DEPARTMENT OF COMPUTER SCIENCE

Statistically significant communities in networks

Xuejun Cao

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

September 2011 | CSMSC-11

Declaration

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Xuejun Cao, September 2011

Acknowledgements

I would like to give thanks to all the people who have been involved in the success of this project. In particular I would like to offer a special thanks to the following people:

Steve Gregory, be as the supervisor of me, and giving me technical suggestions.

Bowen Yan, who provide me vital technical support when met difficulties and save me from wrong research directions.

Ian Holyer and **Julian Gough**, who marked and gave feedback to my interim report, which directed me a right approach to reach my goals.

Erik Reinhard, who in charge of the unit COMSM3100 and gave instructions and support the successfulness of the project.

Xuejun Cao, September 2011

Executive Summary

Objectives

Network can be found everywhere in the real or virtual world. A Community in a network is groups of vertices that are relatively densely connected by edges, which different to the rest of the network. Detect the communities in network efficiently and automatically could be very helpful to statistical analysis whatever in the filed of biology, ecology, society or the virtual reality on Internet that access into human life and make regardless influence on our daily behavior.

There was lots of method was proposed to detect the communities in graphs. However, most of the methods cannot correctly recognize the cluster with appropriate vertices. In some cases, a vertex may not only belong to one cluster, it can be share between several overlapping clusters. In a hierarchical network, most algorithms just focus on the partition of network but ignored the hierarchy structure and its corresponding level. Furthermore, a vertex can be possibly “homeless”, which means it is not included in any cluster. However, most of the existed algorithm will consider it as a part of a cluster. On the other hand, in a random graph, actually, there was no community on the network, whatever the algorithms take each vertex as a single cluster or it take all vertices in a big communities, that was not correct. To avoid these problems, a recent solution is to detect the communities that only considered as statistically significant. So the vertices in a network can be "homeless" (not belonging to any community).

In this project, it aims to compare these statistically significant communities with the communities found by traditional algorithms.

Project Type

This is a completely research project, which will focus on the existed methods and algorithms. It contains a literature review relevant to research topics, followed by a designed experiment to proof the correctness of the thesis/methods. It will be nothing relevant to software/hardware development or model/theories development. All the algorithms and the majority of the codes were taken from other researcher’s implementation.

Implementations

To compare the traditional algorithms with the method that concerned as statistically significant, there’s some research been made to help to made decision of the research environment. After the experiment plan been set, the individual codes for each algorithm has been collected and analyzed. The parameters for each algorithms been planed and set to the methods. Afterwards, the testing been coded to be run automatically without too much manual operations and supervisions, which carried out by shell script. Finally, the results been reformatted, produced to diagrams and been analyzed to draw conclusion of the project.

Excited Elements

a. The width and depth of the research on background technologies
b. A fluent design and realization of experiment
c. Comprehensive understand to the techniques that used in this project
d. Analysis for the experiment data with reasonable conclusion
e. Independent complement of the project and thesis

Table of Contents

Chapter I

1. Introduction

This paper is part of the project Statistically Significant Communities in Network, which complete by Xuejun Cao (1036129), supervised by Steve Gregory. In this section it will review the past stages of this project, describe the design and implementation of the project, and finally give a self-evaluation and conclusion of the whole project.

1.1 Background

To uncover the communities in the network is a hot topic due to the complexity of the network continuous grows. There is lots of methods have been developed to discovery communities, most of them can be considered as traditional algorithms, which considering on their building thesis, they can be classified as the method base on modularity maximization, betweeness, label propagation, clique percolation and fitness function maximization etc.

Recently, a new research was been public, it announced a new algorithm build on statistically significance, which called Order Statistic Local Optimization Method, also as known as OSLOM. Different with the traditional algorithms place every vertices into at least one community, which may not appropriate, even when there's no communities on the network. OSLOM can distinguish the statistically significant communities, and recognize homeless vertices, which not belongs to any community.

