```

```
    Vertex.probability = probability;

    vertices = new Vertex[num_vertices];
    clusters = new Cluster[num_vertices];

    for(int i=0;i<num_vertices;i++){
     vertices[i] = new Vertex(i);
     vertices[i].old_labels.add(i);
    }
    read_graph(fname);
   }

   public void read_graph(String fname){
    BufferedReader in;
    String str;String[] split_str;
    int i,j,num_edges=0;
    try{
     in = new BufferedReader(new FileReader(fname));
        while ((str = in.readLine()) != null) {
         split_str=str.split(" ");
         i=Integer.parseInt(split_str[0]);
         j=Integer.parseInt(split_str[1]);
         vertices[i-1].add_neighbor(vertices[j-1]);
         vertices[j-1].add_neighbor(vertices[i-1]);
         num_edges++;
        }
        in.close();
    }
    catch(Exception e){}
    System.out.println("Number of Vertices : "+Vertex.num_vertices);
    System.out.println("Number of Edges : "+num_edges);
   }

   public void start() throws Exception{
       Random generator = new Random();
       while(Vertex.num_updates>0){
        final ArrayList<Integer> playing_nodes = new ArrayList<Integer
            >();
     Vertex.num_updates=0;
     for(int i=0;i<Vertex.num_vertices;i++){
```

```
    float randomIndex = generator.nextFloat();
    if(randomIndex<Vertex.probability){
     playing_nodes.add(i);
67  }}

   new ParallelTeam(MainClass.Processors).execute(new ParallelRegion
       () {
    @Override
    public void run() throws Exception {
72   execute (0,playing_nodes.size()-1,new IntegerForLoop() {

      public void run(int first, int last) throws Exception {
       for(int i=first;i<last;++i){
        vertices[playing_nodes.get(i)].simultaneous_best_response();
77     }
      }
     });
    }
   });
82  Vertex.global_label_count = Vertex.updated_global_label_count;
   for(int i=0;i<playing_nodes.size();i++){
    vertices[playing_nodes.get(i)].old_labels.clear();
    vertices[playing_nodes.get(i)].old_labels.addAll(vertices[
        playing_nodes.get(i)].labels);
   }
87  rounds++;
   //System.out.println("End of round "+rounds);
   //System.out.println("Number of Updates "+Vertex.num_updates);
  }
  //display();
92  populate_clusters();
 }

 public void populate_clusters(){
  Iterator it;
97  int tmp_label;
  for(int i=0;i<Vertex.num_vertices;i++){
   it=vertices[i].labels.iterator();
   while(it.hasNext()){
    tmp_label=(Integer)it.next();
```

```
102       if(clusters[tmp_label]==null){
           clusters[tmp_label]=new Cluster();
          }
          clusters[tmp_label].members.add(vertices[i]);
         }
107     }
       }

       public void display(){
        System.out.println("Clusters:");
        for(int i=0;i<Vertex.num_vertices;i++){
         vertices[i].display();
        }

       }
117
       public void display_clusters(int thresh){
        for(int i=0;i<Vertex.num_vertices;i++){
         if (clusters[i]!=null && clusters[i].members.size()>=thresh){
          clusters[i].display();
         }
        }

       }


127
       public void write_clusters(String strDirectory, String filename,
           String[] node_list,int thresh,BufferedWriter out1) throws
           FileNotFoundException, IOException{
        String filename1 = strDirectory+"//Clusters//SMDetails";
        FileWriter fstream = new FileWriter(filename1 +".txt");
            BufferedWriter out = new BufferedWriter(fstream);
        int cluster_node_count = 0;
        int number_of_clusters = 0;
        for(int i=0;i<Vertex.num_vertices;i++){
         if (clusters[i]!=null && clusters[i].members.size()>=thresh){
          cluster_node_count = clusters[i].write(i+1,filename,strDirectory,
              node_list);
          out.write((i+1)+" "+cluster_node_count);
          out.newLine();
```

```
      number_of_clusters++;
     }
     cluster_node_count = 0;
142  }
   out1.write(rounds+" ");
   out1.write(number_of_clusters+" ");
   out.close();
 }
147 }
```

Listing B.4. Parallel SMBR Algorithm Java Code

B.5 Cluster Class

```
 import java.util.ArrayList;
 import java.util.Iterator;
3 import java.io.BufferedReader;
 import java.io.BufferedWriter;
 import java.io.FileReader;
 import java.io.FileWriter;
 import java.io.IOException;
8 import java.io.FileNotFoundException;

 public class Cluster {
  ArrayList<Vertex> members;

