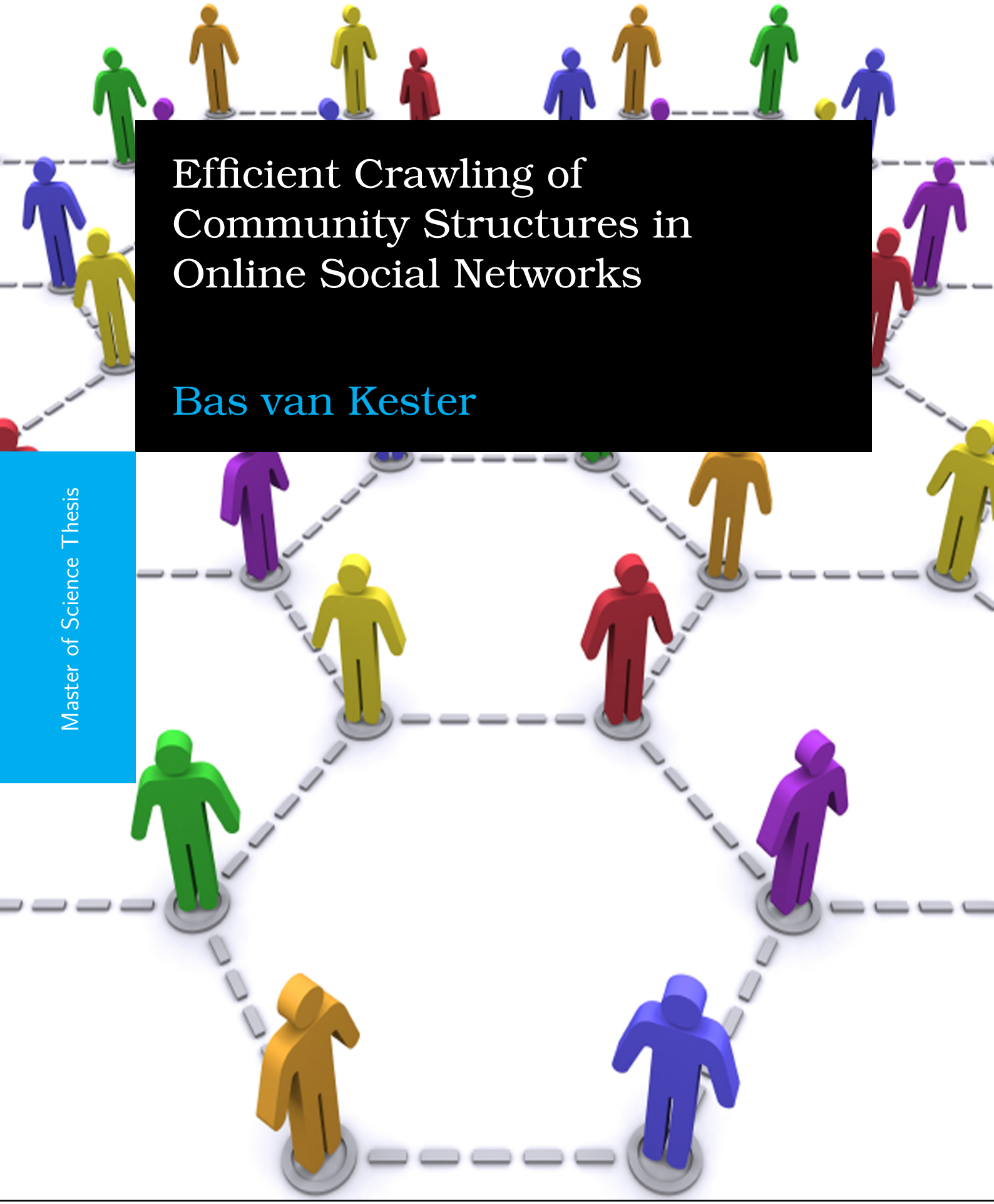


Efficient Crawling of Community Structures in Online Social Networks

Bas van Kester

Master of Science Thesis

TU Delft Delft University of Technology

Network Architectures and Services

PVM 2011-071

Efficient Crawling of Community Structures in Online Social Networks

Master of Science Thesis

For the degree of Master of Science in
Network Architectures and Services Group (NAS)
at Department of Telecommunications
at Delft University of Technology

Bas van Kester
Student no. 1213679
Thesis no. PVM 2011-071

September 15, 2011

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands

TU Delft Delft University of Technology

Abstract

Online social networks showed an enormous growth in the last decade. With the rise of online social networks such as Twitter and Facebook, researchers got the opportunity to access the data of social behavior of millions of people, whereas in the past it was limited to hundreds of people. For these researchers and marketeers it is of great interest to find communities within these large networks, as this is one of the opportunities to see how people behave in groups on a large scale.

The most common approach of analyzing community structures in online social networks is to gather the network by downloading the user profiles one by one (crawling) and afterwards partition the network into groups or communities by community detection algorithms. However, crawling an entire social network is very time consuming and analyzing the networks with community detection algorithms can be computationally expensive.

To overcome these problems, in this thesis a method is proposed for crawling nodes using the community structure of a network. It enables the researcher to start the analysis before completing the crawl. This new method performs between 66% and 480% better than existing crawling techniques such as Breadth First Search (BFS) and Depth First Search (DFS), because a smaller portion of the networks has to be crawled in order to crawl entire communities. The computer-generated networks used in this thesis were created using a new network generator which uniquely combines three features; it creates networks with explicit community structure, arbitrary degree distributions and adaptable community strength.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1-1 Background

Online social networks have grown enormously over the last decade. Online social networks such as Twitter and Facebook enabled social researchers to obtain and analyze the social behavior of millions of people, where in the past it was limited to hundreds of people. It is of great interest to find communities in these large networks for researchers and marketeers, as it is an opportunity to observe how people behave in groups on a large scale.

The most common approach of analyzing community structures in online social networks is to gather the network by downloading the user profiles one by one (crawling) and afterwards partition the network into groups or communities by community detection algorithms.

1-2 Motivation

Obtaining these large networks can be very time consuming. To give an example, downloading all 750 million profiles on Facebook one by one with an average speed of 10 profiles per second, would take more than 2 years. Reducing the problem by sampling only parts of the network does not work well because common crawling methods such as Breadth First Search (BFS) and Depth First Search (DFS) do not crawl entire communities. For instance, BFS has a preference for high degree nodes which are usually the hubs between communities. The low degree nodes are crawled last [2]. To make matters even worse, most community detection algorithms do not scale well with the size of the network and it can take a long time to calculate the partitioning or they result in poor partitioning.

To overcome these problems, in this thesis a new crawling method is proposed that aims to crawl entire communities before continuing to other communities. By crawling entire communities, the analysis of community structure might start before the entire crawl of the network is finished. This could reduces the amount of work that has to be performed in both obtaining the network and analyzing the network.

The performance of the proposed crawling method will be measured by the portion of a network that has to be crawled in order to completely crawl a number of clusters. For this evaluation, each node has to be assigned to a single community, however this explicit community structure is generally not available for real world social networks. For this purpose, community detection algorithms are used. Because there is no consensus on the most appropriate algorithm, multiple algorithms are considered and compared.

In order to be able to compare the existing and new crawling techniques for a variety of scenarios, multiple networks will be used. Real world social networks are considered, but in order to vary the network properties, artificial networks are used. Existing network generators have limitations as they can only create networks with one type of degree distribution. Therefore an improved network generator is proposed, implemented and evaluated. Those generated networks are also used to compare the performance of community detection algorithms.

Finally, when for the real world networks the explicit community structure is found and the artificial networks are generated, the new crawling method is evaluated against them and compared to BFS and DFS.

In short the following goals are set for this thesis:

- Propose and implement a network generator where community strength and degree distributions can be configured
- Compare community detection algorithms using artificial networks and apply the community detection algorithm to a network where no explicit community structure is available
- Propose and implement an algorithm for crawling complete communities
- Evaluate the performance of the proposed crawling algorithm and compare it to existing crawling methods such as BFS and DFS

The means to achieve these goals of this thesis are presented in the rest of the chapters, as described in the following overview.

1-3 Thesis Overview

The remainder of the thesis will be structured as follows. In chapter 2 the related work in literature is presented. Social networks are introduced with a focus on community structure. Furthermore, community detection algorithms are discussed as well as ways to compare partitioning created by those algorithms. Commonly used methods to obtain social networks will be presented. Afterwards, in chapter 3 an algorithm for creating networks with community structure is proposed and evaluated. Using these networks the performance of multiple community detection algorithms is evaluated. In chapter 4, a method for crawling through a networks using its community structure is proposed. The method is evaluated using multiple social networks and networks that will be created using the generator of the previous chapter. Finally, chapter 5 concludes this thesis by summarizing its conclusions and presenting some possible directions for future work.

Chapter 2

Related Work

In this chapter the related work that is needed for the rest of the thesis will be presented. To understand the proposed crawling technique, it is necessary to understand what social networks are and what kind of properties they have (section 2-1). Traditional approaches to analyze community structures use community detection algorithms (section 2-2) to partition after the full dataset of the social network is obtained. In this thesis these algorithms will be used to analyze networks where no "ground truth" for partitioning is available. Because there is no consensus on the optimal community detection algorithm, the partitioning by different algorithms have to be compared (section 2-3). The background and properties of datasets of social networks that will be used later in this thesis, will be addressed (section 2-4). Related work on network generators will be presented, because networks created by such a generator will later be used to compare different crawling methods. Finally existing crawling techniques will be discussed in section 2-5 so they later can be compared with the proposed crawling algorithm.

2-1 Properties of social networks

First, to be able to crawl social network, it is necessary to understand the properties of these networks or graphs. In this section various properties observed in social networks are discussed that are commonly used to characterize them. Two ways in which social networks are obtained are described in section 2-1-1. In this section also the sizes of social networks (section 2-1-2) and degree distribution (section 2-1-3) will be introduced, because they influence the way that the network can be analyzed. Finally community structures are described in section 2-1-4.

2-1-1 Ways to obtain social networks

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

beliefs, knowledge or prestige with links. In this thesis two types of social networks are defined by their way of gathering, traditional social networks and online social networks (OSN).

Social networks are gathered manually by social researchers which follow a group of people over a period of time to investigate their social interaction. This type of network will be called traditional social networks because of the way they are obtained. Famous examples of traditional social networks are the Zachary's karate club [3] and the football network of Girvan et al. [1]. In the karate club network the nodes represent members of a karate club and the links are the social interactions between them. In the football network, the nodes represent football teams and a link exists if they played a match against each other. More details about this network can be found in section 2-4.

In online social networks such as Facebook, Hyves and Twitter, user profiles are represented as nodes and relationships as links. This type of network can be gathered using crawling techniques such as Breadth First Search and Depth First Search which are discussed in section 2-5, or directly by asking the operator. In Facebook and Hyves if a user befriends another one, both users have to agree upon the relation. The network is therefore undirected, whereas in Twitter a user can decide to follow another user without asking for permission which leads to a directed network.

2-1-2 Sizes

The size of a network which is commonly defined by the number of nodes, determines the way a network can be obtained and analyzed. While small social networks can be gathered and interpreted by hand, for larger networks that becomes infeasible and automated methods have to be used. For large networks even automated methods to analyze networks can be infeasible due to their runtime. Therefore the size is an important property of a network. The number of nodes in a social network heavily depends on the type. Traditional social networks are gathered through field studies by researchers and therefore have a limited number up to hundreds of nodes. The process of gathering online social networks can be automatized using the crawling methods discussed in section 2-5 and therefore the networks can be much larger. For instance, Facebook has 750 million users at the moment of writing [4]. One of the largest networks that is fully available for researchers is a Twitter dataset from 2009 which contains 50 million users and 1.5 billion links and was made available by Kwak et al. [5].

2-1-3 Degree distribution

Besides the size of the network, another important property of social networks is how the links are distributed among the nodes, in other words how the distribution of friends is among the users of a social networks. This property is not only relevant for understanding the relationships between entities, it also influences the order in which nodes will be obtained in certain crawling methods. The degree is defined as the number of adjacent neighbors of a node. The degree distribution $P(k)$ is the probability for a fraction of nodes in a network having degree k. A frequently observed property in many online social networks is their power-law degree distribution. A power-law degree distribution is described by the following relation.

$$P(k) \sim k^{-\gamma} \tag{2-1}$$

Here k represents the degree, γ the exponent and $P(k)$ the degree distribution. Nodes with a high degree are called hubs, because they have a more central position in the network. According to Newman [6] social networks are assortatively mixed, which means that hubs are more likely to be connected to other hubs.

2-1-4 Community structure

Community structure is the main focus of this thesis and will be addressed in the following section. In this thesis a general definition of community structure introduced by Newman and Girvan [7] is used, namely "The division of network nodes into groups within which the network connections are dense, but between which are sparser".

An example of a network with community structure is depicted in Figure 2-1. Nodes in a commmunity should share more connections with each other than with nodes in other communities. In the example, nodes within a community are completely connected meaning that all possible links within the community exists, while there are few links between nodes of different communities.

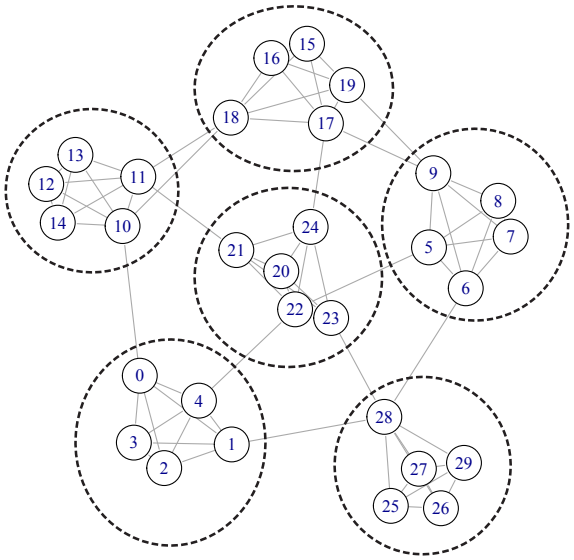


Figure 2-1: Example of a network with 6 communities, highlighted by the dashed circles

Fortunato and Castellano [8] give a good overview on the field of community structures in networks. In this section hierarchy in communities will be discussed and observations in real world social networks are described.

Communities can have various levels of organization, where communities consist of multiple sub-communities. This phenomena is known as hierarchies. For example in a network of university employees, the community of a university can be subdivided in different faculties and even further in departments. Hierarchies are used in community detection algorithms such as Edge Betweenness clustering and Fast Greedy community detection. Because community detection algorithms such as Label Propagation and Spinglass do not find such hierarchies, this type of structure is not considered in the analysis of crawling methods.

Online social networks analyzed with community detection algorithms showed that there is not an average community size, but rather is a power-law distribution for the community size [9]. Small communities seem to coexist with large communities. However, in the networks that were used, relationships or links were formed based on common interest. However, a recent paper of Leskovec et al. [10] showed that in nearly all real world networks they observed, communities gradually get less and less community-like and instead gradually "blend in" with the rest of the network, as the communities steadily grow in size. They observed that for communities size of about 100 nodes, the "quality" of communities gets worse and worse. This agrees well with an intuitive notion of communities where people have interpersonal ties with each other. For instance on schools, each class has a similar size and students are densely connected. On a higher level, students from different classes can know each other, but are less densely connected that within the classes itself. Students from different schools are even less likely to form a relation. For the networks that will be generated, the focus will be on those small and dense groups therefore equally sized communities are used.

2-1-5 Summary

In this section various properties of social networks were presented. Two ways to obtain social networks are considered, traditional social networks constructed by researchers and online social networks which can be automatically obtained by crawlers. The datasets used in traditional social networks are small compared to online social networks. In those online social networks are often power-law degree distributions are observed. Nodes in social networks can be grouped together to form communities. In this thesis the definition of Newman [7] for community structure is used: "The division of network nodes into groups within which the network connections are dense, but between which are sparser". Hierarchy is a property in which communities can be divided into sub-communities, which is used by community detection algorithms such as Fast Greedy and Edge Betweenness community detection. Recent observations showed that community sizes have a limited size and therefore in the networks in section 3 will be generated using equally sized communities of about a 100 nodes.

2-2 Community detection algorithms

The aim of community detection in graphs is to identify groups based on the network topology. These algorithms will be used for networks to assign nodes to clusters where there is no "ground truth" available. This section gives a brief overview of commonly used community detection algorithms. For some community detection algorithms the runtime to complete partitioning does not scale well with the size of the network and do not give results in a reasonable amount of time for large networks; therefore the time complexity of each algorithm is discussed.

2-2-1 Edge betweenness clustering

Girvan and Newman [1] proposed an algorithm one of the first successful algorithms to partition a graph into communities. This paper sparked a big activity in the field of community detection. The results of this algorithm are often used as a reference when there is no "ground

truth" available for testing the performance of new community detection algorithms. Edge betweenness measures the number of all shortest paths passing a link, which gives an intuitive measure for the importance of that link when connecting two communities. The most important link according to the edge betweenness is removed and the edge betweenness is recalculated. This process is repeated until all links are removed. The recalculation of the edge betweenness after each removal is important for the algorithm to give good results, but this step also makes it time consuming to calculate the community structure for large networks. The time complexity of the original algorithm is $O(nm^2)$ where m is the amount of links and n is the amount of nodes. Therefore it is only feasible to use this method on networks containing with up to about 10000 nodes on commodity hardware. Improvements can be made by parallelization [11], sampling and the use of specialized hardware [12]. Because this method is not usable for large networks, three other community detection algorithms with better time complexity are discussed and which are available in the commonly used IGraph library.