1.2 Objectives

In this project, it will compare OSLOM with other appropriate selected algorithms through the communities they have detected. By judging under certain criteria, to find out on what circumstance, which algorithms can do better on communities detecting. In this case, statistically significant method (i.e. OSLOM), or traditional methods? Is that true that statistically significant algorithms can always have advantages to obtain better result of communities than traditional algorithms? Is that any exception existing that OSLOM failed to detect communities?

1.3 Overview

This project can be divided in following stages,

- Preparation, which includes document and thesis research, a project management plan also been made in this phase. Finally, a literature review was given as a conclusion of this phase. This stage was mainly focused on theory.
- Design, which focus on the practical experiment. The feasibility of the theory had been checked, and a detailed experiment plan has been set. The parameters/ input data had been prepared to use.

- Experiment: which is a practical phase that base on all the preparation made in previous stages. The majority work of this stage is running testing and coding to support the experiment. E.G. testing automation etc
- Analysis: after all the result data came out, the raw data had been processed to obtain a conclusion to support this project.
- Conclusion: which is a review of the whole project from the beginning to the end, an evaluation will be given at the end of this paper.

From this paper, in every chapter, a list of abstract of contents can be found below as a reading guidance.

Chapter 1: the introduction and overview of the whole project. The target of this project was clearly highlighted in this section.

Chapter 2: this is a literature review of the topic I was doing. It contains the brief introduction of the relevant theories and implementations that I would reference or take and use in my own project. It covers the materials I was researched on and stated in logic sequence from the catalogued these methods to how to evaluate these methods.

Chapter 3: described the design of the experiment. It stated what have been tested and what not been tested and why. The reasons of why chose the certain methods to been tested through and why used the chosen techniques and tools also gave in this section.

Chapter 4: review of how the testing went through, what have done to support the testing running successfully and what is the condition/environment the testing ran through.

Chapter 5: it shows how the result data been analyzed. It gave the conclusion that obtained from the analyzed result. And how well does the conclusion work for the objectives of this project.

Chapter 6: in this section, an evaluation of the project will be given. It also contains a self-performance evaluation of this project. It will also provide the criteria of how the evaluation been made.

Chapter 7: in the last chapter, it is a conclusion for the whole project. It shows a reflection of this project with the summary of what have done and what had learned. By doing this project, what abilities and knowledge has gained from it.

Chapter II

2. Literature Review

2.1 Introduction

Communities Detection in Graphs

The learning about graphs and their mathematical properties become extremely useful as the representation of a wide variety of systems in different areas since 20 century. For example, in biological, ecological, social, technological, and information networks, all of these topics can be studied as graphs, and graph analysis has become crucial to understand the features of these systems. In the real networks, it can potentially be extremely huge and complex, with millions or even billions of vertices.

The aim of community detection in graphs is to identify the modules and its hierarchical organization by only using the information encoded in the graph topology. There has been lot of approaches been proposed to handle these graphs. It will be discussed in details in the following sections.

2.1.1 Community Basis

- ***Communities:*** it also called clusters or modules, are groups of vertices, which probably share common properties and/or play similar roles within the graph.
- ***Community structure:*** also known as clustering. The real network reveals big inhomogeneities and high level of order and organization. The degree distribution of this kind of network is broad, with a tail that often follows a power law: hence, many vertices with low degree coexist with some vertices with large degree. Moreover, the distribution of the edges is not only globally, but also locally inhomogeneous, with high concentrations of edges within special groups of vertices, and low concentrations between these groups. [1-4] However, a community structure is clustering, a clustering possibly not a community structure.
- ***Vertex:*** a reduced simple point, it is the elementary units of system [5]
- ***Edge:*** the pairwise relationship/interaction between vertices [5]
- ***Modularity:*** used to measure quality of partition [6]

