Design, Analysis and Experimental Evaluation of a Distributed Community Detection Algorithm

Master's Thesis

HUANG, Hong

Design, Analysis and Experimental Evaluation of a Distributed Community Detection Algorithm

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

HUANG, Hong
born in Canton, China

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
wis.ewi.tudelft.nl

Real Impact Analytics
5 Place du Champ de Mars, 1050
Brussels, Belgium
realimpactanalytics.com

© 2015 HUANG, Hong. Coverpicture: some picture.

Design, Analysis and Experimental Evaluation of a Distributed Community Detection Algorithm

Author: HUANG, Hong
Student id: 4281942
Email: ground.huanghong@gmail.com

Abstract

Complex networks are a special type of graph that frequently appears in nature and in many different fields of science and engineering. Studying complex networks is the key to solve the problems in these fields. Complex networks have unique features which we cannot find in regular graphs, and the study of complex networks gives rise to many interesting research questions.

An interesting feature to study in complex networks is community structure. Intuitively speaking, communities are group of vertices in a graph that are densely connected with each other in the same group, while sparsely connected with other nodes in the graph. The notion of community has practical significance. Many different concepts and phenomenons in real world problems can be translated into communities in a graph, such as politicians with similar opinions in the political opinion network.

In this thesis work, a distributed version of a popular community detection method-Louvain method-is developed using graph computation framework *Apache Spark GraphX*. Characteristics of this algorithm, such as convergence and quality of communities produced, are studied by both theoretical reasoning and experimental evaluation. The result shows that this algorithm can parallelize community detection effectively.

This thesis also explores the possibility of using graph sampling to accelerate resolution parameter selection of a resolution-limit-free community detection method. Two sampling algorithms, random node selection and forest fire sampling algorithm, are compared. This comparison leads to suggestions of choice of sampling algorithm and parameter value of the chosen sampling algorithm.

Thesis Committee:

Chair:	Prof. dr. ir. G.J.P.M.Houben, Faculty EEMCS, TUDelft
University supervisor:	Dr. ir. A.J.H.Hidders, Faculty EEMCS, TUDelft
Company supervisor:	Dr. Gautier Krings, Real Impact Analytics, Belgium
Committee Member:	Dr. ir. Alexandru Iosup, Faculty EEMCS, TUDelft

Preface

I first heard about complex networks 5 years ago. At that time the faculty dean of the software college of Northeastern University, Zhiliang Zhu, always showed us his passion for complex networks. He is a researcher in the field of complex networks and had invited several famous scientists to give us lectures on complex network. One of them was Zhigang Xie (Chi K. Tse) from Hong Kong Polytechnic University. From their inspiring lectures I became interested in the study of complex networks. I was therefore very excited when I found this master thesis topic.

This master thesis topic gives me a valuable chance to do research I am interested in. Defining the research questions and attacking the problems in the thesis is not just very challenging, but also exciting. I have learned a lot during the process.

I would like to express my gratitude and thanks to my supervisors: dr. Gautier Krings and dr.ir. Jan Hidders and my supervising professor dr.ir. Geert Jan Houben. Without their help I would not have been able to complete this master thesis. Also I would like to thank dr. ir. Alexandru Iosup for taking part in the thesis committee.

Last, but not least, I would like to thank my family and friends for support during my studies. This thesis would have not been possible without their help and understanding, so it is dedicated to them.

HUANG, Hong
Delft, the Netherlands
November 29, 2015

Contents

Preface iii

Contents v

List of Figures vii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Objectives . . . 3
- 1.3 Research Questions . . . 3
- 1.4 Thesis Outline . . . 3
- 1.5 Thesis Project Timeline . . . 4

2 Background of Complex Networks, Community Detection and Graph Sampling 7
- 2.1 Types of Complex Networks . . . 7
- 2.2 Communities in Complex Networks . . . 8
- 2.3 Community Detection Methods . . . 9
- 2.4 Related Work . . . 10

3 Design of the Distributed Synchronous Louvain Algorithm 15
- 3.1 Understanding the Louvain Algorithm . . . 15
- 3.2 Naive Idea for Distributing the Computation . . . 16
- 3.3 Inspiration from the Gossip Algorithm . . . 16
- 3.4 Idea of the Distributed Synchronous Louvain Algorithm . . . 17
- 3.5 The Distributed Synchronous Louvain Algorithm . . . 17
- 3.6 Difference between the Gossip Algorithm and the Distributed Synchronous Louvain Algorithm . . . 18
- 3.7 Summary . . . 19

4 Analysis of the Distributed Synchronous Louvain Algorithm 21
- 4.1 Idea to Prove Convergence . . . 21
- 4.2 Proof of Convergence . . . 21
- 4.3 An Upper Bound of the Number of Iterations to Converge . . . 22

4.4 Summary 25

5 Experimental Evaluation **27**
5.1 Research Questions 27
5.2 Hypotheses 28
5.3 Implementation in Apache Spark 28
5.4 Measurement 29
5.5 Experiment Setting 30
5.6 Execution Result on Two Graphs 31
5.7 Comparison of the Sequential Louvain Algorithm and the Distributed Synchronous Louvain Algorithm 32
5.8 The Influence of the Update Probability 36
5.9 The Influence of the Mixing Parameter 39
5.10 The Runtime of the Distributed Synchronous Louvain Method on Larger Graphs and With More Machines 41
5.11 Case Study on Real-World Datasets 46
5.12 Evaluation of Hypotheses 49
5.13 Summary 50

6 Improvement of the Resolution Parameter Selection of the CPM Community Detection Algorithm Using Graph Sampling **53**
6.1 Resolution Parameter Selection 53
6.2 Why Improvement Is Needed 55
6.3 Hypotheses 56
6.4 Measurements for Sample Quality 57
6.5 Experiment Setup 64
6.6 Measurement Results of the Forest Fire Sampling Method 64
6.7 Measurement Results of the Random Node Selection Method 74
6.8 Comparison of the Two Sampling Methods 76
6.9 Evaluation of Hypotheses 80
6.10 Summary 84

7 Conclusions and Future Work **87**
7.1 Contributions 87
7.2 Answer to Research Questions 88
7.3 Limitations 89
7.4 Future work 89

Bibliography **91**

List of Figures

1.1 Timeline of This Mater Thesis Project 5

2.1 An Example of Power-Law Degree Distribution 8
2.2 Example to Explain Resolution Limit Problem, from [11] 11

5.1 Community visualization for graph of 1000 nodes 31
5.2 Community visualization for graph of 5000 nodes 32
5.3 Modularity Comparison for Graphs with 1000 Nodes 33
5.4 Modularity Comparison for Graphs with 5000 Nodes 33
5.5 NMI Comparison for Graphs with 1000 Nodes 35
5.6 NMI Comparison for Graphs with 5000 Nodes 35
5.7 Number of Iterations to Converge, With Different Update Probability, N=1000 38
5.8 Number of Iterations to Converge, With Different Update Probability, N=5000 38
5.9 Number of Iterations to Converge, With Different Mixing Parameter, N=1000 40
5.10 Number of Iterations to Converge, With Different Mixing Parameter, N=1000 40
5.11 Number of Vertices Versus Number of Iterations to Converge 42
5.12 Number of Vertices Versus Number of Iterations to Converge With Fitted Straight Line 43
5.13 Number of Vertices Versus Number of Iterations to Converge With Fitted Straight Line 44
5.14 Number of Workers Versus Number of Execution Time per Iteration . . . 45
5.15 Number of Workers Versus Number of Speedup per Iteration 45
5.16 Number of Workers Versus Number of Speedup per Iteration 46
5.17 Number of Machines Versus Execution Time per Iteration on Amazon Dataset 49

6.1 An example resolution profile 54
6.2 Number Of Intervals(Forest Fire,p=0.3) 57
6.3 Interval Length of *Representatives*(1st interval,p=0.3) 61
6.4 Interval Length of *Representatives*(2nd interval,p=0.3) 61
6.5 Interval Length of *Representatives*(3rd interval,p=0.3) 62
6.6 Resolution Profile of the Graph 65
6.7 Precision for Forest Fire Sampling 66
6.8 Precision for Forward Burning Probability 0.6 and 0.9 67

6.9 Length of *Representatives* of 1st and 2nd Longest Interval(p=0.3) 68
6.10 Length of *Representatives* of 1st and 2nd Longest Interval(p=0.3) 69
6.11 Lengths of *Representatives* for Forward Burning Probability 0.3 71
6.12 Lengths of *Representatives* for Forward Burning Probability 0.6 72
6.13 The Precision for Forward Burning Probability 0.3 73
6.14 Precision of Random Node Selection 75
6.15 Length of *Representative* of Random Node Selection 76
6.16 Sample Size vs. Length of *Representatives* for Three Sampling Strategies(2nd Longest Plateau) 77
6.17 Sample Size vs. Length of *Representatives* for Three Sampling Strategies(3rd Longest Plateau) 78
6.18 Sample Size vs. Length of *Representatives* for Three Sampling Strategies(4th Longest Plateau) 78
6.19 Sample Size vs. Length of *Representatives* for Three Sampling Strategies(5th Longest Plateau) 79
6.20 Precision for Different Sampling Strategies 80

Chapter 1

Introduction

A **graph** is an abstract representation of a set of objects and their connections. Another term for **graph** is **network**. In this representation, some pairs of objects are connected by links and some links can have weights indicating the relative importance. The objects are called **vertices** and the links that connect these objects are called **edges**. A graph can be directed or undirected, which means links in a graph have or do not have directions. In an undirected graph, the degree of a vertex is the sum of the weights of the edges connected to it and in a directed graph, the in(out) degree of a vertex is the sum of the weights of the in(out) edges connected to this vertex.

A **Complex network** is a graph with non-trivial topological features, such as a high clustering coefficient, a heavy-tail degree distribution and presence of communities and hierarchical structure. In graph theory, a clustering coefficient is the measurement of the tendency of nodes in a graph forming clusters. A **High clustering coefficient** indicates that nodes tend to form clusters (which are also called **communities**) in the graph. The degree distribution shows the fraction of nodes that have a particular degree. A **Heavy-tail degree distribution** means that the fraction of vertices with a certain degree decays slowly if the degree is large.

Most social networks (such as internet and Facebook), technology networks (such as electric grids) and biological networks (such as transcriptional regulatory networks and virus-host networks) share these features. They are typical examples of complex networks.

Intuitively, groups or communities in a complex network contain nodes that are frequently connected to other nodes in the same community, but are less frequently connected with nodes outside the community. However, no formal definition of groups in a graph is universally accepted[10]. In most cases groups or communities are just the final product of community detection algorithms, which detect communities in a network. To measure the quality of partitions generated by the community detection algorithms, several measurements have been proposed, such as modularity[5] and significance[31].

1.1 Motivation

1.1.1 Social Relevance

Complex networks are a type of network which can very well model the networks in many scientific, engineering and sociological problems. For instance, telecommunication networks which describe the relation of mobile phone calls[5], airport transportation networks[13] and metabolic networks[15]. Understanding certain properties of complex networks is important for us to solve practical problems related to these networks.

Detecting communities in a complex network is of both scientific and societal importance, because this task has appeared in different forms in many practical problems and in different disciplines[10]. For instance, online retailers (e.g. www.amazon.com) need to identify clusters of customers with similar interests in the network of customer-product relationships to set up their recommendation system. In distributed computing an effective way to improve performance of the distributed system is to split the distributed system into groups of computers and processors. Both of these problems can be modeled as a group detection problem in a complex network[10].

1.1.2 Problems of Existing Approaches

We identify two problems in existing approaches of community detection methods. These two problems are what we are trying to solve in this master thesis project.

The first problem is that many community detection algorithms cannot handle large graphs (for instance, with millions of nodes). It is estimated in [30] that the runtime complexity of the popular *Louvain* community detection method has a complexity of $O(m)$, where m is the number of links in the network. When the size of the graph becomes very large, community detection algorithms such as the *Louvain* method can run very slowly, sometimes even without termination in a reasonable period of time.

One possibility to accelerate community detection algorithms is to distribute the execution of these algorithms. In the recent years, many distributed graph processing frameworks, such as *Spark GraphX*[1] and *Apache Giraph*[2], have been proposed, and it is now possible to distribute the execution of community detection algorithms on graphs. However, many community detection algorithms are not horizontally scalable, which means that adding more machines in the execution of these community detection algorithms does not help. As a result these community detection algorithms do not benefit from distributed computing. For example, the original *Louvain* method[5] looks at one single node at a time and decides if the node should stay in its current community or move to one of its neighbouring communities.

The second problem is that the process to select resolution parameters[31] (a particular parameter in the CPM[31] community detection method) is too slow. For a graph with 10,000 nodes the execution time is already more than one hour, and for a graph with 100,000 thousand nodes the execution time is longer than a day. Many real-world networks are quite large, sometimes with up to millions of nodes. The

[1] https://spark.apache.org/graphx/
[2] http://giraph.apache.org/

long execution time of the resolution parameter selection mechanism proposed in [31] limits its application to real-world problems.

1.2 Objectives

The goal of this project is to develop a distributed version of the Louvain community detection algorithm for complex networks, and to improve the CPM algorithms[31] by studying how to accelerate parameter selection for this community detection algorithm.

1.3 Research Questions

The following research questions will be addressed:

- Louvain community detection algorithms have been implemented to work on a single machine. How can we distribute the execution of this algorithm using big data tools such as Apache Spark?

- How is the performance of the distributed version of Louvain algorithm as comparison to the original sequential Louvain method? What characteristics does this distributed version show in execution?

- To speedup the resolution parameter selection of the CPM community detection algorithm, one possibility is to take a sample of the graph and select resolution parameters based on this sample. To do so, the sample graph should give us a good estimation of the resolution profile of the original graph. Among the many graph sampling strategies, which is the best one to use, such that the estimation of the resolution profile[31] of the original graph is as good as possible?

1.4 Thesis Outline

The structure of this thesis report is as follows. Chapter 2 first presents background knowledge of complex networks and community detection problem in complex networks. Then work related to this thesis is presented, including the original sequential version of the Louvain community detection method[5], the CPM community detection method[31] and the method for choosing the right resolution parameter for the CPM community detection method[31].

Chapter 3 presents the Distributed Synchronous Louvain method. At the beginning of this chapter, it will be explained how the Distributed Synchronous Louvain algorithm is inspired by the Gossip algorithm. The chapter then presents a description and pseudocode of the Distributed Synchronous Louvain algorithm.

Chapter 4 first presents the proof of convergence of the Distributed Synchronous Louvain algorithm. Then a weak estimate (in the chapter you will see why it is a "weak" result) of the upper bound of the average number of of iterations for this algorithm to converge is given.

Chapter 5 presents the experimental evaluation result of the synchronous version of the Louvain method on the LFR benchmark. The experiment design is motivated by five questions:

- Is the quality of partitions obtained by the Distributed Synchronous Louvain method better or worse than those produced by the original sequential Louvain method?
- For the Distributed Synchronous Louvain method, how does the probability to adopt the best community influence the quality of the partition obtained (in terms of Normalized Mutual Information) and the number of iterations to converge?
- How does the mixing parameter influence the quality of partitions produced by the Distribute Synchronous Louvain algorithm and the number of iterations for the algorithm to converge?
- How does the Distributed Synchronous Louvain algorithm scale with increasing number of machines?
- How does the size of the graphs influences the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge?

Chapter 6 presents comparison of two sampling strategies: the forest fire sampling method and the random node selection method, from the perspective of resolution profile estimation. First hypotheses about the differences between the two algorithms are proposed. Next two measurements to quantify the quality of the resolution profile produced by graph samples, the *precision* of the longest stable plateau estimate and the length of *representatives*, are presented based on these hypotheses. Then the measurement results for the forest fire sampling method and the random node selection method are given. Lastly, the proposed hypotheses are evaluated based on experiment results and further hypotheses are proposed.

Chapter 7 concludes the thesis. It provides a summary of its contributions and limitations and discusses possible directions for future research.

1.5 Thesis Project Timeline

The timeline of this master thesis project is shown in **Figure1.1**.

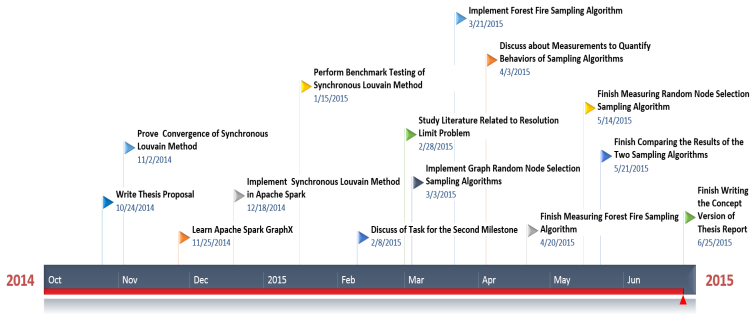

Figure 1.1: Timeline of This Mater Thesis Project

Chapter 2

Background of Complex Networks, Community Detection and Graph Sampling

This chapter presents a review of the literature related to complex networks. The goal of this chapter is to discuss the relevant perspectives that have made complex networks an important research subject in recent years and to introduce the work that this thesis project is built upon. Specifically, the following topics are discussed:

- Types of complex networks
- Definition of communities in complex networks
- Community detection algorithms
- Benchmark for community detection algorithms in complex networks
- Resolution limit problem
- Approach to select the correct resolution parameter
- Graph sampling

2.1 Types of Complex Networks

There are two well-known and much studied types of complex networks: scale-free networks and small-world networks. In the following, the properties of these two types of networks are discussed.

2.1.1 Scale-Free Network

A scale-free network is a network whose degree distribution follows power-law distribution(at least asymptotically). If the degree distribution obeys power law, the degree and the fraction of nodes with that degree shows a linear relation in log-log scale, described as follows:

$$P(k) \sim k^{-\lambda}$$

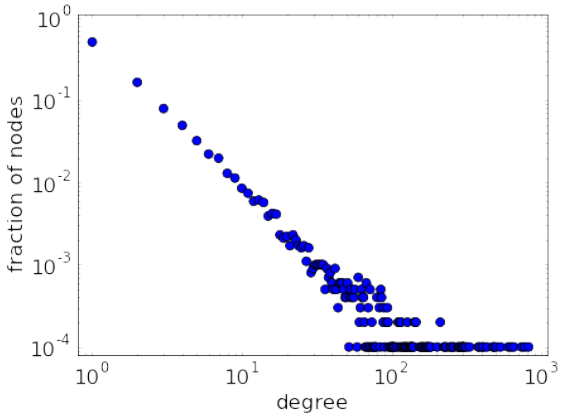

Figure 2.1: An Example of Power-Law Degree Distribution

where k is the degree of a node in the graph and $P(k)$ is the fraction of nodes with degree k. The value of λ is typically between 2 and 3. A typical example of power law distribution is shown in **Figure 2.1**[1].

Many different models have been used to explain how scale-free networks are generated. Some examples include the preferential attachment model[6][3] and the fitness model[4].

2.1.2 Small-World Network

Small-world networks are a type of complex networks in which most nodes are not neighbors, but one node can reach another node in the graph through a relatively small number of steps. In the context of a social network, this means that any user in the network can find any other user in the network after following links in his or her social network and asking on average a small number of users. Many real-world graphs show characteristics of the small-world phenomenon. For example, social networks (such as friendship connections on Facebook[2]), the connectivity of the Internet, Wikipedia[3] or DBpedia, and gene networks in biology.

2.2 Communities in Complex Networks

A partition is a division of a graph in clusters (communities), such that each vertex belongs to one (hard clusters) or more clusters (overlapping communities).

No definition of communities in complex network is universally accepted. It is suggested in [11] that a community in a complex network is a subgraph of a network whose nodes are more tightly connected with each other than with nodes outside the subgraph. This is a fundamental guideline of most community definitions. However,

[1]This image is from http://mathinsight.org/scale_free_network
[2]https://www.facebook.com/
[3]https://zh.wikipedia.org/wiki/

there are many other definitions. Besides, it is quite common that communities are algorithmically defined, which means that they are just the final product of the algorithm, without a precise a-priori definition.

2.3 Community Detection Methods

Many different community detection methods have been proposed. In this section, we only review some of the most important methods.

2.3.1 Clustering-Based Methods

The problem of community detection can be viewed as a special case of clustering problem. Therefore, with properly defined distances between nodes in the graph, many clustering algorithms can be directly applied. The clusters produced by clustering algorithms are communities. For example, a graph can be modelled as an adjacency matrix, and with an adjacency matrix, a Laplacian matrix can be constructed. Spectral clustering can then be applied to the Laplacian Matrix.

2.3.2 Divisive Algorithms

The idea of divisive algorithms is to detect edges that connect nodes from different communities and remove these edges, so that communities are disconnected from the rest of the graph. The most important divisive algorithm for community detection is the algorithm invented by Girvan and Newman[12][23]. In this algorithm, edges to be removed are chosen based on *betweenness*[4]. In each iteration, the edge with the largest *betweenness* is removed. The only edges whose betweenness needs to be recalculated are those edges whose betweenness are influenced by the removal of other edges.

2.3.3 Modularity-Based Methods

Modularity is one of the most popular quality functions for communities. In the case of undirected weighted networks, it is defined as[22]:

$$Q = \frac{1}{2m}\sum_{ij}[A_{ij} - \frac{k_i k_j}{2m}]\sigma(c_i, c_j)$$

where A_{ij} is the weight of the edge between node i and node j, k_i is the sum of the weights of the edges attached to vertex i, c_i is the community that node i belongs to, and $\sigma(c_i, c_j)$ is 1 if node i and node j belong to the same community and 0 otherwise. There are many different ways to optimize modularity. For example, it can be optimized using a greedy approach (e.g. the Louvain method[5]), simulated annealing[14] and external optimization[9].

Besides the algorithms described above, there are many other community detection algorithms. For a complete overview of community detection in complex networks, interested readers could refer to an interesting survey written by Satu Elisa Schaeffer[28] and the survey paper written by Santo Fortunato[10].

[4]The *betweenness* of an edge is the number of shortest paths between pairs of nodes that have a path running along it.

2.4 Related Work

2.4.1 The Louvain Method

The *Louvain* community detection method[5] is a greedy algorithm to optimize modularity. The idea of this algorithm is very simple: the nodes in the graph are visited one by one. Each time a node is visited, the node is first removed from its original community and then added to one of its neighbouring communities or its original community such that the increase of modularity of the whole graph is non-negative and maximum.

2.4.2 Benchmark for Community Detection Algorithm

An early benchmark was proposed by Girvan and Newman[12]. In this benchmark each graph has 128 nodes, divided into four communities with 32 nodes each. This benchmark is regularly used to test community detection algorithms. However, this benchmark does not take into account the complicated features of real-world networks, like the fat-tailed distributions of node degree and different community sizes. The LFR[18][16] benchmark is a popular benchmark for community detection in complex network. In this benchmark the distributions of node degree and community size are both power laws, with tunable exponents.

2.4.3 Resolution Limit Problem

Generally speaking, the algorithms that suffer from resolution limit fail to detect small communities in a large network, even if these communities are clearly defined. This problem is suggested in [11] and an example is given in this paper to illustrate the phenomenon. Another interpretation of this phenomenon is that if you increase the size of the graph while keeping some of the communities clearly defined, you observe that the algorithm is no longer capable of detecting those communities. In this case, the community detection algorithms are said to suffer from resolution limit.

It is suggested in [11] that modularity suffers from resolution limit. An example is given in [11] to illustrate the problem: Assume that we focus on two communities in the network, namely M_1 and M_2, and distinguish three types of links: the internal links of the two communities (l_1 and l_2, respectively), the links between M_1 and M_2 (l_{int}) and between the two communities and the rest of the network M_0(l_1^{out} and l_2^{out}), as is shown in **Figure 2.2**.

Suppose there are two partitions of the graph: partition A and partition B. The only difference between them is that in A, M_1 and M_2 are in separate communities while in B, M_1 and M_2 are in the same community. If we calculate the modularity of partition A and B according to the definition of modularity (Q_A and Q_B respectively), one can see that if we keep l_1, l_2, l_{int}, l_1^{out} and l_2^{out} constant, but increase the size of network, $Q_A > Q_B$ does not always hold. This indicates that when the size of the network is too large, small communities are no longer visible to community detection algorithms that optimize modularity. In other words, modularity suffers from resolution limit.

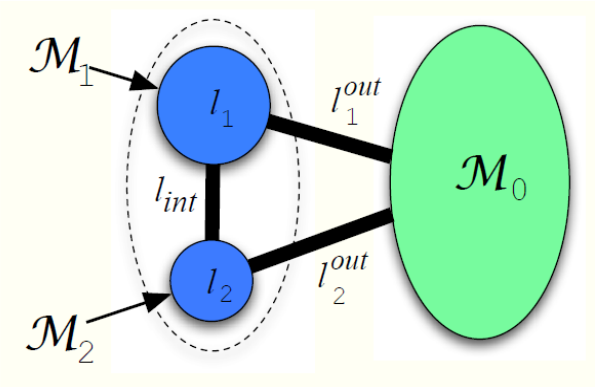

Figure 2.2: Example to Explain Resolution Limit Problem, from [11]

2.4.4 Definition of Resolution-Limit-Free

Resolution-limit-free means that the community detection method does not suffer from the resolution limit problem. The formal definition of resolution-limit-free is given in [32], and is as follows:

Assume the objective function we are using to access to quality of partition is H. Let $C = \{C_1, C_2, ...C_q\}$ *be a H-optimal partition of a graph G. Then the objective function H is called resolution-limit-free if for each subgraph S induced by* $D \subset C$*, the partition D is also H-optimal for S.*

As is suggested in [32], this definition is defined within the framework of first component Potts model, which was developed by Reichardt and Bornholdt[25].