2-2-2 Label propagation

Label propagation works in near linear time and is therefore a good candidate for being used as a community detection algorithm for online social networks. In this method all nodes get a unique label and based on their topology nodes that share many links merge their label until the network is partitioned in communities. Label propagation was first proposed by Raghavan et al. [13]. The low time complexity comes at the expense that the algorithm is greedy and therefore does not give the optimal solution and vary for each run.

2-2-3 Spinglass

The Spinglass community detection uses an analogy with a phenomena from physics. In a ferromagnetic system the spins of elements should have the same orientation to have minimum energy. Links inside a cluster that exist and non-existing links between clusters are rewarded while links that do not exists within a cluster and links between clusters are penalized [14]. This is considered the energy of the system and should be minimal. Because there can be multiple local minima, there should be optimized for the global minimum to get the best partitioning for this algorithm. This can be achieved using simulated annealing (Bertsimas et al. [15]).

The time complexity of the Spinglass algorithm combined with simulated annealing is not provided by the documentation of IGraph or the paper in which Spinglass was proposed. It is determined by the maximum number of iterations which is an input parameter and by the time complexity for calculating the energy of the system.

2-2-4 Fast greedy community detection

The Fast Greedy community detection algorithm is a method which works on modularity and was proposed by Clauset et al. [9]. At the start, each node is assigned to an individual cluster. Initially for connected combinations of clusters the change in modularity ΔQ_{ij} is calculated. The combination with the highest increase is merged together into a new cluster. Only for

the affected clusters, the ΔQ_{ij} is updated. Again the combination with the highest increase in ΔQ_{ij} is merged. This process is repeated until all the nodes are in one giant cluster. The partitioning with the highest modularity is selected as the partitioning.

This process is both memory efficient and fast, because the ΔQ_{ij} only has to be updated for affected clusters. The time complexity of the algorithm for sparse graphs such as online social networks is $O(nlog^2n)$ which is near linear time.

2-2-5 Summary

Community detection algorithms are used to group sets of nodes based on their topology. Four commonly used algorithms are discussed. Edge betweenness community detection is the most known algorithm and although it is commonly used as a test bench for new community detection algorithms, its time complexity makes it unusable for large online social networks (> 10000 nodes). Spinglass community detection can handle larger networks. Label propagation and Fast Greedy community detection have near linear time complexity and can be used for a wide variety of network sizes.

2-3 Comparison of community detection algorithms

To be able to compare algorithms, three different metrics are described in this section. These metrics are meaningful as they quantify differences in partitioning by community detection algorithms. In section 3-3 these comparison methods will be used to find the most appropriate community detection algorithm for partitioning generated networks. This algorithm can then be used to analyze a network without a "ground truth". The three metrics that will be presented are modularity, f_{same} and Jaccard index. Modularity is a global metric and compares the result to random graphs whereas the f_{same} and the Jaccard index compare results to each other.

2-3-1 Modularity

A measure for quantifying how well a graph is partitioned in clusters is modularity [7]. This measure is commonly used to quantify the performance of community detection algorithms against a null hypothesis. Modularity measures the fraction of links in a network that connect nodes of the same community minus the expected value of the same quantity in a random network. It can be written as

$$Q = \sum_i (e_{ii} - a_i^2) \tag{2-2}$$

where e_{ii} is the fraction of links inside community i and a_i^2 is the fraction of links that connect to community i which can be calculated using $a_i^2 = \sum_j e_{ij}$.

If the number of intra-cluster links is as high as in random graph, than $Q = 0$. Values approaching $Q = 1$ indicate a strong community structure. In practice values for strong community structure typically fall in the range from 0.3 to 0.7 [7]. With this measure it is

difficult to compare two graphs with similar modular structure but different sizes because the larger graph will get a higher modularity [16].

2-3-2 F same

The f_{same} measure can be used to find the similarity of two partitions with each other. It uses an intuitive notion of similarity, it quantifies the percentage of nodes put in the same community by different community detection algorithms. To calculate f_{same} first a matrix M has to be constructed in which M_{ij} is the number of nodes assigned to one community by community detection algorithm i and community detection algorithm j. f_{same} is calculated as following in Raghavan et al. [13]:

$$f_{same} = \frac{1}{2}(\sum_{i} max_j\{M_{ij}\} + \sum_{j} max_i\{M_{ij}\})\frac{100}{n} \tag{2-3}$$

When the community structure is known a priori, this can be used as an input for the measure. Then the performance of the community detection algorithms is measured against a "ground truth". The metric can identify how close one solution is to another one, but is not sensitive to the seriousness of the errors. An example given in Raghavan et al. [13] is: when a few nodes form several different communities in one solution are fused together as a single community; this does not change the value of f_{same} much. To compensate this problem, the Jaccard index is introduced.

2-3-3 Jaccard index

A second measure for similarity of partitions is the Jaccard index which is discussed by Fortunato et al. [8]. It looks at the amount of pairs of nodes that are shared in both partitions. Given two partitions A and B into communities, the Jaccard index is defined as

$$I_{jaccard}(A, B) = \frac{n_{11}}{n_{11} + n_{01} + n_{10}} \tag{2-4}$$

where n_{11} is the number of pairs of nodes which are in the same community in both partitions and n_{10} and n_{01} denotes the numbers of pairs of elements which are put in the same community in by algorithm A but not by algorithm B and vice versa. The output of the Jaccard index is between 0 and 1 where 1 is the highest possible similarity. Although it is less intuitive than f_{same} , the Jaccard index is known to be more sensitive to errors in partitioning than the f_{same} index and therefore useful for finding the similarity in partitions of networks.

2-3-4 Summary

Modularity is a measure for the strength of a partitioning of a graph and is often used to quantify the performance of community detection algorithms. The Jaccard index and f_{same} index can be used to compare the partitioning of multiple community detection algorithms. The f_{same} index uses a more intuitive notion of how many nodes are shared in the two partitioning. The Jaccard index is more sensitive to the seriousness of errors in partitioning than the f_{same} index.

2-4 Datasets

For testing the crawling methods, three sources of datasets are used and these are introduced in the following section. A college football network introduced by Girvan and Newman and an email communication network of the Enron corporation [17] are presented, as well as generated cluster networks are discussed.

2-4-1 Football network

Girvan and Newman [1] introduced the United States college football network which is a representation of the schedule of Division I games for the season of year 2000. Nodes in the graph represent teams (identified by their college names) and links represent regular-season games between the two teams they connect. What makes this network interesting is that it incorporates a known community structure which will be called explicit community structure. The teams are divided into conferences containing around 8 and 12 teams each. Games are more frequent between members of the same conference than between members of different conferences, with teams playing on average about seven intra-conference games and four inter-conference games in the 2000 season. Inter-conference play is not uniformly distributed; teams that are geographically close to one another but belong to different conferences are more likely to play one another than teams separated by large geographic distances.

Its relatively small size and explicit community structure makes it suitable for manually verifying the performance of different crawling techniques which will be introduced in section 2-5.

2-4-2 Enron network

The Enron email network is a social network constructed from email communication in a large corporation. Although it has no explicit community structure, due to its size it can be partitioned by algorithms such as Label Propagation, Spinglass and Fast Greedy community detection. Other networks such as Hyves and Twitter are infeasible to partition using those three algorithms due to their run time.

The Enron corpus, a large set of email messages, was made public during the legal investigation concerning the Enron corporation. The raw Enron corpus contains 619,446 messages sent to and received by 158 Enron employees. From this email corpus a network was constructed in which the email addresses are represented by nodes and emails between two addresses are represented by links [17, 18]. This network contains 36692 nodes and 183831 undirected links. Only the connected component is used in crawling techniques and therefore the network can further be reduced to a connected network of 33692 nodes and 180811 undirected links. Non-Enron email addresses act as sinks and sources because only emails from and to Enron employees were included in the corpus. The network does not contain an explicit community structure, but the community structure will be found using community detection algorithms in section 4-2-2.

2-4-3 Computer-generated cluster networks

A generator which creates networks with an explicit community structure where the density of intra-cluster links can be varied, is essential to verify the performance of crawling methods under different conditions. Furthermore, the generator should at least support the generation of networks with power-law degree distribution as they are often observed in online social networks. The most famous and commonly used model for generating networks is the Erdos-Reyni model. This model creates a graph with N nodes where the links are chosen independently with a probability p which results in a binomial degree distribution. This type of network does not have an explicit community structure. Girvan and Newman [1] proposed a generator which uses two probabilities, one for inter-cluster links and one for intra-cluster links. In this type of network an explicit community structure is available, but the shape of the degree distribution can not be influenced. A graph generator in which the degree distribution could be defined would be convenient to test the performance of crawling methods.

Lancichinetti et al. [19] proposed a generator that produces networks with power-law degree distribution and power-law cluster size distribution. However, recent observations showed that communities should have a maximum size which indicate that communities have similar sizes. Furthermore, for verifying the behavior of the crawling methods, equally sized clusters would be more insightful. This generator produces networks in which all nodes have the same ratio of links going inside and outside the community, even for hubs. Hubs interconnect different parts of the network and should therefore have more links to other clusters than low degree nodes. In chapter 3 a generator is proposed that has an explicit equally sized community structure, definable degree distribution and variating community strength.

2-5 Crawling techniques

An important step in the analysis of social networks is to obtain the network. Decisions taken in this step determine the outcome in the further analysis. In this section first ways to access data on online social networks are discussed. Finally two well known crawling methods are presented.

In online social network analysis there are three ways to obtain the network. Getting the corpus from other researchers or the social network operator itself are the least challenging in a technical sense. However most social network operators are not willing to share their corpus. Therefore in a lot of cases it is necessary to obtain the network through crawling. In general there are two ways to interact with the social network; access the online social network through an API or by screen scraping. Both types have advantages and disadvantages which are listed in the table 2-1. Generally API usage is preferred above screen scraping because it uses less resources in data traffic and processing power. However, generally more data is available by screen scraping than by API requests.

For both screen scraping and API access, there are several algorithms to obtain a network, which will be discussed below.

Table 2-1: Comparison of crawling through API requests and screen scraping

API	Screen scraping
+ Bandwidth efficient	- Bandwidth inefficient
+ Easy to parse	- HTML / javascript has to be parsed
+ Less likely to violate the terms of usages	- Likely to violate terms of usage
- Number of request per hour can be limited	o less likely to be limited
- In most cases not all data on website is available using an API	+ In principle all data visible to website visitors is available

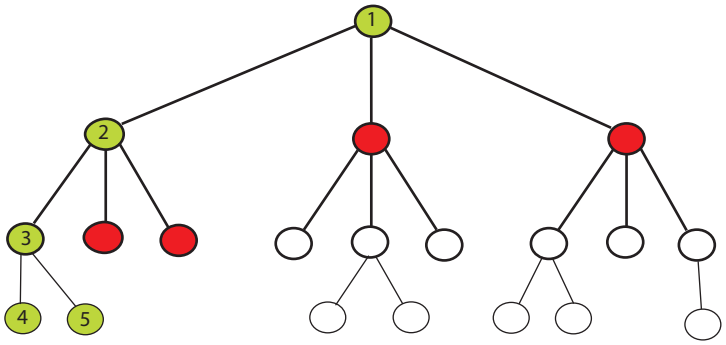


Figure 2-2: Example of DFS

2-5-1 Crawling algorithms

Two well known algorithms to traverse a graph are Breadth First Search (BFS) and Depth First Search (DFS). These algorithms are used in numerous applications. To give a few examples, BFS can be used to find shortest paths in unweighted graphs and verifying if a path exists between two nodes. DFS can be also be used to verify if there is a path between two nodes and find a path out of a maze. In this thesis both algorithms will be used for crawling networks and finding connected components.

The pseudo code for BFS is given by algorithm 1 and an example is shown in Figure 2-3 where a crawl is halted after 5 iterations. The green nodes represent the nodes that have

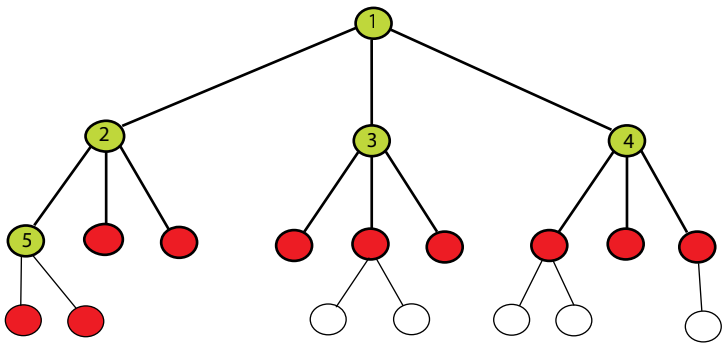


Figure 2-3: Example of BFS

been visited and the red nodes are in the queue.

DFS works in a similar way as BFS, the main difference is that it uses a stack in stead of a queue. The pseudo code can be found in algorithm 2 and an example can be found in Figure 2-2. For both algorithms an iterative implementation is chosen because the recursive variant can cause stack overflows in most common implementations for large networks. The iterative implementation does not have this issue.

A Breadth First Search visits all the successors of a visited node before visiting any successor of any of its child nodes. This in contrast to depth first traversal method, which visits the successor of a visited node before visiting any of its brothers, i.e., children of the same parent. Depth First Search tends to create very long, narrow trees; whereas BFS tends to create very wide, short trees.

BFS is well known to introduce bias towards high degree nodes according to Gjoka et al. [2]. Because hubs have more links pointing towards them, they are more likely to be crawled than low degree nodes. BFS is more commonly used to crawl online social networks than DFS.

None of these two algorithms take advantage of the community structure in networks. Therefore, in chapter 4, a method that aims to crawl a complete community before crawling other communities is proposed. This technique is helpful as the analysis may start while the network is still being crawled.

```
   Data: Node to start s, Graph G
 1 // visitedNodes keeps track which nodes have already been visited
 2 visitedNodes ⟵ ∅;
 3 Q ⟵ EMPTYQUEUE;
 4 Enqueue(Q, s);
 5 while queue ≠ ∅ do
 6 |   v ⟵ Dequeue(Q);
 7 |   INSERT(visitedNodes, v);
 8 |   foreach u ∈ Adj[v] do
 9 |   |   if u ∉ visitedNodes then
10 |   |   |   Enqueue(Q, u);
11 |   |   end
12 |   end
13 end
```

Algorithm 1: Breadth first search

2-6 Summary

In this section the related work that is needed for the rest of the thesis was presented. Social networks and their characteristics were introduced in section 2-1, with a focus on community structures. Ways to detect communities in networks are discussed in section 2-2 as well as ways to compare the partitioning of those algorithms in section 2-3. Various networks will be used in this thesis and were presented in section 2-4. Related work showed that current generators for networks with explicit community structure only support specific degree distributions. For

```
   Data: Node to start s, Graph G
 1 // visitedNodes keeps track which nodes have already been visited
 2 visitedNodes ⟵ ∅;
 3 Q ⟵ EMPTYSTACK;
 4 Push(Q, s);
 5 while queue ≠ ∅ do
 6 |   v ⟵ Pop(Q);
 7 |   INSERT(visitedNodes, v);
 8 |   foreach u ∈ Adj[v] do
 9 |   |   if u ∉ visitedNodes then
10 |   |   |   Push(Q, u);
11 |   |   end
12 |   end
13 end
```

Algorithm 2: Depth first search

verification of the proposed crawling method, it is necessary to use networks with different degree distributions. Therefore in the next chapter an improved network generator will be proposed. Finally methods to obtain online social networks were described in section 2-5. A crawling method that is more useful for obtaining entire communities will be presented in chapter 4.

Chapter 3

Computer-generated cluster networks

In order to compare the performance of the crawling methods for a variety of scenarios, generated networks with community structure are needed. In commonly used computer-generated random networks such as Erdös-Rényi or Barabási-Albert, the generators do not produce graphs with predefined clusters. In this thesis a method is proposed to generate networks with explicit community structure where each node is assigned to one cluster. The strength of the clustering can be varied by defining the probability of intra- and inter-cluster links using a P_{in} value.

Related work showed that most network generators create networks with certain degree distributions. The algorithm proposed below can generate networks with any degree distribution, so that the crawling method can be tested for a variety of scenarios.

First the cluster graph generator algorithm is explained in section 3-1. Secondly the output of the generator is verified in section 3-2. Finally in section 3-3 a set of generated networks is used for verifying the performance of various community detection algorithms.

3-1 Cluster graph generator algorithm

The algorithm uses three stages: Node generation, wiring and rewiring in order to create connected graphs These stages are visualized in figure 3-1, 3-2 and 3-3.

The algorithm uses the concept of slots for generating the desired degree distribution. Initially each node gets an amount of slots determined by a slot distribution which is used as anchor points for creating links in a later stage. This is visualized in figure 3-1. According to the provided number of clusters, each node is assigned to a specific cluster, resulting in an explicit community structure. In the current implementation each cluster holds the same amount of nodes.

When the nodes with their slots are created and grouped into clusters, the nodes are linked using a wiring process. A random number p and the intra-cluster link probability P_{in} determine whether an intra- or inter-cluster link will be created. For intra-cluster links, slots of

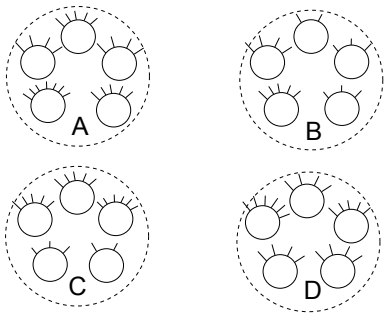


Figure 3-1: Example of a network where nodes with slots are created and assigned to clusters.
This network contains four clusters with each 5 nodes. Each node has between 2 and 7 slots on which links can be attached.

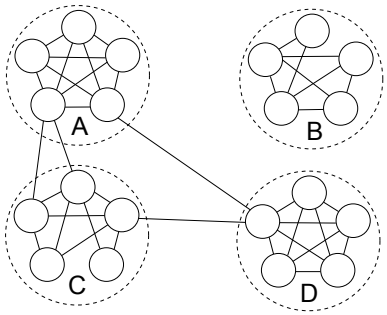


Figure 3-2: Example of a network where nodes are wired
This network contains 36 intra-cluster links and 4 inter-cluster links. Cluster B is not connected to the other clusters and is therefore not reachable when a crawl start in another cluster.