2.1.2 Community Structures

2.1.2.1 Disjoint: every partition in this network is a set of disjoint communities.

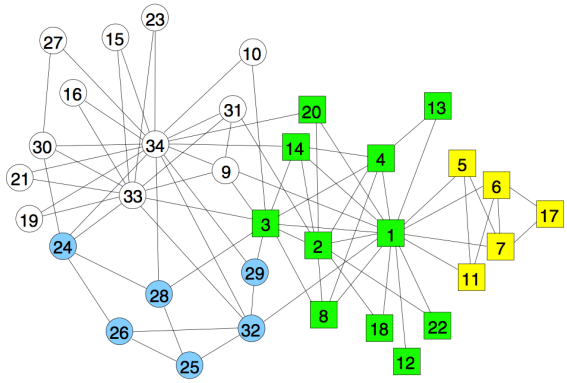


Figure 1 Splitting of the Zachary club network. Squares and circles indicate the two communities observed by Zachary [7]

2.1.2.2 Overlapping

In overlapping communities, a vertex could be assign to not only one, but to a few communities. As the figure shows below, in a word association network, a word is a vertex, and it can possibly belong to several classification of words.

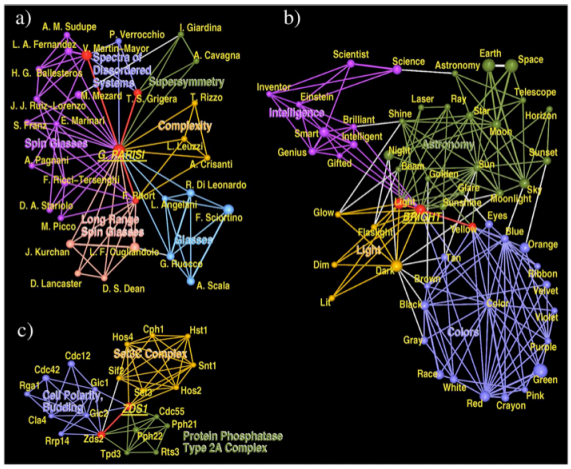


Figure 2 Overlapping communities in a network of word association. The groups, labeled by the colors, were detected with the Clique Percolation Method by Palla et al. (Section 11.1). [8]

2.1.2.3 Hierarchical

Hierarchical communities is a micro-communities nested in a macro-communities. It can have different levels of communities from bottom to top, one include another.

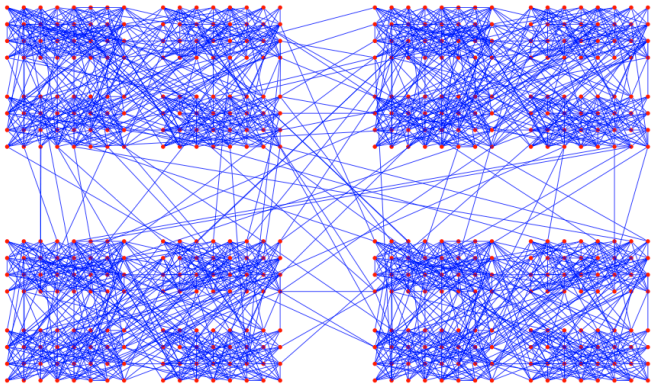


Figure 3 Schematic example of a hierarchical graph. Sixteen modules with 32 vertices each clearly form four larger clusters. All vertices have degree 64. [8]

2.1.2.4 Ordered

Communities are in order. The intercommunity edges are more likely between neighboring communities.