A related definition is *additive objective functions*, which is as follows[32]:

An objective function H for a partition $C = \{C_1, C_2, ...C_q\}$ *is called additive whenever* $H(C) = \sum_i H(C_i)$*, where* $H(C_i)$ *is the objective function defined on the subgraph S induced by* C_i*.*

2.4.5 The Potts Model

This is the model from which several community detection algorithms are derived. The model is based on a very simple idea: In principle, links within communities should be relatively frequent while links between communities should be relatively rare[32]. Based on this idea, we should reward the links within a community and punish the links between communities[25]. This idea can be expressed as the following cost function:

$$H = -\sum_{ij}(a_{ij}A_{ij} - b_{ij}(1 - A_{ij}))\delta(\sigma_i, \sigma_j)$$

where σ_i and σ_j denote the communities that node i and node j belong to. A_{ij} denotes the entries in the adjacency matrix of the graph. $A_{ij} = 0$ if node i and node j are not connected. If node i and node j are connected, in unweighted network A_{ij} is 1 and in weighted network A_{ij} is the weight of the edge. We can see that A_{ij} is the reward term for links inside the community and $(1 - A_{ij})$ is the punish term for links

between communities. In this formula a_{ij} and b_{ij} are two variables controlling the weights of the two terms. The δ function is 1 when the two parameters are the same and 0 otherwise.

2.4.6 Resolution-Limit-Free Methods

The definition of *local weight* of a cost function is as follows[31]:

Let G be a graph, and let a_{ij} and b_{ij} as in the Potts model be the associated weights. Let H be a subgraph of G with associated weights a'_{ij} and b'_{ij} . Then the weights are called local if $a_{ij} = \gamma a'_{ij}$ and $b_{ij} = \gamma b'_{ij}$, where γ depends on subgraph H.

If this condition is satisfied, we say that this cost function has local weights. Whether or not the cost function has local weight is important from the perspective of the resolution limit problem because it is shown in [32] that only two types of cost functions are resolution-limit-free. The first type of cost function has local weights and the second type of cost function must satisfy a very specific requirement. This requirement is so specific that it is impossible to construct such a cost function. Therefore, the only realistic way to construct a cost function that is resolution-limit-free is to make it in a way that it has local weights.

From the Potts model, the cost functions of many community detection methods can be derived. For example, the Reichardt and Bornholdt[25] method sets $a_{ij} = w_{ij} - b_{ij}$ and $b_{ij} = \gamma_{RB} p_{ij}$ where p_{ij} represents the probability of a link between i and j, which is determined by the random null model chosen.

Following the previous inference, if we set $\gamma_{RB} = 1$, we arrive at modularity. We can see that there is no local weight in the cost function, so community detection methods based on modularity optimization suffer from the resolution limit problem.

2.4.7 The Constant Potts Model Method

As is suggested in [32], by defining $a_{ij} = w_{ij} - b_{ij}$ and $b_{ij} = \gamma$, another cost function is obtained:

$$H = -\sum_{ij}(A_{ij}w_{ij} - \gamma)\delta(\sigma_i, \sigma_j)$$

which is called the constant Potts Model(CPM). We can see that the cost function of the CPM method does have local weight: therefore, the CPM method is resolution-limit-free.

2.4.8 Resolution Profile

In the CPM method, the only free parameter is γ, which is called the resolution parameter. For a graph with n nodes, the valid range of the resolution parameter is $[\frac{1}{n}, 1]$. Larger resolution parameters give rise to smaller communities. An optimal partition of the CPM method will remain optimal for a continuous interval of resolution parameters. The larger the interval, the more clear-cut and robust the community structure[31]. The intervals that are significantly larger than the other intervals in the whole valid range of resolution parameters are called the "stable plateau". A *resolution profile* of a graph shows the intervals of resolution parameters where optimal

partitions have the same significance (a measurement defined in [31] to measure the quality of partitions) value and the measurements of these optimal partitions (such as significance[31], the total of internal edges or the sum of squares of community sizes).

The resolution parameter of the CPM method has to be determined. It is suggested in [31] that the partitions which correspond to the stable plateaus in the resolution profile are the planted partitions of the graph. These are the partitions we are looking for, so the resolution parameters should be chosen within these stable plateaus.

To reveal these stable plateaus, the resolution profile of a graph should be constructed. This can be done by bisectioning on the sum of squares of community sizes[31].

2.4.9 Graph Sampling

The goal of graph sampling is to identify a small set of representative nodes and links of a social network[29]. Graph sampling can be beneficial in many ways. For instance, graph sampling can be used to compress graphs or to accelerate algorithms running on the graphs[19]. Graph sampling can be performed on both homogeneous networks (with one type of nodes and one type of edges) and heterogeneous networks (with multiple types of nodes and multiple types of edges)[29]. Graph sampling algorithms are supposed to preserve the properties of the original graph, such as the in/out degree distribution, the path length distribution and the clustering coefficient distribution[29].

Over the years, many different approaches of graph sampling have been proposed. Below, the representative approaches are presented:

In general, there are three strategies of graph sampling algorithms[29]:

- **Node Selection.** This strategy works by randomly selecting nodes from the graph and then the associative links.
- **Edge Selection.** This strategy works by randomly selecting edges and then including the associated nodes.
- **Sampling by Exploration.** This strategy works by randomly selecting the nodes to start with and then performing random walks from these nodes.

A detailed overview of graph sampling algorithms can be found in [29], which summarizes many graph sampling algorithms proposed in recent years. Below, some of the most important graph sampling algorithms are presented.

Node Selection

A naive approach is to randomly select a set of nodes from the graph with a uniform probability and then include the associated links. This method is called random node selection. However, this approach does not preserve any information from the original graph, as it is independent from any property of the original graph. A better idea is to make the properties of the original graph play a role in the probability of choosing a node. If the degrees of the nodes are known before graph sampling, we can make the probability of a node being chosen proportional to the degree of the node[1]. If the PageRank values of the nodes are known before graph sampling, we can make the probability of a node being selected proportional to its PageRank value[19].

Edge Selection

An easy method one can think of is to uniformly select edges at random, and then include the associated nodes (Random Edge Sampling). This approach suffers from the same problem as the naive approach of node selection. A more complicated approach is to uniformly select a node at random, then uniformly select an edge associated to it[29], which is called Random Node-Edge (RNE) Sampling. In [19] a hybrid approach is proposed to combine Random Edge Sampling and Random Node-Edge Sampling. This approach works by running Random Edge Sampling with probability p and Random Node-Edge Sampling with probability $1 - p$.

A method called Induced Edge Sampling[2] is a simple variant of the edge selection strategies mentioned above. First the algorithm uniformly select edges at random and the nodes associated with these edges from the graph. In this way, a set of nodes is selected. This step is repeated several times. Next the algorithm add all the edges that exists between the nodes selected in the previous step. The method proposed in [26] is another example of the edge selection graph sampling method.

Sampling by Exploration

The *Forest Fire* sampling algorithm is one of the algorithms that obtain graph samples by exploration. An exact definition of this algorithm is given in [19], and it can be summarized as follows:

1. Choose a node v uniformly at random to start from.
2. Generate a random number m which is geometrically distributed with mean $\frac{p_f}{1-p_f}$, where p_f is the forward burning probability.
3. Select m out-links of v that have not been visited.
4. Recursively apply step 2 and 3 to all newly added nodes.
5. When the fire dies out, go to step 1.

The sample graph is the graph induced by the selected nodes.

Another graph sampling algorithm is called *Random Walk* sampling. In this algorithm we uniformly at random pick a starting node and then simulate a random walk on the graph.

Chapter 3

Design of the Distributed Synchronous Louvain Algorithm

This chapter presents the synchronous version of the Louvan community detection algorithm. At the beginning of this chapter, the original version of the Louvain community detection algorithm is briefly described. Next it will be explained how the Distributed Synchronous Louvain algorithm is inspired by the Gossip algorithm. This chapter concludes with a description and pseudocode of the Distributed Synchronous Louvain algorithm.

3.1 Understanding the Louvain Algorithm

As a starting point, let us look deeper into the original Louvain community detection algorithm. Recall that this algorithm visits nodes in the graph one by one. Each time a node i is visited, it moves to a community C such that the increase in modularity is positive and maximized. The gain of modularity by adding a node to a neighbouring community is defined in the following formula [5],

$$\Delta Q = \left[\frac{\Sigma_{in}^{C} + k_{i,in}^{C}}{2m} - \left(\frac{\Sigma_{tot}^{C} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}^{C}}{2m} - \left(\frac{\Sigma_{tot}^{C}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{3.1}$$

Let us take a deeper look at this equation. The first term is the modularity of the community with the node inside (Σ_{in}^{C} means the sum of internal links without i, $k_{i,in}^{C}$ is the number of internal links that node i is bringing to the community, while $\Sigma_{tot}^{C} + k_i$ is the total number of links connected to the community, including both internal and external links of the community). The second term is the modularity where the node i and the community C are separated (the first two terms are for the community, while the third term is for the single node).

By simple algebra, equation (3.1) can be simplified as:

$$\Delta Q = \frac{k_{i,in}^{C}}{2m} - \frac{\Sigma_{tot}^{C} k_i}{m} \tag{3.2}$$

which means it is no longer necessary to compute Σ_{in}^{C}. In each iteration, only $k_{i,in}^{C}$, m, k_i and Σ_{tot}^{C} need to be calculated. Among these terms, k_i and m can be computed beforehand, so in each iteration, we only need to update $k_{i,in}^{C}$ and Σ_{tot}^{C}.

Notice that this equation applies only to the gain of modularity if a node is added to a community. When we decide whether or not a node should be moved to another community, we have to consider both the decrease of modularity by removing this node from its current community, and the gain in modularity by adding this node to another community.

The following sections describe how to adjust the original Louvain community detection algorithm to work in the synchronous set-up. The definition of the synchronous version of Louvain community detection algorithm is given and it will be explained how this algorithm is devised.

3.2 Naive Idea for Distributing the Computation

A naive idea to distribute the Louvain method is that in each iteration, every node takes a step to move to the community, which maximize the gain of modularity for that node. However, if we do so, there is a risk that this process will never terminate. In the original Louvain algorithm, once a node has moved to a new community, the state of the community structure (such as the variables Σ^C_{tot} and $k^C_{i,in}$) is updated accordingly, before the next node starts to take a decision to choose the best community for it. In this case the node under consideration can rely on its local knowledge. However, this is not the case in the synchronous scenario, because if all the nodes make their steps to move to another community at the same time, the local knowledge of each node soon becomes inconsistent with the actual global state of community structure. Once you make a move to the optimal community for you, your neighbours also make a move to their optimal communities and your knowledge about your neighbours is no longer valid. For example, node 1 may decide that the community of one of its neighbours, say, node 2, leads to the maximum gain in modularity for node 1. However, at the same time node 2 may decide that the community that node 1 belongs to leads to the maximum gain in modularity for mode 2. As a result, the process enters an infinite loop.

This example might be too extreme. However, one can imagine that if every node in the graph takes the same action at the same time, there is a risk that the process will never terminate, which means that there are always some nodes who always decide to move to another community. This is the reason why we cannot use the naive approach that all nodes update their communities at the same time and why the Louvain algorithm should be adjusted to ensure convergence in synchronous setting.

3.3 Inspiration from the Gossip Algorithm

One way to ensure convergence of the synchronous Louvain method is to modify the algorithm to make it work like a gossip algorithm. Gossip algorithms are used in distributed systems to achieve consensus. The idea of this algorithm is simple and intuitive: In cases where the gossip algorithm is used, for example, algorithms for the averaging problem, at each time step each node contacts one of its neighbours at random, and forms a pair. Then the nodes of the pair do something, such as averaging their current value. In the end the whole distributed system enters a global stable state. The Louvain method can be distributed in a similar way.

3.4 Idea of the Distributed Synchronous Louvain Algorithm

The synchronous version of the Louvain Algorithm differs from the original version of the Louvain algorithm in that the nodes in a graph are not visited one by one. Instead all nodes make their decision to move to other communities at the same time. The idea to achieve global convergence is to modify the original Louvain method using a similar principle as the Gossip algorithm (making a move only with some probability that is smaller than 1). Specifically, at each iteration of the synchronous setting, each node decides with a fixed probability q to update its community following the rules of the original Louvain algorithm, and with probability $1 - q$ it does not do anything. Because of the introduction of this probability, it is possible for a node to stay in its current community and wait for the other nodes to join its community. Intuitively, this should ensure the convergence of the Louvain method in a synchronous set-up.

3.5 The Distributed Synchronous Louvain Algorithm

In this section, the definition of the synchronous version of Louvain algorithm is proposed. First a high-level description of the algorithm is given, independent of programming languages and programming models.

1. Initially, each node is in its own community.
2. Update Σ_{tot}^{C}, which is the sum of the weights of the edges incident to nodes in community C.
3. All nodes in the graph exchange community membership information with their neighbours simultaneously.
4. All nodes in the graph simultaneously do the following: update their community membership with probability p and do nothing with probability $1 - p$. The new community is chosen such that based on their knowledge of this iteration, the gain of modularity is non-negative and is maximum.
5. If there exist nodes who decide that their current communities are not optimal for them, repeat step 2 to 4.
6. Merge nodes in the same community into one node and create a new graph. In this graph, the sum of the weights of the edges between any two communities in the original graph becomes the weight of the edges between the two corresponding nodes in the new graph. Repeat step 1 to 5. If all the nodes remain in their original community after step 5, the algorithm terminates.

One can see that the only difference between the Distributed Synchronous Louvain algorithm and original sequential Louvain algorithm is that in the Distributed Synchronous Louvain algorithm, nodes update their communities with a certain probability, while in the original sequential version of the Louvain algorithm, nodes are visited one by one and the node being visited will always go to the community which gives the highest increase in modularity.

In the following, the pseudocode of the Distributed Synchronous Louvain method for one level is presented. *Community* contains the community membership information of the nodes in the graph, G is the graph to be processed and *prob* is the update probability which indicates how likely a node is going to join the neighbouring community that gives the maximum increase in modularity:

INPUT: Graph $G = (V, E)$, update probability *prob*
OUTPUT: *Community*

1: **function** SYNCHRONOUS-LOUVAIN(*G*,*prob*)
2: $m \leftarrow total\ weights\ of\ graph\ G$
3: **for all** $v \in V$ **do**
4: $Community(v) \leftarrow IndexOf(v)$
5: $InBestCommunity(v) \leftarrow False$
6: Calculate k_v
7: **end for**
8: **while** there exist $v \in V$ such that $InBestCommunity(v)$=$False$ **do**
9: update Σ^C_{tot} based on graph G and $Community$
10: **parfor** $v \in V$ **do**
11: exchange community membership information with all n such that $n \in Neighbours(v)$, update $k^C_{i,in}$ accordingly.
12: $\alpha \leftarrow$ uniformly at random from interval $[0, 1]$
13: **if** $\alpha < prob$ **then**
14: $Q = max_{i \in Neighbours(v)} \{\frac{k^C_{i,in}}{2m} - \frac{\Sigma^C_{tot} k_i}{m}\}$
15: $LossOfModularity = \frac{k^{Current}_{v,in}}{2m} - \frac{\Sigma^{Current}_{tot} k_v}{m}$
16: **if** $Q > LossOfModularity$ **then**
17: $Community(v) = \arg\max_C \{\frac{k^C_{i,in}}{2m} - \frac{\Sigma^C_{tot} k_i}{m}\}$
18: $InBestCommunity(v) \leftarrow False$
19: **else**
20: $InBestCommunity(v) \leftarrow True$
21: **end if**
22: **end if**
23: **end parfor**
24: **end while**
25: **return** *Community*
26: **end function**

Notice that $LossOfModularity$ is computed using a formula similar to Equation 3.2. This term means the loss of modularity by removing the node v from its current community *Current* and making v an isolated node.

3.6 Difference between the Gossip Algorithm and the Distributed Synchronous Louvain Algorithm

Although the Distributed Synchronous Louvain Algorithm is inspired by the Gossip algorithm, they are actually quite different, in the following ways: In each iteration of the gossip algorithm, a node will share its belief with one of its neighbors with a fixed

probability, while in the Distributed Synchronous Louvain method, one node has to consider all its neighbours in order to decide which community to join. In fact, what happens in the Distributed Synchronous Louvain method is not exactly "gossip"–each node will share its knowledge with every neighbour. Therefore, these two algorithms are quite different, and we cannot directly apply the mechanism of gossip algorithms to prove the convergence of the Distributed Synchronous Louvain algorithm.

3.7 Summary

This chapter first discussed how the Distributed Synchronous Louvain method is inspired by the Gossip algorithm. It then presented a description and pseudocode of the Distributed Synchronous Louvain algorithm. In the next chapter a theoretical analysis of the convergence of the Distributed Synchronous Louvain method will be presented.

Chapter 4

Analysis of the Distributed Synchronous Louvain Algorithm

This chapter first presents the proof of convergence of the Distributed Synchronous Louvain algorithm. Then a weak estimate (in the chapter you will see why it is a "weak" result) of upper bound of the number of iterations it takes for this algorithm to converge is given.

Key Findings:

- The Distributed Synchronous Louvain method can converge in a finite number of steps.
- For a given graph, if it is possible to reach a state of convergence from any assignment of community membership in one step, then the expected number of iterations it takes for the Distributed Synchronous Louvain method to converge on this graph has an upper bound.

4.1 Idea to Prove Convergence

The hint for this proof is given by Julien Hendrickx, professor at Université Catholique de Louvain. The idea is to model the transition of community membership as a Markov chain. From the perspective of Louvain method, the state where none of the nodes will shift community correspond to absorbing state in a Markov chain. Therefore, proving convergence of the synchronous version of the Louvain method is equivalent to proving that the transition of community membership forms an absorbing Markov chain.

4.2 Proof of Convergence

Lemma 1: In each move (transfer a node from one community to another) of the sequential Louvain method, the gain of modularity is lower bounded by a positive constant.

Proof: This argument follows from the fact that the elements that are used to compute gain of modularity are positive integers, as a result, the positive gain of modularity (if it triggers the update of community) cannot be arbitrarily close to 0. Given a <partition, node> pair, the gain of modularity by moving a node to a neighboring community

is determined. The number of <partition,node> pairs is finite. Therefore there exists a positive constant that is the global minimum of the positive gain of modularity for all possible <partition, node> pairs, and the positive gain of modularity (which triggers the update of community) is lower bounded by this positive constant.

Lemma 2: Given any partition of the network, by using the update in sequential Louvain method (original version of Louvain algorithm), we can always reach a state where a local maximum is reached.

Proof: The range of modularity is between -1 and 1. By Lemma 1 it follows that the number of moves for an arbitrary partition to reach local maximum is finite. This is because the modularity of a network cannot be arbitrarily high, and the positive gain of modularity, given the topology of the network, is lower bounded by a positive constant.

The proof for convergence:

Theorem 1: The execution of the Distributed Synchronous Louvain algorithm will converge after a finite number of iterations, which means that after a finite number of iterations, every node in the graph will end up in its locally optimal community and none of them will do any update to the community they belong to.

Proof: By lemma 2, given any partition pi of the network, by running the sequential Louvain algorithm, we can always reach a local maximum for modularity by running a sequence of actions $a_1, a_2, ...a_n$, where each action a_i corresponds to moving one node n_i in a community to another community (the cases where the node under consideration does not take any move are ignored, n_i and n_j may be the same node, since a node may be considered several times in the algorithm). The local maximum we reach is an absorbing state in the Markov chain, where none of the nodes does any update.

Then, in the Distributed Synchronous Louvain method, we can start from the same partition pi. We can execute the same sequence of events $a_1, a_2, ...a_n$, such that in each step i, node i in the network that takes action a_i in sequential case still takes action a_i to update his community and the other nodes don't do anything. Following this sequence of events we can always reach a local maximum. In this way, we prove that it is possible to go from any state to at least one absorbing state in the Markov chain.

In the Markov chain there is at least one absorbing state, and it is possible to go from any state to at least one absorbing state in a finite number of steps, so the Markov chain is an absorbing Markov chain. In an absorbing Markov chain, the probability that the process will be absorbed is 1, which means convergence.

The proof is complete.

4.3 An Upper Bound of the Number of Iterations to Converge

In the previous section we were able to prove that the synchronous version of the Louvain method always converges in a finite number of steps. From a practical point of view, the next question to ask is "What is that finite number?". If the number of steps required for this algorithm to converge is too high, this algorithm is almost useless in practice. Therefore, if this algorithm can always terminate in a reasonable number of steps in practice, I hope to find an upper-bound for this number in theory.

The idea of modelling the transition of community membership as a Markov chain inspired me, and I continued on this idea. In the following I present the upper bound of the number of iterations it takes to converge in a special case.

The Markov chain describing the state transition of the Distributed Synchronous Louvain method is an absorbing Markov chain, as proved in the previous section.

Therefore, the transition matrix P (with t transient states and r absorbing states) of this Markov chain is as follows:

$P = \begin{bmatrix} Q & R \\ O & I_r \end{bmatrix}$

where I_r is the identity matrix, Q is a t by t matrix describing the probability of moving from one transient state to another transient state. R is a t by r matrix which describes the probability of moving from a transient state to an absorbing state.

The fundamental matrix of this transition matrix is:

$$N = \sum_{k=0}^{\infty} Q^k = (I_t - Q)^{-1}$$

Starting from transient state i, the expected number of steps before being absorbed is:

$$E_i = \sum_{j=1}^{t} N_{ij}$$

Notice that this is the expected number of iterations it takes for the Distributed Synchronous Louvain algorithm to reach its locally optimal state.

Recall that the infinity norm of matrix N is as follows:

$$\|N\|_\infty = \max_i \sum_{j=1}^{t} |N_{ij}|$$

Moreover, for all i, we have the following relation:

$$\sum_{j=1}^{t} N_{ij} \leq \sum_{j=1}^{t} |N_{ij}|$$

Therefore, we have:

$$E_i = \sum_{j=1}^{t} N_{ij} \leq \sum_{j=1}^{t} |N_{ij}| \leq \|N\|_\infty$$

Therefore, the infinity norm of the fundamental matrix is an upper bound of the expected number of iterations before reaching the absorbing state.

Moreover, if it is possible to reach an absorbing state from any transient state in one step, which means in every row of R there is at least one non-zero element, then $I_t - Q$ is a *strictly diagonal dominant matrix* (SDD matrix). In the following, I will prove this statement.

Lemma 1: If it is possible to reach an absorbing state from any transient state in one step, then $I_t - Q$ is a *strictly diagonal dominant matrix* (SDD matrix).

Proof: Let $L = I_t - Q$. Suppose the ith row of L is as follows:

$$L_{i1}, L_{i2}, ..., L_{ii}, ..., L_{it}$$

Then we have

$$L_{i1} = -Q_{i1}$$
$$L_{i2} = -Q_{i2}$$
$$.....$$
$$L_{ii} = 1 - Q_{ii}$$
$$.....$$
$$L_{it} = -Q_{it}$$

Since the entries in Q are probabilities of moving from a state i to another transient state, we have:

$$Q_{ii} + Q_{i2} + ... + Q_{it} \leq 1$$

and all entries in Q are non-negative.
Therefore we have

$$Q_{ii} + Q_{i2} + ... + Q_{i,i-1} + Q_{i,i+1} + ... + Q_{it} \leq 1 - Q_{ii}$$

Because Q_{ii},Q_{i2},...$Q_{i,i-1}$,$Q_{i,i+1}$,... Q_{it} and $1 - Q_{ii}$ are non-negative (all entries in Q are probabilities). We can add an absolute value sign, like this:

$$|Q_{ii}| + |Q_{i2}| + ... + |Q_{i,i-1}| + |Q_{i,i+1}| + ... + |Q_{it}| \leq |1 - Q_{ii}|$$

Which means:

$$\sum_{i \neq j} |Q_{ij}| \leq |1 - Q_{ii}|$$

Recall that we have:

$$|L_{i1}| = |Q_{i1}|$$
$$|L_{i2}| = |Q_{i2}|$$
$$.....$$
$$|L_{ii}| = |1 - Q_{ii}|$$
$$.....$$
$$|L_{it}| = |Q_{it}|$$

So we have the following:

$$\sum_{i \neq j} |L_{ij}| \leq |L_{ii}|$$

If it is possible to reach an absorbing state from any transient state in one step, then in each row of R there is at least one positive element. Because P is a transition matrix, each line of this matrix adds up to 1. Therefore, we have the following:

$$Q_{ii} + Q_{i2} + ... + Q_{it} < 1$$

which leads to

$$Q_{ii} + Q_{i2} + ... + Q_{i,i-1} + Q_{i,i+1} + ... + Q_{it} < 1 - Q_{ii}$$

Again, because all these Qs are probabilities, we can add an absolute value sign:

$$|Q_{ii}| + |Q_{i2}| + ... + |Q_{i,i-1}| + |Q_{i,i+1}| + ... + |Q_{it}| < |1 - Q_{ii}|$$

then we have

$$\sum_{i \neq j} |L_{ij}| < |L_{ii}|$$

which means L is a strictly diagonal dominant matrix, so $I_t - Q$ is a strictly diagonal dominant matrix.