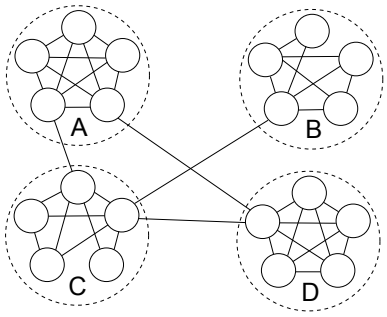


Figure 3-3: Example of a network where nodes are rewired to it one connected component (GCC)
An inter-cluster link is moved from connecting cluster A and C to connecting cluster B and C.

different nodes inside one cluster are randomly selected, the corresponding nodes connected using a link and the two slots are removed. If an inter-cluster link is created, two slots of nodes in different clusters are selected, their nodes linked and the slots are removed. When there are no slots available at a particular node, that node cannot be selected for creating a link and therefore the number of slots of a node determines its maximum possible degree. The process of determining the type of link, selecting the slots, creating the link and removing the slots is repeated until the defined amount of links is created.

When all links are created, nodes can be rewired until all nodes are in one connected component (optional) which will be referred to as "force GCC". In crawling it is necessary that all nodes are reachable from every starting point and therefore the network can be rewired in such a way that a path exists between every pair of nodes.

In short the graph generation algorithm contains the following stages:

1. node generation and slot assignment
2. assigning nodes to clusters
3. creating the links
4. force giant connected component (GCC)

These steps are described in more detail in the following paragraphs.

3-1-1 Steps in cluster graph generator algorithm

Node generation The predefined number of nodes is created where each node gets a number of slots assigned. These slots are later required for connecting the nodes using links. The number of slots is determined by the desired degree distribution which is given to the algorithm as an input parameter. The current implementation supports three types of slot distributions: an equal number of slots for each node, a power-law distributed number of slots for each node and an uniformly distributed number of slots between a lower and upper boundary. For an equal distribution, each node get the same fixed amount of slots assigned. For an uniform distribution, each node gets a random number of slots assigned which is uniformly distributed between lower bound a and upper bound b. These two distributions are useful for verifying the performance of the crawling methods. For a power-law distribution, the slot distribution follows equation 2-1 with a parameter γ and this distribution is useful because it is often observed as the degree distribution in online social networks.

Cluster assignment In order to have an explicit community structure, each node is assigned to a cluster, using the provided number of clusters. Each cluster has the same size except the last cluster.

Creating links The nodes are randomly wired to slots of nodes inside and outside their cluster, depending on the P_{in} value. This value determines the ratio between inter- and intra-cluster links and therefore determines the community strength of the generated network. The pseudo code can be found in algorithm 3. L represents the number of links to be created,

N the number of nodes and $Nodes$ is the set of nodes in the graph. The p value is a uniform random variable with values between 0 and 1. If $p < P_{in}$ then an intra-cluster link is created, otherwise an inter-cluster link.

When creating an intra-cluster link, two nodes with available slots inside the same community are linked. This is done in the following way: First a source node is randomly selected from the set *selection* which contains all nodes with available slots. This *selection* set makes sure that only nodes are selected that have available slots. Then the destination node is selected from all the nodes in the same clusters that have slots available. The nodes are randomly selected weighted by their number of available slots. A node with a high number of available slots is more likely to be selected than a node with only a few available slots. The source node and its neighbors are excluded from the random selection, so that no duplicate links can be created. If no destination node can be found matching these criteria, the source node is excluded from the *selection* and another iteration is started until the source node and destination node are successfully selected. When no source and destination node pair can be found, the process is halted.

When creating an inter-cluster link, the source and destination node are selected in such a way that they belong to different clusters. First, the source node is selected in the same way as an intra-cluster link. Then the destination node is selected from all the nodes with available slots, but excluding the nodes in the same cluster so that only nodes in other clusters are left. Again the source node and its neighbor are excluded to prevent duplicate links.

When the source node and destination node are successfully selected, the link is created and in case there is no slot available anymore, the node is excluded from the set of nodes with slots available. The whole process is repeated until all links are created or when it is not possible to find a source and destination node.

Force GCC The network is not necessarily connected. The graph can be rewired in such a way that all nodes are connected in one component (GCC). Because each link is randomly generated, nodes with a low number of slots can be unconnected. Using Breadth First Search (BFS) the largest connected component is discovered. If not all nodes belong to this connected component, the unconnected nodes are rewired until the network forms a connected component. P_{in} remains unchanged, because if a complete cluster is disconnected, an inter-cluster link will added and a random other inter-cluster link will be removed. In all other cases where only parts of a single community are unconnected, a random intra-cluster link will be removed and an unconnected node will be connected to another node inside the same cluster.

However, in some cases it is not possible to generate a network with the provided input parameters. In the next section a selection of problems is introduced, with corresponding checks that can detect problems in an early stage of the graph generation to prevent long and unnecessary run times which can not result in successfully generating a network.

3-1-2 Verification of input parameters

Some of the input parameter can be in conflict with each other. In this section two scenarios that can not result in successfully generating a network are discussed. First a check is provided

```
   Data: Nodes, L
 1 nodesWithOpenSlots ⟵ Nodes;
 2 for l ⟵ 1 to L do
 3 |   selection ⟵ nodesWithOpenSlots;
 4 |   success ⟵ FALSE;
 5 |   if p < P_in then
 6 |   |   // Find two nodes for an intra-cluster link
 7 |   |   while !success do
 8 |   |   |   source ⟵ getRandomNode(selection, NULL);
 9 |   |   |   success ⟵ slotAvailableInSameCluster(source);
10 |   |   |   if success then
11 |   |   |   |   destination ⟵
   |   |   |   |   getRandomNode(nodesInSameCluster(source), source);
12 |   |   |   end
13 |   |   |   else
14 |   |   |   |   selection ⟵ selection − source;
15 |   |   |   end
16 |   |   end
17 |   end
18 |   else
19 |   |   // Find two nodes for an inter-cluster link
20 |   |   source ⟵ getRandomNode(selection, NULL);
21 |   |   selection ⟵ selection − nodesInSameCluster(source);
22 |   |   destination ⟵ getRandomNode(selection, source);
23 |   end
24 |   // Wire source and destination node
25 |   connect(source, destination);
26 |   // Remove slots
27 |   if !hasSlots(source) then
28 |   |   nodesWithOpenSlots ⟵ nodesWithOpenSlots − source;
29 |   end
30 |   if !hasSlots(destination) then
31 |   |   nodesWithOpenSlots ⟵ nodesWithOpenSlots − destination;
32 |   end
33 end
```

Algorithm 3: Wiring

that can detect whether the required number of links can be created given a certain slot distribution. Secondly an algorithm is presented that can estimate if a certain P_{in} can be achieved given a certain slot distribution. Both checks are performed after the slot and cluster assignment and prior to the link creation.

Total number of links

For each link two slots are required so if the number of links $L > (2 \times$ Total number of slots) the network cannot be generated. This check is performed after the slot assignment and before the links are created. In this case an exception is thrown and the program ended.

P_{in}

The P_{in} value can be in conflict with the number of clusters and the slot distribution. The P_{in} value determines how many intra-cluster links will be created. On the other hand, the number of clusters results in a certain cluster size, which in turn leads to a maximum amount of intra-cluster links per cluster. This number is constrained even further by the slot distribution.

A mathematical description follows below. Furthermore a numerical example will be given to clarify the problem. Finally an algorithm is proposed which can detect whether the required P_{in} can be achieved given a certain slot distribution and number of clusters.

The P_{in} value and its resulting number of intra-cluster links can be modeled in the following way. Every time a link is created, a uniform random number between 0 and 1 is generated to determine whether an intra-cluster link or an inter-cluster link should be created. This can be considered a Bernoulli trial. This is done independent of the previously generated links and therefore the generation of all links follows a binomial distribution. The probability that a number k of intra-cluster links L_{i-i} is generated is given by:

$$Pr[L_{i-i} = k] = \binom{L}{k} P_{in}^{L} P_{out}^{L-k} \tag{3-1}$$

where L is the number of links in the network.

Based on the P_{in} value and the number of links follows an expected value for the number of intra-cluster links which can be calculated as following:

$$E[L_{i-i}] = LP_{in} \tag{3-2}$$

The standard deviation is given by

$$\sigma[L_{i-i}] = \sqrt{LP_{in}P_{out}} \tag{3-3}$$

However, this does not mean that all intra-cluster links can be created. The number of possible intra-cluster links is also determined by the number of nodes inside a cluster. Because all clusters except one are equally sized, the cluster size can be calculated from the number of clusters and the number of nodes in the network. No duplicate links are allowed, so in the optimal scenario each cluster forms a complete subgraph with the nodes in its own cluster which follows the following equation.

$$L_{max} = \frac{N(N-1)}{2} \tag{3-4}$$

where $N = |N_{cluster}|$ is the number of nodes in a cluster and $L_{max} = L_{i-i}$ is the maximum number of links inside a cluster.

An example network was generated using the following parameters: 100 nodes with each 22 slots and grouped into 20 clusters; 1000 links with a $P_{in} = 0.7$. All clusters contain 5 nodes. According to equation 3-4, 10 intra-cluster links can be created per cluster. In total $10 * 20 = 200$ intra-cluster links are possible. The required $P_{in} = 0.7$ cannot be achieved because according to equation 3-2 this would require $0.7 * 1000 = 700$ intra-cluster links.

Networks with other slot distributions are checked whether it is possible to create all intra-cluster links after the assignment of slots and assignment of clusters. This is done prior to the link phase of the algorithm. The pseudo code can be found in algorithm 4 and an example is given in table 3-1. This check counts the amount of possible intra-cluster links per cluster and is based on the distribution of available slots in a cluster. It basically emulates the wiring of a cluster in a scenario where the nodes are connected from a low amount of slots to a high amount of slots until no more links inside that cluster can be created. The sum over all clusters gives an upper bound for the amount intra-cluster links that can be created. A more detailed description of the algorithm is as followed.

For each cluster the number of possible links inside that cluster is counted. Because nodes inside a cluster can have different numbers of free slots, this has to be taken into account. The number of possible links inside a particular cluster is done in the following way. The number of free slots per node in a cluster A is put in a list. This list is sorted and the function $LinksPossible(slots)$ is called for the list having an ascending and descending sorting. This function emulates the wiring of nodes in cluster A based on its sorting. The first node of the list is selected. If the second node has a slot available, one slot is subtracted from the first and second node. The counter for possible intra-cluster links is increased. When the first and third node have a slot available, from both nodes the number of slots is decreased by one and the counter is increased. This process is continued until one of the two nodes has no slot available or the end of the list is reached. Then the second node is selected as a starting point and the process is repeated for all nodes next to this starting node. This is done until all nodes are used as a starting point. When the list is sorted from large to small, this gives an upper bound for the amount of possible links in cluster A. When the list is sorted from small to large, this shows how many intra-cluster links can be at least created. In table 3-1 an example is given for a cluster with 5 nodes. If the expected number of intra-cluster links given by equation 3-2 is higher than the upper bound, it is unlikely that a network can be created with the required amount of links and an error is shown and the wiring process is not started. If the expected number of intra-cluster links is lower than the lower bound the network it is very likely that the network can be created and the wiring process is started. If the expected number of intra-cluster links is between the lower upper bound but higher than the lower bound, it might not be possible to create the desired number of links. The wiring process is started, but a warning is shown.

```
   Data: Clusters, L, P_in
 1 num_intralinkslower ⟵ 0;
 2 num_intralinksupper ⟵ 0;
 3 for cluster in clusters do
 4 |   for i ⟵ 1 to cluster.size do
 5 |   |   node ⟵ cluster[i];
 6 |   |   slotslower[i] ⟵ getNumSlots(node);
 7 |   |   slotsupper[i] ⟵ getNumSlots(node);
 8 |   end
 9 |   SortAscending(slotslower);
10 |   num_intralinkslower+ = linksPossible(slotslower);
11 |   SortDescending(slotsupper);
12 |   num_intralinkslower+ = linksPossible(slotsupper);
13 end
```

Algorithm 4: Check the possible number of intra-cluster links

```
   Data: slots
   links ⟵ 0;
   for j ⟵ 0 < slots.length − 1 do
   |   slotsavailable ⟵ slots[j];
   |   for k ⟵ 0 < slotsavailable do
   |   |   item ⟵ k + j + 1;
   |   |   if item ≥ slots.length then
   |   |   |   BREAK;
   |   |   end
   |   |   if slots[item] > 0 then
   |   |   |   slots[item] − −;
   |   |   |   slots[j] − −;
   |   |   |   links + +;
   |   |   end
   |   end
   end
```

Function LinksPossible(slots) Calculates the number of possible links in a cluster

Table 3-1: Example of intra-cluster link check. The cluster has 5 nodes with a slot distribution of {4, 5, 2, 3, 2}. The slots are sorted ascendingly to give a lower bound of the number of intra-cluster links in this cluster. In this worst case scenario, only 5 links can be created.

start	item						counter
-	-	2	2	3	4	5	0
0	1	1	1	3	4	5	1
0	2	0	1	2	4	5	2
1	2	0	0	1	4	5	3
2	3	0	0	0	3	5	4
3	4	0	0	0	2	4	5

3-2 Verifying the output of the cluster graph generator

Multiple networks were created to evaluate the network generator. The generator will be used for creating networks ranging from 100 to 10000 nodes and 1000 to 100000 links. First the amount of nodes and links will be verified. Then the P_{in} and degree distribution will be checked. The number of clusters and the cluster size distribution will be verified. These networks will be generated for two slot distributions; equal distribution and power-law distribution. In order to successfully test the crawling methods, the generated networks that they are tested against should have the previously mentioned properties and are therefore verified in this section.

3-2-1 Equal distribution

The degree distribution of traditional networks such as the Football network can be approximated by equal degree distribution. In such a network each node has the same degree. For the equal slot distribution of 22 slots, the P_{in} and the size of the network are varied. The ratio between nodes and links is a factor of 10, which implies an average degree of 20. Each network has 20 clusters so the cluster size varies as a function of the number of nodes. Each node gets 22 slots assigned. It is necessary to assign more slots than required by the ratio between nodes and links, because not every combination of free slots can be used for creating a link. It could happen that when almost all links are created and few slots are available, the following scenario occurs. There are only available slots on nodes which have already been connected by links. These slots cannot be used and therefore extra slots are required.

The result of networks with equal distributions are shown in appendix B. It was possible to create networks with 100 nodes with $P_{in} \leq 0.2$ as expected in the calculation above. Networks with larger P_{in} are not possible to generate because not enough intra-cluster links can be created. The same issue arises for networks $P_{in} > 0.4$ with 200 nodes. All successfully generated networks have the desired number of nodes, number of links an average degree of 20, which is caused by the ratio between the number of links and the number of nodes. The standard deviation of the degree ranges from 1.24 to 1.42 as can be expected because of the extra slots that were provided. 20 clusters are created with each containing the same amount of nodes, just as expected. All networks are connected which means that each node is reachable from every other node. All input parameters are exactly matched in the resulting networks except for the degree distribution and P_{in} . We also introduce a local P_{in} value, which represents the ratio between intra-cluster degree and total degree for each individual node. In figure 3-4 the local P_{in} is plotted as a function of the degree and shows that there is a variation in the local P_{in} . This corresponds to real world social networks where not every entity has the exact same ratio between neighbors in their community and total neighbors.

3-2-2 Power-law distribution

In online social networks, power-law degree distributions are observed quite often and therefore networks with this type of distribution are generated and used to test the crawling methods. The same parameters for equal distribution were used, except that a power-law slot distribution was applied with a $\gamma = 1.5$. These generated networks have the correct number

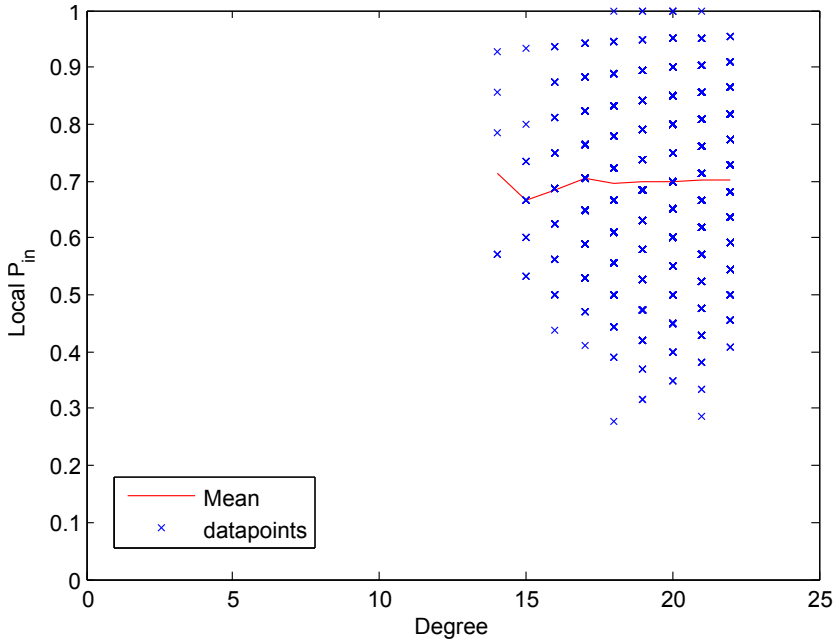


Figure 3-4: Local P_{in} for a network of 10000 nodes with 20 clusters, P_{in}=0.7 and an equal slot distribution of 22

of nodes and number of links and all networks are connected. In figure 3-5 the local P_{in} is plotted as a function of the degree and shows that there is a variation in the local P_{in} . The average local P_{in} is higher than the global P_{in} . This means that low degree nodes share a higher portion of their links in their cluster than their high degree counterparts. In real world social networks high degree nodes also act as hubs between clusters and should therefore also have lower local P_{in} values.