2.2 Problems Outlines

1. Many methods are designed to find clusters in undirected graphs, it cannot be easily, and sometimes it is impossible to be extended to directed graphs.
2. Similar as the problem state above, when edges carry weights, indicating the strength of the interaction/affinity between vertices, many methods cannot handle this even as extension.
3. In overlapping communities, a vertex may be shared by more than one cluster. There is very few methods account for this possibility. So how the shared vertices ca be appropriate handled.
4. In a hierarchical network, there should be clusters nested in another cluster. Most community finding methods typically focus on the "best" partition of a network, disregarding the possible existence of hierarchical structure [5]. The algorithms should be able to find out the hierarchy structure and its corresponding levels.
5. In some cases, there will be some vertices that not belong to any clusters, which considered as the noise vertices in communities detecting. Like in a random graph, if the linking probability is the same for all pairs of vertices, it will not be any real communities within the graphs, how to distinguish communities from pseudo-communities.
6. Detect communities in the dynamic network structure, which very little is known [5]. How to detect the dynamic communities also is a highlighted problem that intends to find an efficient solution in this project.

2.3 Methods and Algorithms

In this section, it will discuss the methods to detect communities in graphs. The methods that list below are roughly classified into two catalogues. First are the traditional methods, which include the most popular or typically methods used on community detection. The second catalogue oriented to the methods that based on the recent proposed idea, which to detect the communities that only considered as statistically significant. In this catalogue, it will only discuss Order Statistical Local Optimization Method (OSLOM), which is the only method that was found with the feature that mentioned above.

2.3.2 Traditional Methods

2.3.2.3 Modularity Maximization

Modularity is known as a measure to the partition quality. So if high values of modularity indicate good partitions, we can find the best partition by maximizing modularity.

Problems with modularity:

1. Modularity peaks at a small number of communities because of resolution limit. [11]
2. Network has an exponential number of distinct partitions with modularity near maximum. It is impossible to find global maximum. [12]

- **CNM Algorithm**

CNM algorithm is a typical modularity maximization method, which was proposed by A. Clauset, M.E.J. Newman and C. Moore [9]. It is a greedy technique to detect communities.

It starts with n communities, each of the communities with one vertex. For each pair of communities joined by an edge, the value of ΔQ, which stands for the increase of modularity if the communities will be merged, will be computed. Then, merge the two communities with the max value of ΔQ that computed from the previous step. Finally, repeat the two steps that mentioned above until the desired number of communities obtained.

The running time of this method is $O(n \log^2 n)$. The complexity of the algorithm is $O(md \log n)$, d is the depth of the dendrogram describing the successive partitions that found during the execution of the algorithm. This algorithm is allowed to analyze the community structure of very large graphs, up to 10^6 vertices.

The code can be freely downloaded from http://cs.unm.edu/~aaron/research/fastmodularity.htm.

- **CliqueMod Algorithm**

In CNM algorithm, because of the initial communities are singletons, so the first few steps are random. It was considered as a problem of CNM algorithm.

CliqueMod algorithm is a community detection algorithm that combines the concepts of cliques and modularity optimization [10]. It consists of two phases: first is "find the cliques", the network will be partitioned into a set of disjoint communities which are cliques, larger cliques are preferred; second, the communities found from the first phase will be merged by using a hill-climbing greedy algorithm to maximize the modularity of the partition.

The result of CliqueMod algorithm was proved better than CNM algorithm and most of the modularity-maximizing algorithms that exist.

2.3.2.4 Betweenness

Betweenness is a variable expressing the frequency of the participation of edges to a process. Edge betweenness is the number of shortest paths between all vertex pairs that run along the edge. [14]

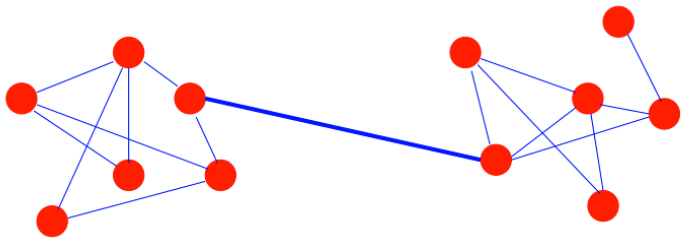


Figure 4 Edge betweenness is highest for edges connecting communities. In the figure, the edge in the middle has a much higher betweenness than all other edges, because all shortest paths connecting vertices of the two communities run through it. [13]

- **Givan-Newman Algorithm [15]**

Givan-Newman algorithm was a method proposed by Girvan and Newman. It is a greedy division algorithm and it is the most popular algorithm. It use following steps to detect communities,

1. Calculate the betweenness of all the edges within the network
2. Remove the edge with the highest betweenness obtained from the above step.
3. Recalculate thee betweenness for all edges on the running graph
4. Repeat the cycle from the second step until no edges remain.