The proof is complete.

Since we know that $I_t - Q$ is a strictly diagonal dominant matrix under certain condition, according to **Ahlberg-Nilson-Varah bound**[33], there is an upper bound for the infinity norm of its inverse:

$$\|N\|_\infty = \|L^{-1}\|_\infty = \|(I_t - Q)^{-1}\|_\infty \leq \frac{1}{\min_i\{|L_{ii}| - \sum_{i \neq j} |L_{ij}|\}}$$

This relation indicates that, if it is possible to reach an absorbing state from any transient state in one step, the expected number of iterations it takes for the algorithm to converge is upper-bounded by a constant.

However, this is just a weak estimate about the upper bound of the number of iterations to converge. The condition for this bound to hold is very strict and almost never happens in real life. Moreover, this is an upper bound for an average number of iterations to converge, not for the worst-case scenario. Even if this upper bound does hold, it is still not guaranteed that the actual number of iterations will always stay below this upper bound.

4.4 Summary

In this chapter the proof of convergence for the Distributed Synchronous Louvain method was given, which shows that the Distributed Synchronous Louvain method can converge in a finite number of iterations. Then, it was shown that under very specific conditions the expected number of iterations it takes for the Distributed Synchronous Louvain method to converge has an upper bound. Although in theory the Distributed Synchronous Louvain algorithm can converge in a finite number of steps, I am not sure about its performance in general. In the next chapter the performance characteristics of Synchronous Louvain method will be studied by a comprehensive performance testing.

List of Key Findings:

- The Distributed Synchronous Louvain method can converge in a finite number of steps.

- For a given graph, if it is possible to reach a state of convergence from any assignment of community membership in one step, then the expected number of iterations it takes for the Distributed Synchronous Louvain method to converge on this graph has an upper bound.

Chapter 5

Experimental Evaluation

This chapter studies the performance characteristics of the synchronous version of the Louvain community detection algorithm. First the research questions about the performance characteristics of the Distributed Synchronous Louvain method are proposed. Then the hypotheses for these research questions are proposed. Different options of measurements used in the benchmark testing are proposed based on the hypotheses. Using these measurements, the benchmark testing are carried out using the LFR benchmark[1]. Finally, based on the observations in the experiments, the hypotheses about the performance characteristics of the Distributed Synchronous Louvain method are evaluated.

5.1 Research Questions

In general, for the Distributed Synchronous Louvain method we are interested in two things: The quality of the partitions produced and the execution time of the algorithm. Since the Distributed Synchronous Louvain algorithm is based on the original sequential Louvain community detection method, for the quality of communities we have to compare both algorithms (synchronous and sequential version). For the execution time of the algorithm we have to study the execution time of the Distributed Synchronous Louvain method for each iteration and the number of iterations for the algorithm to converge. The research questions for the performance characteristics of the Distributed Synchronous Louvain method are proposed as follows:

1. Is the quality of the partitions obtained by the Distributed Synchronous Louvain method better or worse than those produced by the sequential Louvain method?

2. The Distributed Synchronous Louvain algorithm has a single parameter-the probability to join the neighbouring community (or stay in its own community) that gives a maximum increase in modularity. For the Distributed Louvain method, how does this parameter influence the quality of the partitions obtained and the number of iterations it takes for the algorithm to converge?

[1] https://sites.google.com/site/santofortunato/inthepress2

3. How does the mixing parameter influence the number of iterations it takes for the Distributed Synchronous Louvain method to converge and the quality of the partitions obtained by the Distributed Synchronous Louvain method?

4. How does the size of the graphs influence the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge on the graphs?

5. How does the number of machines influence the execution time of the Distributed Synchronous Louvain algorithm for each iteration?

In the following, when speaking of "update probability", I refer to the free parameter of the Distributed Synchronous Louvain method. Specifically, I mean the probability for a node to join the neighbouring community (or stay in its own community) that gives a maximum increase in modularity.

5.2 Hypotheses

The following hypotheses are proposed based on the research questions:

1. **Hypothesis 1:** The quality of the partitions generated by the Distributed Synchronous Louvain method is similar to those produced by the original sequential Louvain algorithm.

2. **Hypothesis 2:** The higher the update probability (but smaller than 1) , the better the quality of the partitions and the lower the smaller the number of iterations for the algorithm to converge.

3. **Hypothesis 3:** The higher the mixing parameter of a graph, the higher the number of iterations to converge and the poorer the quality of the partitions obtained.

4. **Hypothesis 4:** As the size of graphs increases, we will see a slow increase in the number of iterations for the Distributed Synchronous Louvain method to converge on these graphs.

5. **Hypothesis 5:** With an increasing number of machines, the execution time of the Distributed Synchronous Louvain algorithm for each iteration is significantly reduced.

5.3 Implementation in Apache Spark

This algorithm is implemented in Apache Spark GraphX[2]. In the following, for the meaning of mathematical notations please refer to **Equation 3.1** in **Section 4**. The update of Σ_{tot}^{C} is calculated by using the *map* operation to emit a (community, degree per node) tuple for each node in the graph, and then applying *reduceByKey* to the resulting RDD. The *mapReduceTriplet* operation is used for the neighbouring nodes in the graph to exchange information. This is usually needed when nodes in the graph have

[2]https://spark.apache.org/graphx/

to collect information from their neighbours, such as the calculation of $k^C_{i,in}$. Specifically, $k^C_{i,in}$ is calculated in this way: nodes exchange their membership of communities and the weights of the edges. Then each node adds up the weights of the edges from each community.

In the next section I will motivate my choice of measurements to validate these hypotheses.

5.4 Measurement

5.4.1 Execution Time of the Algorithm

Since we are studying the algorithm it self, the time for initializing the cluster, loading the test graphs in-memory, and verifying results is not included in the measurements. The Distributed Synchronous Louvain algorithm is an iterative algorithm, so the measurement of the execution time can be broken down into two measurements:

- The number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge
- The execution time per iteration.

The number of iterations it takes to converge is determined by the algorithm itself, not by the machine settings. Therefore, to study the change in this quantity, we vary the size of graphs, not the number of machines. To study the change of execution time per iteration, we will vary the number of machines to reveal the scalability of the algorithm.

5.4.2 Quality of Produced Communities

Many different cost functions have been proposed to measure the quality of the partitions produced by community detection algorithms, such as significance[31] and modularity. However, not all of them are appropriate for our experiments. This is for two reasons: First they might be biased in one way or another. Second, most of them are derived only from the communities produced by the community detection algorithms, without using the ground truth of the LFR benchmark.

From all the available measurements, two measurements were chosen for this benchmark testing: Modularity and Normalized Mutual Information. Modularity is used because the core idea of Louvain algorithm (both the sequential and the synchronous version) is to optimize modularity in a greedy way. By using this measurement, we are able to see how effective a method is in terms of modularity optimization. Normalized Mutual Information (NMI) is used because it utilizes the ground truth produced by the LFR benchmark. Intuitively, it measures the similarity between the ground truth and the partition generated by a community detection method. If the partitions found and the ground truth are completely the same, NMI takes its maximum value of 1. If the partition found by the algorithm is totally irrelevant of the ground truth partition, NMI takes its smallest value of 0. NMI is an independent indicator that does not depend on any assumptions of a particular community detection method.

The average degree	20
The maximum degree of a node	50
Minimum community size	10
Maximum community size	50
Minus exponent for the degree sequence	2
Minus exponent for the community size distribution	3

Table 5.1: Configuration of Benchmark Graphs

Therefore, this measurement is considered to be an objective measurement that does not favor any specific community detection algorithm.

5.5 Experiment Setting

For the benchmark the LFR[3] benchmark is used. There are several versions of the LFR benchmark. In this benchmark testing, the version that generates undirected binary network (package 1) is used. The network size is 1000 nodes and 5000 nodes or larger. The two sizes 1000 nodes and 5000 nodes are chosen because they are used in many previous papers, such as [31], [17], [18] and [16]. If the same sizes are used for this experiment, the experiment results can be compared with previous work, producing more useful insights.

For the benchmark testing we mainly control graph size (the number of nodes in the graph) and the mixing parameter. The mixing parameter controls how clear the hidden communities are in the benchmark graphs. If the mixing parameter is close to 0, the community structure is quite clear and can be easily revealed by a community detection algorithm. If the mixing parameter is close to 1, the community structure is quite messy.

The experiment setting is as follows: For both graph sizes, vary the mixing parameter from 0.05 to 0.9 with steps of 0.05. For each mixing parameter, generate 1 graph. In total 18 graphs are generated for each network size. The configuration of benchmark graphs for our experiments is shown in **Table5.1**.

For other parameters the default values are used.

This implementation[4] is used as the implementation for the sequential Louvain method in the experiment. The updated version is used.

The NMI (Normalized Mutual Information) is defined in [7]. This measure is based on defining a confusion matrix N, where rows correspond to the "real" communities (in our case, the ground truth communities produced by the LFR benchmark) and the columns correspond to communities found by the community detection algorithm being tested. The elements of N, N_{ij} is the number of nodes in the real community i that appear in detected community j. Normalized Mutual Information is then defined as:

[3] https://sites.google.com/site/santofortunato/inthepress2

[4] https://sites.google.com/site/findcommunities/

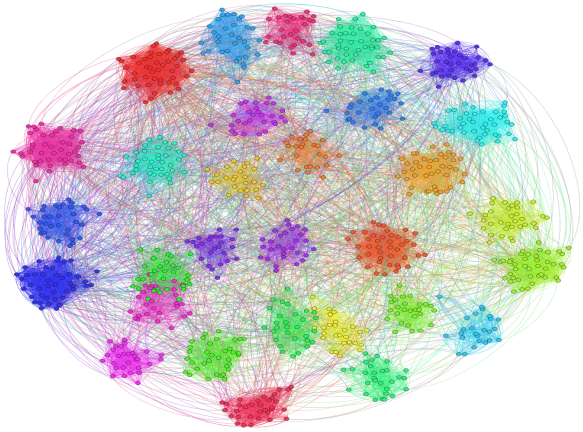

Figure 5.1: Community visualization for graph of 1000 nodes

$$I(A,B) = \frac{-2\sum_{i=1}^{c_A}\sum_{j=1}^{c_B} N_{ij} log(\frac{N_{ij}N}{N_{i.}N_{.j}})}{\sum_{i=1}^{c_A} N_{i.} log(\frac{N_{i.}}{N}) + \sum_{j=1}^{c_B} N_{.j} log(\frac{N_{.j}}{N})}$$

where c_A denotes the number of real communities and c_B denotes the number of detected communities, $N_{i.}$ denotes the sum of row i of matrix N_{ij} and $N_{.j}$ denotes the sum of column j.

5.6 Execution Result on Two Graphs

Key Finding:

- In some cases the Distributed Synchronous Louvain algorithm can converge in a small number of iterations.

To have an idea about the behaviour of the Distributed Synchronous Louvain Algorithm, I decided to first try this algorithm on two small graphs. First my implementation of the Distributed Synchronous Louvain method was tested on a graph with one thousand nodes. This graph was generated by the LFR benchmark[5], package 1, which generates undirected and unweighted networks. The configuration of this graph is shown in **Table5.1**.

The visualization of result is shown in **Figure 5.1**. It seems that the Distributed Synchronous Louvain method can converge in a fairly small number of iterations (with only 11 iterations). What happens if the size of the graph increases? Next the number of nodes was increased from 1000 to 5000, while the other configurations remain the same (the visualization of communities is shown in **Figure 5.2**). Still the number of iterations it takes to converge is small (with only 13 iterations). Furthermore, it is observed that when the size of the graph increases, the increase in the number of iterations is very small. In general, the execution result indicates that there are cases when

[5]https://sites.google.com/site/santofortunato/inthepress2

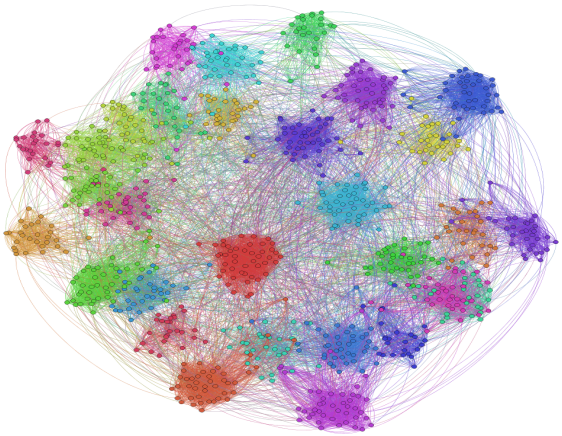

Figure 5.2: Community visualization for graph of 5000 nodes

the Distributed Synchronous Louvain method can converge in a very small number of iterations.

So far only two graphs have been tested. A comprehensive experimental evaluation will be presented to reveal the performance characteristics of the Distributed Synchronous Louvain algorithm.

5.7 Comparison of the Sequential Louvain Algorithm and the Distributed Synchronous Louvain Algorithm

This section compares the quality of the communities produced by the sequential Louvain algorithm and the Distributed Synchronous Louvain algorithm. The quality of communities is measured both in modularity and in the Normalized Mutual Information. In general, this section tries to answer the first research question.

Key findings:

- Both the original sequential Louvain method and the Distributed Synchronous Louvain algorithm produce communities with similar quality, both in terms of modularity and in terms of Normalized Mutual Information.

5.7.1 Quality of Communities in Terms of Modularity

This section compares the quality of communities produced by the sequential Louvain algorithm and the Distributed Synchronous Louvain Algorithm in terms of modularity.

Key findings:

- Both the original sequential Louvain method and the Distributed Synchronous Louvain algorithm produce communities with similar modularity.

The experiment is carried out in this way: vary the mixing parameter from 0.05 to 0.9 with steps of 0.05. Generate one graph for each mixing parameter. The properties of the benchmark graphs used are listed in **Table 5.1**. For each graph, run both algorithms

(the sequential Louvain method and the Distributed Synchronous Louvain Algorithm) 10 times. Take the average value for measurements. For the Distributed Synchronous Louvain Algorithm, the update probability is 0.6.

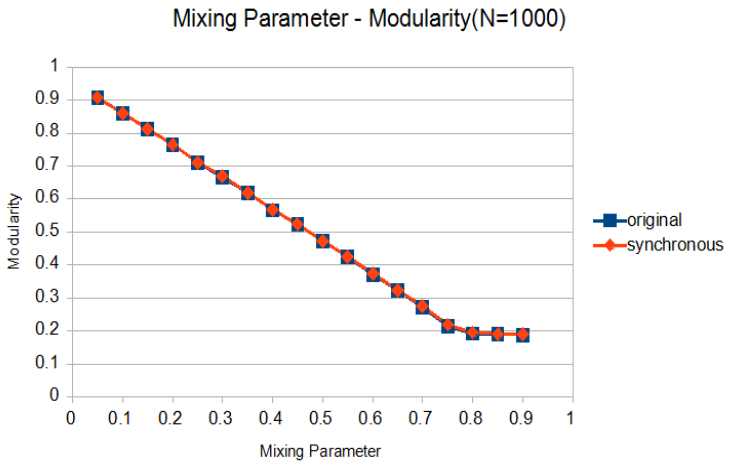

Figure 5.3: Modularity Comparison for Graphs with 1000 Nodes

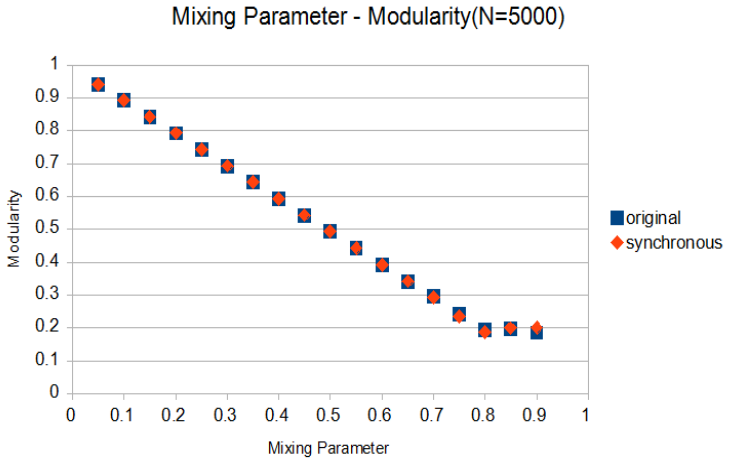

Figure 5.4: Modularity Comparison for Graphs with 5000 Nodes

The results are shown in **Figure 5.3** and **Figure 5.4**. The horizontal axis shows the mixing parameter and the vertical axis shows the modularity of the communities. In both figures the modularity of communities that both algorithms produce are plotted. The measurement value of modularity is the average value of 10 runs.

When the size of the graph is 1000 nodes, we can see that the maximum modularity that can be achieved by both algorithms is almost identical, as is shown in **Figure 5.3**. When the size of the graph is 5000 nodes, we can observe a similar phenomenon, except that when the mixing parameter is 0.9, the modularity obtained by the Distributed

Synchronous Louvain method is slightly better. Since it happens only once this might not be a general phenomenon.

From the experiment results, we can see that in terms of modularity optimization, the synchronous and sequential Louvain methods are quite similar.

An interesting observation is that in both **Figure 5.3** and **Figure 5.4**, we can observe a linear relation between the mixing parameter and the corresponding modularity obtained by both the sequential and the synchronous Louvain method. More experiments are needed to see if this is a general phenomenon or a random effect. If this is a general phenomenon, it will become a very interesting research question to study, since there is no obvious connection between the mixing parameter and the maximum modularity obtained by the Louvain method.

5.7.2 Quality of Communities in Terms of Normalized Mutual Information

This section compares the quality of communities produced by the sequential Louvain algorithm and the Distributed Synchronous Louvain Algorithm in terms of Normalized Mutual Information.

Key findings:

- Both the original sequential Louvain method and the Distributed Synchronous Louvain algorithm produce communities with similar value in Normalized Mutual Information.
- When the mixing parameter is smaller than 0.5, the Distributed Synchronous Louvain method can generate communities with very high quality, regardless of the actual value of the mixing parameter.

In this experiment, the setting was the same as the previous section: Generate one graph for each mixing parameter (from 0.05 to 0.9 with steps of 0.05). The properties of the benchmark graphs used are listed in **Table 5.1**. For each graph, run both algorithms (the sequential Louvain method and the Distributed Synchronous Louvain Algorithm) 10 times. Take the average value for measurements. For the Distributed Synchronous Louvain Algorithm, the update probability is 0.6.

The results are shown in **Figure 5.5** and **Figure 5.6**. The horizontal axis shows the mixing parameter and the vertical axis shows the Normalized Mutual Information value of the corresponding mixing parameter. The value of Normalized Mutual Information for a mixing parameter is the average value of 10 runs on the benchmark graphs with that mixing parameter.

The mixing parameter of 0.5 separates graphs with a clear community structure and graphs without. From **Figure 5.5** and **Figure 5.6**, we can see that when the mixing parameter is smaller than 0.5, both algorithms can generate partitions of very high quality, almost identical to the ground truth. When the mixing parameter is larger than 0.5, the quality of partitions decreases very rapidly as the mixing parameter increases.

We can also see that as the mixing parameter increases and the mixing parameter becomes larger than 0.5, the quality of partitions produced by both community detection methods decreases. This is because as the mixing parameter increases, the community structures in the benchmark graphs become less and less obvious. As a

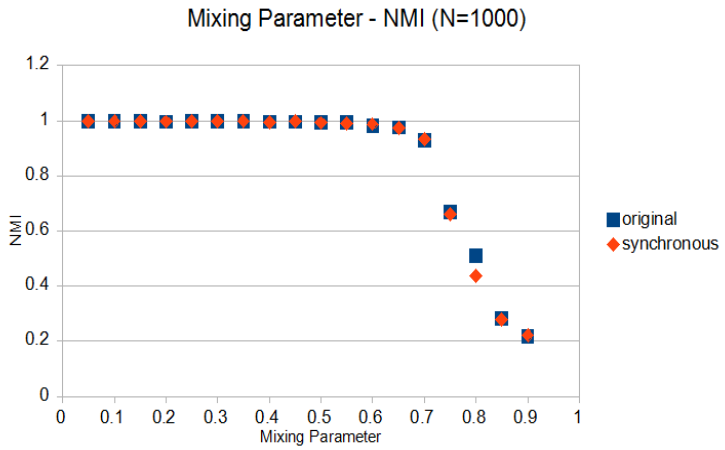

Figure 5.5: NMI Comparison for Graphs with 1000 Nodes

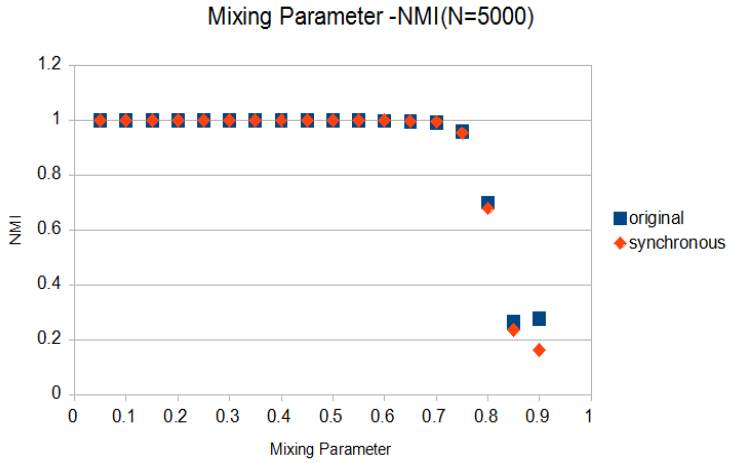

Figure 5.6: NMI Comparison for Graphs with 5000 Nodes

result, it becomes more and more difficult to find communities in the graphs, and the detected communities deviate more and more from the ground truth.

Moreover, we can see that in terms of NMI, both algorithms again achieve very similar results. The only difference is that for the graph of 1000 nodes, the Distributed Synchronous Louvain achieves worse results when the mixing parameter is 0.8, and for the graph of 5000 nodes, the Distributed Synchronous Louvain method achieves worse results when the mixing parameter is 0.9. The difference is not significant, so this might be a random phenomenon.

5.8 The Influence of the Update Probability

This section studies the influence of the update probability on the quality of communities produced by the Distributed Synchronous Louvain algorithm and the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge. In general, this section deals with the second research question.

Key findings

- The update probability does not have a significant influence on the quality of communities produced by the Distributed Synchronous Louvain method.
- When the update probability is smaller than 0.5, a larger value of the update probability leads to a smaller number of iterations for the Distributed Synchronous Louvain algorithm to converge.
- When the update probability is larger than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is roughly the same, regardless of the value of the update probability.
- To reduce the number of iterations for the algorithm to converge, making the update probability as close as possible to 1 does not help. We can achieve a small number of iterations with a more or less neutral update probability (around 0.5 or 0.6).

5.8.1 The Influence of the Update Probability on the Quality of Communities Produced

This section studies the influence of the update probability on the quality of communities produced by the Distributed Synchronous Louvain algorithm.

Key findings

- The update probability does not have a significant influence on the quality of communities produced by the Distributed Synchronous Louvain method.

In this experiment, the graph with mixing parameter 0.5 is chosen, with configurations listed in **Table 5.1**. This graph is chosen because the mixing parameter of 0.5 is a threshold to separate graphs with a clear community structure and graphs whose community structure is not very clear. The update probability varies from 0.1 to 0.9 with steps of 0.1. The Normalized Mutual Information of the communities produced is the average value of 10 runs. The same experiments are carried out for both graph sizes, 1000 and 5000.

The results are listed in **Table 5.2** and **Table 5.3**. We can see that the update probability does not have a significant influence on the quality of communities produced by the Distributed Synchronous Louvain method in terms of Normalized Mutual Information, as can be seen from **Table 5.2** and **Table 5.3**.

Update Probability	NMI
0.1	0.9943
0.2	0.9919
0.3	0.9933
0.4	0.9946
0.5	0.9929
0.6	0.9926
0.7	0.9933
0.8	0.9963
0.9	0.9968

Table 5.2: Update Probability vs. NMI for N=1000

Update Probability	NMI
0.1	0.9991
0.2	0.9995
0.3	0.9993
0.4	0.9990
0.5	0.9994
0.6	0.9992
0.7	0.9993
0.8	0.9992
0.9	0.9990

Table 5.3: Update Probability vs. NMI for N=5000

5.8.2 Influence of the Update Probability on the Number of Iterations to Converge

This section studies the influence of update probability on the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge.