The network generator works as expected and can create networks with arbitrary degree distributions and with an explicit community structure. In the following section a number of community detection algorithms is tested on networks generated by the above proposed algorithm.

3-3 Evaluation community detection algorithms

In this section the networks generated in section 3-2 are used as a test bench for multiple community detection algorithms. The results will be used to select the most appropriate algorithm to partition real world networks without explicit community structure. The partitioning created by the community detection algorithms are compared to the "ground truth" of the cluster generator algorithm using modularity, Jaccard index and f_{same} (section 2-2 and 2-3) as a function of P_{in} which relates to the community strength. The algorithm that produces the partitioning that is most similar to the "ground truth" will be considered the most appropriate algorithm for partitioning social networks. The three algorithms used for partitioning the networks in communities are Spinglass, Fast Greedy and Label Propagation where Spinglass is the most computationally expensive algorithm and Label Propagation is

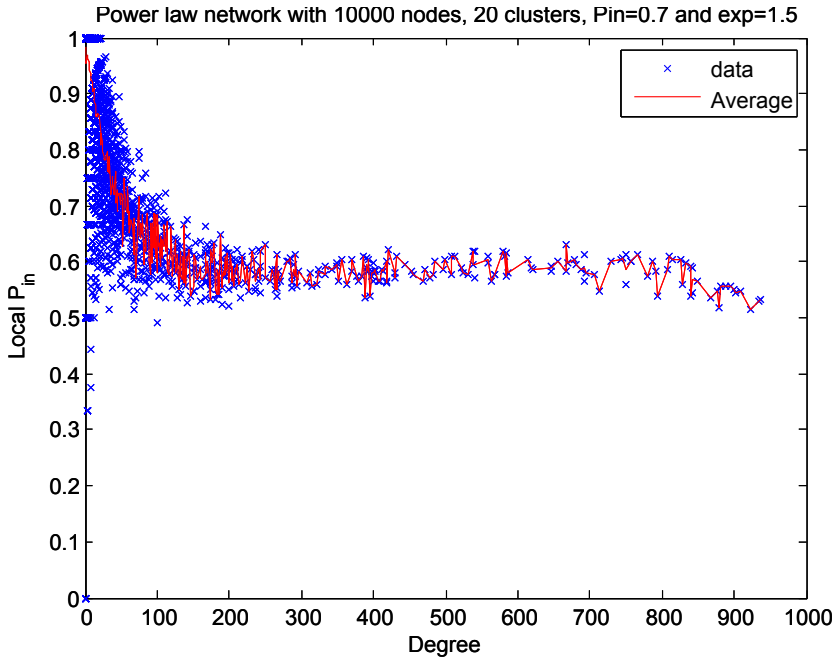


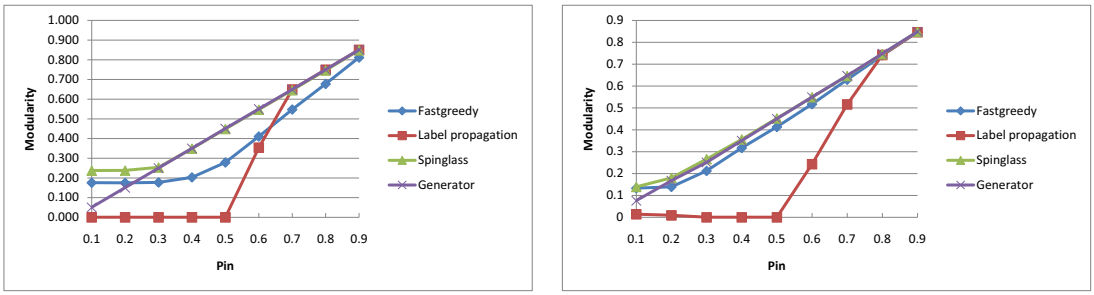
Figure 3-5: Local P_{in} for a network of 10000 nodes with 20 clusters, P_{in}=0.7 and a power-law distribution with exponent $\gamma = 1.5$

computationally the cheapest. There are 10 networks generated per configuration and the results are averaged.

In figure 3-6 to 3-8 the results of the comparison are shown for networks generated with an equal slot distribution. Figure 3-6(a) shows that the modularity of Fast Greedy and Spinglass are highly correlated to the modularity of the generator. For $P_{in} < 0.6$ Label Propagation has low modularity which means that the partitioning is no better than random partitioning. When looking at the f_{same} and Jaccard index in Figure 3-7 and Figure 3-8 the Spinglass community detection algorithm performs well above $P_{in} \geq 0.4$. Although the modularity score of Fast Greedy seems to be similar to the cluster generator, the f_{same} and Jaccard index scores the lowest of all algorithms. Clearly the Spinglass algorithm performs best of the three algorithms for partitioning the generated networks with an equal distribution based on all three measurements. However this algorithm is also the most computationally expensive.

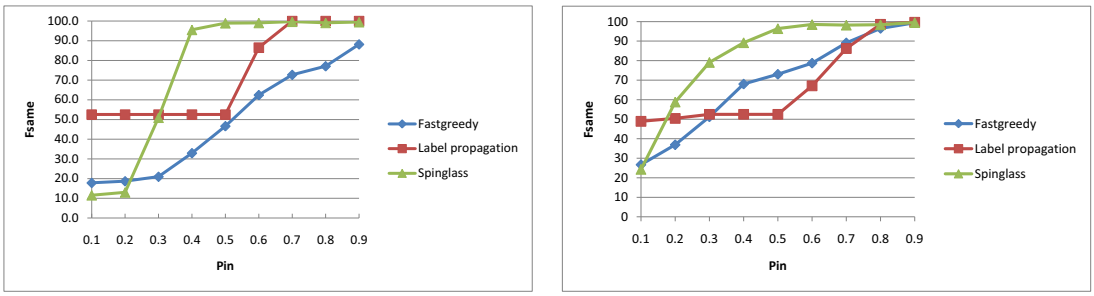
In figure 3-6 to 3-8 the same analysis is performed for networks generated with a power-law slot distribution. Again the Spinglass community detection algorithm performs best of the three, because this method shows highest similarity with the "ground truth" of the generator for networks with $P_{in} \geq 0.5$. It shows the highest similarity for both the f_{same} and Jaccard index. This means that the Spinglass algorithm is able to partition graphs where on average each node shares about half of its neighbors inside the same community. In laymen terms, Spinglass is able to find groups when users share half of their friends are in the same community. Both other community detection algorithms only give reasonable results for $P_{in} \geq 0.8$ and resulting $modularity \geq 0.75$. However, in real world online social networks these high values for modularity are not commonly found. The Spinglass algorithm performs well for $P_{in} \geq 0.5$ with $modularity \geq 0.5$ which can be found in online social networks with

community structures. Therefore the Spinglass algorithm will give the best partitioning in communities out of these three algorithms.



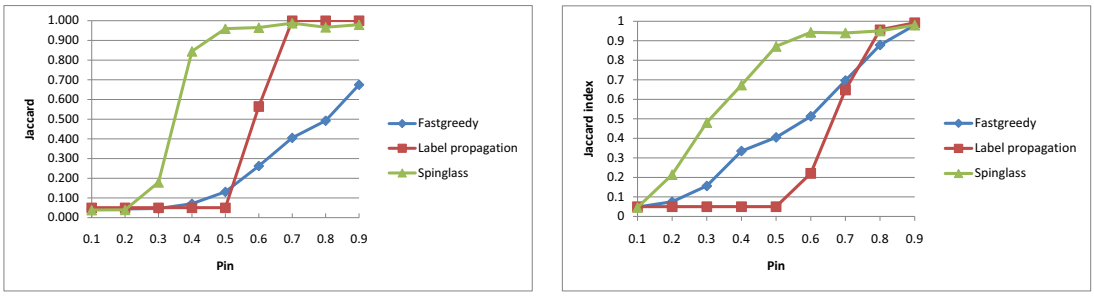
(a) Equal (b) Power-law

Figure 3-6: Comparison of community detection algorithms using modularity



(a) Equal (b) Power-law

Figure 3-7: Comparison of community detection algorithms using f_{same} to the "ground truth"



(a) Equal (b) Power-law

Figure 3-8: Comparison of community detection algorithms for networks using Jaccard index to the "ground truth"

3-4 Conclusion

The network generator successfully creates graphs with explicit community structure and the desired degree distribution. Three commonly used community detection algorithms were

tested using the “ground truth” created by the network generator. This analysis showed that partitioning created by the Spinglass community detection algorithm shows the highest similarity with the “ground truth”. The Spinglass algorithm clearly performs better than Label Propagation and Fast Greedy community detection, but is computationally more expensive.

Chapter 4

Mutual friend crawling

Frequently used crawling techniques such as Breadth First Search (BFS) and Depth First Search (DFS) do not take advantage of the topological structure of the network to find communities. In this section an approach is investigate to steer the crawl to stay inside a community.

In section 4-1, a crawling algorithm is proposed that aim to crawl a smaller portion of the network in order to crawl entire communities. In section 4-2 the performance of the algorithm is evaluated for the Football network, the Enron network and computer generated networks and compared to existing crawling methods such as BFS and DFS. For the Enron network, no explicit community structure is available and therefore created using community detection algorithms.

4-1 Explanation of the algorithm

Suppose you crawl an online social network (OSN) with users and their friends. In this network the users are represented as nodes and the friendships between users as the links. The algorithm for crawling inside a cluster is based on an intuitive heuristic. Users within a community should share more links between them than with users in other communities. The priority in which friends are crawled is based on the number of users that are referring to them divided by their degree. An example is given in figure 4-1. The red nodes represent nodes that have been discovered (G_s) and the green nodes are in the queue. Both nodes in the queue {1,8} have 2 links referring from the discovered network to them which is defined as a found reference. With the found reference and degree, the score for the node is calculated using:

$$CalculateScore = \begin{cases} \frac{foundreference(u)}{degree(u)} & \text{if } u \notin startpoints, \\ 1 & \text{if } otherwise, \end{cases} \tag{4-1}$$

Node 1 has a score of $\frac{2}{2} = 1$, node 8 has a score of $\frac{2}{5}$, thus node 1 will first be crawled.

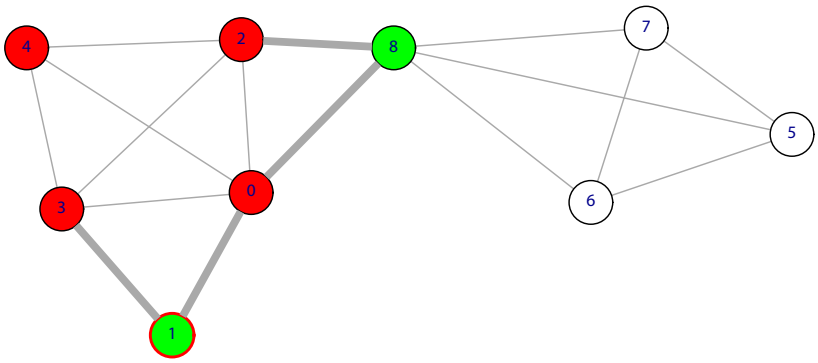


Figure 4-1: Explanation of the found reference metric using an example graph.
The red nodes denote the visited nodes which already have been crawled, the green nodes represent the uncrawled nodes which are in the scores queue to be crawled. The number indicates the identifier of the node.

Iteration	Node	Q (node: score)	Next node
0	2	0: $\frac{1}{5}$ 1: $\frac{1}{5}$ 3: $\frac{1}{4}$ 4: $\frac{1}{5}$	3
1	3	0: $\frac{2}{5}$ 1: $\frac{2}{5}$ 4: $\frac{2}{5}$	0
2	0	1: $\frac{3}{5}$ 4: $\frac{3}{5}$ 10: $\frac{1}{6}$	1
3	1	4: $\frac{4}{5}$ 10: $\frac{1}{6}$ 28: $\frac{1}{6}$	4
4	4	10: $\frac{1}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	10
5	10	11: $\frac{1}{6}$ 12: $\frac{1}{4}$ 13: $\frac{1}{4}$ 14: $\frac{1}{4}$ 18: $\frac{1}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	12
6	12	11: $\frac{2}{6}$ 13: $\frac{2}{4}$ 14: $\frac{2}{4}$ 18: $\frac{1}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	13
7	13	11: $\frac{3}{6}$ 14: $\frac{3}{4}$ 18: $\frac{1}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	14
8	14	11: $\frac{4}{6}$ 18: $\frac{1}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	11
9	11	18: $\frac{2}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	18
10	18	15: $\frac{1}{4}$ 16: $\frac{1}{4}$ 17: $\frac{1}{6}$ 19: $\frac{1}{6}$ 22: $\frac{1}{6}$ 28: $\frac{1}{6}$	15

Table 4-1: Crawl of graph 4-2 using mutual algorithm.
The first 10 iterations are shown.

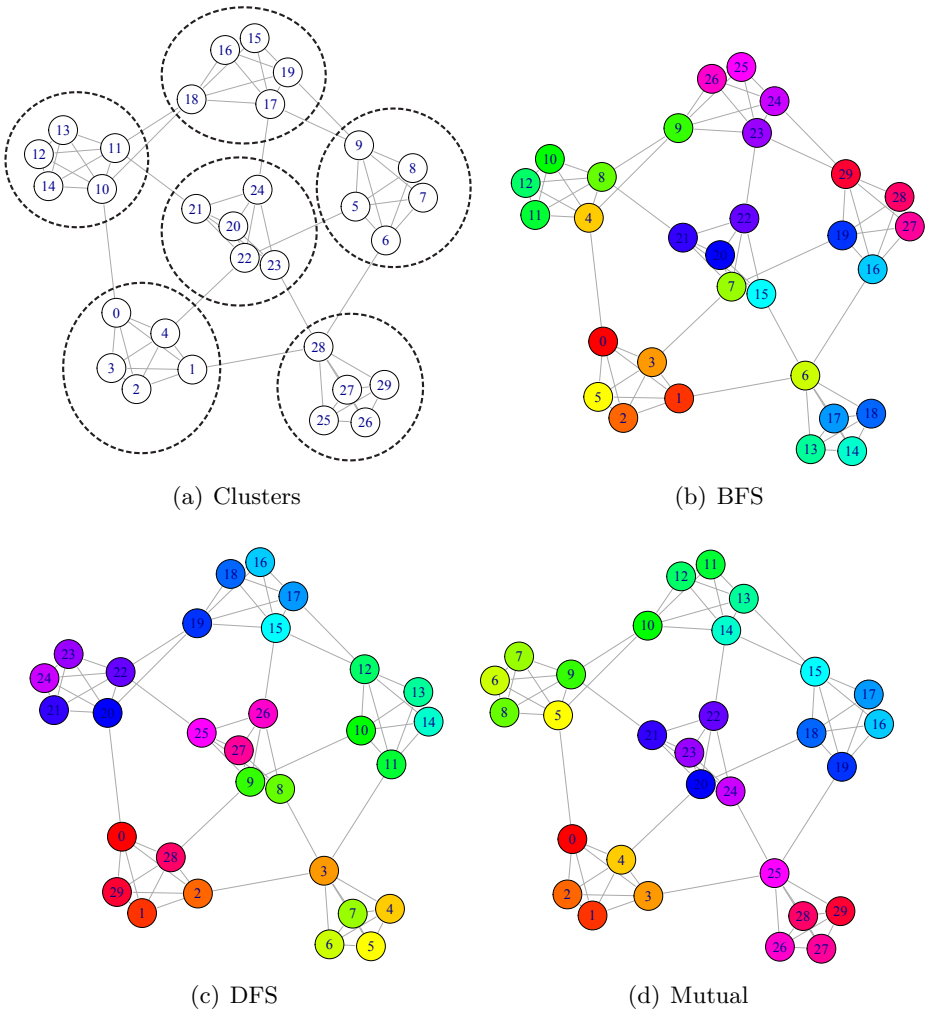
An example of crawling a small network is given figure 4-2. This example shows that the proposed algorithm crawls entire communities before continuing to other communities, while BFS and DFS do not. This indicates that the concept works.

For this algorithm, the following assumptions are taken:

1. The network is connected. As in BFS and DFS, there needs to be a path between every pair of nodes in the graph. Otherwise the unconnected nodes will not be discovered.

2. The network is un-directed. All links in the network need to be bi-directional.

3. The degree of all nodes in the discovered subgraph G_s and the direct neighbors of the discovered subgraph G_s is known. In other words, the degree of nodes that are discovered (G_s) and nodes in the queue Q are known. The nodes in the queue are not yet crawled for their adjacency lists. In most social networks, gathering profile information such as the number of friends (degree) costs very few resources compared



(a) Clusters (b) BFS

(c) DFS (d) Mutual

Figure 4-2: Example of a graph being crawled using three crawling methods. Nodes are grouped into communities(a), Nodes colored and labeled according to crawl iteration (b), (c) and (d)

to gathering the full friends list (adjacency list). Gathering the profile data can be done right when it is placed in the queue, so that the degree information is available and is therefore not a hard constraint on the algorithm.

4. The number of friends found in the adjacency list should be the same as the degree found in the profile information. This implies that the degree in the profile information is public as well as all the identifiers of the neighboring nodes. Therefore both information sources should be public and none of the neighboring nodes should be hidden by privacy settings of the social network.