13  public Cluster(){
   members = new ArrayList<Vertex>();
  }
  public void display(){
   Iterator it=members.iterator();
18  Vertex v;
   while(it.hasNext()){
    v=(Vertex) it.next();
    System.out.print((v.ID+1)+" ");
   }
23  System.out.println();
  }
```

```
  public int write(int i,String filename,String strDirectory,String[]
      node_list) throws FileNotFoundException, IOException{
   Iterator it=members.iterator();
   Vertex v;
28 int nodes_in_cluster = 0;
   String filename2 = strDirectory+"//Clusters//"+i;
   FileWriter fstream2 = new FileWriter(filename2 +".txt");
       BufferedWriter out2 = new BufferedWriter(fstream2);
   out2.write(node_list[0]);
33 out2.newLine();
   while(it.hasNext()){
    v=(Vertex) it.next();
    out2.write(node_list[v.ID+1]);
    out2.newLine();
38  nodes_in_cluster++;
   }
   out2.close();
   return nodes_in_cluster;
  }
43}
```

Listing B.5. Cluster Class Java Code

B.6 Vertex Class

```
  import java.util.ArrayList;
2 import java.util.Arrays;
  import java.util.HashSet;
  import java.util.Iterator;

  public class Vertex {
7
   public static int[] global_label_count;
   public static int[] updated_global_label_count;
   public static double rho;
   public static double lambda;
12 public static int num_vertices;
   public static int num_updates;
   public static double probability;
```

```
ArrayList<Vertex> neighbors;
HashSet<Integer> labels,old_labels;
int degree,ID;
private int[] local_label_count;
boolean updated;

public Vertex(int id){
 ID=id;
 degree=0;
 neighbors=new ArrayList<Vertex>();
 labels=new HashSet<Integer>();
 old_labels=new HashSet<Integer>();
 labels.add(ID);
 global_label_count[ID]++;
 updated_global_label_count[ID]++;
}

public void add_neighbor(Vertex v){
 neighbors.add(v);
 degree++;
}

public void compute_best_response(){

 Iterator it=labels.iterator();
 while(it.hasNext()){
  global_label_count[(Integer)it.next()]--;
 }
 old_labels.clear();
 old_labels.addAll(labels);
 labels.clear();

 local_label_count=new int[num_vertices];
 Vertex neighbor;
 HashSet<Integer> neighbor_labels;
 HashSet<Integer> unique_neighbor_labels=new HashSet<Integer>();
 int tmp_label;
 int num_unique_neighbor_labels=0;
```

```
for(int i=0;i<degree;i++){
 neighbor=neighbors.get(i);
 neighbor_labels=neighbor.labels;
 it=neighbor_labels.iterator();
 while(it.hasNext()){
  tmp_label=(Integer)it.next();
  local_label_count[tmp_label]++;
  unique_neighbor_labels.add(tmp_label);
 }
}

num_unique_neighbor_labels=unique_neighbor_labels.size();
Dummy_Object[] dummy_objs=new Dummy_Object[
    num_unique_neighbor_labels];
int unique_label;
it=unique_neighbor_labels.iterator();
int counter=0;
while(it.hasNext()){
 unique_label=(Integer)it.next();
 dummy_objs[counter]=new Dummy_Object(unique_label,
     local_label_count[unique_label]-rho*global_label_count[
     unique_label]);
 counter++;
}

local_label_count=null;
Arrays.sort(dummy_objs);

Dummy_Object cur_dummy_obj;
for(int i=0;i<num_unique_neighbor_labels;i++){
 cur_dummy_obj=dummy_objs[i];
 if(cur_dummy_obj.reward>(lambda*i)){
  labels.add(cur_dummy_obj.label);
  global_label_count[cur_dummy_obj.label]++;
 }
}



if(!labels.equals(old_labels))
```

```
92      num_updates++;
    }

    public void simultaneous_best_response(){

97    Iterator it=old_labels.iterator();
    labels.clear();
    while(it.hasNext()){
     updated_global_label_count[(Integer)it.next()]--;
    }
102   local_label_count=new int[num_vertices];
    Vertex neighbor;
    HashSet<Integer> neighbor_labels;
    HashSet<Integer> unique_neighbor_labels=new HashSet<Integer>();
    int tmp_label;
107   int num_unique_neighbor_labels=0;

    for(int i=0;i<degree;i++){
     neighbor=neighbors.get(i);
     neighbor_labels=neighbor.old_labels;
112    it=neighbor_labels.iterator();
     while(it.hasNext()){
      tmp_label=(Integer)it.next();
      local_label_count[tmp_label]++;
      unique_neighbor_labels.add(tmp_label);
117    }
    }

    num_unique_neighbor_labels=unique_neighbor_labels.size();
    Dummy_Object[] dummy_objs=new Dummy_Object[
        num_unique_neighbor_labels];
122   int unique_label;
    it=unique_neighbor_labels.iterator();
    int counter=0;
    while(it.hasNext()){
     unique_label=(Integer)it.next();
127    dummy_objs[counter]=new Dummy_Object(unique_label,
         local_label_count[unique_label]-rho*global_label_count[
         unique_label]);
     counter++;
```