Key findings

- When the update probability is smaller than 0.5, a larger value of the update probability leads to a smaller number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge.
- When the update probability is larger than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is roughly the same, regardless of the value of the update probability.
- To reduce the number of iterations to converge, making the update probability as close as possible to 1 does not help. We can achieve a similar number of iterations with a more or less neutral update probability (around 0.5 or 0.6).

In this experiment, the setting is the same as in the previous section: the graph with a mixing parameter of 0.5 is chosen, with configurations listed in **Table 5.1**. The update probability varies from 0.1 to 0.9 with steps of 0.1. The number of iterations

it takes to converge is the average value of 10 runs. The same experiments are carried out for both graph sizes, 1000 and 5000.

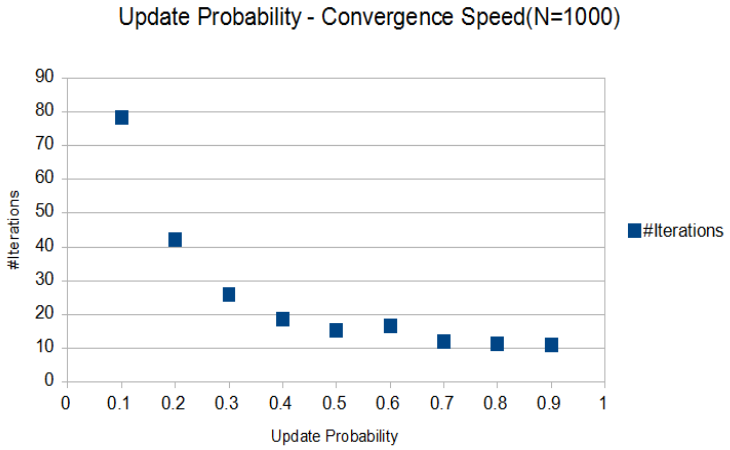

Figure 5.7: Number of Iterations to Converge, With Different Update Probability, N=1000

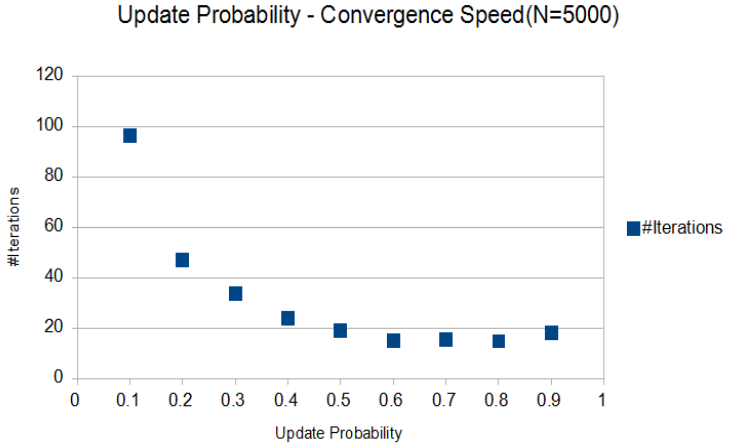

Figure 5.8: Number of Iterations to Converge, With Different Update Probability, N=5000

The results are shown in **Figure 5.7** and **Figure 5.8**. The horizontal axis shows the update probability and the vertical axis shows the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge.

When the size of the graph is 1000 and the update probability is 0.1, the number of iterations it takes to converge is very large, around 80 iterations. The number of iterations it takes to converge drops drastically as the update probability increases. This is expected, because the higher the update probability, the more likely a node will join the community that leads to maximum increase in modularity. As a result the quality

of the community improves faster. However, it can also be observed that the minimum number of iterations is reached around the probability of 0.5 and 0.6. If the update probability is larger than that, the number of iterations remains around a stable value. For size of 5000 a similar phenomenon can be observed. This observation indicates that to reduce the number of iterations to converge, making the update probability as close as possible to 1 does not help. We can achieve a similar number of iterations with a more or less neutral update probability (around 0.5 or 0.6).

An interesting observation is that in both **Figure 5.7** and **Figure 5.8**, when the update probability is smaller than 0.5, the products of the update probabilities and the corresponding number of iterations to converge are roughly the same. A rough guess is that in general we expect an inversely-proportional relationship between the update probability and the corresponding number of iterations to converge when the update probability is smaller than 0.5.

5.9 The Influence of the Mixing Parameter

This section studies the influence of the mixing parameter on the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge and the quality of communities produced by the algorithm. In general, this section deals with the third research question.

Key findings:

- When the mixing parameter is smaller than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is roughly the same.
- When the mixing parameter is smaller than 0.5, the quality of communities produced by the Distributed Synchronous Louvain algorithm is roughly the same.
- When the mixing parameter is larger than 0.5, as the mixing parameter increases, the number of iterations for the Distributed Synchronous Louvain algorithm to converge will increase and the quality of communities produced by the algorithm will decrease.

In this experiment, the same settings are used as the first research question: Vary the mixing parameter from 0.05 to 0.9 with steps of 0.05. Generate one graph for each mixing parameter. The properties of the benchmark graphs used are listed in **Table 5.1**. For each graph, run the Distributed Synchronous Louvain Algorithm 10 times. The update probability is 0.6, and the average number of iterations it takes to converge is collected.

The results are shown in **Figure 5.9** and **Figure 5.10**. The horizontal axis shows the mixing parameter and the vertical axis shows the number of iterations it takes for the algorithm to converge. The number of iterations to converge is the average value of 10 runs.

In the two figures, we can observe that if the mixing parameter is not larger than 0.5, the number of iterations is roughly the same. When the mixing parameter is larger than 0.5, the number of iterations to converge increases as the mixing parameter increases. Moreover, the increase is rapid.

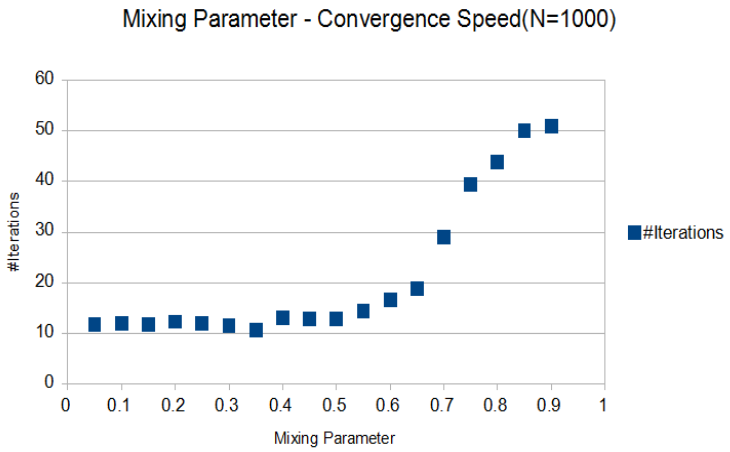

Figure 5.9: Number of Iterations to Converge, With Different Mixing Parameter, N=1000

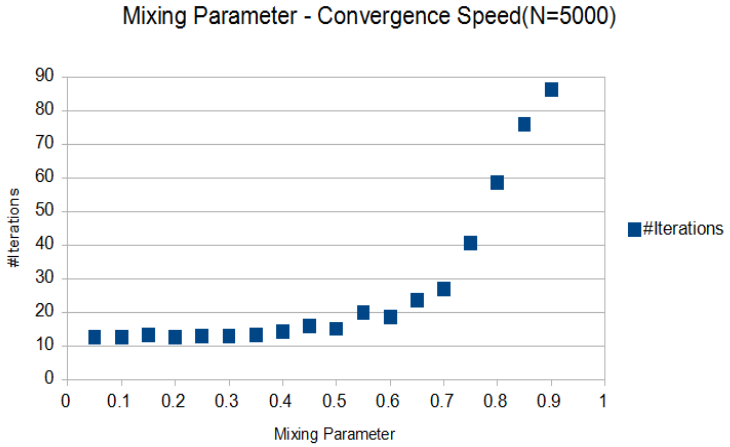

Figure 5.10: Number of Iterations to Converge, With Different Mixing Parameter, N=1000

If we compare this observation with what we observed in the experiments for research question one, which measured Normalized Mutual Information, we can see the importance of mixing parameter 0.5: From the perspective of the LFR benchmark, the mixing parameter of 0.5 separates graphs with clear community structure and graphs without. From the perspective of the Distributed Synchronous Louvain method, graphs with a mixing parameter less than 0.5 are "equally easy". This can be explained by three observations: First, from **Figure 5.5** and **Figure 5.6**, we can observe that when the mixing parameter is smaller than 0.5, the communities generated by the Distributed Synchronous Louvain method are almost always identical to the ground truth, no matter what the mixing parameter is. Second, from **Figure 5.9** and **Figure 5.10**, we can observe that when the mixing parameter is smaller than 0.5, it takes a similar number

of iterations for the Distributed Synchronous Louvain method to reach a partition that is almost perfect, no matter what the mixing parameter is. Third, from **Figure 5.5**, **Figure 5.6**, **Figure 5.9** and **Figure 5.10**, one can see that when the mixing parameter is larger than 0.5, as the mixing parameter increases, the quality of the partitions obtained decreases and the number of iterations to converge increases faster and faster.

5.10 The Runtime of the Distributed Synchronous Louvain Method on Larger Graphs and With More Machines

In this section, we evaluate how the actual runtime of the Distributed Synchronous Louvain Algorithm will change for different graph sizes and different number of machines.

Since the Distributed Synchronous Louvain Algorithm is an iterative algorithm, there are two main factors that will influence the actual runtime of the algorithm:

- Number of iterations to converge
- Execution time per iteration

The number of iterations to converge mainly depends on the algorithm itself and the size of graphs being processed by the algorithm. Here we are interested in how the size of the input graph influences the number of iterations it takes for the Distributed Synchronous Louvain method to converge.

The execution time per iteration is influenced by the machine settings and how the algorithm is implemented (some operations may be more expensive than the others, while they do the same thing). Since the Distributed Synchronous Louvain Algorithm is a distributed algorithm, we are interested in finding out how the number of machines influences the execution time for each iteration.

Key Findings:

- Assume that all the other properties are the same. The number of iterations it takes for the algorithm to converge grows linearly as the number of vertices of a graph grows exponentially.
- The Spark implementation of the Distributed Synchronous Louvain algorithm shows good scalability with an increasing number of machines.
- It is in general not a good idea to run the Spark program with a small number of workers because the cost of distributing the computation is too high for a small number of workers (One worker node in the Spark cluster is one machine).

5.10.1 The Influence of the Graph Size on the Number of Iterations to Converge

In this section we look at how the size of the input graph influences the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge.

Key Finding:

- Assume that all the other properties are the same. The number of iterations it takes for the algorithm to converge grows linearly as the number of vertices of a graph grows exponentially.

The test graphs are generated using the LFR benchmark. In total 4 graphs are generated, with 1000 nodes, 5000 nodes, 10,000 nodes and 100,000 nodes respectively. They are different only in the number of edges and vertices. For all the other properties they are the same (as listed in **Table 5.1**). The Distributed Synchronous Louvain Algorithm is run on these graphs. For each graph, the number of iterations to converge is the average value of 5 runs.

The results are shown in **Figure 5.11**. The horizontal axis shows the number of vertices of the graphs and the vertical axis shows the number of iterations it takes for the Distributed Synchronous Louvain method to converge on the graph.

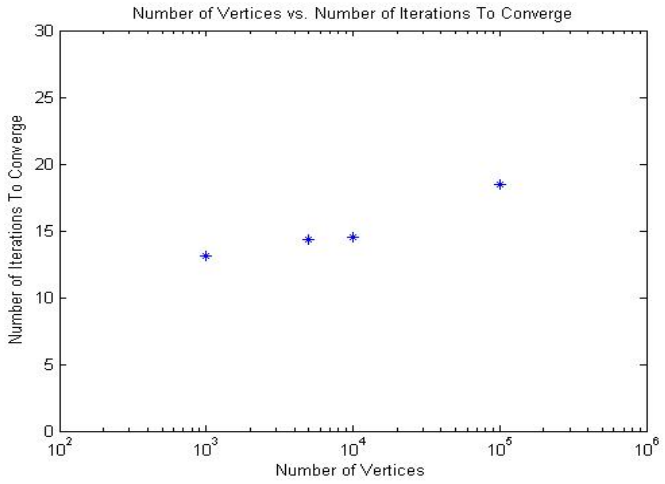

Figure 5.11: Number of Vertices Versus Number of Iterations to Converge

It can be seen from **Figure 5.11** that as the number of vertices of the graphs grows exponentially and all the other properties are the same, the number of iterations it takes for the algorithm to converge grows linearly. We can fit a straight line to the observations in **Figure 5.11**, which is shown in **Figure 5.12**. The straight line in this figure is the line we can fit to the four observation points.

Therefore, we can suggest the following hypothesis:

Assume that all the other properties are the same. The number of iterations it takes for the algorithm to converge grows linearly as the number of vertices of a graph grows exponentially.

Two experiments have been carried out to verify this hypothesis.

Experiment 1:

Suppose this hypothesis is true. Then, if we generate a graph with the LFR benchmark with the same configuration as is listed in **Table 5.1**, except that the number of nodes increases to 1 million, it should take the algorithm about 21 iterations to converge on this graph, which is shown in **Figure 5.12**.

So I carry out this experiment. Such a graph was generated, with the configuration listed in **Table 5.1** and 1 million nodes. The algorithm was run on this graph for 5

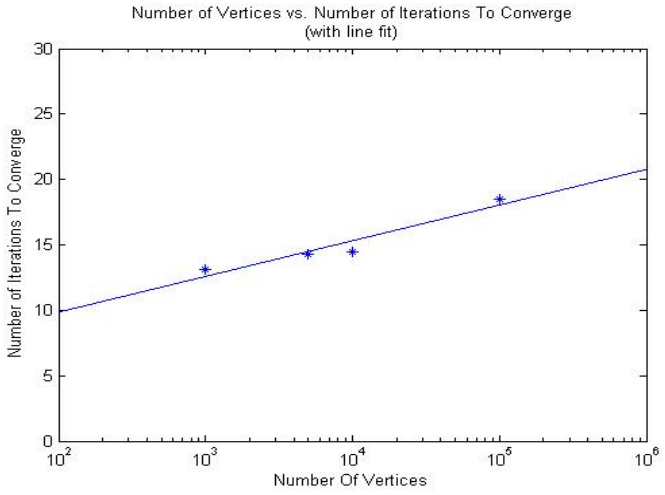

Figure 5.12: Number of Vertices Versus Number of Iterations to Converge With Fitted Straight Line

times and the average number of iterations to converge was 20.8, which is very close to the estimated number of iterations derived from the hypothesis.

Experiment 2:

It was shown in the previous sections that if we only vary the mixing parameter and keep all the other configurations the same for the LFR benchmark, the number of iterations to converge will be similar as long as the mixing parameter is smaller than 0.5. We have derived the hypothesis from the graphs with mixing parameter 0.4. Suppose we change this value to 0.2, the number of iterations to converge should be similar, and the observations points should be close to the straight line we have derived from the graphs with the mixing parameter 0.4.

In this experiment, four graphs are generated again with the four sizes, 1000, 5000, 10,000 and 100,000 nodes. All the configurations are the same as in **Table 5.1**, except that the mixing parameter is now 0.2. Again we run the algorithm on these graphs and collect the average number of iterations to converge over 5 runs.

The result is shown in **Figure 5.13**. The horizontal axis shows the number of vertices and the vertical axis shows the number of iterations for the algorithm to converge. The four data points are obtained from the execution results of graphs with mixing parameter 0.2 and the blue straight line is fitted to the execution result of graphs with mixing parameter 0.4. One can see that the data points and the straight line are very close, which indicates that the hypothesis is correct. If the mixing parameter is different, but all the other configurations are the same, the number of iterations to converge is similar.

5.10.2 Relation between the Number of Machines and the Execution Time Per Iteration

In this section we try to understand how the number of workers in a Spark cluster influences the execution time of the Spark implementation of the Distributed Synchronous

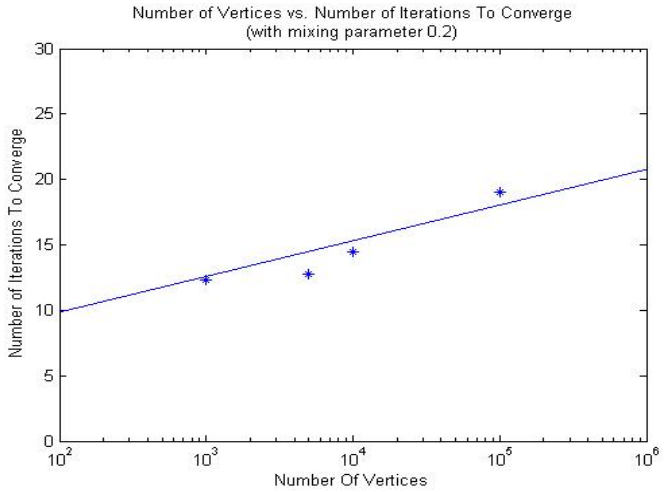

Figure 5.13: Number of Vertices Versus Number of Iterations to Converge With Fitted Straight Line

Louvain Algorithm for each iteration, and how much speedup can we obtain by distributing the execution of the Louvain community detection algorithm.

Key Findings:

- The Spark implementation of the Distributed Synchronous Louvain algorithm shows good scalability with an increasing number of machines.
- It is in general not a good idea to run the Spark program with a small number of workers because the cost of distributing the computation is too much for a small number of workers.

This experiment has been performed on the Dutch national e-infrastructure with support from the SURF SARA cooperative. To conduct the experiment, a graph with 1 million nodes is generated with the configurations listed in **Table 5.1**. This graph has 5007712 edges in total. The execution of the algorithm is carried out on a Spark cluster provided by SURF SARA. The configuration of each worker in the cluster is shown in **Table 5.4**. Vary the number of workers in this way: 1, 2, 4, 8 and 16. The execution time per iteration is shown in **Figure 5.14** and the speedup is shown in **Figure 5.15**.

CPU	AMD Opteron
Operating System	Linux CentOS 6
Executor Memory	2.1G
Number of cores used per worker	4
Storage	4 SATA Disks with 2 to 4 TB storage each
Hadoop Distribution	Hortonworks HDP 2.2
Spark Distribution	Spark 1.4.1

Table 5.4: Configuration per Node

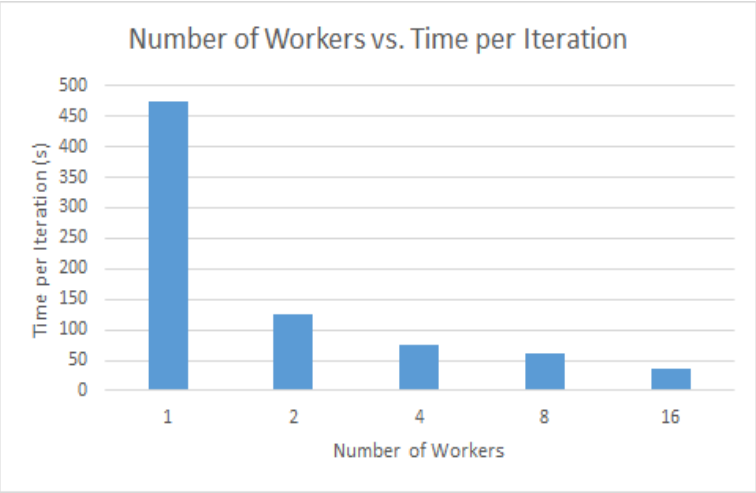

Figure 5.14: Number of Workers Versus Number of Execution Time per Iteration

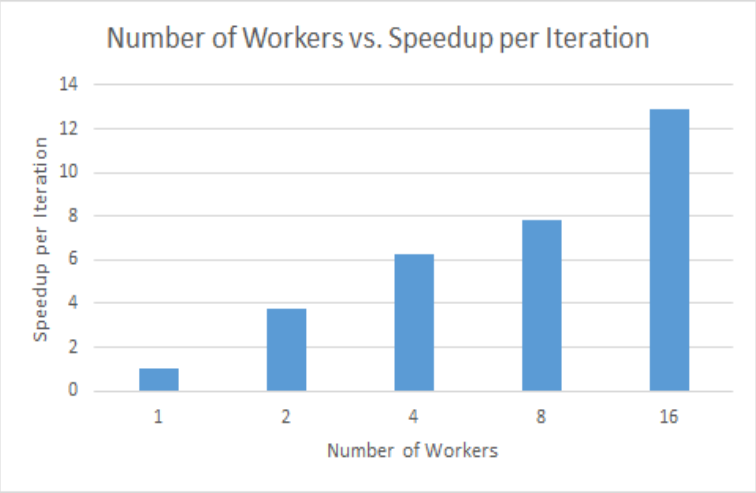

Figure 5.15: Number of Workers Versus Number of Speedup per Iteration

In **Figure 5.14**, the horizontal axis shows the number of workers and the vertical axis shows the execution time per iteration. In **Figure 5.15** the horizontal axis shows the number of workers and the vertical axis shows the corresponding speedup. The speedup is calculated with formula $\frac{T_1}{T_n}$, where T_1 means the execution time per iteration for one worker node, and T_n means the execution time per iteration for n worker nodes. The **Figure 5.16** shows the same information as **Figure 5.15**, except that in **Figure 5.16** there is a dotted line indicating the ideal linear speedup.

One can see from **Figure 5.14** that the Spark implementation of the Distributed Synchronous Louvain Algorithm is scalable with an increasing number of machines. As a result, we can observe an increasing speedup in **Figure 5.15** when the number of workers increases. What is interesting here is the speedup shown in **Figure 5.15** when there are 2 and 4 workers. Ideally, when we double the number of machines, we would expect a 2X and 4X speedup respectively. However, what we observe here

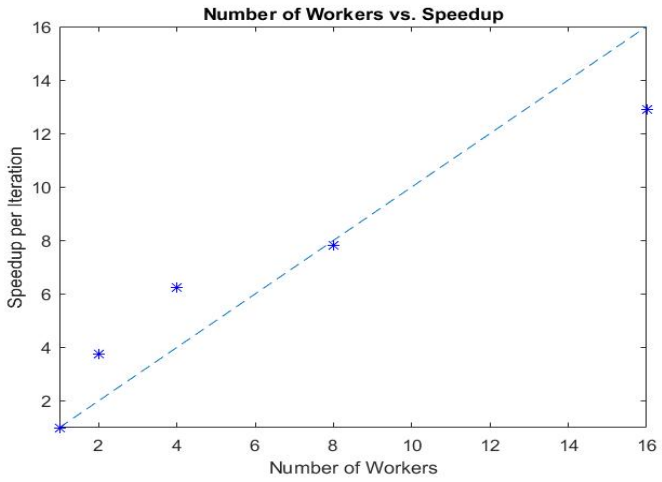

Figure 5.16: Number of Workers Versus Number of Speedup per Iteration

in **Figure 5.15** is that when there are 2 workers, the speedup is much higher than 2X. When there are 4 workers, the speedup reaches 6X. This can be seen more clearly from **Figure 5.16**. The possible cause is that the execution of Spark with a single worker introduces too much overhead by distributing the computation in a single machine, which significantly increases the execution time. With more machines, this overhead is largely reduced and as a result we can observe a much lower execution time for each iteration in **Figure 5.14** and a much higher speedup in **Figure 5.15**.

5.11 Case Study on Real-World Datasets

In this section the Distributed Synchronous Louvain method is applied to a real-world dataset to verify our findings so far. Before we proceed, let us summarize what we know about the Distributed Synchronous Louvain algorithm so far.

5.11.1 Summary of Findings of the Distributed Synchronous Louvain Method

Comparison of the sequential Louvain method and the Distributed Synchronous Louvain method:

- Both the original sequential Louvain method and the Distributed Synchronous Louvain algorithm produce communities with similar quality, both in terms of modularity and in terms of Normalized Mutual Information.

About update probability:

- When the update probability is smaller than 0.5, a larger value for the update probability leads to a smaller number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge.

- When the update probability is larger than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is more or less the same, regardless of the value of the update probability.
- To reduce the number of iterations for the Distributed Synchronous Louvain algorithm to converge, making the update probability as close as possible to 1 does not help. We can achieve a small number of iterations with a more or less neutral update probability (around 0.5 or 0.6).
- The update probability does not have a significant influence on the quality of communities produced by the Distributed Synchronous Louvain method.

About the mixing parameter:

- When the mixing parameter is smaller than 0.5, the quality of communities produced by the Distributed Synchronous Louvain algorithm is more or less the same, regardless of the actual value of the mixing parameter.
- When the mixing parameter is smaller than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is more or less the same, regardless of the actual value of the mixing parameter.
- When the mixing parameter is larger than 0.5, as the mixing parameter increases, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge increases and the quality of communities produced decreases.