With these assumptions the following algorithm is constructed. There are three important data structures used in algorithm 5. $VisitedNodes$ is a set of all nodes that have been visited so far and prevents that nodes are crawled multiple times. $FoundReference$ is an integer which represents the number of discovered links to an uncrawled node. This value is important for calculating the priority in which the nodes are crawled. $FoundReferences$ is a set of nodes and its corresponding found references. The scores data structure works as a queue for nodes that have to be crawled. It consists of the nodes and their corresponding score.

```
   Data: Startpoints, GraphG
 1 // foundreferences counts the number of links towards a node
 2 foundreferences ⟵ ∅;
 3 // scores works as a queue for the nodes to be crawled
 4 scores ⟵ ∅;
 5 // visitedNodes keeps track which nodes have already been visited
 6 visitedNodes ⟵ ∅;
 7 foreach v ∈ V[G] do
 8 |   foundreferences[v] ⟵ 0;
 9 end
10 foreach s ∈ startpoints do
11 |   scores[s] ⟵ 1;
12 end
13 while scores ≠ ∅ do
14 |   v ⟵ MaximumScore(scores);
15 |   scores[v] ⟵ NIL;
16 |   INSERT(visitedNodes, v);
17 |   foreach u ∈ Adj[v] do
18 |   |   if u ∉ visitedNodes then
19 |   |   |   foundreferences[u] ⟵ foundreferences[u] + 1);
20 |   |   |   scores[u] ⟵ CalculateScore(u);
21 |   |   end
22 |   end
23 end
```

Algorithm 5: Mutual friend crawling algorithm

First the starting points are initialized with the highest possible score, to let them be the first nodes to crawl (line 10). Then the node with the highest score is selected (line 14) and

removed from the scores (line 15), so it does not exists in the queue anymore. The node is added to the visitedNodes so it is not crawled again (line 16). Then the adjacency list is obtained (line 19). When crawling a social network this would mean that the list of friends is downloaded. Then for all the unvisited friends, the value of their foundreferences is increased (line 17) and their score is recalculated (line 20). This process is looped until all the nodes in the scores data structure are found.

The algorithm would ideally crawl a full cluster before starting on another cluster. An example of such an 'ideal' network is given in figure 4-2(a). Node 2 is the starting point for the crawl. The labels inside the node depict the iteration of the crawl. The nodes in 4-2(b) to figure 4-2(d) are colored using a rainbow coloring scheme starting with red and labeled according to the iteration in the crawl. The values in the score data structure are shown in table 4-1.

4-2 Performance evaluation

The performance of BFS, DFS and the proposed algorithm is tested in a variety of topologies. The networks should have an implicit community structure, community structure where the partitioning in communities is unknown. An explicit community structure where the partitioning is known is not required for the algorithm, but such a network is only selected to simplify the validation. These networks are not very common, but there are a few examples of social networks where each node is assigned to a specific group, such as a Football network [1]. This network is selected because of its small scale and therefore the results can easily be verified by hand. Afterwards a large scale social network is analyzed, but unfortunately there are no large social networks (>10000 users) available that have an explicit community structure, in which each node is assigned to a single community. Because of the time complexity of Spinglass community detection algorithm (see section 3-3), a relatively small corporate email network, Enron, was selected. Because the nodes are not assigned to specific communities, it is first clustered using the community detection algorithms and afterwards analyzed using different crawling techniques. Finally computer-generated networks are tested using the cluster graph generator described in chapter 3. In this way, the crawling method can be tested under various scenarios, such as different degree distributions and changing community strength.

4-2-1 Football network

The first network that is analyzed is a college football network (not soccer) in the United States which was introduced in Girvan et al [1].

In this network there are 115 teams which played 616 games, thus the resulting interaction network contains 115 nodes and 616 links. The degree distribution is shown in figure 4-5(a), which shows that each team plays between 7 and 12 games. All teams are grouped into one of the 12 leagues or 'conferences', which results in 12 clusters. The cluster size distribution is shown in figure 4-5(b). This figure shows that the sizes of the clusters are between 5 and 13 nodes, so there are 5 to 13 teams in a conference.

We define a P_{in} and P_{out} where $P_{in} + P_{out} = 1$ and P_{in} is the probability that a link is between two nodes of the same cluster and P_{out} is the probability that a link is between nodes of different clusters. In total there are 397 games played within conferences and 219 games played

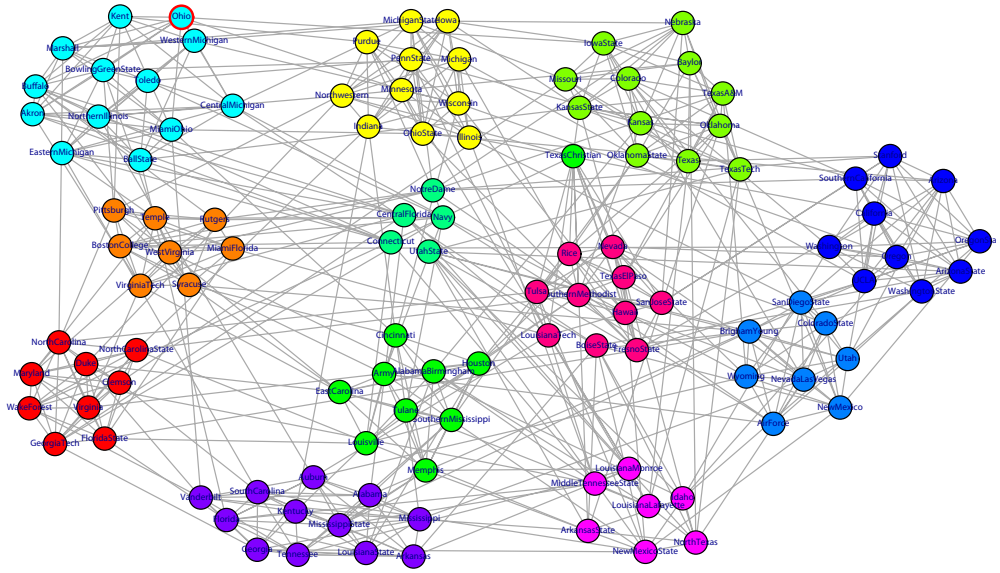


Figure 4-3: Graph of the football network introduced in [1]

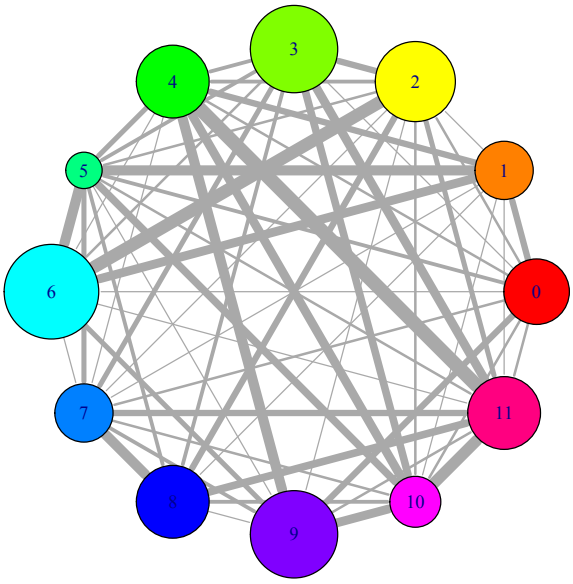


Figure 4-4: Graph of the clusters in the football network

i	Clustername	$\|N_{cluster}\|$	L_{i-i}	$L_{i-i_{max}}$	$\frac{L_{i-i}}{L_{i-i_{max}}}$	L_{i-j}
0	Atlantic Coast	9	36	36	1.0000000	10
1	Big East	8	28	28	1.0000000	9
2	Big Ten	11	44	55	0.8000000	11
3	Big Twelve	12	49	66	0.7424242	9
4	Conference USA	10	31	45	0.6888889	10
5	Independents	5	1	10	0.1000000	11
6	Independents	13	51	78	0.6538462	9
7	Mountain West	8	28	28	1.0000000	11
8	Pacific Ten	10	40	45	0.8888889	8
9	Southeastern	12	49	66	0.7424242	9
10	Sun Belt	7	10	21	0.4761905	10
11	Western Athletic	10	30	45	0.6666667	11

Table 4-2: Properties of the clusters in the football network

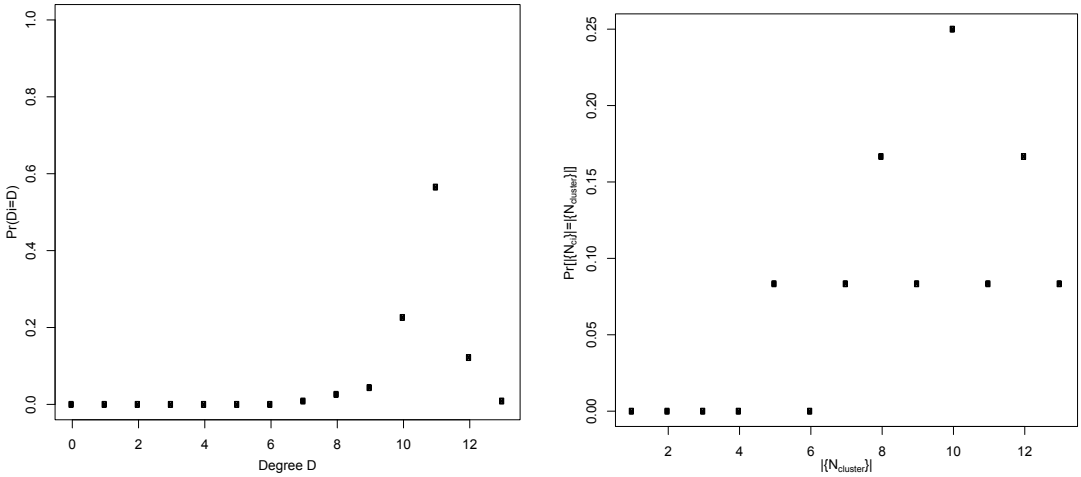
between teams of different conferences. For the football network $P_{in} = \frac{\#gamesinsidecluster}{\#games} = \frac{397}{616} = 0.6444805$. This means that a team has a probability of P_{in} to play against a team of its own conference. This metric was also used for generating networks in section 3.

More properties of the network can be found in table 4-2. This table shows how well connected the nodes inside a cluster are connected which is related to the property the algorithm steers its crawl on. In conference 5 called "independents" the 5 teams played only one single game (L_{i-i}). This conference contains teams such as the naval team and other teams that are not geographically connected. From a topological point of view this can hardly be regarded as a cluster. This will probably effect the results of our algorithm. The maximum number of possible games is given by the number of links in a complete graph, which is shown in equation 3-4

In all other clusters the ratio of actually played games versus the number of possible games is much higher ($\frac{L_{i-i}}{L_{i-i_{max}}}$). When the ratio is 1 all possible games are played inside that conference, which means that cluster is full mesh. Finally L_{i-j} shows to how many clusters a cluster is connected. This does not show how often a game is played between two clusters. In figure 4-4 the graph of the clusters is shown. The size of the nodes shows how many teams there are in the conference. The width of the links show how many games are played between the conferences. The data for plotting these widths can be found in appendix A-1.

Results The network is crawled using a Breadth First Search, Depth First Search and the proposed crawling technique. Because of the relatively small size of the network, it is feasible to compare all crawling techniques for all possible starting points.

Figure 4-6 shows a box plot of the part of the network that is crawled as a function of number of completely crawled clusters for crawls from all possible starting points. Goal of the proposed algorithms is to crawl the nodes of a cluster before starting on another cluster and therefore the part of the network that is crawled should be as low as possible. The box plot shows the average portion of the network that has to be crawled in order to complete a number of clusters and also depicts the outliers. The figure clearly shows that the proposed



(a) Degree distribution (b) Cluster size distribution

Figure 4-5: Characteristics of football network

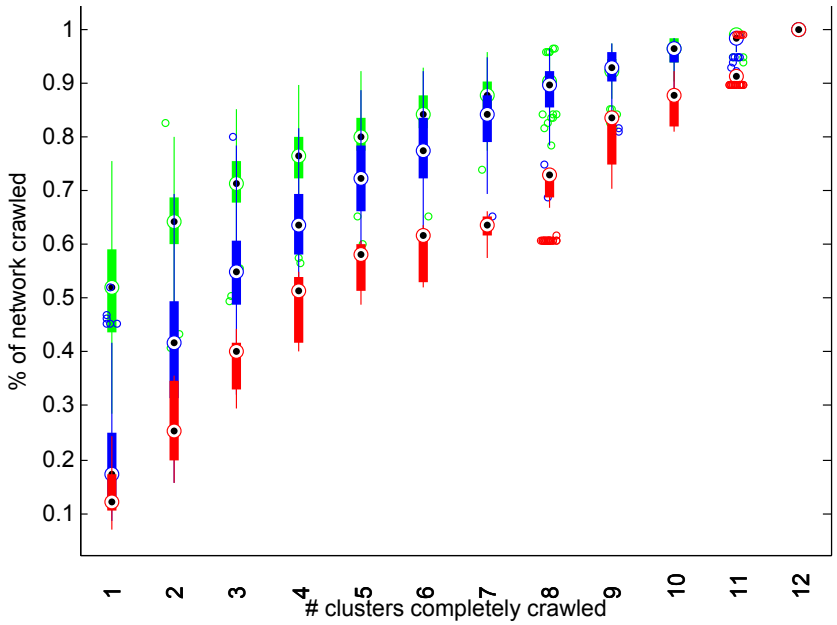


Figure 4-6: Part of the network crawled before completing a cluster of the football network (box plot)

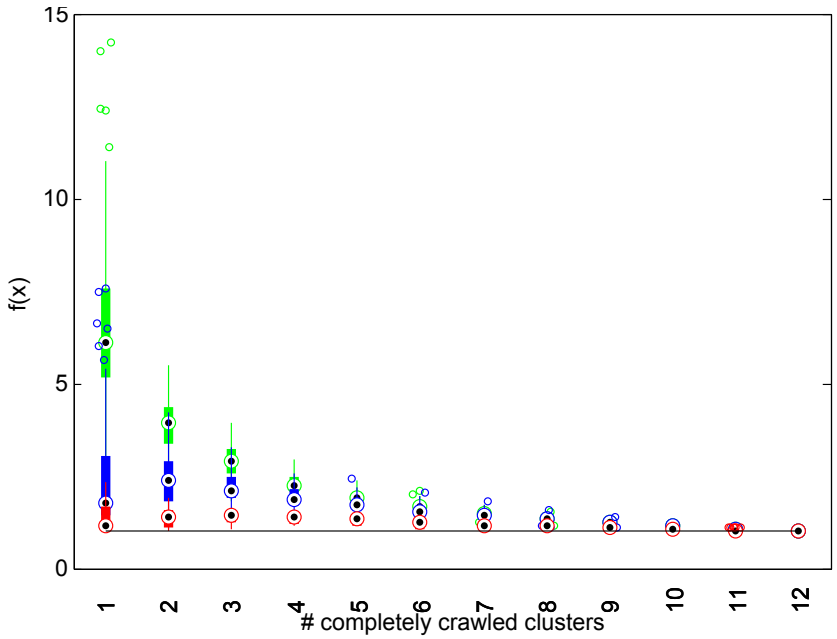


Figure 4-7: f(x) for football network

algorithm needs crawl a smaller portion of the network in order to complete clusters. Also the variance is lower than in BFS and DFS, meaning that the performance depends less on the starting point.

Figure 4-7 shows how many times the size of a cluster has to be crawled in order to complete a cluster, which is defined as f(x). DFS clearly needs to crawl much more nodes to complete a cluster and does not stay inside a cluster before going to another cluster.

Figure 4-8 shows a zoom in on the average f(x) and ideally only one time the size of the cluster has to be crawled to completely crawl one cluster. The figure shows that approximately 1.5 times the cluster size has to be crawled before a full cluster is completely crawled using the proposed algorithm. For BFS one needs to approximately crawl 2.5 times the cluster in order to achieve the same result and for DFS it is necessary to crawl more than 5 times the size of the first cluster in order to completely crawl it.

The proposed algorithm performs better than BFS and DFS for the football network, because it has to crawl a smaller portion of the network in order to completely crawl a community. In the next section the performance is tested on a larger email network.

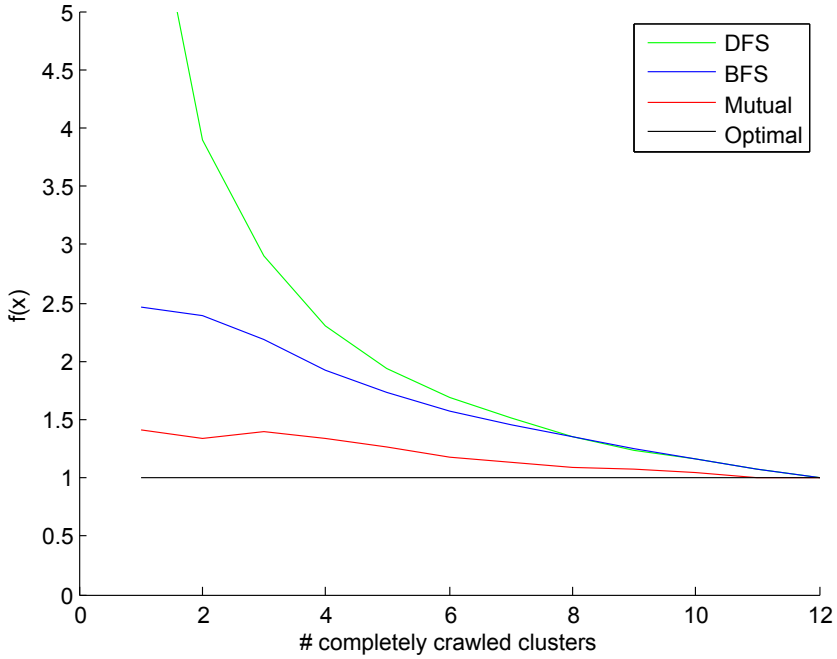


Figure 4-8: f(x) for football network (zoom)

4-2-2 Enron email network

The Enron email network is a social network constructed from email communication in a large corporation. A large set of email messages, the Enron corpus, was made public during the legal investigation concerning the Enron corporation. The raw Enron corpus contains 619,446 messages sent to and received by 158 Enron employees [17]. It was chosen because of its relatively small size which made it feasible to partition it using the community detection algorithms discussed in section 2-2.