The run time of this algorithm is $O(n^3)$.

- **CONGA Algorithm [16]**

CONGA is short for Community Overlap NG Algorithm. It is an extension of GN algorithm that described above to detect overlapping communities.

If the vertex has high split betweenness, which is the number of shortest paths that would run between two parts of a vertex if the latter were split, it will be split between two communities.

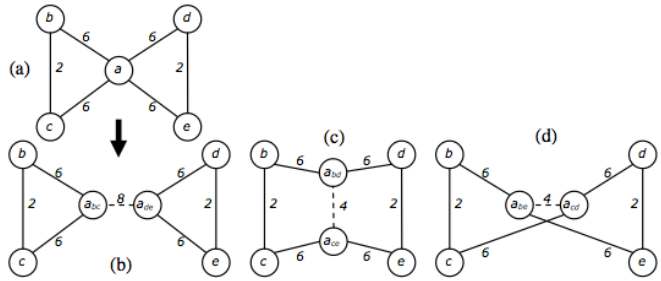


Figure 5 (a) Network. (b) Best split of vertex a. (c), (d) Other splits of vertex a. [16]

The code of this method can be found at http://www.cs.bris.ac.uk/~steve/networks/index.html.

2.3.2.5 Label Propagation

- **RAK Algorithm**

In the label propagation method designed by Raghavan et al.[17], firstly, every vertices will initially be assigned a unique identifier. Then replaces the vertex's label by the label used by the majority of its neighbors. If there is no unique majority, it will randomly pick one majority. Repeat the second step until convergence.

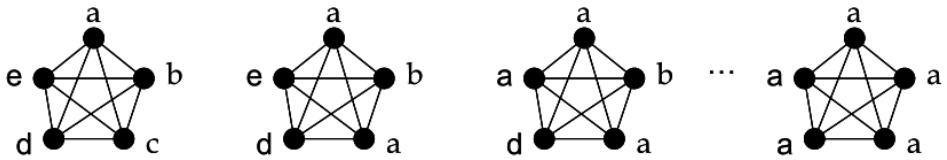


Figure 6 Nodes are updated one by one as we move from left to right. Due to a high density of edges (highest possible in this case), all nodes acquire the same label. [17]

The running time of each iteration is O(m).

- **COPRA Algorithm [18]**

COPRA is the short for Community Overlap Propagation Algorithm. It extends the label propagation algorithm to detect overlapping communities.

In this algorithm, it labels each vertex x with a set of pairs (c,b) corresponding to the figure shows below, where c is a community identifier and b is a belonging coefficient, indicating the strength of x's membership of community c, then all belonging coefficients for x are sum to 1. Each propagation step would set x's label to the union of its neighbours' labels, sum the belonging coefficients of the communities over all neighbours and normalize.

It can be assuming as a function bt(c,x) that maps a vertex x and community identifier c to its belonging coefficient in iteration t, where N(x) denotes the set of neighbours of x.