About runtime:

- Assume that all the other properties are the same. The number of iterations it takes for the algorithm to converge grows linearly as the number of vertices of a graph grows exponentially.
- The Spark implementation of Distributed Synchronous Louvain algorithm shows good scalability with an increasing number of machines.
- It is in general not a good idea to run the Spark program with a small number of workers because the cost of distributing the computation is too much for a small number of workers.

5.11.2 Introduction of Datasets

In this section the Distributed Synchronous Louvain algorithm is applied to two real world data sets from SNAP[6]-*Amazon* and *DBLP*. These two data sets are used because they are similar in sizes (both in terms of number of nodes and number of edges). *Amazon* data set has 334863 vertices and 925872 edges. *DBLP* data set has 317080 vertices and 1049866 edges. *Amazon* data set contains a graph which represents a network of products. Two products are connected if they are always co-purchased. *DBLP* data set contains a network of co-author relationship. Each vertex is an author and two authors are connected if they have co-authored a paper.

[6]http://snap.stanford.edu

In the following we shall see that the Distributed Synchronous Louvain Algorithm will show different behaviour on the two data sets *Amazon* and *DBLP*. Then we shall see how this difference is related to our findings on the Distributed Synchronous Louvain Method.

5.11.3 Number of Iterations to Converge on the Two Data Sets

Although *Amazon* and *DBLP* have a similar number of nodes and a similar number of edges, the number of iterations it takes for the algorithm to converge is quite different for the two datasets. For *Amazon* it takes on average 51 iterations in 5 runs for the algorithm to converge. However, for *DBLP* dataset the algorithm fails to converge even after 100 iterations. In terms of the number of iterations to converge, the algorithm shows very different behaviour on each of the two data sets. How does this difference relate to our knowledge of the Distributed Synchronous Louvain method? It is suggested in [24] that the sequential Louvain method can achieve around 0.3 for Normalized Mutual Information on the *Amazon* dataset, while it achieves a much lower value for NMI on the *DBLP* dataset. One of our findings about the Distributed Synchronous Louvain method is that it generates communities with very similar quality as the sequential Louvain method. Therefore, for the Distributed Synchronous Louvain method, the quality of communities for the *Amazon* dataset would be much better than the quality of communities for the *DBLP* dataset. One of our findings is that the higher the mixing parameter is, the worse the quality of communities produced by the Distributed Synchronous Louvain method will be, and the higher the number of iterations it takes to converge. Therefore, suppose both the *Amazon* and the *DBLP* datasets are generated by the LFR benchmark, the mixing parameter associated with the *Amazon* dataset would be much smaller than the mixing parameter associated with the *DBLP* dataset, and we can expect that it takes a much smaller number of iterations for the algorithm to converge on the *Amazon* dataset than on the *DBLP* dataset.

5.11.4 Scalability on the Amazon Data Set

In this section we look at the scalability of the Spark implementation of the Distributed Synchronous Louvain algorithm on the *Amazon* dataset (The *DBLP* dataset is not considered here because the Distributed Synchronous Louvain algorithm can not terminate on this dataset in a reasonable number of iterations). This experiment has been performed on the Dutch national e-infrastructure with support from the SURF SARA cooperative. Specifically, the experiment is carried out on a Spark cluster provided by SURF SARA. The hardware configuration is as shown in **Table 5.4**. We vary the number of workers: 2, 4, 8 and 16, and run the Distributed Synchronous Louvain Algorithm on the *Amazon* dataset. The execution time per iteration is shown in **Figure 5.17**.

In **Figure 5.17**, the horizontal axis shows the number of workers and the vertical axis shows the execution time per iteration. One can see from the figure that the Distributed Synchronous Louvain algorithm can scale well with an increasing number of machines, which is in line with our findings about the scalability of the Distributed Synchronous Louvain Algorithm. Also, we can observe that when we increase the number of machines from 2 to 4, the speedup is more than 2. This is because the

Spark implementation has introduced too much overhead when running with a small number of workers. By increasing the number of workers, this overhead is significantly reduced and as a result we can observe a much higher speedup.

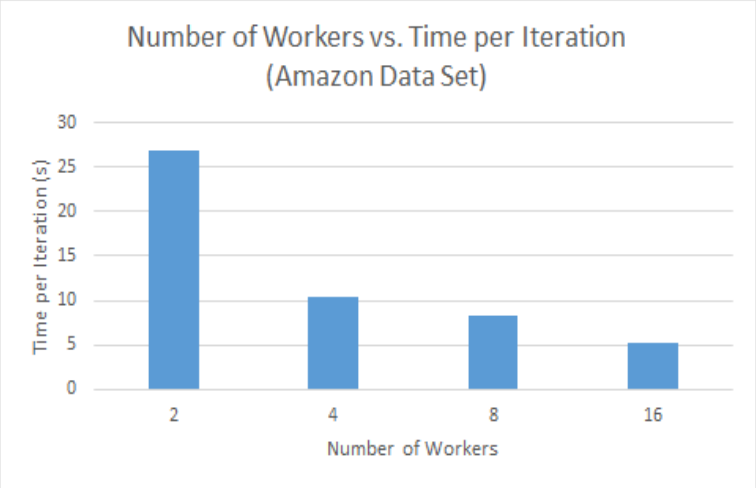

Figure 5.17: Number of Machines Versus Execution Time per Iteration on Amazon Dataset

5.12 Evaluation of Hypotheses

In this section, the hypotheses proposed at the beginning of this chapter are evaluated.

5.12.1 Hypothesis One

The original hypothesis:

The quality of partitions generated by the Distributed Synchronous Louvain method is similar to those produced by the original sequential Louvain algorithm.

In terms of quality, both the sequential and the synchronous Louvain method generate a similar result, both in terms of NMI and modularity. There is no evidence in the experiment result which invalidates this hypothesis.

5.12.2 Hypothesis Two

The original hypothesis:

The higher the update probability(but smaller than 1) , the better the quality of partitions and the lower the number of iterations to converge.

From the result of the experiments it seems that the update probability does not influence the quality of the partitions obtained in the Distributed Synchronous Louvain method, but it significantly influences the number of iterations to converge. When the update probability is larger than 0.5, the number of iterations is roughly the same. When the update probability is smaller than 0.5, the number of iterations increases drastically as the probability decreases.

Therefore, the second hypothesis is not completely correct. The update probability does not influence the quality of partitions obtained. When the update probability is larger than 0.5, no matter the value of the update probability, the number of iterations to converge remains more or less the same. When the update probability is smaller than 0.5, the number of iterations increases as the update probability decreases.

5.12.3 Hypothesis Three

The original hypothesis:

The higher the mixing parameter of a graph, the higher the number of iterations to converge and the poorer the quality of the partitions obtained.

From the result of the experiments we can see that the mixing parameter influences both the quality of the partitions and the number of iterations to converge.

In terms of the quality of the partitions generated, when the mixing parameter is smaller than 0.5, the partitions generated are always almost perfect. When the mixing parameter is larger than 0.5, the quality of the partitions obtained decreases as the mixing parameter increases.

In terms of the number of iterations to converge, when the mixing parameter is smaller than 0.5, the number of iterations to converge remains roughly the same. When the mixing parameter is larger than 0.5, the number of iterations increases as the mixing parameter increases.

Therefore, it seems that hypothesis three is not completely correct.

5.12.4 Hypothesis Four

The original hypothesis:

As the size of graphs increases, we will see a slow increase in the number of iterations for the Distributed Synchronous Louvain method to converge.

From the result of **Section 5.10.1** we can conclude that the number of iterations it takes for the algorithm to converge grows linearly as the number of vertices of a graph grows exponentially, assuming that all the other properties are the same. Therefore, we can say that as the size of graphs increases, the increase in the number of iterations to converge is slow.

5.12.5 Hypothesis Five

The original hypothesis:

With an increasing number of machines, the execution time of the Distributed Synchronous Louvain algorithm for each iteration is significantly reduced.

As is discussed in **Section 5.10.2**, given the same graph, the execution time significantly reduces as more workers are included in the Spark cluster. The speedup is very close to linear. Therefore, this hypothesis is valid.

5.13 Summary

This chapter presents an experimental evaluation to compare the original sequential Louvain method and the synchronous Louvain method that we have developed. The

influence of the mixing parameter (which controls how clear the community structure is) and the update probability on the number of iterations for the Distributed Synchronous Louvain method to converge and the quality of partitions produced by this algorithm are studied. In general, it seems that the original sequential Louvain method is very similar to the synchronous Louvain method. Furthermore, the scalability of the Distributed Synchronous Louvain method is evaluated, with an increasing number of machines and an increasing size of graphs.

List of Findings:

Comparison of the sequential Louvain method and the Distributed Synchronous Louvain method:

- Both the original sequential Louvain method and the Distributed Synchronous Louvain algorithm produce communities with similar quality, both in terms of modularity and in terms of Normalized Mutual Information.

About update probability:

- When the update probability is smaller than 0.5, a larger value of the update probability leads to a smaller number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge.
- When the update probability is larger than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is roughly the same, regardless of the value of the update probability.
- To reduce the number of iterations to converge, making the update probability as close as possible to 1 does not help. We can achieve a small number of iterations with a more or less neutral update probability (around 0.5 or 0.6).
- The update probability does not have a significant influence on the quality of communities produced by the Distributed Synchronous Louvain method.

About the mixing parameter:

- When the mixing parameter is smaller than 0.5, the quality of communities produced by the Distributed Synchronous Louvain algorithm is roughly the same, regardless of the actual value of the mixing parameter.
- When the mixing parameter is smaller than 0.5, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge is roughly the same, regardless of the actual value of the mixing parameter.
- When the mixing parameter is larger than 0.5, as the mixing parameter increases, the number of iterations it takes for the Distributed Synchronous Louvain algorithm to converge increases and the quality of communities produced decreases rapidly.

About runtime on larger graphs and with more machines:

- Assume that all the other properties are the same. The number of iterations it takes for the algorithm to converge grows linearly as the number of vertices of a graph grows exponentially.

- Given the same graph, the Spark implementation of the Distributed Synchronous Louvain algorithm shows good scalability with an increasing number of machines.
- It is in general not a good idea to run the Spark implementation of the Distribtued Synchronous Louvain algorithm with a small number of workers because the cost of distributing the computation is too high for a small number of workers.

Chapter 6

Improvement of the Resolution Parameter Selection of the CPM Community Detection Algorithm Using Graph Sampling

This chapter discusses the application of graph sampling techniques to the resolution parameter selection for the CPM method. First the method used to select the resolution parameter for the CPM method is briefly described. Next the reason why graph sampling would be useful for resolution parameter selection is explained. Then two sampling algorithms, the forest fire sampling and the random node selection sampling algorithm, are compared from the perspective of the resolution parameter selection: First hypotheses on the difference between the two graph sampling algorithms are proposed. Then the measurements to quantify the graph samples produced by these two graph sampling strategies are presented and the rationale of choosing such measurements is discussed. Finally, the measurement results of the two sampling strategies are presented, and proposed hypotheses are evaluated.

6.1 Resolution Parameter Selection

Many community detection methods, such as methods based on modularity, suffer from the problem of resolution limit, which means that the method fails to detect comparatively small communities in a large network. The Constant Potts Model method does not suffer from this problem[31]. In the following, when speaking of the CPM method, we refer to the Constant Potts Model method. The cost function of the CPM method is as follows[31]:

$$H = \sum_{ij} [A_{ij} - \lambda] \delta(\sigma_i, \sigma_j)$$

This cost function can be rewritten as follows:

$$H = -[E - \lambda N] \tag{6.1}$$

where $E = \sum_c e_c$ is the total of internal edges of communities in a partition, e_c is the number of internal edges in community c, and $N = \sum_c n_c^2$ is the sum of squared community sizes (n_c is the number of nodes in community c).

There is one free parameter in the Constant Potts Model method–the resolution parameter λ. This parameter means that communities have an internal density of at least λ and an external density of at most λ. A higher λ gives rise to smaller communities. For a graph with n nodes, the valid range of resolution parameters is $[\frac{1}{n}, 1]$.

It is proved in [31] that if a partition is optimal for resolution parameter λ_1 and λ_2, then the partition is optimal for any resolution parameter λ within $[\lambda_1, \lambda_2]$. This means that if you gradually increase λ, the optimal partition will change, but each optimal partition will remain optimal for a continuous interval of λ. As a result, the whole valid range of resolution parameters $[\frac{1}{n}, 1]$ is cut into smaller intervals, where the same optimal partition remains optimal within the same interval of resolution parameters.

The question now is how to detect these intervals. In other words, how to detect whether or not an optimal partition remains optimal over some interval of λ. By looking at Formula (6.1), one may have the intuition that the values of λ where E or N changes correspond to the λ values where the optimal partition changes. This is true. In fact, it is proved in [31] that if two partitions σ_1 and σ_2 are optimal for both resolution parameters λ_1 and λ_2, then necessarily $N_1 = N_2$ and $E_1 = E_2$. Therefore, to detect whether or not a partition remains optimal over an interval of λ, we only need to detect the points where $N(\lambda)$ or $E(\lambda)$ changes, which can be done effectively using bisectioning on λ.

By plotting resolution parameter values versus N or E, one can construct the resolution profile of a graph. The resolution profile of a graph shows two things: First the intervals within the whole valid range of resolution parameters. Second the corresponding measurement of the optimal partition in each interval (Significance, total of internal edges of communities in a partition(E) or sum of squares of community sizes (N)). An example of a resolution profile is shown in **Figure 6.1**.

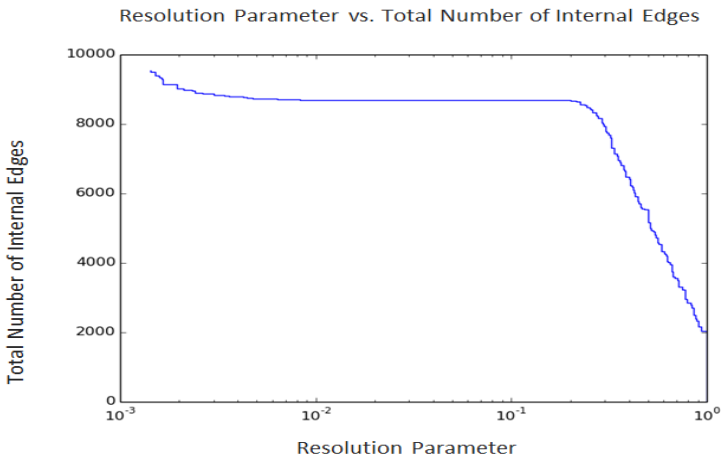

Figure 6.1: An example resolution profile

In this figure, the vertical axis shows the total number of internal edges of communities in a partition (E) and the horizontal axis shows the resolution parameter. One can see from the figure that E is a stepwise function of λ.

Why is the resolution profile of a graph useful? The free parameter λ of the Con-

stant Potts Model needs to be determined. There is no a-priori way to choose a particular resolution parameter[31], but the resolution profile of a graph can be used to choose a value for the resolution parameter. To determine whether or not a particular resolution parameter value is a good choice, a common approach is to look at how much the optimal partition changes after some perturbation[21] [27] [8] [20]. Based on this idea, [31] looked at the intervals in the whole valid range of resolution parameters. As is described before, these intervals are ranges of λ where the same partition remains optimal. The longer the interval, the more clear-cut the community structure. Those intervals that are significantly longer than other intervals are called stable "plateaus". It is suggested in [31] that stable plateaus correspond to the planted partitions for the graph. When the resolution parameter λ is within the range of stable plateaus, the partitions obtained have a very low variation and comparatively high *Significance* (a measurement proposed in [31] for partition quality) value. Therefore, the stable plateaus are the ranges of resolution parameters we are looking for.

In summary, to locate the range of resolution parameters which correspond to planted partitions in a graph, one can perform bisectioning on the resolution parameter and construct the resolution profile of this graph. Once the resolution profile is constructed, the "stable plateaus" in the resolution profile correspond to the range of good resolution parameters.

In the following discussion, when speaking of the "resolution profile construction method", we refer to the resolution profile construction method with bisectioning proposed in [31].

Note that in the resolution profile construction method, "an optimal partition remains optimal over a continuous range of resolution parameters". This does not mean, however, that there is only one optimal partition for each interval of resolution parameters. In fact there could be many possible optimal partitions in the same interval, all with the same N and E.

6.2 Why Improvement of the Resolution Profile Construction Method Is Needed

This new method to construct the resolution profile is still too slow in practice. Even with bisectioning, hundreds of runs of the CPM method are still needed to construct the resolution profile of a graph. To get a general idea, the Python package[1] is used to construct the resolution profile for three graphs: one with 3000 nodes, one with 5000 nodes and one with 10,000 nodes. The execution time is listed in **Table 6.1**. The experiment is run on a machine with 4GB memory, 4-core Intel i7 CPU with 100GB SSD Disk.

One can see that this method still takes a long time. An execution of this method on a graph with 100,000 nodes can not even terminate in one day. It is very common for real-world graphs to contain as many as millions of nodes. In such graphs the method of resolution profile construction using bisectioning is simply not applicable. Therefore we need to improve this method by accelerating it.

[1] https://github.com/vtraag/louvain-igraph

#Nodes	Execution Time
3000	9 minute
5000	23 minute
10000	1 hour 10 minute

Table 6.1: Execution Time for Three Different Graphs

One possible approach to accelerate this method is this: instead of running the method on a whole graph, run it on a representative sample. The sample is smaller and the execution time will be much shorter. The question now is the following:

Which graph sampling method gives the most representative sample from the perspective of the resolution profile construction method?

6.3 Hypotheses

In this chapter, we only consider two graph sampling algorithms: the forest fire sampling algorithm and the random node selection algorithm. Since the forest fire sampling algorithm has a free parameter (forward burning probability)[2], it is also necessary to study the influence of this parameter on the quality of graph samples. The following hypotheses about the difference between the two sampling strategies are proposed:

1. **Hypothesis 1:** In general, the accuracy of the forest fire sampling algorithm is at least as good as random node selection algorithm in estimating stable plateaus in the original graph's resolution profile.

2. **Hypothesis 2:** For the forest fire sampling algorithm, the higher the forward burning probability (but smaller than 1), the more accurate it is in estimating stable plateaus in the original graph's resolution profile.

3. **Hypothesis 3:** As the sample size decreases, the accuracy of sampling algorithms in estimating stable plateaus decreases.

I generated graph samples from several graphs, ran the resolution profile construction method on these graphs and then compared the resolution profiles of the graph samples and the resolution profiles of the original graphs from which these samples were generated. It turns out that the longest stable plateau in the sample graph's resolution profile and the stable plateau of the original graph's resolution profile sometimes have no intersection at all. I became curious about the cause of this phenomenon and I looked into the resolution profiles. In general, I noticed two things: First the longest stable plateau in the original graph's resolution profile was cut into small intervals in the resolution profile of the samples. Second, the smaller intervals in the original graph's resolution profile were merged into larger intervals in the resolution profile of the samples.

[2]In fact, in the original forest fire model, there are two free parameters-forward burning probability and backward burning probability. Since we are only looking at undirected network, backward burning probability does not really make sense here.

To understand this phenomenon more clearly, I did a small experiment: I generated a graph with 10,000 nodes with configurations as listed in **Table 5.1** and I ran the forest fire sampling algorithm on the original graph with a forward burning probability of 0.3. I counted the number of intervals within the range of resolution parameters [0.5,1] (I took the average value for the number of intervals). The result is shown in **Figure 6.2**. The horizontal axis shows the size of the sample graphs and the vertical axis shows the number of intervals within the range [0.5,1] of the resolution parameter. The figure shows that as the size of the graph sample decreases, the number of intervals decreases, which indicates that smaller intervals are merged into larger intervals.

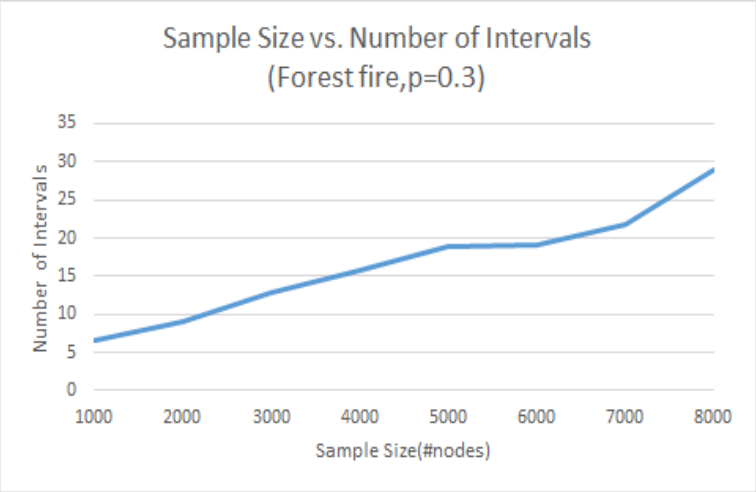

Figure 6.2: Number Of Intervals(Forest Fire,p=0.3)

The resolution profile of a graph sample can be viewed as an estimation of the resolution profile of the original graph. Therefore, for the phenomenon that the long plateaus in the resolution profile of a sample graph do not match the stable plateau in the resolution profile of the original graph, the guess of cause of this phenomenon can be formulated from the perspective of estimation. By doing so the fourth hypothesis is obtained, as follows:

Hypothesis 4: *The cause of the mismatch of stable plateaus between the sample graph's resolution profile and the original graph's resolution profile is that: the graph samples have underestimated long intervals and overestimated short intervals in the resolution profile of the original graph.*

In this hypothesis, "overestimate" means that an interval in the original graph's resolution profile is shorter (in log scale) than the corresponding interval in the sample's resolution profile.

6.4 Measurements for Sample Quality

This section presents the measurements that are used to quantify the behaviour of graph sampling algorithms.

6.4.1 Precision of Plateau Estimation

In order to verify the first three hypotheses, we need to define a measurement which is able to show how much the resolution profile of the sample "agrees" with the resolution profile of the original graph. In the following it will be explained how this measurement is devised:

The first idea is to look at the estimation of the significance of the graph samples. The rationale is that the closer the significance values of the sample graph's communities and the original graph's communities, the more similar the resolution profiles of the two graphs. However, it is suggested in[31] that for an Erdős-Rényi random graph, the maximum value of significance that a partition can achieve in a graph scales with graph size. The significance of communities of Erdős-Rényi random graphs scale approximately as $n\log n$, where n is the number of nodes in the graph. Therefore, the significance value of the original graph's partition cannot be directly compared to the significance value of the graph sample's partition. A possible solution would be to divide the significance value by the scaling factor $n\log n$, but for graphs which are not Erdős-Rényi random graph, the scaling factor might not be $n\log n$. This indicates that it is not a good idea to compare the significance of a partition of the original graph with the significance of a partition of a sample of this graph.

The second idea is to look at the resolution profile directly. Since in the resolution profile construction method, we only care about the intervals in the resolution profile that are significantly longer than the others (stable plateaus), we do not need to compare all the intervals. We only consider the longest intervals (the longer the interval, the more clear-cut the community structure).

Based on this idea, the *precision* of stable plateaus is defined as follows: *Suppose the longest interval in the resolution profile of the original graph is S, the longest interval in the resolution profile of this graph's sample is S'. The precision is the length (in log scale) of the intersection between S and S' divided by the length (in log scale) of S'.*

Similarly, if the original graph shows a hierarchical structure, the precision for the second longest interval, third longest interval...et cetera, could be defined in a similar way.

For example, if the longest interval of the original graph is $[0.2, 0.5]$ and the longest interval of the sample is $[0.15, 0.4]$, then the precision is $\frac{log0.4-log0.2}{log0.4-log0.15}$.

A disadvantage of the precision of plateau estimate is that it is very sensitive to relative ordering of lengths (in log scale) of intervals in the resolution profile. Suppose A is the actual stable plateau and B is an interval in the resolution profile of the sample and B has a long intersection with A. If B is slightly shorter than the stable plateau C in the resolution profile of this graph sample, the precision result could be very bad, because in this case precision is calculated from the perspective of C.

Despite this disadvantage, the measurement of precision can still give us an idea about how a sample algorithm performs. The measurement is reasonable, because when we try to estimate stable plateaus in a graph's resolution profile from this graph's sample, we do not have prior knowledge about this graph's resolution profile. The relative ordering of lengths of intervals does affect our judgement about the positions of stable plateaus, so it is reasonable to reflect this in the measurement of precision. In other words, *precision* is unstable because the process of estimating the resolution

profile and stable plateaus using graph samples is unstable.

6.4.2 The Length of Representative of Intervals in a Resolution Profile

To examine hypothesis 4, we need to answer the following question:

Given the resolution profile of a graph sample, which interval in this resolution profile is the sample's "estimate" of a particular interval in the resolution profile of the original graph?

In other words, given an interval in the resolution profile of a graph, which is its corresponding interval in the resolution profile of this graph's sample? Precision is not enough to answer this question. It is necessary to define another measurement. The new measurement is defined in this way: for each of the intervals in the resolution profile of the original graph, find a corresponding interval in the resolution profile of the sample. When this interval is found, measure the length (in log scale) of this interval. Then we can see if there is any overestimation or underestimation. The interval (in the resolution profile of graph samples) which corresponds to interval A in the resolution profile of the original graph is called the *representative* of A.