Characteristics of the network From this email corpus a network is constructed in which the email addresses are represented by nodes and emails are represented by links. This network contains 36692 nodes and 183831 links. Only the connected component is used in crawling techniques and therefore the network can further be reduced to a connected network of 33696 nodes and 180811 undirected links. Non-Enron email addresses act as sinks and sources because only emails from and to Enron employees were included in the corpus.

Community detection The network does not contain an explicit community structure. For the analysis, each node has to be assigned to specific community. Using various community detection algorithms mentioned in section 2-2, each node is assigned to a community. This analysis showed that Spinglass creates the best partitioning. However, the other two algorithms, Fast Greedy and Label Propagation community detection are also considered. Unfortunately it was not feasible to use Edge Betweenness community detection, because the run time would be too large. The run time was estimated using Erdos-Renyi graphs with different sizes. From this trend an expected run time for edge betweenness community detection on the Enron network would be 2500 days, which is not feasible. Label propagation, Spinglass and Fast Greedy run on the Enron network in 28 seconds, 45 minutes and 88 seconds respectively.

Comparing crawling results for community detection algorithms There is no "ground truth" available for the Enron network. Therefore Spinglass is taken as the most proper community detection algorithm. The partitioning of the different community detection algorithms are compared among themselves to see if they give similar partitions. A commonly used indicator for testing how well a network is divided into communities is modularity which is explained in section 2-3-1 and the results are shown in table 4-3. The similarity between the results of the algorithms are compared using the Jaccard index and the f_{same} index which are introduced in section 2-3 and the results are shown in Appendix C-1(a) and C-1(b).

A direct pairwise comparison of the partitioning of the algorithms is not possible because of differences in number of clusters and cluster size distributions, Therefore the algorithms are compared to a partitioning where all nodes are placed in the same community called "oneclust"'. If the similarity with "oneclust" is high, this shows that a large portion of the nodes are put in a single community. The measurements can be found in Appendix on page 65. For Label Propagation, there is a high similarity with "oneclust" indicated with an f_{same} index of 0.836.

To be able to compare the results to randomly assigning clusters, the f_{same} and Jaccard index are used to compare the real partitioning for the three algorithms with a cluster assignment

Algorithm	Clusters	inside	outside	P	Modularity	Run time
Label	880	169886	10925	0.940	0.296	28.19 seconds
Spinglass	25	135248	45563	0.748	0.606	2737 seconds
Fastgreedy	559	145875	34936	0.807	0.506	87.88 seconds

Table 4-3: Community detection algorithms

based on randomly assigning nodes into the given community size distribution. If the similarity to this shuffled partitioning is high this means that the partitioning is no better than random and indicates poor partitioning.

Spinglass has the highest modularity (0.606) of the three algorithms which indicates the best partitioning in communities. 25 communities were found in a network of 33 thousand nodes. In Figure C-1(c) the cluster size distribution is depicted and shows that the sizes of the clusters are large (234-5408 nodes) for social networks. Because of the relatively low number of clusters, it is not possible to infer a reasonable distribution by visual inspection.

Fast greedy has a modularity of 0.506 which is according to Girvan et al. a reasonable partitioning in communities. However the three largest clusters contain 67 % of the nodes of the full network which is high for a social network. This observation together with the poor performance for generated networks, makes this partitioning questionable.

Label propagation has the lowest modularity of the three algorithms. In Girvan et al. [1] modularity's between 0.3 and 0.7 are considered normal values for communities therefore the 0.3 value of Label Propagation is reasonably low. This indicated that the algorithm does not partition the Enron network well. The algorithm puts most of the nodes in a single cluster and puts all other nodes in tiny clusters containing only a few nodes. This can be observed in the cluster size distribution in figure C-1(a). Having such a diverse distribution does not correspond to an intuitive notion of communities. The P value is relatively high because most nodes are in the same cluster.

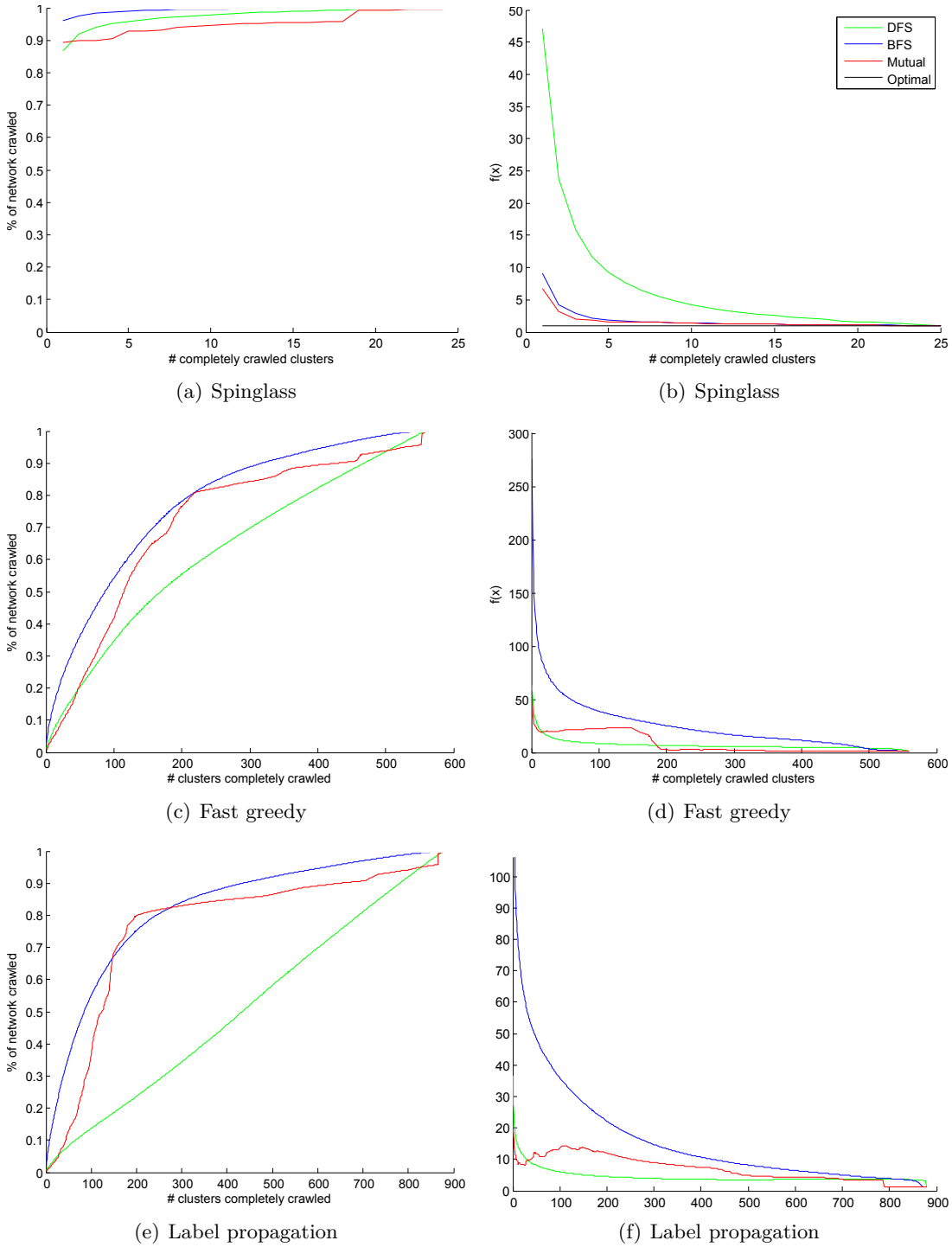
All three community detection algorithms have difficulty to partition the Enron network and that shows that the network does not have a strong community structure. In the following section the three crawling techniques are tested on the results of the three community detection algorithms.

Crawling results Figure 4-9 shows the results for crawling the network using the different crawling techniques and comparing them to the output of the three community detection algorithms. Figure 4-9 a and b shows the average trajectory of the proposed crawling method for the partitioning of the Spinglass algorithm. A large part ($> 90\%$) of the network has to be crawled in order to completely crawl a cluster. Still the proposed crawling method performs better than BFS and DFS.

As previously shown, the similarity of Label Propagation to "oneclust" and the randomly assigned clusters is high and therefore this partitioning is not optimal. Because of the poor

results of the community detection algorithms, it is difficult to assess and compare the performance of the crawling techniques. According to the other two algorithms Figure 4-9 c,d,e and f, it is necessary to crawl a lot of extra nodes in order to completely crawl a cluster ($>$ 10 times as much nodes as are in the clusters itself). A possible explanation for the "knee" shaped trajectory in Figure 4-9 c and e is caused the existence of a small number of large clusters.

As observed in the figures, DFS appears to perform quite well. An explanation for this behavior could be the preference for DFS to crawl towards the "whiskers" of the network. Because of their skewed cluster size distribution defined by Label Propagation and the resulting existence of a huge number of small clusters, a high proportion of "whiskers" is assigned to single communities. This could explain the apparent good performance of DFS.



(a) Spinglass (b) Spinglass

(c) Fast greedy (d) Fast greedy

(e) Label propagation (f) Label propagation

Figure 4-9: Part of network crawled before completely crawling clusters of Enron network

4-2-3 Computer-generated networks

The three crawling methods are also tested against synthetic networks created by the algorithm which was in introduced in chapter 3. By variating the community strength, it is possible to see how the crawling method functions in networks with very strong to very weak community structure. The networks have the following configuration: 10000 nodes, 100000 links and 100 clusters. Two degree distributions are investigated, equal degree distribution where each nodes as approximately a degree of 22 and power-law distribution with a $\gamma = 1.5$. The strength of the community structure is varied using a $0.1 \leq P_{in} \leq 0.9$. Each network configuration is generated 10 times and the results below are averaged over those 10 samples unless otherwise noted.

Figure 4-10 and Figure 4-11 shows which part of the network has to be crawled before a number of clusters is fully crawled. Because the clusters are equally sized, the optimal scenario is when a crawl follows the diagonal of the plot. This figure shows that for all large P_{in} the mutual crawling algorithm performs better than BFS and DFS, because a smaller portion of the network has to be crawled in order to completely crawl clusters. There is also a clear difference between the performance of BFS and DFS; BFS performs better than DFS.

Figure 4-12 shows how many times the size of the cluster has to be crawled in order to completely crawl a cluster for networks generated using an equal slot distribution. This figures show that the proposed algorithm requires the least amount of nodes to crawl a complete cluster. For networks with community structure ($P_{in} \geq 0.4$) at least 2.0 times the cluster size has to be crawled in order to completely crawl the first cluster. For BFS it is necessary to crawl 53.9 times the cluster size and for DFS it is 92.5 times the cluster size which is almost the entire network. In order to complete the first community, you need to crawl 26.9 times more nodes in BFS than with the proposed method. For DFS it is even worse, you need to crawl 46.2 times more nodes! Figure 4-13 shows that for power-law distributed networks you need to crawl 1.6, 7.7 and 73.5 times the first community size for the proposed algorithm, BFS and DFS respectively. Here the proposed algorithm shows an improvement of 4.8 times compared to BFS and 45.9 times for DFS.

The proposed algorithm performs significantly better than BFS and DFS for the computer-generated networks if there sufficient community structure.

4-3 Conclusion

The performance of proposed method for crawling nodes inside a community structure of a network is determined by the community size distribution and the degree distribution itself and the community strength of the network. The method performs better than existing crawling techniques such as BFS and DFS for the football network and computer-generated networks. The proposed algorithm has to crawl a smaller portion of a network in order to completely crawl communities. This improvement is measured to be between 66% and 480%.

Comparing the crawling techniques for real world social networks proved to be difficult, because there is no “ground truth” community structure available for existing networks. Finding the community structure using community detection algorithms such as Spinglass, Fast Greedy and Label Propagation put constrains on the selection of the network under test.

Figure 4-10: Part of the network crawled before completing a cluster for networks generated using an equal slot distribution

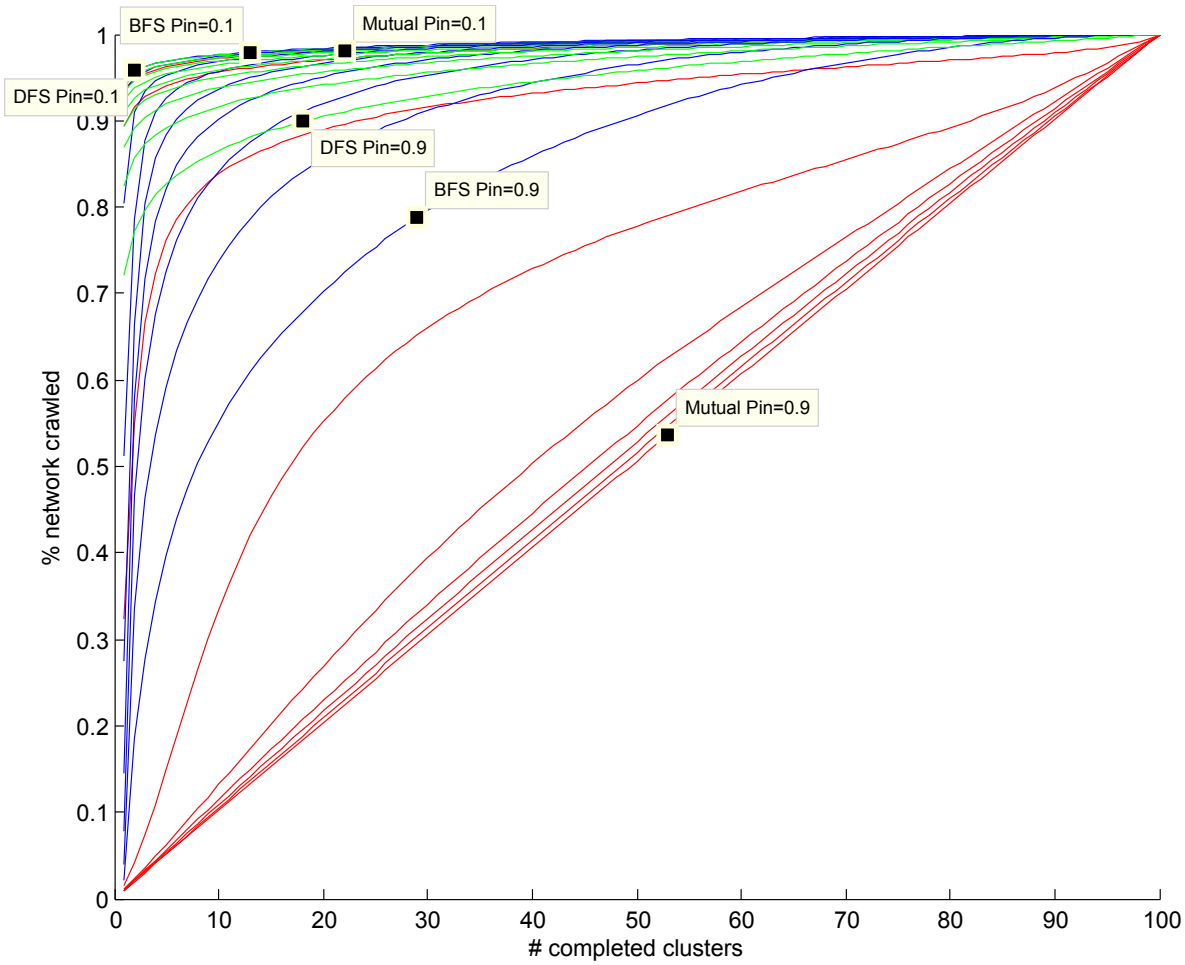
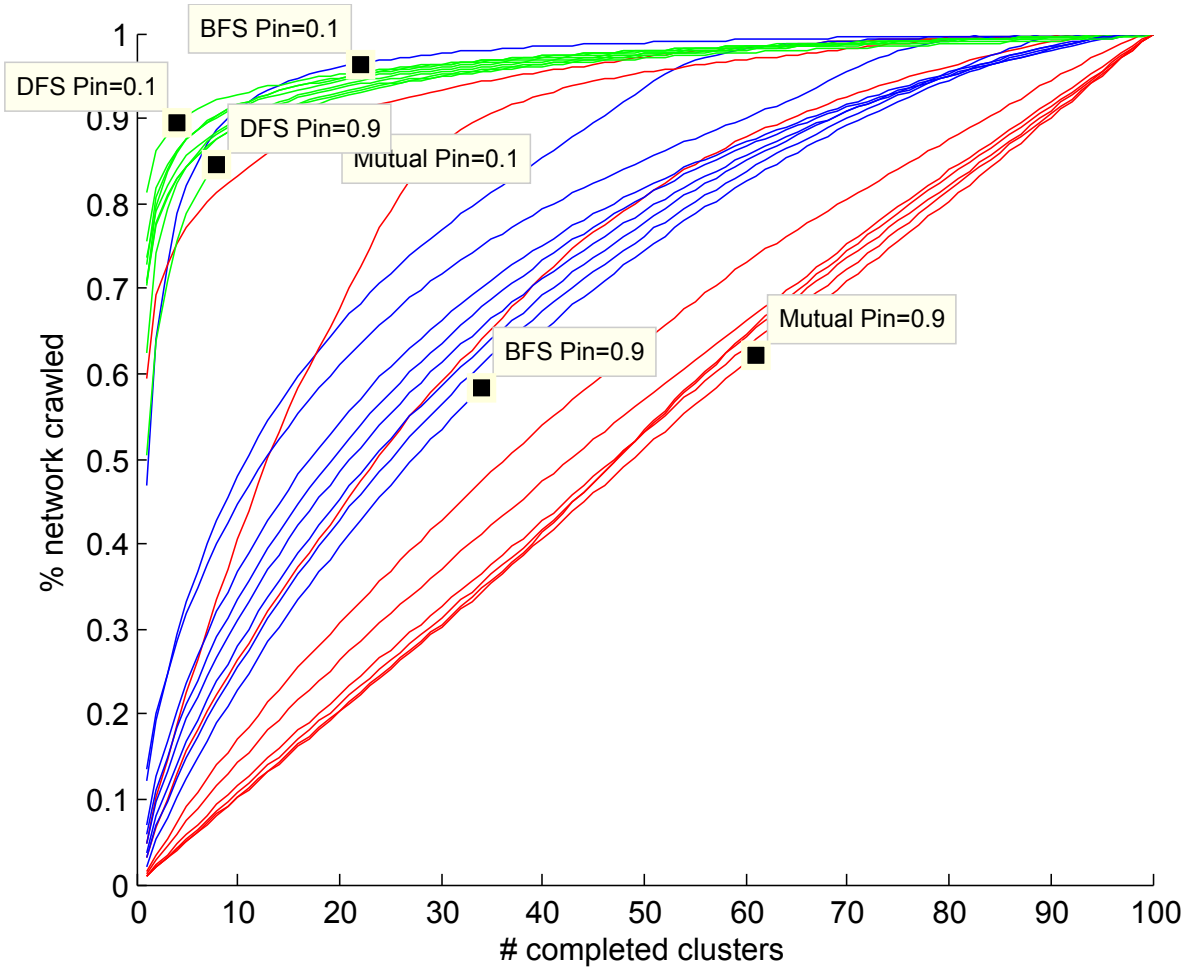
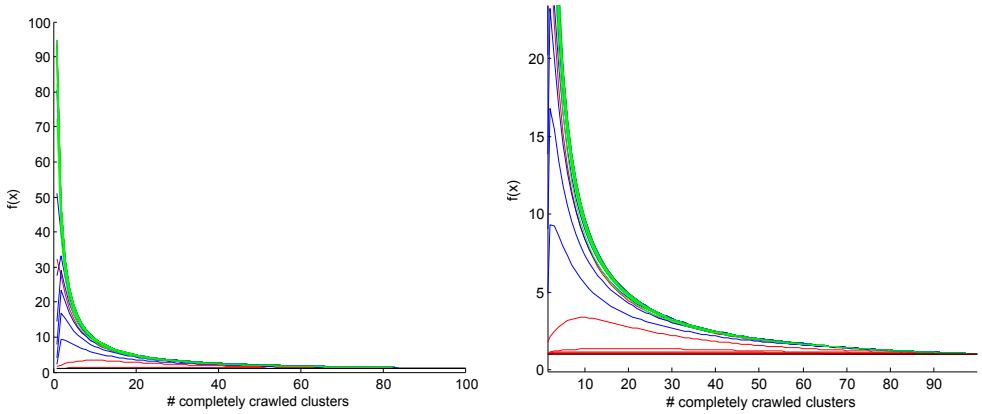