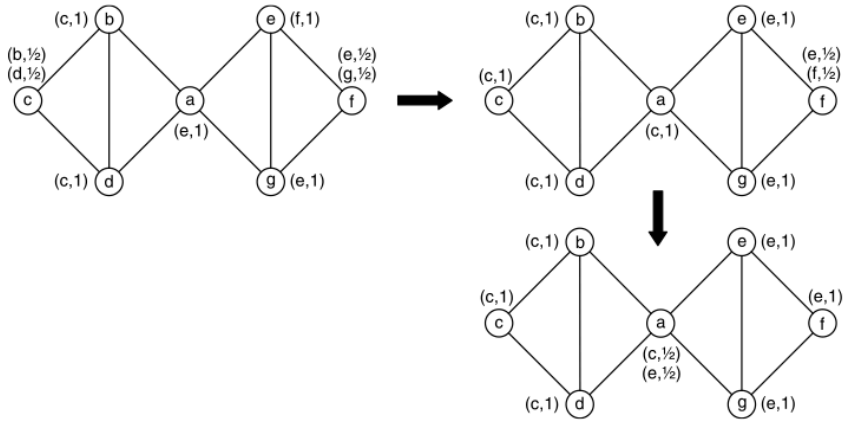


Figure 7 COPRA gives vertex a set of labels and belonging coefficiency.

2.3.2.6 Random Walks

Random walks [19] can also be useful to find communities. It spends a long time inside a community if a graph has a strong community structure.

- **Walktrap Algorithm** [20]

Walktrap is a measure of similarities between vertices based on random walks. It has several important advantages,

1) it captures well the community structure in a network;
2) it can be computed efficiently;
3) it can be used in an agglomerative algorithm to compute efficiently the community structure of a network.

The algorithm of walktrap can be described as the steps below,

1. Calculate P^t_{ij}, the probability of reaching j from I in t steps.
2. Calculate distance r_{ij} between all pairs of vertices:

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{ik} - P^t_{jk})^2}{d(k)}} = \left\| D^{-\frac{1}{2}} P^t_{i\bullet} - D^{-\frac{1}{2}} P^t_{j\bullet} \right\|$$ [20]

3. Start with n communities each with 1 vertex.

4. For each pair of communities: compute mean distance between each vertex and its community if they were merged.
5. Merge the two communities that minimize this distance.
6. Repeat from step 4.

It is a greedy agglomerative algorithm, which runs in time $O(mn^2)$ and space $O(n^2)$ in the worst case, and in time $O(n^2 \log n)$ and space $O(n^2)$

The software of the algorithm can be found at http://www-rp.lip6.fr/~latapy/PP/walktrap.html.

- **Infomap** [21]

Infomap is an information theoretic approach, which can be used to uncover community structure in weighted and directed networks. It use the probability flow of random walks on a network as a proxy for information flows in the real system and separate the network into modules by compressing a description of the probability flow. The result comes out as a map that both simplifies and highlights the regularities in the structure and their relationships.

2.3.2.7 Clique Percolation

- **Clique Percolation Method (CPM)**

It is based on the concept that the internal edges of a community are likely to form cliques due to their high density.

It uses the term k-clique to indicate a complete graph with k vertices. K-clique community is the largest subgraph of adjacent k-vertex cliques. Two k-cliques are adjacent if they share k − 1 vertices. K-clique communities are naturally overlapped.

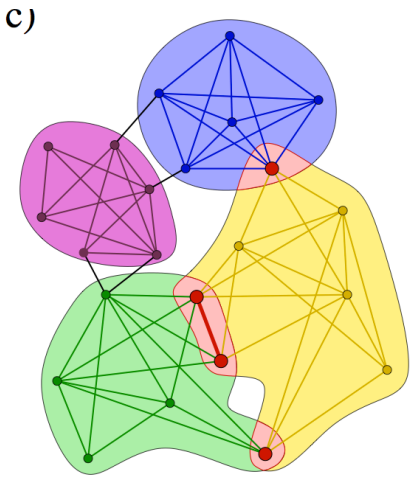


Figure 8 An example of overlapping k-clique-communities at k = 4. The yellow community overlaps with the blue one in a single node, whereas it shares two nodes and a link with the green one. These overlapping regions are emphasised in red. Notice that any k-clique (complete subgraph of size k) can be reached only from the k-cliques of the same community through a series of adjacent k-cliques. Two k-cliques are adjacent if they share k – 1 nodes