Choice of *Representative*

Now the question is:

Given the resolution profile of a sample, which interval should we choose as the ***representative*** *of an interval in the original graph's resolution profile?*

There are many possible candidate intervals in the resolution profile of a sample that can represent a certain interval in the resolution profile of the original graph.

First, it is reasonable to say that all the intervals in a sample's resolution profile that have a non-empty intersection with an interval P in the original graph's resolution profile are potential candidates for the *representative* of P. This is because all these intervals are estimations of P that are at least partially correct.

Second, from the perspective of the resolution profile construction method, the resolution significance of a resolution parameter λ is determined by the length (in log scale) of the interval that λ belongs to. The longer the interval, the more clear-cut the community structure. Given that we have no a-priori knowledge of the resolution profile of the original graph, the long intervals in a sample's resolution profile are those that are likely to influence our judgement of the position of stable plateaus. Therefore, it is reasonable to require the *representative* of an interval to be as long as possible (in log scale).

In terms of "long", there are three possible choices for the *representative* of interval P:

1. The interval that has the largest intersection with P
2. The longest interval within P.
3. The longest interval which has a non-empty intersection with P.

One can see that the *representative* chosen by the third criterion is always larger than or equal to the *representatives* chosen by the first two criteria. In other words, the length (in log scale) of *representatives* chosen by the third criterion is an upper bound.

The second criterion is not a good choice because this criterion is too restricted. This criterion requires the *representative* to be inside interval *P*. However, in the experiments, it is often observed that intervals in the sample's resolution profile do not lie within the range of interval *P* of the original graph's resolution profile, while they still have a very long intersection. Therefore this criterion is not used.

The third criterion has the advantage that it is a reasonable choice from the perspective of the resolution profile construction method. This is because from this method's point of view, the longest intervals in a graph's resolution profile correspond to the planted partition of the graph. The *representative* chosen by criterion 3 has upper-bound length (in log scale) among all the intervals that are potential candidates to represent *P*. Therefore criterion 3 is consistent with the spirit of the resolution profile construction method. In other words, among all the intervals that are partially correct about interval *P*, the interval chosen by the third criterion is the one that is most likely to affect our judgement of resolution significance.

The disadvantage of the third criterion is that it does not take into account the length (in log scale) of the intersection. An interval can be chosen to be *representative* even if it has a very short intersection with the interval it represents in the original graph's resolution profile.

The first criterion seems to offer a very good trade-off between the length (in log scale) of the intersection and the length (in log scale) of the chosen *representative* itself. However, the *representative* chosen by this criterion can be much shorter than the *representative* chosen by the third criterion.

To find out the difference between the *representatives* chosen by the first and the third criterion, a small experiment is carried out:

1. Use the forest fire sampling algorithm to generate graph samples. Set the forward burning probability to 0.3.

2. Vary the sample sizes from 1000 to 8000 with steps of 1000. For each graph size generate 5 samples from the original graph .

3. Run the resolution profile construction method on the samples and generate one resolution profile for each sample.

4. In each sample, locate the representative intervals of 1st, 2nd and 3rd longest intervals (the 1st longest interval is the stable plateau in our case) chosen by the first and the third criterion respectively. Measure the length of these intervals.

The result is shown in **Figure 6.3**,**Figure 6.4** and **Figure 6.5**. Data points with the same color show the length of *representatives* chosen by the same criterion. The measurement result of the length of *representatives* for a certain sample size is the average value of all the sample graphs of that size. Each figure shows the measurement result of the *representatives* of a certain interval in the original graph's resolution profile.

One can see that: First, the average length (in log scale) of *representatives* chosen by the first criterion and the third criterion are in most cases exactly the same. This indicates that in most cases the *representatives* chosen by the first and the third criterion are the same. Second, even in the cases where the lengths (in log scale) of *representatives* are different, the difference is really small. This indicates that even when the two

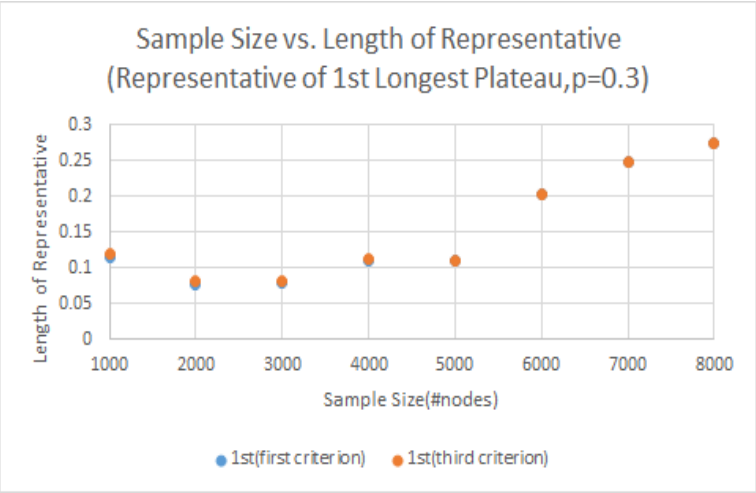

Figure 6.3: Interval Length of *Representatives*(1st interval,p=0.3)

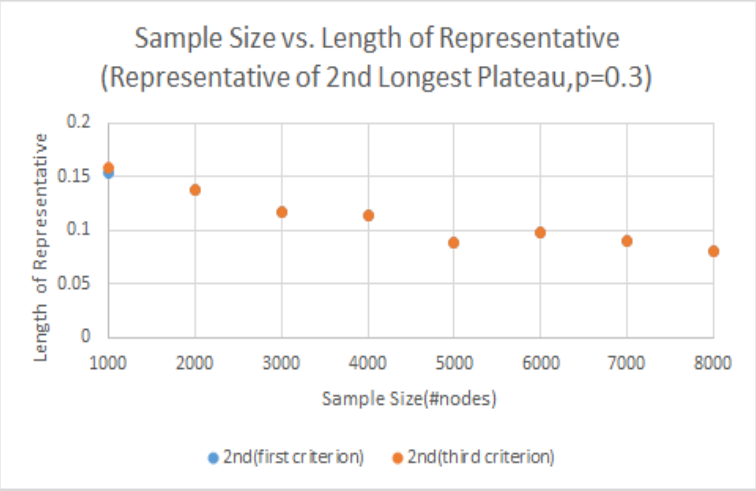

Figure 6.4: Interval Length of *Representatives*(2nd interval,p=0.3)

criteria choose different *representatives*, the lengths (in log scale) of *representatives* are quite similar. Therefore, it seems that if we only care about the length of *representatives*, the *representatives* chosen by the first and the third criteria are quite similar. In practice we can use both. Here the 3rd criterion is chosen, because it can guarantee that the *representatives* chosen are as long as possible (upper bound).

Therefore, the *representative* of an interval in the resolution profile of the original graph is defined as follows:

For an interval (m,n) in the original graph's resolution profile, its representative in the sample's resolution profile is defined as the interval in the sample's resolution profile that has a non-empty overlap with (m,n), and at the same time it has the largest length (in log scale) among all the intervals that have non-empty overlap with (m,n).

The *representative* interval chosen by criterion 3 may be a poor estimate of the original interval (e.g. it is possible that a *representative* and the interval it represents

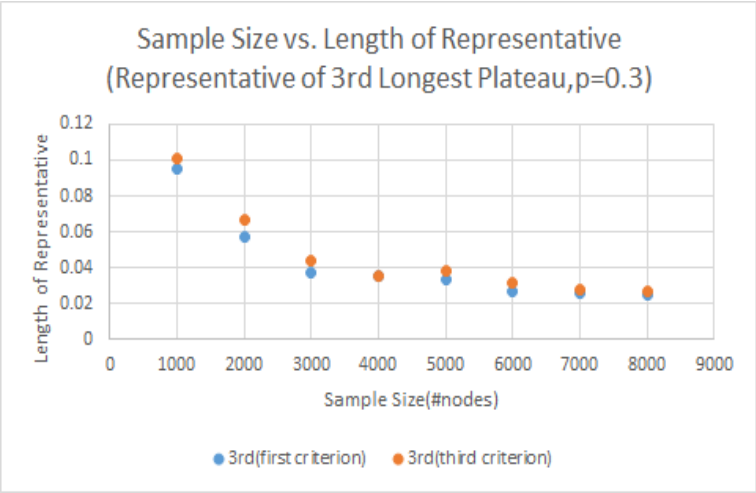

Figure 6.5: Interval Length of *Representatives*(3rd interval,p=0.3)

only have a short intersection), but this is not a problem, because the notion of *representative* is introduced mainly to help us understand what is happening when the estimate of the stable plateau in the resolution profile is completely wrong, i.e. the estimate of the stable plateau and the real stable plateau have no intersection at all. In this case, the accuracy of the representative of the stable plateau is not important, since anyway it is dominated by another plateau which has no intersection with the actual stable plateau in the original graph's resolution profile.

Notice that the notion of *representative* is not just defined for the stable plateaus. In fact a *representative* can be defined for each interval in the resolution profile of the original graph.

Interpretation of *Representative*

This section describes the interpretation of *representative*.

For example, let's consider two intervals in the original graph's resolution profile, A and B. A is the longest stable plateau in the original graph's resolution profile. Suppose in a sample of this graph, the *representative* of A is A' and the *representative* of B is B'. The ideal situation would be: A' is larger than any other interval in the sample's resolution profile. However, in practice we are likely to observe that B' is larger than A' and B' is the largest stable plateau. This indicates that the sampling process has overestimated B and underestimated A, leading to the situation where the *representative* of B dominates the *representative* of A. If B' is longer than any other intervals in the sample's resolution profile, the user might think that interval B' corresponds to the planted partitions of the original graph, which is not correct.

If two plateaus A and B in the original graph's resolution profile share the same *representative* R, it indicates that the sampling procedure has "merged" these two plateaus in the sample graph's resolution profile, because all the other intervals in the sample graph's resolution profile that have a non-empty intersection with A or B are shorter than R.

Relation between *Representative* and *Precision*

In the following cases, the precision of the estimate of the longest stable plateau will be high:

1. When considering the average length of *representatives*, the *representative* of the longest stable plateau is always the longest among all the intervals in the sample graph's resolution profile.
2. When considering the average length of *representatives*, the *representative* of the longest stable plateau is not the longest, but it is close to the longest *representative* in the sample graph's resolution profile.

In the other cases, the precision of estimation of longest stable plateau will be poor, or even 0.

Problem of *Representative*

Although the notion of *representative* is useful, there are problems with it:

1. **The measurement may be unstable.** The resolution profile may be influenced by many different features of the sample. Moreover, very different samples may be generated by the same graph sampling configuration. As a result, it is likely that the *representative* of intervals in the original graph's resolution profile shows a large variety across different samples. In the worst case, the length (in log scale) of *representatives* becomes similar to noise. In fact, the measurement of *precision* also suffers from the same problem.
2. **The interpretation of *representative* may be oversimplified.** The interpretation of *representative* described in the previous section may be oversimplified. For example, if two plateaus A and B in the original graph share the same *representative* R, it is not completely true to say that the sampling procedure has "merged" these two intervals in the original graph's resolution profile, since there may be other plateaus that have non-empty intersection with A or B.
3. ***Representative* does not have a solid theoretical basis.** The definition of *representative* is directly motivated by the resolution profile construction method. This definition is independent of any other measurements for graph sample quality, such as degree distribution and clustering coefficient.

In general, the largest risk is that:
Instead of measuring something meaningful, it is likely that I am simply measuring noise.

However, these three problems with *representative* also shows how complex it is to quantify the behavior of a sampling algorithm.

The first problem is caused by the fact that in practice you find *representatives* for only a few intervals in the resolution profile of the original graph. As a result, the *representatives* of these intervals only occupy a very small fraction of the whole range of valid resolution parameters, and the positions of these *representatives* may be very different in different samples.

Range	Log Length
(0.0078125,0.0441941738242)	0.75257498916
(0.800354433289,1.0)	0.0967176450914
(0.00185017185702,0.0020437432591)	0.0432142669554
(0.505102622537,0.551938001461)	0.0385106732733
(0.456025846352,0.489617341633)	0.0308673335394

Table 6.2: Top 5 Stable Plateau of the Graph

To avoid this problem, a possible approach is to use a measurement that considers all intervals in the resolution profile. This is difficult, because comparing the similarity of intervals is difficult. For example, can you tell how similar $[0, 0.5, 0.6, 1]$ and $[0, 0.3, 0.4, 1]$ are? Besides, it is unnecessary to consider every interval in the whole range of valid resolution parameters because not all intervals are equally important. In the resolution profile construction method we only consider those intervals that are significantly longer than other intervals. The second problem can be solved by taking into account more subtle cases in the interpretation, but this drastically increases the complexity of the measurement. Therefore in general, although there might be a better measurement, measuring length (in log scale) of *representative* is already good enough as the first step to solve the problem of understanding the behavior of a sampling algorithm and its influence on the resolution profile of a graph.

Although we can measure the length (in log scale) of *representatives* as a first step, we cannot turn a blind eye to these problems. We must be very cautious about the experiment design and the interpretation of the experiment results.

6.5 Experiment Setup

This section presents the setting of the experiments. A graph with 10,000 nodes is generated. The top five intervals (in terms of length in log scale) in the resolution profile of this graph are shown in **Table 6.2** and a visualization of the resolution profile is shown in **Figure 6.6**.

In **Figure 6.6**, the horizontal axis shows the resolution parameters in log scale and the vertical axis shows the total number of internal edges of the communities. One can see from the figure that there is only one stable plateau. Therefore, we only calculate the precision from the perspective of this stable plateau (the longest interval in the original graph's resolution profile, in log scale). In the following discussion, "the stable plateau in the resolution profile" means "the longest interval in the resolution profile".

In the following, when speaking of "the original graph", I mean this graph.

6.6 Measurement Results of the Forest Fire Sampling Method

In this section we generate graph samples using the forest fire sampling strategy and evaluate the precision of the stable plateau estimate of different sizes of graph samples.

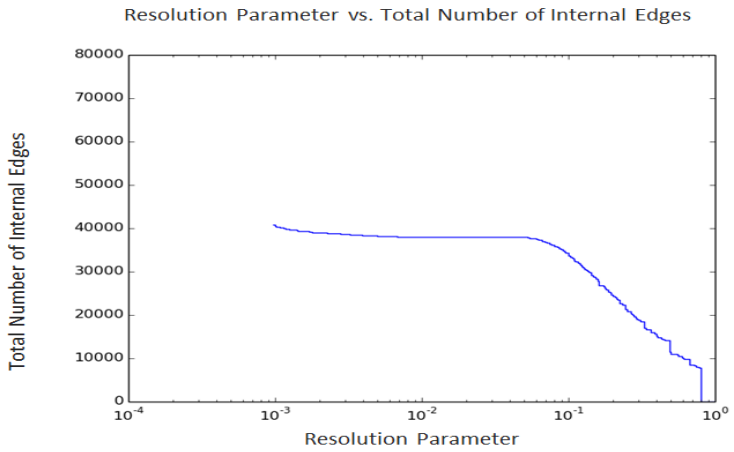

Figure 6.6: Resolution Profile of the Graph

Then the lengths of the *representatives* of certain intervals in the original graph's resolution profile are measured. A detailed description of the forest fire sampling algorithm can be found in **Section 2.4.9**.

Key Findings:

- For some values of the forward burning probability (0.6 and 0.9), as the size of graph sample decreases, the precision of the stable plateau estimate first decreases and then increases.

- In general there are two different types of behaviour for lengths (in log scale) of *representatives*: One type is the *representative* of the stable plateau in the original graph's resolution profile. As sample size decreases, its length first decreases and then increases, showing a "U" shape. The other type is the *representatives* of the other intervals in the original graph's resolution profile. In general, the length (in log scale) of these *representatives* goes up in a wave-like manner as the sample size decreases.

- The results of the length of *representatives* and the precision of stable plateau estimates are consistent.

6.6.1 The Precision of Stable Plateau Estimates

In this section we use the forest fire sampling algorithm to generate graph samples and measure the precision of stable plateau estimates of the resolution profile of these graph samples.

Key Findings:

- For some value of the forward burning probability, as the size of graph sample decreases, the precision of the stable plateau estimate first decreases and then increases.

The experiment is carried out in this way: Use the forest fire sampling algorithm to generate samples of the original graph. Set the forward burning probability to 0.3. Vary the size of samples from 1000 to 4000 with steps of 1000 nodes. For graph sizes from 1000 to 4000 nodes generate 10 samples for each size. Run the resolution profile construction method on the graph samples and generate one resolution profile for each graph sample. Measure the precision of the stable plateau estimate with respect to the resolution profile of the original graph. The precision of the stable plateau estimate is measured according to the definition described in **Section 6.4.1**.

Figure 6.7 shows the precision of sampling as sample size changes. The horizontal axis shows the sizes of graph samples in terms of number of vertices and the vertical axis shows the precision of the stable plateau estimate. The measurement value of the precision for a certain sample size is the average value of all the samples of that size (for instance, there are 10 samples with 4000 nodes, so the value of precision is averaged over the ten samples).

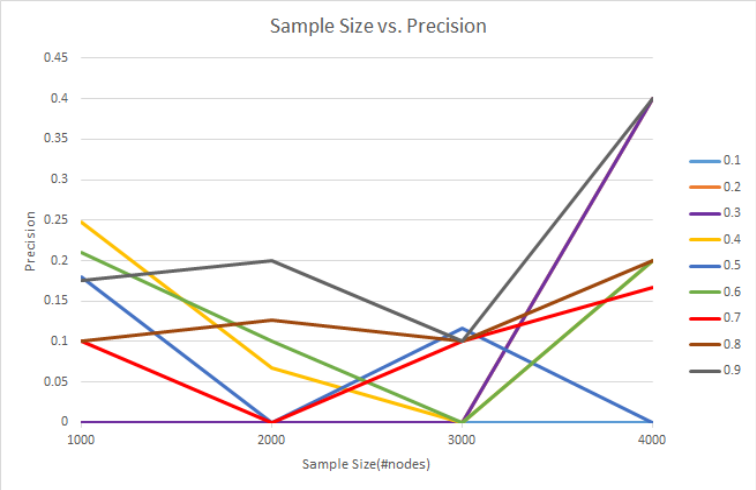

Figure 6.7: Precision for Forest Fire Sampling

In **Figure 6.7**, there is one line for each forward burning probability. It is clear from this figure that hypothesis 3 is not correct. Specifically, if we look at the change of precision with forward burning probability 0.6 and 0.9, as is shown in **Figure 6.8**, we can see that as the sample size decreases, the precision first decreases and then increases.

Therefore, the precision of stable plateau estimates do not always decrease as the sample size decreases. For some values of the forward burning probability, as sample size decreases, the precision of the stable plateau estimate first decreases and then increases.

6.6.2 Length of *Representatives*

This section studies how the length of *representatives* changes as the sample size changes if we use the forest fire sampling algorithm to generate graph samples.

Key Findings:

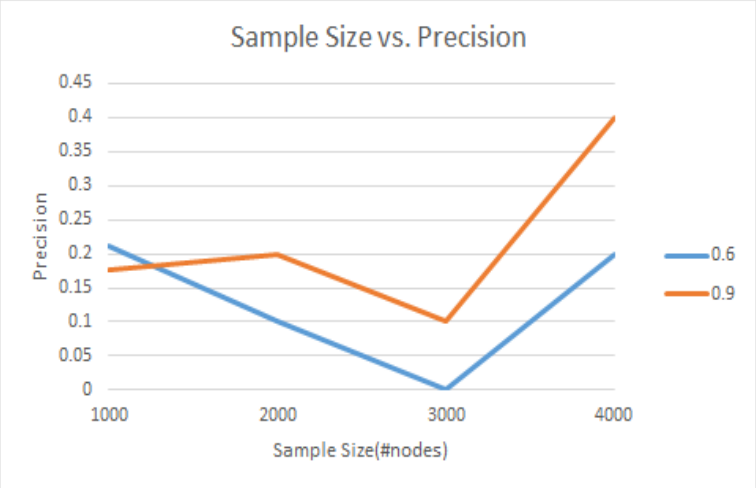

Figure 6.8: Precision for Forward Burning Probability 0.6 and 0.9

- In general there are two different types of behaviour for lengths (in log scale) of *representatives*: One type is the *representative* of the stable plateau in the original graph's resolution profile. As sample size decreases, its length first decreases and then increases, showing a "U" shape. The other type is the *representatives* of the other intervals in the original graph's resolution profile. In general, the length (in log scale) of these *representatives* goes up in a wave-like manner as the sample size decreases.

- The results of the length of *representatives* and the precision of stable plateau estimates are consistent.

Result of Length of *Representatives*

This section presents the result of the measurement of the length of *representatives* of certain intervals in the original graph's resolution profile.

Key Findings:

- As the size of graph samples decreases, the length of *representative* of the stable plateau first decreases and then increases, showing a "U" shape, while the length of *representative* of the second longest interval keeps increasing.

Experiment 1:

Because the longest stable plateaus in the resolution profile of many samples are in fact *representatives* of (0.800354433289,1.0), which is the second longest interval in the resolution profile of the original graph, it would be interesting to look into the *representative* of this interval in addition to the *representative* of the stable plateau in the original graph's resolution profile.

In the experiment, the forward burning probability is set at 0.3. Graph samples with four different sizes are studied: 1000, 2000, 3000 and 4000 nodes. This correspond to 10%, 20%, 30% and 40% of the original graph. The experiment is carried out in the following steps:

1. Use the forest fire sampling algorithm to generate graph samples. Set the forward burning probability to 0.3. Use the graph described in **Section 6.5** to generate graph samples with 1000 nodes, 2000 nodes, 3000 nodes and 4000 nodes respectively (10 samples for each graph size).

2. Run the resolution profile construction method on the graph samples and generate a resolution profile for each sample.

3. In each resolution profile, locate the *representative* intervals of the stable plateau and the second longest interval in the original graph's resolution profile (listed in **Table 6.2**). Measure the length of these intervals.

The result is shown in **Figure 6.9**. In this figure, the horizontal axis shows the sizes of graph samples in terms of the number of vertices and the vertical axis shows the length of *representatives*. Each line shows the length of the *representative* of a certain interval in the original graph's resolution profile. The value of the length of *representatives* for a sample size is the average value of all the sample graphs of that size.

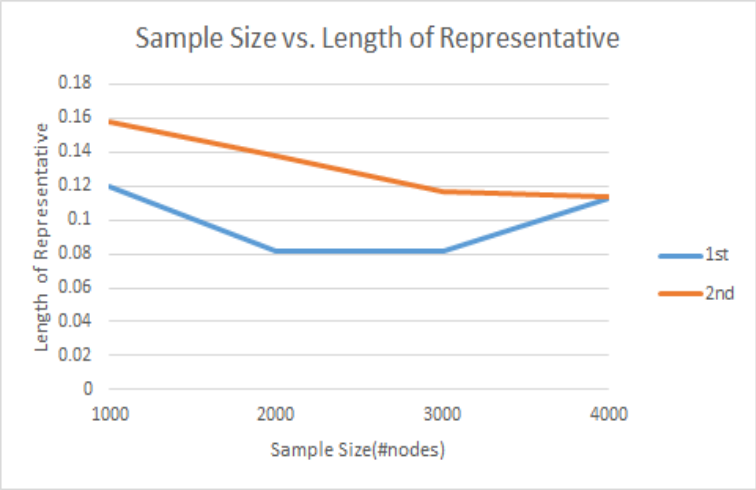

Figure 6.9: Length of *Representatives* of 1st and 2nd Longest Interval(p=0.3)

One can see that the *representative* of the stable plateau is dominated by the *representative* of the second longest interval. The length (in log scale) of the *representative* of the second longest interval keeps increasing while the length (in log scale) of the *representative* of the stable plateau first decreases and then increases, showing a "U" shape. This observation is consistent with the precision for forward burning probability 0.3: for sample size 1000, 2000 and 3000, the precision is 0 and for sample size 4000, where the two lines in **Figure 6.9** intersect, the precision increases to 0.4.

Experiment 2:

What happens if the size of the sample increases? I further increase the size of sample and see what happens. Larger samples are studied. Specifically, the experiment is carried out in the following steps:

1. Use the forest fire sampling algorithm. Set the forward burning probability to 0.3 and use the graph described in **Section 6.5** to generate graph samples with 1000 nodes, 2000 nodes, 3000 nodes and 4000 nodes respectively (10 samples for each graph size) and generate graph samples with 5000 nodes, 6000 nodes, 7000 nodes, 8000 nodes and 9000 nodes respectively (5 samples for each graph size).

2. Run the resolution profile construction method on the graph samples and generate a resolution profile for each sample.

3. In each resolution profile, locate the *representative* intervals of the stable plateau and the second longest interval in the original graph's resolution profile (listed in **Table 6.2**). Measure the length of these intervals.

The result is shown in **Figure 6.10**. The horizontal axis shows the size of samples in terms of the number of vertices and the vertical axis shows the length (in log scale) of *representatives*. Each line shows the length of the *representative* of a certain interval in the original graph's resolution profile. In this figure, the measurement value of the length of the *representative* for a sample size is the average value of all the sample graphs of that size.