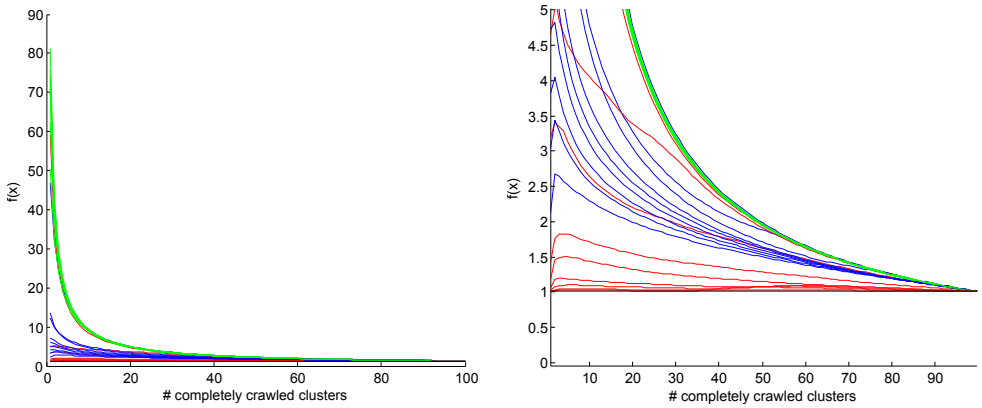
Figure 4-11: Part of the network crawled before completing a cluster for networks generated using a power-law slot distribution





(a) f(x) (b) f(x) zoom

Figure 4-12: f(x) for networks generated with an equal slot distribution



(a) f(x) (b) f(x) zoom

Figure 4-13: f(x) for networks generated with a power-law distribution

The network size is the main influence on the time complexity of those algorithms. For the relatively small Enron network (compared to Facebook) the community detection algorithms gave skewed cluster size distributions. The community detection algorithms showed no similar partitioning, which indicates a weak community structure. Therefore comparing the crawling techniques for this network proved to be difficult. However, the proposed algorithm performed similar to BFS and DFS.

Chapter 5

Conclusion & Future work

5-1 Summary

In this thesis, a new crawling method is proposed that aims to crawl entire communities before continuing to other communities. By crawling entire communities, the analysis of community structure can start before the complete crawl of the network is finished.

A method was proposed for crawling nodes inside a community structure of a network. This new method performs better than existing crawling techniques such as Breadth First Search (BFS) and Depth First Search (DFS) for a football network and computer-generated networks, because a smaller portion of the networks had to be crawled in order to completely crawl communities. For these networks the proposed method performs between 66% and 480% better than BFS and DFS.

Comparing the crawling techniques for real world social networks proved to be difficult, because there is no "ground truth" community structure available for most available networks. Therefore, community detection algorithms were used to find the community structure. Three commonly used community detection algorithms were compared using computer-generated networks. The Spinglass algorithm clearly performed better than Label Propagation and Fast Greedy community detection, but is computationally more expensive. The time complexity of Spinglass puts constrains on the network size and therefore the Enron network was selected. For the relatively small Enron network all community detection algorithms gave inconclusive results and therefore comparing the crawling techniques for this network proved to be difficult.

The computer-generated networks used in this thesis where created using a new network generator. This network generator uniquely combines three features; it creates networks with explicit community structure, arbitrary degree distributions and adaptable community strength. The generator was validated for equal and power-law degree distributions and variating community strengths. This generator enabled us to create networks that were used to verify the performance of the crawling method and comparing the community detection algorithms.

5-2 Future Work

This section concludes this thesis by suggesting ideas for future work. Some of these proposals are the following:

1. Network generator
   (a) Support multiple cluster size distribution: Currently only networks with equally sized communities are generated. With some modifications it should be possible to generate networks with arbitrary cluster size distributions.
   (b) Support overlapping communities: The current implementation assigns each node to a single community. However, in real world social networks, a person can be part of multiple communities. A useful addition to the generator would be the support of overlapping communities.
2. Proposed crawling method
   (a) Multiple starting points in the same community: In the first iteration of the crawling method, a neighbor in a different community can be selected. By using multiple starting point in the same community, this "startup problem" can be avoided.
   (b) Verify the performance for large scale online social networks: Because of the time complexity of Spinglass, we were unable to obtain the community structure of a large scale online social network. More work should be put in finding the community structure of a network such as Facebook or Hyves, so that the algorithm can fully be verified.
   (c) Parallelization of the proposed crawling method: The proposed algorithm supports only single threaded download of profiles. With a parallel algorithm, the time it takes to obtain the network can be reduced. This might be achieved by starting multiple crawls which are "far way" from each other.
   (d) Community detection using the proposed crawling technique: Manual verification of the performance of the crawler for the Football network showed leads for identifying communities using the proposed crawler. The iteration in which the method completed crawling a community coincided with drops in the 'score'. These drops might be a way to identify the community strength or even identify communities.

Appendix A

Football network

Table A-1: Number of games between conferences

clusterid	1	2	3	4	5	6	7	8	9	10	11	12
1	36	5	2	1	2	3	1	2	0	5	2	2
2	5	28	1	0	5	8	7	1	1	0	1	1
3	2	1	44	5	3	2	10	1	5	1	2	4
4	1	0	5	49	3	3	2	4	3	0	6	7
5	2	5	3	3	31	4	1	1	0	8	7	11
6	3	8	2	3	4	1	8	4	3	1	6	2
7	1	7	10	2	1	8	51	1	0	4	0	1
8	2	1	1	4	1	4	1	28	8	3	2	5
9	0	1	5	3	0	3	0	8	40	1	3	6
10	5	0	1	0	8	1	4	3	1	49	7	2
11	2	1	2	6	7	6	0	2	3	7	10	9
12	2	1	4	7	11	2	1	5	6	2	9	30

Appendix B

Cluster graph generator

Table B-1: Results of cluster graph generator for networks with equal distribution of 22 slots and 20 clusters

N	P_{in} input	global P_{in}	local P_{in} avg	local P_{in} std	N in GCC	Avg D	std D	Avg com. size	std com. size
100	0.1	0.097	0.097	0.054	100	20	1.331	5	0
100	0.2	0.192	0.193	0.026	100	20	1.442	5	0
200	0.1	0.1	0.1	0.064	200	20	1.34	10	0
200	0.2	0.2	0.2	0.077	200	20	1.339	10	0
200	0.3	0.3	0.301	0.081	200	20	1.361	10	0
200	0.4	0.4	0.403	0.066	200	20	1.408	10	0
500	0.1	0.097	0.097	0.065	500	20	1.321	25	0
500	0.2	0.198	0.199	0.086	500	20	1.324	25	0
500	0.3	0.301	0.301	0.097	500	20	1.33	25	0
500	0.4	0.4	0.4	0.104	500	20	1.331	25	0
500	0.5	0.501	0.502	0.103	500	20	1.329	25	0
500	0.6	0.6	0.601	0.101	500	20	1.342	25	0
500	0.7	0.699	0.7	0.096	500	20	1.318	25	0
500	0.8	0.799	0.8	0.085	500	20	1.288	25	0
500	0.9	0.9	0.901	0.065	500	20	1.249	25	0
1000	0.1	0.1	0.1	0.066	1000	20	1.35	50	0
1000	0.2	0.199	0.199	0.088	1000	20	1.361	50	0
1000	0.3	0.3	0.3	0.1	1000	20	1.358	50	0
1000	0.4	0.399	0.4	0.107	1000	20	1.342	50	0
1000	0.5	0.5	0.5	0.109	1000	20	1.342	50	0
1000	0.6	0.601	0.602	0.107	1000	20	1.345	50	0
1000	0.7	0.7	0.7	0.101	1000	20	1.332	50	0
1000	0.8	0.8	0.801	0.088	1000	20	1.32	50	0
1000	0.9	0.901	0.901	0.065	1000	20	1.31	50	0
2000	0.1	0.1	0.1	0.067	2000	20	1.353	100	0
2000	0.2	0.2	0.2	0.09	2000	20	1.349	100	0
2000	0.3	0.3	0.3	0.101	2000	20	1.339	100	0
2000	0.4	0.398	0.398	0.109	2000	20	1.343	100	0

Continued on next page

Table B-1: Results of cluster graph generator for networks with equal distribution of 22 slots and 20 clusters

N	P_{in} input	global P_{in}	local P_{in} avg	local P_{in} std	N in GCC	Avg D	std D	Avg com. size	std com. size
2000	0.5	0.499	0.499	0.11	2000	20	1.34	100	0
2000	0.6	0.6	0.6	0.109	2000	20	1.347	100	0
2000	0.7	0.7	0.7	0.102	2000	20	1.344	100	0
2000	0.8	0.801	0.801	0.089	2000	20	1.343	100	0
2000	0.9	0.9	0.9	0.067	2000	20	1.338	100	0
10000	0.1	0.1	0.1	0.067	10000	20	1.346	500	0
10000	0.2	0.2	0.2	0.09	10000	20	1.35	500	0
10000	0.3	0.3	0.3	0.103	10000	20	1.345	500	0
10000	0.4	0.4	0.4	0.11	10000	20	1.348	500	0
10000	0.5	0.501	0.501	0.112	10000	20	1.347	500	0
10000	0.6	0.6	0.6	0.11	10000	20	1.349	500	0
10000	0.7	0.699	0.699	0.103	10000	20	1.35	500	0
10000	0.8	0.799	0.799	0.089	10000	20	1.35	500	0
10000	0.9	0.9	0.9	0.067	10000	20	1.348	500	0

Table B-2: Results of cluster graph generator for networks with powerlaw slot distribution with $\gamma = 1.5$ and 20 clusters

N	P_{in} input	global P_{in}	local P_{in} avg	local P_{in} std	N in GCC	Avg D	std D	Avg com. size	std com. size
200	0.1	0.1	0.139	0.138	200	20	27.217	10	0
200	0.2	0.196	0.319	0.204	200	20	25.279	10	0
500	0.1	0.099	0.149	0.198	500	20	37.024	25	0
500	0.2	0.204	0.332	0.238	500	20	36.021	25	0
500	0.3	0.302	0.501	0.256	500	20	33.35	25	0
500	0.4	0.4	0.635	0.237	500	20	30.378	25	0
500	0.5	0.497	0.735	0.227	500	20	26.821	25	0
1000	0.1	0.104	0.188	0.278	1000	20	45.918	50	0
1000	0.2	0.2	0.359	0.292	1000	20	45.217	50	0
1000	0.3	0.302	0.516	0.283	1000	20	42.703	50	0
1000	0.4	0.4	0.633	0.256	1000	20	39.8	50	0
1000	0.5	0.496	0.738	0.222	1000	20	36.075	50	0
1000	0.6	0.602	0.828	0.184	1000	20	31.827	50	0
1000	0.7	0.699	0.88	0.155	1000	20	27.482	50	0
1000	0.8	0.799	0.921	0.126	1000	20	22.582	50	0
2000	0.1	0.108	0.227	0.333	2000	20	57.029	100	0
2000	0.2	0.205	0.399	0.335	2000	20	55.077	100	0
2000	0.3	0.301	0.544	0.305	2000	20	53.564	100	0
2000	0.4	0.398	0.659	0.27	2000	20	50.863	100	0
2000	0.5	0.501	0.759	0.225	2000	20	46.898	100	0
2000	0.6	0.602	0.836	0.178	2000	20	42.851	100	0
2000	0.7	0.7	0.89	0.141	2000	20	37.767	100	0
2000	0.8	0.802	0.936	0.102	2000	20	32.884	100	0
2000	0.9	0.899	0.969	0.066	2000	20	27.389	100	0
5000	0.1	0.119	0.33	0.403	5000	20	74.743	250	0
5000	0.2	0.215	0.492	0.388	5000	20	69.789	250	0
5000	0.3	0.306	0.6	0.337	5000	20	69.334	250	0
5000	0.4	0.403	0.704	0.28	5000	20	67.59	250	0

Continued on next page

Table B-2: Results of cluster graph generator for networks with powerlaw slot distribution with $\gamma = 1.5$ and 20 clusters

N	P_{in} input	global P_{in}	local P_{in} avg	local P_{in} std	N in GCC	Avg D	std D	Avg com. size	std com. size
5000	0.5	0.5	0.787	0.231	5000	20	64.486	250	0
5000	0.6	0.601	0.854	0.183	5000	20	59.96	250	0
5000	0.7	0.699	0.902	0.137	5000	20	55.463	250	0
5000	0.8	0.8	0.942	0.096	5000	20	49.481	250	0
5000	0.9	0.899	0.975	0.055	5000	20	43.437	250	0
10000	0.1	0.128	0.427	0.441	10000	20	91.265	500	0
10000	0.2	0.219	0.539	0.39	10000	20	87.446	500	0
10000	0.3	0.303	0.607	0.35	10000	20	88.757	500	0
10000	0.4	0.402	0.717	0.296	10000	20	83.187	500	0
10000	0.5	0.504	0.812	0.235	10000	20	78.647	500	0
10000	0.6	0.601	0.866	0.184	10000	20	75.545	500	0
10000	0.7	0.7	0.913	0.138	10000	20	71.239	500	0
10000	0.8	0.8	0.949	0.092	10000	20	65.338	500	0
10000	0.9	0.9	0.978	0.052	10000	20	58.814	500	0

Table B-3: Results of cluster graph generator for networks with equal distribution of 22 slots and 100 clusters and 10000 nodes

P_{in} input	global P_{in}	local P_{in} avg	local P_{in} std	N in GCC	Avg D	std D	Avg com. size	std com. size
0.1	0.100	0.100	0.067	10000	20	1.344	100	0
0.2	0.200	0.200	0.089	10000	20	1.343	100	0
0.3	0.300	0.300	0.102	10000	20	1.347	100	0
0.4	0.400	0.400	0.109	10000	20	1.351	100	0
0.5	0.500	0.500	0.111	10000	20	1.341	100	0
0.6	0.600	0.600	0.109	10000	20	1.348	100	0
0.7	0.700	0.700	0.102	10000	20	1.340	100	0
0.8	0.801	0.801	0.089	10000	20	1.340	100	0
0.9	0.900	0.900	0.067	10000	20	1.335	100	0

Table B-4: Results of cluster graph generator for networks with powerlaw slot distribution with $\gamma = 1.5$ and 100 clusters and 10000 nodes

P_{in} input	global P_{in}	local P_{in} avg	local P_{in} std	N in GCC	Avg D	std D	Avg com. size	std com. size
0.1	0.122	0.502	0.428	10000	20	85.555	100	0
0.2	0.210	0.619	0.360	10000	20	80.347	100	0
0.3	0.302	0.705	0.303	10000	20	76.822	100	0
0.4	0.401	0.794	0.242	10000	20	69.044	100	0
0.5	0.501	0.852	0.196	10000	20	61.775	100	0
0.6	0.600	0.901	0.154	10000	20	54.048	100	0
0.7	0.699	0.934	0.124	10000	20	46.140	100	0
0.8	0.799	0.960	0.097	10000	20	37.420	100	0
0.9	0.900	0.980	0.066	10000	20	28.915	100	0