```
    }

    local_label_count=null;
    Arrays.sort(dummy_objs);

    Dummy_Object cur_dummy_obj;
    for(int i=0;i<num_unique_neighbor_labels;i++){
     cur_dummy_obj=dummy_objs[i];
     if(cur_dummy_obj.reward>(lambda*i)){
      labels.add(cur_dummy_obj.label);
      updated_global_label_count[cur_dummy_obj.label]++;
     }
    }

    if(!labels.equals(old_labels))
     num_updates++;
   }

   public void display(){
    Iterator it=labels.iterator();
    System.out.print("Vertex "+ID+" : ");
    while(it.hasNext()){
     System.out.print((Integer)it.next()+" ");
    }
    System.out.println();
   }


}
```

Listing B.6. Vertex Class Java Code

B.7 Dummy Object Class

```

public class Dummy_Object implements Comparable{

 int label;
 double reward;
```

```
 public Dummy_Object(int l,double r){
  this.label=l;
  this.reward=r;
 }

 public int compareTo(Object obj){
  Dummy_Object other_obj=(Dummy_Object)obj;
  if(other_obj.reward>this.reward)
   return 1;
  else
   return -1;
 }


}
```

Listing B.7. Dummy Object Java Code

Bibliography

[1] Easley, D. and J. Kleinberg (2010) *Networks, Crowds, and Markets*, Cambridge Books, Cambridge University Press.

[2] Papadopoulos, S., A. Skusa, A. Vakali, Y. Kompatsiaris, and N. Wagner (2009) "Bridge Bounding: A Local Approach for Efficient Community Discovery in Complex Networks," *ArXiv e-prints*, 0902.0871.

[3] Tang, L. and H. Liu (2010) *Community Detection and Mining in Social Media*, Morgan & Claypool Publishers.

[4] Kaminsky, A. (2009) *Building Parallel Programs: SMPs, Clusters and Java, International Edition*, Cengage South-Western.

[5] Kaplan, A. M. and M. Haenlein (2010) "Users of the world, unite! The challenges and opportunities of Social Media," *Business Horizons*, **53**(1), pp. 59 – 68.

[6] (2009) "Social media: The new hybrid element of the promotion mix," *Business Horizons*, **52**(4), pp. 357 – 365.

[7] Goldberg, M., S. Kelley, M. Magdon-ismail, K. Mertsalov, and A. Wallace (2009), "Finding Overlapping Communities in Social Networks," .

[8] Xie, J., S. Kelley, and B. K. Szymanski (2011) "Overlapping Community Detection in Networks: the State of the Art and Comparative Study," *CoRR*, **abs/1110.5813**.

[9] Mandala, S., S. Kumara, and K. Chatterjee (2011) *A Game Theoretic Approach to Graph Clustering, Tech. rep.*, Department of Industrial Engineering,The Pennsylvania State University, University Park, Pennsylvania.

[10] Traud, A. L., E. D. Kelsic, P. J. Mucha, and M. A. Porter (2008) "Comparing Community Structure to Characteristics in Online Collegiate Social Networks," *ArXiv e-prints*, 0809.0690.

[11] Traud, A. L., P. J. Mucha, and M. A. Porter (2011) "Social Structure of Facebook Networks," *ArXiv e-prints*, 1102.2166.

[12] McKee, J. (2006), "Community Guy," .
URL http://moblogsmoproblems.blogspot.com/2006/10/

[13] Thadakamalla, H. P., R. Albert, and S. R. T. Kumara (2006) "Complexity and Large-scale Networks," in *Operations Research and Management Science Handbook*, CRC Press, p. chap 11.

[14] Brin, S. and L. Page (1998) "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, **30**(1-7), pp. 107 – 117, ¡ce:title¿Proceedings of the Seventh International World Wide Web Conference¡/ce:title¿.

[15] Kleinberg, J. M. (1999) "Authoritative sources in a hyperlinked environment," *J. ACM*, **46**, pp. 604–632.

[16] Albert, R. and A.-L. Barabási (2002) "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, **74**, pp. 47–97.

[17] Haykin, S. (1994) *Neural Networks: A Comprehensive Foundation*, 1st ed., Prentice Hall PTR, Upper Saddle River, NJ, USA.

[18] Grossman, J., P. Ion, and R. Castro (2008), "Erdos Number Project," .
URL http://www.oakland.edu/enp/

[19] Raghavan, U. N., R. Albert, and S. Kumara (2007) "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E*, **76**, p. 036106.

[20] Barabsi, A.-L. and R. Albert (1999) "Emergence of Scaling in Random Networks," *Science*, **286**(5439), pp. 509–512.