The implementation of CPM, which called CFinder is freely available at www.cfinder.org

2.3.2.8 Fitness Function Maximization

Fitness function maximization required finding local maximum of a fitness function. For the communities, the fitness is:

$$f_G = \frac{k_{in}^G}{\left(k_{in}^G + k_{out}^G\right)^\alpha}$$

where in this function, G stand for a community, k_{in}^G a internal degree of G, k_{in}^G is external degree of G. For the global maximization, in f_G, G is the whole network.

- **LFM Algorithm** [8]

LFM algorithm can be used to detect natural community. It can detect community follow the steps below,

1. Look at all the neighbours of the vertices of G but not in G.
2. If all the neighbours have negative fitness, progress finish.
3. Otherwise, add the neighbours with large fitness into G.
4. Look through all vertices that within G.
5. If a vertex has negative fitness, remove it from G and repeat step 4.
6. Repeat from the beginning.

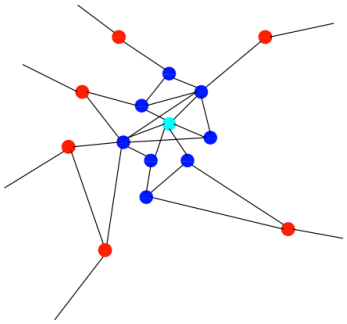


Figure 9 In this figure, the blue nodes are the vertices inside the community. Choose any vertex not yet assign to the community, i.e. red nodes in this case. Then find natural community of the node choose from previous step, including the vertices already assigned to communities. Repeat from beginning.

The running time of LFM is $O(n^2 \log n)$.

- **GCE Algorithm** [24]

GCE is short for the algorithm Greedy Clique Expansion. It identifies distinct cliques as seeds and expands these seeds by greedily optimizing a local fitness function.

1. Find all the maximal cliques.
2. For each maximal clique C, expand it to its natural community using LFM technique.
3. Then repeat the second step.

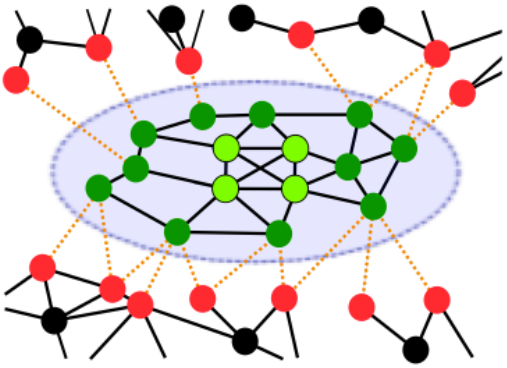


Figure 10 An idealized representation of a community, showing the seed clique in the center, surrounded by nodes added through greedy expansion. The external edges, shown dashed, connect the community to its frontier.

2.3.2.9 Model Based Method

- **MOSSES** [23]

MOSES is a scalable algorithm, which based on a statistical model of community structure that with distinct ability on detecting highly overlapping community structure in the large network. It is also similar with modularity-based algorithm that can be extended as MOSSES maximization algorithm.

The source code of MOSSES can be found at http://sites.google.com/site/aaronmcdaid/moses.

2.3.3 OSLOM (Order Statistic Local Optimization Method) [5]

OSLOM was a statistically significant communities detection method, which was proposed by A. Lancichinetti, A. Radicchi, J. Ramasco and S. Fortunato. It is a method, which to optimizes locally the statistical significance of clusters that defined with respect to a global null model. It declared itself can handle all the problems that outlined in the second part of this review, it can run alone or cooperate with other high efficiency algorithms.

In the following section, the features of OSLOM will be discussed, the algorithm it used will be described and the example of comparison of the performance with OSLOM and some other methods will be provided.