Table B-5: Comparison of community detection algorithms for networks created with cluster graph generator. Networks have a equal distribution of 22 slots and 20 clusters

		Modularity				Fsame			Jaccard		
N	P_{in}	G	F	L	S	F	L	S	F	L	S
100	0.1	0.047	0.155	0.000	0.184	35.7	52.5	34.9	0.046	0.040	0.053
100	0.2	0.141	0.164	0.000	0.194	45.5	52.5	52.6	0.078	0.040	0.131
200	0.1	0.050	0.177	0.000	0.208	30.2	52.5	27.7	0.047	0.045	0.049
200	0.2	0.150	0.181	0.000	0.218	38.2	52.5	43.5	0.068	0.045	0.108
200	0.3	0.250	0.212	0.000	0.266	49.2	52.5	76.1	0.109	0.045	0.411
200	0.4	0.350	0.285	0.236	0.352	58.3	63.1	90.3	0.164	0.196	0.683
500	0.1	0.047	0.188	0.000	0.225	23.3	52.5	19.0	0.043	0.048	0.040
500	0.2	0.148	0.191	0.000	0.229	28.7	52.5	30.4	0.054	0.048	0.069
500	0.3	0.251	0.210	0.000	0.263	41.9	52.5	73.2	0.092	0.048	0.382
500	0.4	0.350	0.268	0.035	0.350	52.6	52.5	95.9	0.144	0.048	0.854
500	0.5	0.451	0.357	0.447	0.449	59.1	95.8	97.3	0.195	0.829	0.906
500	0.6	0.550	0.467	0.549	0.547	65.3	98.9	99.2	0.271	0.961	0.969
500	0.7	0.649	0.578	0.649	0.643	71.8	99.8	99.3	0.374	0.991	0.972
500	0.8	0.749	0.695	0.749	0.744	78.7	100.0	99.5	0.493	1.000	0.979
500	0.9	0.850	0.823	0.850	0.845	90.4	100.0	99.5	0.717	1.000	0.980
1000	0.1	0.050	0.190	0.000	0.231	20.3	52.5	16.2	0.043	0.049	0.039
1000	0.2	0.149	0.190	0.000	0.233	24.5	52.5	22.0	0.048	0.049	0.050
1000	0.3	0.250	0.204	0.000	0.261	33.7	52.5	69.9	0.070	0.049	0.349
1000	0.4	0.349	0.255	0.000	0.350	48.0	52.5	96.1	0.128	0.049	0.863
1000	0.5	0.450	0.341	0.444	0.448	57.0	93.4	98.2	0.186	0.840	0.932
1000	0.6	0.551	0.458	0.550	0.548	64.0	99.6	99.3	0.266	0.988	0.974
1000	0.7	0.650	0.568	0.650	0.644	69.4	99.9	99.5	0.353	0.997	0.980
1000	0.8	0.750	0.689	0.750	0.743	76.1	100.0	99.2	0.460	1.000	0.971
1000	0.9	0.851	0.816	0.851	0.843	86.6	100.0	99.5	0.644	1.000	0.982
2000	0.1	0.050	0.190	0.000	0.235	19.1	52.5	13.9	0.043	0.050	0.038
2000	0.2	0.150	0.191	0.000	0.236	20.6	52.5	18.1	0.045	0.050	0.045

Continued on next page

Table B-5: Comparison of community detection algorithms for networks created with cluster graph generator. Networks have a equal distribution of 22 slots and 20 clusters

		Modularity				Fsame			Jaccard		
N	P_{in}	G	F	L	S	F	L	S	F	L	S
2000	0.3	0.250	0.198	0.000	0.259	30.0	52.5	66.1	0.061	0.050	0.309
2000	0.4	0.348	0.241	0.000	0.348	44.1	52.5	97.2	0.110	0.050	0.898
2000	0.5	0.449	0.326	0.310	0.447	54.5	74.4	98.9	0.173	0.449	0.959
2000	0.6	0.550	0.434	0.549	0.547	61.7	99.7	99.6	0.238	0.990	0.983
2000	0.7	0.650	0.562	0.649	0.645	68.4	99.6	99.6	0.344	0.989	0.983
2000	0.8	0.751	0.686	0.751	0.745	75.6	100.0	99.6	0.462	1.000	0.983
2000	0.9	0.850	0.811	0.850	0.843	86.4	100.0	99.6	0.634	1.000	0.985
5000	0.1	0.051	0.183	0.000	0.237	18.1	52.5	12.0	0.043	0.050	0.038
5000	0.2	0.150	0.183	0.000	0.237	19.3	52.5	14.7	0.044	0.050	0.041
5000	0.3	0.250	0.188	0.000	0.256	24.0	52.5	56.9	0.050	0.050	0.224
5000	0.4	0.351	0.224	0.000	0.351	38.4	52.5	96.7	0.088	0.050	0.883
5000	0.5	0.451	0.312	0.000	0.449	53.9	52.5	99.1	0.172	0.050	0.967
5000	0.6	0.550	0.434	0.546	0.546	63.3	94.5	98.3	0.278	0.784	0.939
5000	0.7	0.650	0.564	0.650	0.645	72.0	99.5	99.3	0.395	0.982	0.975
5000	0.8	0.749	0.678	0.749	0.745	75.8	100.0	99.7	0.469	1.000	0.988
5000	0.9	0.850	0.812	0.850	0.848	86.4	100.0	99.9	0.642	1.000	0.995
10000	0.1	0.050	0.175	0.000	0.237	17.8	52.5	11.5	0.043	0.050	0.038
10000	0.2	0.150	0.175	0.000	0.238	18.6	52.5	13.0	0.044	0.050	0.039
10000	0.3	0.250	0.177	0.000	0.253	20.9	52.5	50.9	0.046	0.050	0.179
10000	0.4	0.350	0.203	0.000	0.349	32.9	52.5	95.5	0.070	0.050	0.844
10000	0.5	0.451	0.278	0.000	0.449	46.6	52.5	98.9	0.131	0.050	0.959
10000	0.6	0.550	0.410	0.354	0.547	62.4	86.5	99.0	0.262	0.564	0.965
10000	0.7	0.649	0.547	0.649	0.646	72.7	100.0	99.6	0.405	1.000	0.987
10000	0.8	0.749	0.677	0.749	0.746	77.0	100.0	99.1	0.492	1.000	0.966
10000	0.9	0.850	0.811	0.850	0.848	88.1	100.0	99.4	0.674	1.000	0.979

Table B-6: Comparison of community detection algorithms for networks created with cluster graph generator. Networks have a power-law slot distribution of $\gamma = 1.5$ and 20 clusters

		Modularity				Fsame			Jaccard		
N	P_{in}	G	F	L	S	F	L	S	F	L	S
500	0.1	0.043	0.128	0	0.141	24.7	52.5	22.9	0.046	0.048	0.047
500	0.2	0.148	0.144	0	0.169	40.4	52.5	54.6	0.094	0.048	0.195
500	0.3	0.247	0.205	0	0.252	56.3	52.5	84.2	0.176	0.048	0.568
500	0.4	0.347	0.297	0	0.346	67.1	52.5	94.2	0.293	0.048	0.803
500	0.5	0.443	0.391	0.003	0.442	72.5	52.8	97.8	0.369	0.049	0.922
1000	0.1	0.049	0.131	0	0.142	23.1	52.5	20.3	0.045	0.049	0.044
1000	0.2	0.145	0.141	0	0.169	34	52.5	51.1	0.075	0.049	0.173
1000	0.3	0.247	0.202	0	0.253	53.7	52.5	83.4	0.163	0.049	0.556
1000	0.4	0.344	0.288	0	0.345	63.4	52.5	94.2	0.256	0.049	0.81
1000	0.5	0.442	0.388	0	0.442	70.1	52.5	96.7	0.344	0.049	0.884
1000	0.6	0.549	0.507	0.078	0.548	78.8	57.6	99.2	0.5	0.123	0.971
1000	0.7	0.647	0.613	0.479	0.645	85.8	80.6	99.4	0.616	0.408	0.977
1000	0.8	0.748	0.727	0.748	0.747	95.3	100	99.7	0.813	1	0.986
2000	0.1	0.054	0.134	0	0.143	23.2	52.5	20.4	0.046	0.05	0.043
2000	0.2	0.151	0.141	0	0.171	35	52.5	50.1	0.074	0.05	0.17
2000	0.3	0.247	0.201	0	0.253	52.5	52.5	82.8	0.159	0.05	0.543
2000	0.4	0.344	0.286	0	0.347	61.9	52.5	94.8	0.244	0.05	0.822
2000	0.5	0.447	0.392	0	0.447	69.5	52.5	97.6	0.339	0.05	0.912
2000	0.6	0.549	0.503	0.108	0.548	76	61.1	98.6	0.448	0.204	0.953
2000	0.7	0.648	0.626	0.473	0.646	89.2	82.7	98.8	0.692	0.524	0.952
2000	0.8	0.75	0.744	0.733	0.749	98	95.8	99.6	0.929	0.824	0.986
2000	0.9	0.848	0.847	0.847	0.846	99.9	99.9	99.6	0.997	0.998	0.984
5000	0.1	0.066	0.136	0	0.141	24.5	52.5	22.1	0.047	0.05	0.043
5000	0.2	0.163	0.137	0	0.177	34.4	52.5	55.8	0.072	0.05	0.199
5000	0.3	0.253	0.206	0	0.261	51.9	52.5	81.3	0.157	0.05	0.513
5000	0.4	0.35	0.294	0	0.352	63.2	52.5	93.7	0.26	0.05	0.792

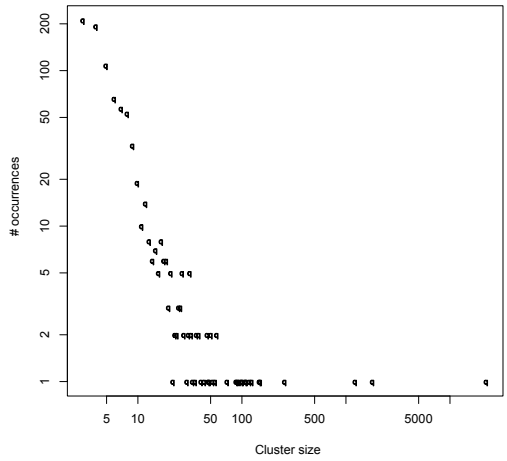
Continued on next page

Table B-6: Comparison of community detection algorithms for networks created with cluster graph generator. Networks have a power-law slot distribution of $\gamma = 1.5$ and 20 clusters

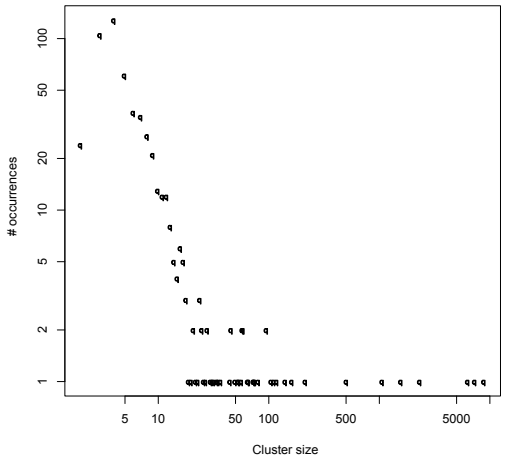
		Modularity				Fsame			Jaccard		
N	P_{in}	G	F	L	S	F	L	S	F	L	S
5000	0.5	0.447	0.396	0	0.448	69.3	52.5	97.7	0.35	0.05	0.914
5000	0.6	0.548	0.51	0.087	0.549	76.8	58.6	98.8	0.476	0.142	0.953
5000	0.7	0.647	0.627	0.389	0.645	89.1	75.7	98.2	0.679	0.361	0.945
5000	0.8	0.748	0.741	0.728	0.747	96.5	96.5	99.5	0.879	0.866	0.98
5000	0.9	0.847	0.847	0.841	0.846	100	98.7	99.5	0.998	0.946	0.981
10000	0.1	0.076	0.133	0.014	0.139	26.7	48.9	24.3	0.048	0.05	0.046
10000	0.2	0.167	0.139	0.009	0.18	36.9	50.4	58.8	0.075	0.05	0.214
10000	0.3	0.251	0.212	0	0.266	51.2	52.5	79.1	0.156	0.05	0.481
10000	0.4	0.35	0.317	0	0.356	68	52.5	89.2	0.335	0.05	0.673
10000	0.5	0.45	0.413	0	0.452	73	52.5	96.4	0.405	0.05	0.871
10000	0.6	0.549	0.517	0.243	0.549	78.7	67.1	98.5	0.513	0.221	0.943
10000	0.7	0.647	0.629	0.516	0.646	89.1	86.2	98.2	0.696	0.648	0.94
10000	0.8	0.748	0.74	0.742	0.746	96.5	98.6	98.4	0.878	0.956	0.95
10000	0.9	0.848	0.847	0.846	0.848	99.4	99.7	99.5	0.98	0.992	0.98

Appendix C

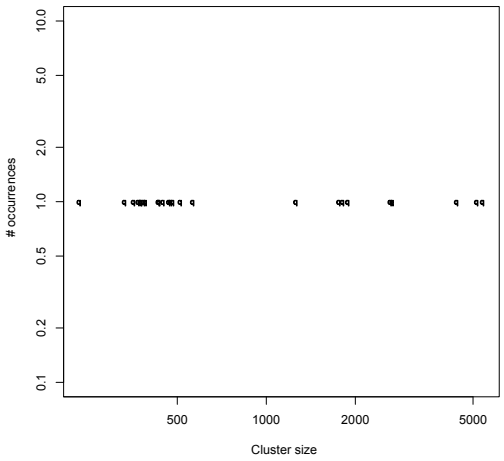
Enron network



(a) Label propagation



(b) Fast greedy



(c) Spinglass

Figure C-1: Cluster size distribution for partitioning of the enron network by three community detection algorithms.

Table C-1: Comparison of different community detection algorithms for the Enron network

(a) fsame index

Fsame	Fast Greedy	Fast Greedy shuffle	Label	Label shuffle	Spinglass	Spinglass shuffle	oneclust
Fast Greedy	1.000	0.278	0.684	0.479	0.573	0.222	0.632
Fast Greedy shuffle	0.278	1.000	0.478	0.479	0.221	0.221	0.632
Label	0.684	0.478	1.000	0.672	0.611	0.430	0.836
Label shuffle	0.479	0.479	0.672	1.000	0.429	0.429	0.836
Spinglass	0.573	0.221	0.611	0.429	1.000	0.164	0.580
Spinglass shuffle	0.222	0.221	0.430	0.429	0.164	1.000	0.580
oneclust	0.632	0.632	0.836	0.836	0.580	0.580	1.000

(b) Jaccard index

Jaccard	Fast Greedy	Fast Greedy shuffle	Label	Label shuffle	Spinglass	Spinglass shuffle	oneclust
Fast Greedy	1	0.09	0.23	0.14	0.22	0.06	0.16
Fast Greedy shuffle	0.09	1	0.14	0.13	0.06	0.06	0.16
Label	0.23	0.14	1	0.29	0.16	0.08	0.46
Label shuffle	0.14	0.13	0.29	1	0.08	0.08	0.46
Spinglass	0.22	0.06	0.16	0.08	1	0.05	0.09
Spinglass shuffle	0.06	0.06	0.08	0.08	0.05	1	0.09
oneclust	0.16	0.16	0.46	0.46	0.09	0.09	1

Bibliography

[1] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, p. 7821, 2002.

[2] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou, "Walking in facebook: A case study of unbiased sampling of osns," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –9.

[3] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, no. 4, pp. pp. 452–473, 1977. [Online]. Available: http://www.jstor.org/stable/3629752

[4] "Facebook - statistieken," http://www.facebook.com/press/info.php?statistics. [Online]. Available: http://www.facebook.com/press/info.php?statistics

[5] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 591–600.

[6] M. Newman, "The structure and function of complex networks," *SIAM review*, pp. 167–256, 2003.

[7] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113, Feb 2004.

[8] S. Fortunato and C. Castellano, "Community Structure in Graphs," *ArXiv e-prints*, Dec. 2007.

[9] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, p. 066111, Dec 2004.

[10] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *CoRR*, vol. abs/0810.1355, 2008.

[11] S. L. Qiaofeng Yang, "A parallel edge-betweenness clustering tool for protein-protein interaction networks," 2006. [Online]. Available: http://www.inderscience.com/link.php?id=11611

[12] D. Ediger, K. Jiang, J. Riedy, D. A. Bader, and C. Corley, "Massive social network analysis: Mining twitter for social good," in *2010 39th International Conference on Parallel Processing*, 2010, p. 583Ú593.

[13] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E*, vol. 76, no. 3, p. 036106, Sep 2007.

[14] J. Reichardt and S. Bornholdt, "Statistical mechanics of community detection," *Phys. Rev. E*, vol. 74, no. 1, p. 016110, Jul 2006.

[15] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1998.

[16] B. Good, Y. De Montjoye, and A. Clauset, "Performance of modularity maximization in practical contexts," *Physical Review E*, vol. 81, no. 4, p. 046106, 2010.

[17] B. Klimt and Y. Yang, "Introducing the enron corpus." in *CEAS'04*, 2004, pp. –1–1.

[18] J. Leskovec, "SNAP: network datasets: Enron email network," http://snap.stanford.edu/data/email-Enron.html, date: 14-9-2011. [Online]. Available: http://snap.stanford.edu/data/email-Enron.html

[19] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, p. 046110, Oct 2008. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.78.046110