[21] Diesner, J., T. Frantz, and K. Carley (2005) "Communication Networks from the Enron Email Corpus It's Always About the People. Enron is no Different," *Computational & Mathematical Organization Theory*, **11**, pp. 201–228.

[22] Leskovec, J. and E. Horvitz (2008) "Planetary-scale views on a large instant-messaging network," in *Proceeding of the 17th international conference on World Wide Web*, WWW '08, ACM, New York, NY, USA, pp. 915–924.

[23] Nanavati, A. A., S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea, and A. Joshi (2006) "On the structural properties of massive telecom call graphs: findings and implications," in *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pp. 435–444.

[24] Mislove, A., M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee (2007) "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pp. 29–42.

[25] Palla, G., A.-L. Barabási, and T. Vicsek (2007) "Quantifying social group evolution," *Nature*, **446**, pp. 664–667.

[26] Hechter, M. (1988) *Principles of Group Solidarity*, California Series on Social Choice and Political Economy, University of California Press.

[27] Zachary, W. W. (1977) "An Information Flow Model for Conflict and Fission in Small Groups," *Journal of Anthropological Research*, **33**(4).

[28] Papadopoulos, S., Y. Kompatsiaris, A. Vakali, and P. Spyridonos "Community detection in Social Media," *Data Mining and Knowledge Discovery*, pp. 1–40.

[29] Kovacs, I. A., R. Palotai, M. S. Szalay, and P. Csermely (2010) "Community Landscapes: An Integrative Approach to Determine Overlapping Network Module Hierarchy, Identify Key Nodes and Predict Network Dynamics," *PLoS ONE*, **5**(9), p. e12528.

[30] Wasserman, S. and K. Faust (1994) *Social network analysis: methods and applications*, Structural analysis in the social sciences, Cambridge University Press.

[31] Scott, J. (2000) *Social network analysis: a handbook*, SAGE Publications.

[32] Borgatti, S. P., M. G. Everett, and P. R. Shirey (1990) "LS sets, lambda sets and other cohesive subsets," *Social Networks*, **12**(4), pp. 337 – 357.

[33] ma, J. and S. Schaeffer (2006) "On the NP-Completeness of Some Graph Cluster Measures," in *SOFSEM 2006: Theory and Practice of Computer Science* (J. Wiedermann, G. Tel, J. Pokorn, M. Bielikov, and J. tuller, eds.), vol. 3831 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 530–537, 10.1007/11611257_51.

[34] Clauset, A., M. E. J. Newman, and C. Moore (2004) "Finding community structure in very large networks," *Phys. Rev. E*, **70**, p. 066111.

[35] Luo, F., J. Wang, and E. Promislow (2006) "Exploring Local Community Structures in Large Networks," in *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on*, pp. 233 –239.

[36] Shi, J. and J. Malik (2000) "Normalized Cuts and Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, **22**, pp. 888–905.

[37] Kannan, R., S. Vempala, and A. Vetta (2004) "On clusterings: Good, bad and spectral," *J. ACM*, **51**, pp. 497–515.

[38] Newman, M. E. J. and M. Girvan (2004) "Finding and evaluating community structure in networks," *Phys. Rev. E*, **69**, p. 026113.

[39] Pons, P. and M. Latapy (2005) "Computing Communities in Large Networks Using Random Walks," in *Computer and Information Sciences - ISCIS 2005* (p. Yolum, T. Gngr, F. Grgen, and C. zturan, eds.), vol. 3733 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 284–293, 10.1007/11569596_31.

[40] Lorrain, F. and H. C. White (1971) "Structural equivalence of individuals in social networks," *The Journal of Mathematical Sociology*, **1**(1), pp. 49–80.

[41] Palla, G., I. Derenyi, I. Farkas, and T. Vicsek (2005) "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, **435**(7043), pp. 814–818.

[42] von Luxburg, U. (2007) "A tutorial on spectral clustering," *Statistics and Computing*, **17**, pp. 395–416, 10.1007/s11222-007-9033-z.

[43] Duch, J. and A. Arenas (2005) "Community detection in complex networks using extremal optimization," *Phys. Rev. E*, **72**, p. 027104.