Figure 6.10: Length of *Representatives* of 1st and 2nd Longest Interval(p=0.3)

One can see that the length (in log scale) of the *representative* of the second longest interval increases as the size of the graph samples decreases. The length (in log scale) of the *representative* of the stable plateau first decreases and then increases, showing a "U" shape.

Notice that when the sample size is 4000, length (in log scale) of the the *representative* of the stable plateau is slightly larger than the length (in log scale) for sample size 5000. This slightly distorts the trend of the "U" shape, but the difference is relatively small. This "noisy" behavior indicates that the *representatives* of intervals are not stable.

Length of *Representatives* of More Intervals

This section studies the characteristics of the length of *representatives* of the top 5 intervals in the original graph's resolution profile.

Key Findings:

- In general there are two different types of behaviour for lengths (in log scale) of *representatives*: One type is the *representative* of the stable plateau in the original graph's resolution profile. As sample size decreases, its length first decreases and then increases, showing a "U" shape. The other type is the *representatives* of the other intervals in the original graph's resolution profile. In general, the length (in log scale) of these *representatives* goes up in a wave-like manner as the sample size decreases.

Experiment 1:

Now more intervals in the original graph's resolution profile are included in the experiment to quantify the behaviour of sampling algorithms. First the forward burning probability is set at 0.3. Instead of just looking at the length (in log scale) of *representatives* of the stable plateau and the second longest interval in the original graph's resolution profile, the *representatives* of the top 5 intervals in the original graph's resolution profile are studied. The reason for doing so is to reveal the general behavior of intervals in the resolution profile as the sample size decreases. Specifically, the experiment is carried out in the following steps:

1. Use the forest fire sampling algorithm to generate graph samples. Set the forward burning probability to 0.3 and use the graph described in **Section 6.5** to generate samples with 1000 nodes, 2000 nodes, 3000 nodes and 4000 nodes respectively (10 samples for each size) and generate samples with 5000 nodes, 6000 nodes, 7000 nodes, 8000 nodes and 9000 nodes (5 samples for each size).

2. Run the resolution profile construction method on the graph samples and generate a resolution profile for each sample.

3. In each resolution profile, locate the *representative* intervals of the top 5 intervals in the original graph's resolution profile (listed in **Table 6.2**). Measure the length of these intervals.

The result is shown in **Figure 6.11**. The horizontal axis shows the size of graph samples and the vertical axis shows the length (in log scale) of *representatives*. Each line shows the length of the *representative* of a certain interval in the original graph's resolution profile. In this figure, the measurement value of the length of the *representative* of a sample size is the average value of all the sample graphs of that size.

One can see from **Figure 6.11** that there are two different types of behaviour for lengths (in log scale) of *representatives*:

One type is the *representative* of the stable plateau in the original graph's resolution profile. As sample size decreases, its length first decreases and then increases, showing a "U" shape. This corresponds to the black line in the figure.

The other type is the *representatives* of the second, third, fourth and fifth longest interval in the original graph's resolution profile. In general, the length (in log scale)

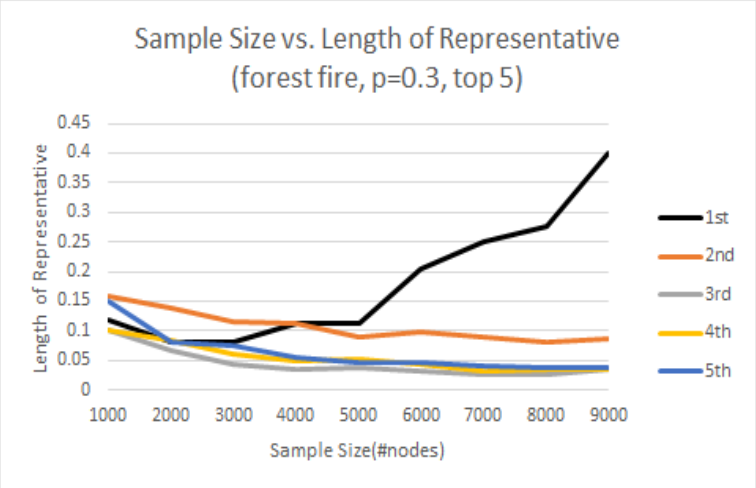

Figure 6.11: Lengths of *Representatives* for Forward Burning Probability 0.3

of these *representatives* goes up in a wave-like manner as the sample size decreases. When the sample size is large (roughly larger than 50% of the original graph), the speed of increase is not high. When the size of samples are much smaller(smaller than 50%), the increase is much faster.

Experiment 2:

Is this a general phenomenon? I repeated the same experiment with a forward burning probability of 0.6. The setting was the same as the previous experiment:

1. Use the forest fire sampling algorithm to generate graph samples. Set the forward burning probability to 0.6 and use the graph described in **Section 6.5** to generate graph samples with 1000 nodes, 2000 nodes, 3000 nodes and 4000 nodes respectively (10 samples for each graph size) and generate graph samples with 5000 nodes, 6000 nodes, 7000 nodes, 8000 nodes and 9000 nodes (5 samples for each graph size).

2. Run the resolution profile construction method on the graph samples and generate a resolution profile for each sample.

3. In each resolution profile, locate the *representative* intervals of the top 5 intervals in the original graph's resolution profile (listed in **Table 6.2**). Measure the length of these intervals.

The result is shown in **Figure 6.12**. The horizontal axis shows the sample sizes and the vertical axis shows the length of *representatives* in log scale. Each line shows the length of the *representative* of a certain interval in the original graph's resolution profile. In this figure, the measurement value of the length of the *representative* of a sample size is the average value of all the sample graphs of that size.

In general one can observe the same phenomenon. Compared with a forward burning probability of 0.3, there are two differences: First, the increase of the length (in log scale) of the second type of *representatives* (*representatives* of the 2nd, 3rd, 4th and 5th longest intervals in the original graph's resolution profile) is not as fast as in

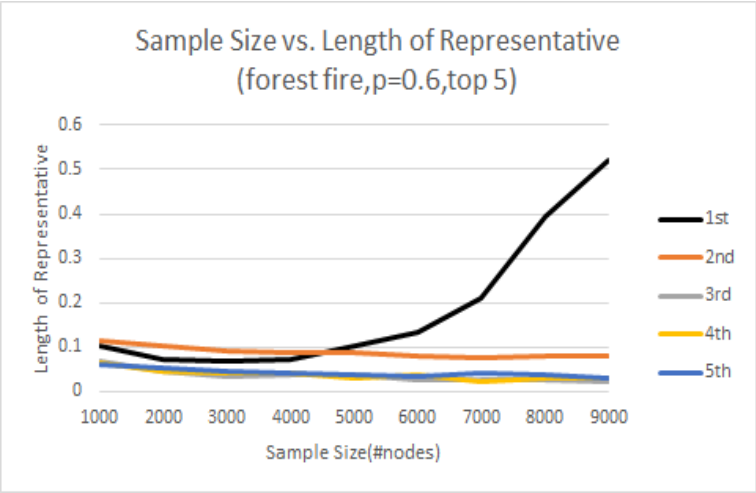

Figure 6.12: Lengths of *Representatives* for Forward Burning Probability 0.6

the case of a probability of 0.3. Second, the fluctuation of the length (in log scale) of *representatives* is not as significant as in the case of a probability of 0.3.

If we look at the length (in log scale) of the top 5 intervals (**Table 6.2**), we can see that the length (in log scale) of the stable plateau is much larger than the lengths of the other 4 intervals, while the lengths (in log scale) of the other 4 plateaus are more or less similar. This is consistent with what we observed in lengths of their *representatives*. There are two different types of behaviours: one for the *representative* of the stable plateau and the other for the *representatives* of the 2nd, 3rd, 4th and 5th longest intervals in the original graph's resolution profile.

Notice that the lengths of the 2nd, 3rd, 4th and 5th largest intervals in the original graph's resolution profile are larger than the lengths of their corresponding *representatives* when the sample size is 9000, which seems to indicate that graph samples do not always overestimate smaller intervals in the original graph's resolution profile. However, the difference between them is quite small, and for most sizes of graph samples(from 80% to 10% of the original graph), the graph samples tend to overestimate intervals that are not the longest in the original graph's resolution profile.

Relation Between Length of *Representatives* and Precision of Stable Plateau Estimates

This section examines the relation between the length of *representatives* and the precision of plateau estimates of the graph samples.

Key Findings:

- The results of the length of *representatives* and the precision of stable plateau estimates are consistent.

To see the connection between the precision and length of *representatives*, the precision vs. sample size relation is plotted: Use the Forest Fire sampling algorithm to generate samples. Set the forward burning probability at 0.3. Vary the size of samples from 1000 to 9000 with steps of 1000 nodes. For graph sizes from 1000 to 4000

nodes generate 10 samples for each size and for graph sizes from 5000 to 9000 nodes generate 5 samples for each size. Run the resolution profile construction method on the graph samples and generate one resolution profile for each graph sample. Measure the precision of stable plateau estimates with respect to the resolution profile of the original graph.

The result is shown in **Figure 6.13**. The horizontal axis shows the size of graph samples and the vertical axis shows the precision of the stable plateau estimates for a certain sample size. The measurement value of the precision of stable plateau estimates is the average value of all the graph samples of that size.

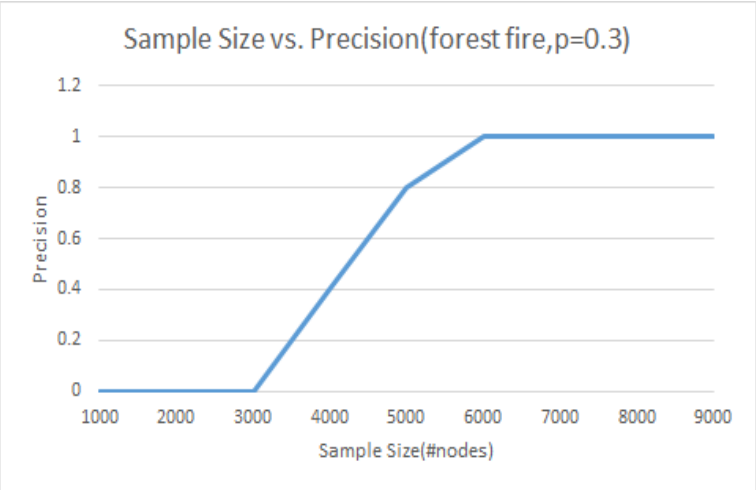

Figure 6.13: The Precision for Forward Burning Probability 0.3

By comparing **Figure 6.13** with **Figure 6.10**, one can see that the intersection point in **Figure 6.10**, at a sample size of 4000 nodes, corresponds to the shifting from non-zero precision to 0 precision in the precision result in **Figure 6.13**. This indicates that the measurement of the length of *representatives* is consistent with the measurement of the precision of stable plateau estimates. Therefore, I am more confident that the length (in log scale) of *representatives* is a good indicator to quantify the behaviour of graph sampling algorithms.

6.6.3 Summary of Observations

In this section, the observations of this set of experiments for the forest fire sampling algorithm are summarized.

In terms of the precision of the estimate of the stable plateau in the original graph's resolution profile, the precision decreases as the size of sample decreases. However, in some cases, if the sample size is smaller than a certain amount, precision may begin to increase again (in some cases in a wave-like manner) as sample size decreases. There are two different types of behavior for the length (in log scale) of *representatives* of intervals in the original graph's resolution profile:

1. One type is the *representative* of the stable plateau in the original graph's resolution profile. As sample size decreases, the length of *representative* (in log scale)

first decreases and then increases, showing a "U" shape.

2. The other type is the *representatives* of shorter intervals. In general the lengths (in log scale) of these *representatives* increases in a wave-like way as sample size decreases.

What do these observations indicate? Ideally, in the graph samples, the length of *representatives* of the stable plateau should be larger than the length of *representatives* of the other intervals in the original graph's resolution profile. This indicates that in most cases, in the sample graph's resolution profile, the *representative* of the stable plateau in the original graph's resolution profile will be the longest one, and it is unlikely for this sample to completely miss the range of the stable plateau in the original graph's resolution profile. If we observe that the *representative* of the stable plateau in the original graph's resolution profile is always smaller than the *representatives* of the other intervals, the sample's estimation of the stable plateau in the original graph's resolution profile is likely to be completely wrong.

With this in mind, we find something interesting in **Figure 6.11** and **Figure 6.12**. In both cases, the *representative* of the longest plateau shows a "U" shape. Suppose the sample size gradually decreases: 9000, 8000, 7000,...,2000,1000. In the case of a forward burning probability of 0.3(**Figure 6.11**), when the sample size is very small (around 1000), the increase in the lengths of *representatives* of the 2nd, 3rd, 4th and 5th longest intervals is more significant than the increase in the length of *representative* of the stable plateau. As a result it is very likely that the samples completely miss the stable plateau in the original graph's resolution profile, and the precision remains 0. On the other hand, in the case of a forward burning probability of 0.6(**Figure 6.12**), the increase in the lengths of *representatives* of the 2nd, 3rd, 4th and 5th longest intervals is always very limited. As a result, when the sample size is as small as 1000 nodes, the increase in the length of the *representative* of the stable plateau leads to an recovery of precision.

6.7 Measurement Results of the Random Node Selection Method

For random node selection, the same things are studied: the length (in log scale) of *representatives* and the precision of estimates of stable plateaus.

6.7.1 Precision of Stable Plateau Estimate

In this section the precision of the estimates of the stable plateau is measured.

Key Findings:

1. The precision of the stable plateau estimate that the Random Node Selection can achieve is very similar to the forest fire sampling algorithm with forward burning probability 0.3.

The experiment is carried out in this way: Use the Random Node Selection algorithm to generate samples of the original graph. Vary the size of samples from 1000 to 9000 with steps of 1000 nodes. For graph sizes from 1000 to 4000 nodes generate

10 samples for each graph size and for graph sizes from 5000 to 9000 nodes generate 5 samples for each graph size. Run the resolution profile construction method on the graph samples and generate one resolution profile for each graph sample. Measure the precision of the stable plateau estimate with respect to the resolution profile of the original graph.

The result is shown in **Figure 6.14**. The horizontal axis shows the size of graph samples and the vertical axis shows the precision of the stable plateau estimate. Each measurement value in the figure for a certain sample size is the average value of all the graph samples of that size.

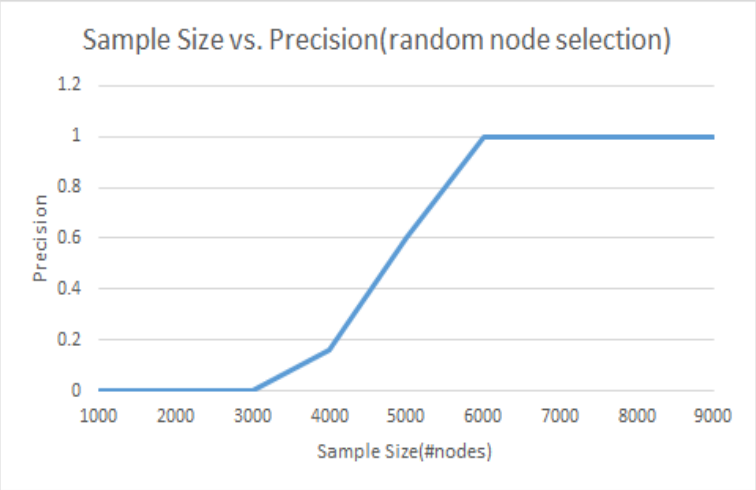

Figure 6.14: Precision of Random Node Selection

One can see that the precision is similar to that of the forest fire sampling algorithm when the forward burning probability is 0.3.

6.7.2 Length of *Representative*

In this section the length (in log scale) of the *representatives* of the top 5 intervals are studied.

Key Findings:

- One can see a similar trend for the random node selection method: As we decrease the size of samples, the length (in log scale) of the *representative* of stable plateau first goes down and then goes up, showing a "U" shape, while the length (in log scale) of the other four intervals goes up in a wave-like manner.

The experiment is carried out in this way:

1. Use the Random Node Selection sampling algorithm to generate graph samples. Use the graph described in **Section 6.5** to generate graph samples with 1000 nodes, 2000 nodes, 3000 nodes and 4000 nodes respectively (10 samples for each graph size) and generate graph samples with 5000 nodes, 6000 nodes, 7000 nodes, 8000 nodes and 9000 nodes (5 samples for each graph size).

2. Run the resolution profile construction method on the graph samples and generate a resolution profile for each sample.

3. In each resolution profile, locate the *representative* intervals of the top 5 interval in the original graph's resolution profile (listed in **Table 6.2**). Measure the length of these intervals.

The result is shown in **Figure 6.15**. The horizontal axis shows the sample sizes and the vertical axis shows the length of the *representatives* in log scale. Each line shows the length of the *representative* of a certain interval in the original graph's resolution profile. In this figure, the measurement value of the length of the *representative* of a sample size is the average value of all the sample graphs of that size.

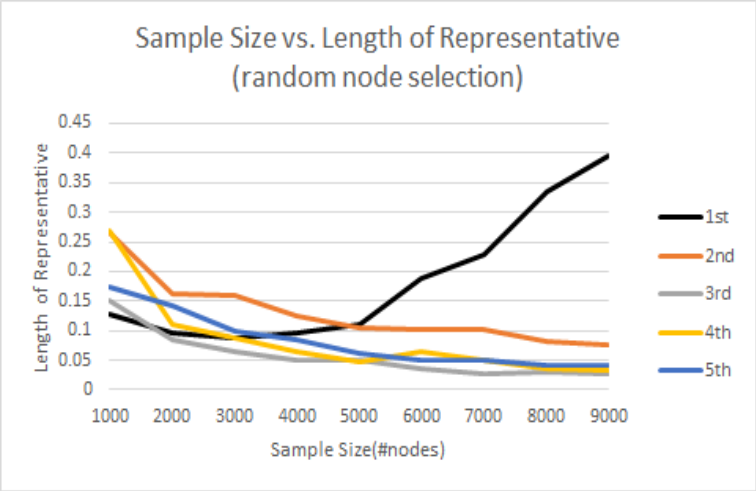

Figure 6.15: Length of *Representative* of Random Node Selection

In general, for the random node selection method, we can see a trend similar to the forest fire sampling method: As we decrease the size of samples, the length (in log scale) of the *representative* of the stable plateau first goes down and then goes up, showing a "U" shape, while the length (in log scale) of the other four intervals goes up in a wave-like manner.

6.8 Comparison of the Two Sampling Methods

In this section, we compare the measurement result of three sampling configurations (random node selection, forest fire sampling with p=0.3 and forest fire sampling with p=0.6).

6.8.1 Comparison of *Representative* Length of the Two Sampling Methods

In this section, we look at how the sampling algorithms behave by comparing the measurement results of different sampling configurations on the same interval in the

original graph's resolution profile. For each interval in the original graph's resolution profile, we show the length of its *representative* in the graph sample's resolution profile, and we compare the result from different graph sampling configurations. The result is shown in **Figure 6.16**, **Figure 6.17**, **Figure 6.18** and **Figure 6.19**. In each figure, the horizontal line shows the size of graph samples and the vertical axis shows the length of *representatives*. Each line corresponds to one graph sampling configuration and each figure shows the result of *representatives* of a certain interval in the original graph's resolution profile.

Key Findings:

- For intervals that are not the longest in the original graph's resolution profile, the Random Node Selection tends to overestimate more than the forest fire sampling algorithm.
- The smaller the forward burning probability, the more the forest fire sampling algorithm will overestimate intervals that are not the longest in the original graph's resolution profile.

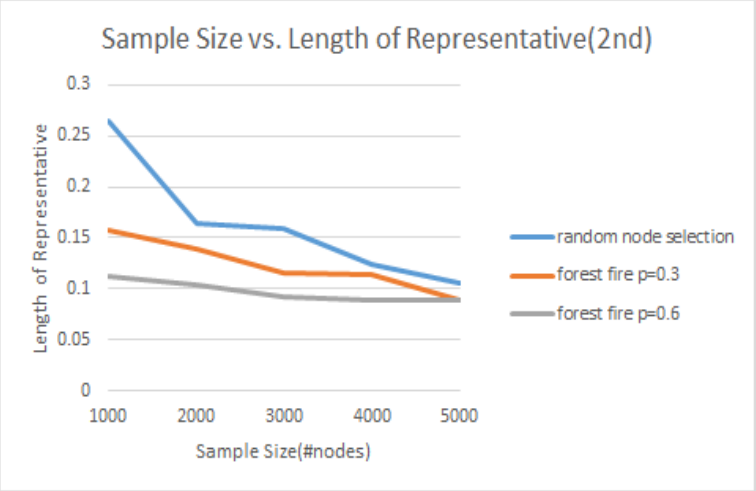

Figure 6.16: Sample Size vs. Length of *Representatives* for Three Sampling Strategies(2nd Longest Plateau)

In practice, the size of samples is always smaller than 50% of the original graph to give a significant reduce in execution time, so in this section we only consider the cases where the sample size is smaller than 50% of the original graph. The findings for the comparison of the two sampling strategies are as follows:

First, when the sample size is 1000 (10% of the original graph), for random node selection, the average length (in log scale) of the *representatives* of all the other four intervals are larger than the average length (in log scale) of the *representative* of the stable plateau in the original graph's resolution profile(**Figure 6.15**), which is worse than in the case of the forest fire sampling with forward burning probability 0.3 (**Figure 6.11**) and forest fire sampling with forward burning probability 0.6 (**Figure 6.12**).

Second, in the plot of the length (in log scale) of *representatives* of 2nd, 3rd, 4th and 5th longest intervals in the original graph's resolution profile (**Figure 6.16, 6.17,**

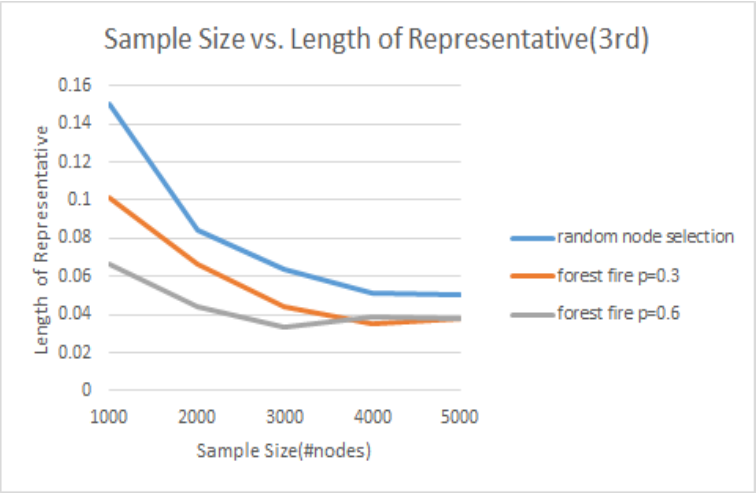

Figure 6.17: Sample Size vs. Length of *Representatives* for Three Sampling Strategies(3rd Longest Plateau)

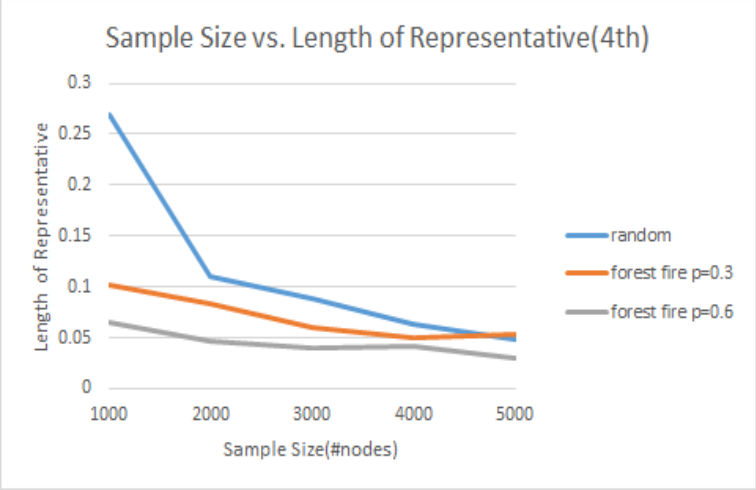

Figure 6.18: Sample Size vs. Length of *Representatives* for Three Sampling Strategies(4th Longest Plateau)

6.18, 6.19), the line of the forest fire sampling method with forward burning probability 0.3 is almost always between the line of random node selection and the line of the forest fire sampling method with forward burning probability 0.6. Moreover, the line of random node selection is always the highest in the plot. This indicates that for the 2nd, 3rd, 4th and 5th longest intervals in the original graph's resolution profile, the random node selection sampling algorithm tends to give a more severe overestimation than the forest fire sampling method with a forward burning probability of 0.3, and the forest fire sampling method with a forward burning probability of 0.3 tends to give a more severe overestimation than the forest fire sampling method with a forward burning probability of 0.6. This is reasonable, because as the forward burning probability of the forest fire sampling algorithm decreases, the behaviour of the forest fire sampling

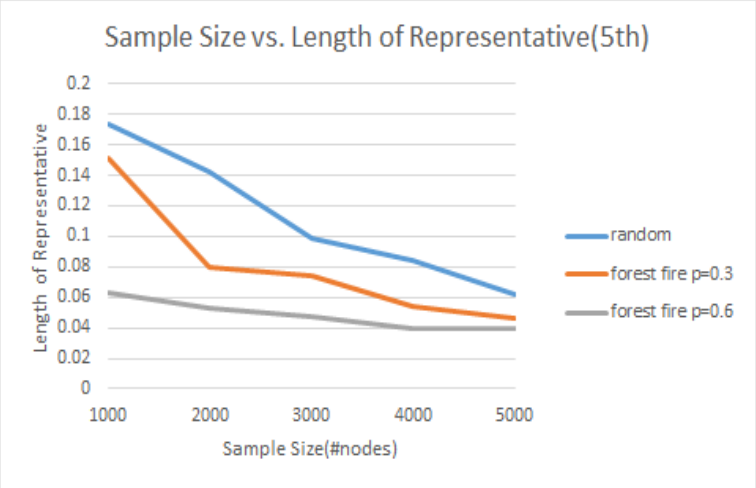

Figure 6.19: Sample Size vs. Length of *Representatives* for Three Sampling Strategies(5th Longest Plateau)

algorithm is closer and closer to random node selection.

6.8.2 Comparison of Precision of the Two Sampling Methods

In this section the two sample algorithms' precision of the stable plateau estimate are compared. The result is shown in **Figure 6.20**. The horizontal axis shows the graph sample sizes and the vertical axis shows the precision of the stable plateau estimate. Each line in the figure shows the precision result of one graph sampling configuration.

Key Finding:

- The forest fire sampling method with a forward burning probability of 0.6 seems to be a better choice when the sample size is small (i.e. the number of nodes is only 10% of original graph), though none of the three configurations can generate a good estimate when the sample size is very small.

In **Figure 6.20**, the horizontal axis shows the sample size and the vertical axis shows the precision of the stable plateau estimate. Each line shows the result of precision for each graph sampling configuration. In this figure only sample sizes smaller than 50% are considered. From the precision of the estimation of stable plateau (**Figure 6.20**), we can see that for the graph we are looking at, none of the three configurations can generate a satisfactory result when the sample size is smaller than 50%.

When the number of nodes is between 4000 and 5000, it seems that the forest fire sampling method with a forward burning probability of 0.3 has better performance, and the precision of the forest fire (p=0.6) and random node selection methods are quite similar. However, due to the instability of the precision of stable plateau estimate, more experiments are needed to draw a valid conclusion. When the number of nodes is around 1000 to 2000, the precision of the forest fire sampling method with a forward burning probability of 0.6 is slightly better than the other two configurations, but it is

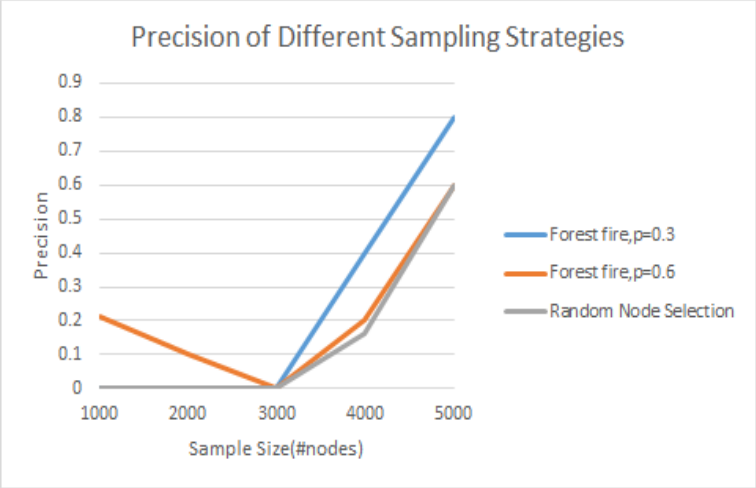

Figure 6.20: Precision for Different Sampling Strategies

still very low. This observation is consistent with what we observe in the result of the lengths (in log scale) of *representatives*. In general, the forest fire sampling method with a forward burning probability of 0.6 seems to be a better choice when the sample size is small (i.e. the number of nodes is only 10% of original graph), although none of the three configurations can generate a good estimate in this size.

For the cases where acceleration of the resolution profile construction method is needed, the number of nodes is usually very large, and 50% is usually still too large. Therefore the superiority of p=0.3 when the sample size is 40% to 50% of the original graph is less important than the superiority of p=0.6 when the sample size is 10% to 20% of the original graph.

6.9 Evaluation of Hypotheses

6.9.1 Validity of Hypotheses

In this section the validity of the four hypotheses proposed is examined using the experiments results.

Hypothesis 1

The original hypothesis:

In general, the accuracy of the forest fire sampling algorithm is at least as good as the random node selection algorithm in estimating stable plateaus in the original graph's resolution profile.

From the perspective of precision, it seems that the accuracy of the plateau estimation of the two configurations of the forest fire sampling algorithm(forward burning probability 0.3 and forward burning probability 0.6) is at least as good as random node selection. When the sample size is large (the number of nodes is more than 50% of original graph), all three configurations have 100% accuracy. When the sample size is around 40% to 50% of the original graph, the precision of the forest fire sampling

method with a forward burning probability of 0.3 is better than the other two configurations. When the sample size is even smaller, the forest fire sampling method with a forward burning probability of 0.6 shows better performance.

So far, there is no evidence in the experiment results which invalidates this hypothesis. However, only two forward burning probabilities are included in the experiments. To draw a solid conclusion, more experiments are needed to see what is happening for other values of forward burning probability.

Hypothesis 2

The original hypothesis:

For the forest fire sampling algorithm, the higher the forward burning probability (but smaller than 1), the more accurate it is in estimating stable plateaus in the original graph's resolution profile.

It is shown in **Figure 6.8** and **Figure 6.20** that this hypothesis is not valid. When the sample size is comparatively large (the number of nodes is around 40% to 50% of the original graph), it is shown in **Figure 6.20** that a forward burning probability of 0.3 has better precision than a forward burning probability of 0.6. The phenomenon may be caused by the instability of the measurement of precision itself (described in **Section 6.4**), but this observation is enough to invalidate the hypothesis.

Hypothesis 3

The original hypothesis:

As sample size decreases, the accuracy of sampling algorithms in estimating stable plateaus decreases.

This hypothesis is invalidated in **Figure 6.8**. As the sample size decreases, we do not see a monotonic decrease of precision. Even taking into account the instability of precision measurement (as described in **Section 6.4**), this hypothesis is unlikely to be true.

Hypothesis 4

The original hypothesis:

The cause of frequent mismatch between stable plateaus in the original graph's resolution profile and stable plateaus in the sample graph's resolution profile is that: the graph samples have underestimated long intervals and overestimated short intervals in the resolution profile of the original graph.

The notion of *representative* is introduced to evaluate this hypothesis. The result of lengths (in log scale) of *representatives* is consistent with the result of precision, which indicates that it is a good choice to interpret the precision change of the stable plateau estimate from the perspective of underestimation and overestimation of intervals in the original graph's resolution profile. In general, this hypothesis is consistent with what we observed in the experiments.

6.9.2 More Hypotheses

Based on experiment results so far, several more hypotheses are proposed, which are described in the following sections. Unfortunately, due to time constraints, I have not been able to conduct more experiments to verify these hypotheses, but they are interesting questions to study in the future.

Hypothesis about Interval Length

Based on this set of experiments for the two sampling algorithms (the forest fire sampling algorithm and the random node selection sampling algorithm) two more hypotheses about the comparison of lengths (in log scale) of *representatives* of intervals in the original graphs' resolution profile are proposed:

1. **Hypothesis 5:** *When the size of samples is smaller than 50% of the original graph, the overestimation of intervals (intervals that are not the longest in the original graph's resolution profile) is more severe in the random node selection algorithm than in the forest fire sampling algorithm. For the forest fire sampling algorithm, the overestimation is more severe in the case of a forward burning probability of 0.3 than in the case of a forward burning probability of 0.6.*

2. **Hypothesis 6:** *As the size of the sample graphs decreases, the lengths of representatives show two different types of trends: the length (in log scale) of the representative of the stable plateau decreases, and when the sample size is as small as 10-20% the length starts to increase again, showing a "U" shape. The length (in log scale) of the representatives of other intervals keeps increasing in a wave-like manner as sample size decreases.*

To validate these two hypotheses, more experiments are needed. Unfortunately, the period of this thesis project is too short to conduct more experiments on the topic of sampling.

Hypothesis about the Accuracy of the Stable Plateau Estimate

Using the previous three hypotheses about length of *representatives*, we can gain a deeper insight into the precision results of the three graph sampling configurations (random node selection and two configurations of forest fire sampling). The fact that the precision for forest fire sampling (p=0.6) achieves a significantly better precision when sample size is small (around 10% of original graph) can be interpreted in terms of the length (in log scale) of *representatives*: when the sample size is as small as 10% of the original graph, the length of the *representative* of the stable plateau starts to increase as the sample size decreases. Its length becomes closer and closer to the length of the longest *representative*. As a result we see an increase of precision as the sample size decreases. In the case of the random node selection and forest fire(p=0.3) method, the lengths of the *representative* of the longest plateau also increases when the number of nodes is around 10% of the original graph. However the overestimation of the other plateaus is so severe that the precision stays 0 for these configurations. Based on this understanding, the following hypothesis is proposed:

Hypothesis 7: *The sampling strategy of the forest fire sampling method(p=0.6) has a much smaller overestimation for intervals that are not the longest in the original graph's resolution profile. Therefore, when the sample size is very small (the number of nodes is around 10% of original graph), this configuration may generate a more accurate stable plateau estimate than the other two configurations.*

Hypothesis about Choice of Sampling Methods

From the result of the length of *representative*, it seems that the forest fire sampling algorithm with a forward burning probability of 0.6 has the smallest overestimation among the three sampling configurations when the sample size is small. This is also indicated in the result of precision. When the sample size is small (the number of nodes is around 10% of the original graph), the precision of the stable plateau estimate of the forest fire sampling algorithm with a forward burning probability of 0.6 is slightly better than that of the other two configurations. Therefore, the following hypothesis is proposed:

Hypothesis 8: *When the number of nodes in graph samples is as small as 20%(or smaller) of the original graph, the forest fire sampling method with a forward burning probability of 0.6 will be the best choice among the three configurations (random node selection, forest fire sampling p=0.3 and forest fire sampling p=0.6).*

6.9.3 Threat to Validity

In fact in this chapter, the experiments I have done are quite limited. No solid conclusion is given.I only propose hypotheses about what could be the case. This is because the topic of graph sampling to accelerate the resolution profile construction method is the second goal of my master thesis, and the topic itself is too time-consuming and complicated to be extensively studied in the limited period of time available for the master thesis project. In fact, to increase the number of experiments, many of the experiments have to be run overnight. In the following, the factors that may harm the validity of the hypotheses will be discussed.

Instability of *Representatives*

There are parts of the result that seem to invalidate the hypotheses.

For example, in the experiment of the forest fire sampling method (with forward burning probability 0.3), when the sample size is 4000, the length of the *representative* of the stable plateau is larger than in the case of 5000 nodes (**Figure 6.11**). This violates the hypothesis that the length of the *representative* of the longest plateau has a "U" shape as the sample size decreases. However, the difference is not significant. In fact, this kind of small "noise" which violates the hypotheses more or less can be found in many figures in the experiments.

There are two possible causes for these "exceptions" in the result: The first cause is that the hypotheses are not correct, and the second cause is the instability of *representative*. In fact, the wave-like appearance of the length (in log scale) of the *representatives* of the 2nd, 3rd, 4th and 5th longest intervals when the sample size decreases are suspicious. The "wave" also indicates that the *representatives* of the intervals are not stable. To determine which is the actual cause, more experiments are needed.

Oversimplified Interpretation

The definition of *representative* is based on the third criterion (described in **section 6.4.2**). It is possible that the interpretation of the notion of *representative* is oversimplified and in fact more complicated things are taking place. For example, if two intervals *A* and *B* share the same *representative* in the resolution profile of a sample, can we simply say "this implies that these two intervals in the original graph's resolution profile are 'merged' in the sample graph's resolution profile"? This is in fact questionable: it is likely that this *representative* has a very small intersection with both *A* and *B*. In this case, the indication of "merging" is not valid.

However, as is discussed in **section 6.4.2**, in most cases the *representative* chosen by the first criterion and the third criterion are the same. Even if there is a difference, it is small. This indicates that even though the third criterion does not require the intersection to be long, the *representative* chosen by the third criterion has a similar length as the *representative* chosen by the first criterion. Therefore, this problem does not have a significant influence on the comparison of the lengths of *representatives*.

Characteristics of the Resolution Profile of the Original Graph

Particular characteristics of the resolution profile may invalidate the hypotheses. For example, intervals that are closer to 1 may be less "damaged" by the decrease of the graph size. As a result, if the stable plateau is close to 1, for example (0.5,1), it is possible that the length of the *representative* of the stable plateau in the original graph's resolution profile continuously increases as the sample size decreases, instead of showing a "U" shape.

Graph Configuration

The graph samples studied in this chapter are all generated from one graph. The difference in graph types and configurations is not taken into account in the experiments. In fact, different graph configurations and different graph types may invalidate the hypotheses.

First, if the graph is generated from the Erdős-Rényi model, it is possible that the *representative* of the longest stable plateau behaves in a different way as sample size decreases. It is also possible that you cannot find a clear pattern in the change of the length (in log scale) of the *representatives* as sample size decreases.

Second, if I choose different values for the minimum and maximum community size, the stable plateaus in the resolution profile of the graph may change. It is possible that the length of their *representatives* may change in a different way as sample size decreases, which invalidates the hypotheses for the length of *representatives* and the precision of stable plateau estimation.

6.10 Summary

In this chapter the possibility to accelerate the resolution profile construction method proposed in [31] by graph sampling was explored. First, four hypotheses about sampling methods were proposed. Based on these hypotheses, two different metrics, precision and length (in log scale) of *representative*, were proposed in order to measure

the quality of the resolution profile of graph samples. With these two measures the accuracy and behavior of the resolution profile estimation of two sampling methods (the random node selection method and the forest fire sampling method) were compared. Based on the result of the experiments, the four hypotheses were evaluated and more hypotheses were proposed. Lastly the threats to the validity of these hypotheses were discussed.

Key Findings:

- When the sample size is small (the number of vertices is around 10% of the original graph), neither the Random Node Selection nor the forest fire sampling algorithm can generate graph samples with resolution profiles which can accurately reveal the position of stable plateau in the original graph's resolution profile.
- The forest fire sampling algorithm with p=0.6 can generate slightly better results in estimating the stable plateau in the resolution profile of the original graph, because compared with the other two configurations (The Random Node Selection method and the forest fire sampling method with p=0.3), it has less overestimation for the intervals that are not the longest in the original graph's resolution profile.

Chapter 7

Conclusions and Future Work

This chapter first gives an overview of the contributions of this master thesis project and then presents answers to the research questions of the project. Next the limitations of our approach are discussed. The chapter is concluded with discussions about possible future research directions.

7.1 Contributions

The first contribution of this thesis is that a synchronous version of the Louvain community detection algorithm, which is inspired by the idea of the gossip algorithm, is proposed. Moreover, the convergence of this algorithm is proved using the model of the absorbing Markov chain. Then the convergence speed of this algorithm is discussed. The upper-bound for the average number of iterations to converge under a very specific condition is given. Although this is not a strict upper-bound, it does give us an idea about the speed of convergence of this algorithm.

Then a comprehensive experimental evaluation for the proposed distributed Louvain algorithm is performed, which reveals the performance characteristics of this algorithm. Specifically, this experiment shows how the mixing parameter (which controls how clear the community structure is in a graph) and the update probability (the probability for a node to join a locally optimal community) influence the number of iterations for the algorithm to converge and the quality of partitions generated. There are some interesting observations in the experiments. First, when the mixing parameter is less than 0.5, the number of iterations to converge is almost the same for the same update probability. Second, the update probability influences the number of iterations to converge, but it does not affect the quality of the partitions generated. Third, to reduce the number of iterations to converge, we do not need to arbitrarily increase the update probability, because when the update probability is higher than 0.5, the number of iterations to converge is almost the same. Also, it seems that the number of iterations to converge is acceptable in most cases. It has been proved that the Distributed Synchronous Louvain community detection algorithm does converge in a finite number of steps, but we have no estimate about what the maximum number of iterations it takes. From the experiments it seems that even for the most "difficult" graphs (graphs with a mixing parameter of 0.9), the program can terminate in around 100 iterations. With larger graphs, this number may be larger but it is still acceptable.

The second contribution of this thesis is that the possibility to speedup the resolution parameter selection with graph sampling is explored. Two sampling strategies, the forest fire sampling algorithms and the random node selection algorithm, are compared. Hypotheses on the difference between these two sampling algorithms are proposed and measurements to quantify the behavior of these sampling algorithms are devised based on these hypotheses. In the experiments it seems that the forest fire sampling algorithm with a forward burning probability of 0.6 is more likely to generate better graph samples when the sample size is small. However, when the number of nodes of graph samples is less than 50% of the original graph, neither of the two sampling strategies is able to give satisfactory results, even though the forest fire sampling algorithm with a forward burning probability of 0.6 generates a slightly better result.

7.2 Answer to Research Questions

Question 1: *How to distribute the execution of the Louvain community detection algorithm using big data tools such as Apache Spark?*

The original sequential Louvain community detection algorithm looks at nodes in the graph one by one. Each node joins one of its neighbouring communities which gives a maximum increase of modularity. If no positive increase of modularity is possible, the node will stay in its original community. This algorithm can be distributed by introducing the idea of the gossip algorithm: in each iteration, all nodes make their decisions to join one of their neighbouring communities or stay in its original community. Each node has a probability p to join the community that gives a maximum increase of modularity locally and a probability $1-p$ to do nothing. We can implement this synchronous algorithm using Apache Spark. We then have a distributed version of the Louvain community detection algorithm.

Question 2: *How is the performance of the distributed version in comparison with original sequential Louvain method? What characteristics does the distributed version shows in execution?*

In terms of the quality of partitions generated, the Distributed Synchronous Louvain algorithm is very similar to the original sequential Louvain algorithm. The Distributed Synchronous Louvain algorithm has some very interesting characteristics. First, the update probability (the probability that a node will join the neighbouring community or its own original community that gives the maximum increase of modularity) influences the number of iterations for the algorithm to converge, but it does not influence the quality of partitions generated. Second, for the LFR benchmark, when the mixing parameter is less than 0.5 and all the other configurations are the same (such as number of nodes, minimum and maximum communities sizes), the number of iterations for the algorithm to converge is almost the same , no matter what the mixing parameter is.

Question 3: *To accelerate the resolution parameter selection of the CPM community detection algorithm, one possibility is to take a sample of the graph and select resolution parameters based on this sample. To do so, the sample graph should give us a good estimation of the resolution profile of the original graph. Among the many graph sampling strategies, which is the best one to use, so that the estimation of the resolution profile[31] of the original graph is as good as possible?*

In the thesis, two sampling algorithms, the forest fire sampling algorithm and random node selection, are compared. Specifically, three graph sampling configurations are compared: random node selection, forest fire sampling with a forward burning probability of 0.3 and forest fire sampling with a forward burning probability of 0.6. It seems that the forest fire sampling algorithm with a forward burning probability of 0.6 gives the smallest overestimation for intervals in the original graph's resolution profile which are not the stable plateau. As a result, it seems that this configuration is able to provide more reliable results for stable plateau estimation when the sample size is small. However, in general none of the three configurations (random node selection and the forest fire sampling with a forward burning probability of 0.3 or 0.6) give very good results for the stable plateau estimation when the sample size is very small (as small as 10% of the original graph).

7.3 Limitations

For the experimental evaluation section (described in **Section 5**), it would be more convincing if more graphs were generated for each mixing parameter (not just one graph is generated for each mixing parameter).

The second limitation is that the types of graphs that are used for testing are quite limited. For the experimental evaluation for the Distributed Synchronous Louvain method, only undirected binary networks are used. In fact, the implementation of the Distributed Synchronous Louvain method in this thesis can work on undirected weighted networks. It would be nice if weighted networks are included in the experiments.

The third limitation is that in the study on graph sampling, only samples from one graph are studied and only two forward burning probabilities of forest fire sampling are included in the experiment. In fact, as is mentioned in **Section 6.9.3**, these might be very important factors that may harm the validity of conclusions.

7.4 Future work

This thesis has looked into two topics: the distribution of the Louvain method and using graph sampling to accelerate the resolution parameter selection for the CPM method. In the following, directions for future research are given.

For the study of the Distributed Synchronous Louvain method, the following issues can be studied in the future:

1. Apply this algorithm to larger real-world graph datasets and verify the hypotheses proposed for the Distributed Synchronous Louvain method. The Distributed Synchronous Louvain method is proposed with a practical purpose, and applying this algorithm on larger real-world graphs can show its usefulness for real-world problems.

2. Extend the experiments to weighted networks. The implementation of the Distributed Synchronous Louvain method can work on weighted undirected networks, however in this thesis, the experiments were carried out only on undirected binary networks. This could be done in the future.

For the study of using graph sampling to accelerate the resolution parameter selection of the CPM community detection method, the following things can be done in the future:

1. For now the graph samples are generated from one single graph. Due to time constraints for this master thesis project, no more graphs are included in the experiments. To further verify the validity of hypotheses proposed in this thesis, more graphs of different types need to be studied in the future.

2. As is suggested in [31], the resolution profile of the Erdős-Rényi random graph has a special characteristic: the density of the graph corresponds to the transition point in the resolution profile (the place where the curve of N/n^2 flattens. Here n is the number of nodes and N is the sum of squares of community sizes). It would be interesting to study the influence of graph sampling strategies to the resolution profile of samples of Erdős-Rényi random graph.

Bibliography

[1] Lada A Adamic, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. Search in power-law networks. *Physical review E*, 64(4):046135, 2001.

[2] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *arXiv preprint arXiv:1211.3412*, 2012.

[3] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[4] Ginestra Bianconi and A-L Barabási. Competition and multiscaling in evolving networks. *EPL (Europhysics Letters)*, 54(4):436, 2001.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[6] Cumulative Advantage Distribution CAD. A generai theory of bibiiometric and other cumulative advantage processes. *Journal of the American society for Information science*, page 293, 1976.

[7] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005.

[8] J-C Delvenne, Sophia N Yaliraki, and Mauricio Barahona. Stability of graph communities across time scales. *Proceedings of the National Academy of Sciences*, 107(29):12755–12760, 2010.

[9] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[11] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[12] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[13] Roger Guimera, Stefano Mossa, Adrian Turtschi, and LA Nunes Amaral. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proceedings of the National Academy of Sciences*, 102(22):7794–7799, 2005.

[14] Roger Guimera, Marta Sales-Pardo, and Luís A Nunes Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004.

[15] Hawoong Jeong, Bálint Tombor, Réka Albert, Zoltan N Oltvai, and A-L Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000.

[16] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.

[17] Andrea Lancichinetti and Santo Fortunato. Consensus clustering in complex networks. *Scientific reports*, 2, 2012.

[18] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008.

[19] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

[20] Atieh Mirshahvalad, Olivier H Beauchesne, Eric Archambault, and Martin Rosvall. Resampling effects on significance analysis of network clustering and ranking. *PloS one*, 8(1):e53943, 2013.

[21] Atieh Mirshahvalad, Johan Lindholm, Mattias Derlen, and Martin Rosvall. Significant communities in large sparse networks. *PloS one*, 7(3):e33721, 2012.

[22] Mark EJ Newman. Analysis of weighted networks. *Physical Review E*, 70(5):056131, 2004.

[23] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[24] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-LLuis Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd international conference on World wide web*, pages 225–236. ACM, 2014.

[25] Jörg Reichardt and Stefan Bornholdt. Partitioning and modularity of graphs with arbitrary degree distribution. *Physical Review E*, 76(1):015102, 2007.

[26] Bruno Ribeiro and Don Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 390–403. ACM, 2010.

[27] Peter Ronhovde and Zohar Nussinov. Multiresolution community detection for megascale networks by information-based replica correlations. *Physical Review E*, 80(1):016109, 2009.

[28] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[29] ChengTe Li ShouDe Lin, MiYen Yeh. Sampling and Summarization for Social Networks. `http://www.siam.org/meetings/sdm13/lin.pdf`, 2013. [SDM 2013 Tutorial].

[30] VA Traag. Faster unfolding of communities: speeding up the louvain algorithm. *arXiv preprint arXiv:1503.01322*, 2015.

[31] Vincent A Traag, Gautier Krings, and Paul Van Dooren. Significant scales in community structure. *Scientific reports*, 3, 2013.

[32] Vincent A Traag, Paul Van Dooren, and Yurii Nesterov. Narrow scope for resolution-limit-free community detection. *Physical Review E*, 84(1):016114, 2011.

[33] James M Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its Applications*, 11(1):3–5, 1975.