[44] Massen, C. P. and J. P. K. Doye (2005) "Identifying communities within energy landscapes," *Phys. Rev. E*, **71**, p. 046101.

[45] Newman, M. E. J. (2006) "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E*, **74**, p. 036104.

[46] Girvan, M. and M. E. J. Newman (2002) "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, **99**(12), pp. 7821–7826.

[47] Tsatsou, D., S. Papadopoulos, I. Kompatsiaris, and P. C. Davis (2011) "Distributed Technologies for Personalized Advertisement Delivery." in *Online Multimedia Advertising: Techniques and Technologies* (X.-S. Hua, T. Mei, and A. Hanjalic, eds.), Hershey: IGI Global, pp. 233–261.

[48] Zhao, Q., P. Mitra, and B. Chen (2007) "Temporal and information flow based event detection from social text streams," in *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI Press, pp. 1501–1506.

[49] Rees, B. S. and K. B. Gallagher (2010) "Overlapping Community Detection by Collective Friendship Group Inference," in *2010 International Conference on Advances in Social Networks Analysis and Mining*, IEEE, pp. 375–379.

[50] Nguyen, N. P., T. N. Dinh, D. T. Nguyen, and M. T. Thai (2011) "Overlapping Community Structures and Their Detection on Social Networks," in *3rd IEEE International Conference on Social Computing (SOCIALCOM)*.

[51] Padrol-Sureda, A., G. Perarnau-Llobet, J. Pfeifle, and V. Muntes-Mulero (2010) "Overlapping Community Search for social networks," *Data Engineering, International Conference on*, **0**, pp. 992–995.

[52] Pizzuti, C. (2009) "Overlapped community detection in complex networks," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09.

[53] Goldberg, M., S. Kelley, M. Magdon-Ismail, K. Mertsalov, and A. Wallace (2010) "Finding Overlapping Communities in Social Networks," in *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, SOCIALCOM '10, IEEE Computer Society, Washington, DC, USA, pp. 104–113.

[54] Baumes, J., M. Goldberg, M. Krishnamoorthy, M. Magdon-Ismail, and N. Preston (2005) *Finding communities by clustering a graph into overlapping subgraphs*, pp. 97–104.

[55] Sawardecker, E. N., M. Sales-Pardo, and L. A. Amaral (2009) "Detection of node group membership in networks with group overlap," *The European Physical Journal B - Condensed Matter and Complex Systems*, **67**, pp. 277–284.

[56] Chen, W., Z. Liu, X. Sun, and Y. Wang (2010) "A game-theoretic framework to identify overlapping communities in social networks," *Data Mining and Knowledge Discovery*, **21**, pp. 224–240.

[57] Psorakis, I., S. Roberts, M. Ebden, and B. Sheldon (2011) "Overlapping community detection using Bayesian non-negative matrix factorization," *Phys. Rev. E*, **83**, p. 066114.

[58] Xie, J., S. Kelley, and B. K. Szymanski (2011) "Overlapping Community Detection in Networks: the State of the Art and Comparative Study," .

[59] Gregory, S. (2009) "Finding Overlapping Communities Using Disjoint Community Detection Algorithms," *Complex Networks*, **207**, p. 4761.

[60] Li, X., B. Liu, and P. Yu (2006) "Discovering Overlapping Communities of Named Entities," in *Knowledge Discovery in Databases: PKDD 2006* (J. Frnkranz, T. Scheffer, and M. Spiliopoulou, eds.), vol. 4213 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 593–600, 10.1007/11871637_60.

[61] Wei, F., C. Wang, L. Ma, and A. Zhou (2008) "Detecting Overlapping Community Structures in Networks with Global Partition and Local Expansion," in *Progress in WWW Research and Development* (Y. Zhang, G. Yu, E. Bertino, and G. Xu, eds.), vol. 4976 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 43–55, 10.1007/978-3-540-78849-2_7.

[62] Shannon, C. E. (2001) "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, **5**, pp. 3–55.

[63] Balakrishnan, A., S. R. T. Kumara, and S. Sundaresan (1999) "Manufacturing in the Digital Age: Exploiting Information Technologies for Product Realization," *Information Systems Frontiers*, **1**, pp. 25–50.

[64] Lee, C. (2011), "The Sociograph," .
URL `http://sociograph.blogspot.com/2011/03/`