The code of this method can be found at (http://www.oslom.org).

2.3.3.3 Features

- ***A. Significant Clusters***

OSLOM analyse single cluster to optimize the cluster significant. It can distinguish whether the cluster is significant, even in the random graph.

B. Homeless Vertices

Homeless vertices are the vertices that not assigned to any cluster. All the vertices in a random graph are the homeless vertices. There very few clustering techniques can handle the homeless vertices in the network, but in OSLOM, the homeless vertex comes as the natural output.

C. Overlapping Communities

The possibility of cluster overlap is another natural output of OSLOM. It can easily uncovering overlapping vertices.

D. Cluster Hierarchy

When OSLOM processing on the at the network level, it will take care of the possibility of hierarchy structure.

E. Weighted Network

OSLOM also can be generalized and well performed on weighted graphs. Base on the equation below,

$$p(w_{ij} > x | k_i, k_j, s_i, s_j) = \exp(-x / \langle w_{ij} \rangle)$$

F. Directed Graphs

It can be easily generalized to handle directed graphs either by define two uniformly distributed random variable.

G. Dynamical Networks

OSLOM can well handle the dynamical networks. It can start from any partition/cover from any initial partition/cover that can be use as input.

H. Complexity

The complexity of OSLOM depends on the object that it working on, it can not be exactly estimated.

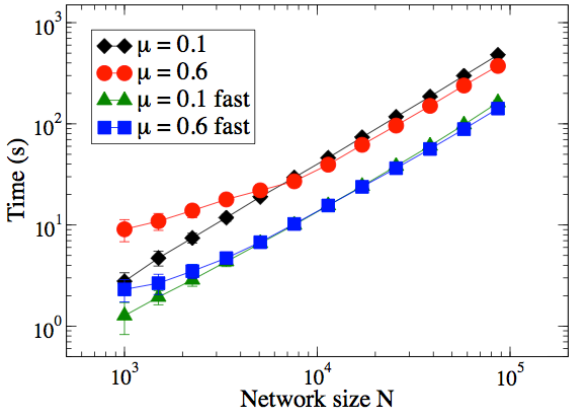


Figure 11 Complexity of OSLOM. The diagram shows how the execution time of two different implementations of the algorithm scales with the network size (expressed by the number of vertices), for LFR benchmark graphs. [5]

2.3.3.4 Methods

In general, OSLOM analysis graphs in three phases, it roughly finds the clusters first, to specify the correctness and boundary of the clusters, finally analyze the result on the network level.

A. Statistical Significant of Network

This phase aim to estimate the statistical significance of a given cluster. It uses the significance as a fitness measure in order to evaluate the clusters. In the configuration model that used, the networks are generated by joining randomly vertices under the constraint, which each vertex has a fixed number of neighbors, taken from the pre-assigned degree distribution.

It starts from the graph G with N vertices and E edges. The significant will be assessed in the given subgraph C. The equation below shows the enumerates the possible configurations of the network with k_{in} connections between i and C.

$$p(k_i^{in}|i,\mathcal{C},\mathcal{G}) = A\frac{2^{-k_i^{in}}}{k_i^{out}!\; k_i^{in}!\; (m_{\mathcal{C}}^{out} - k_i^{in})!\; (M*/2)!}.$$

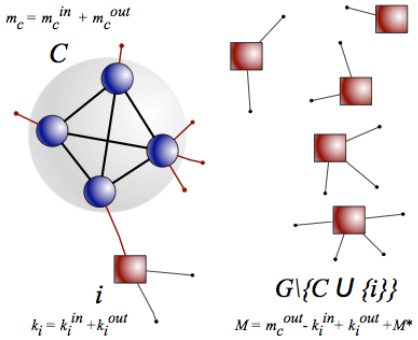


Figure 10 A schematic representation of a subgraph C, whose significance is to be assessed. The subgraph C is embedded within a random graph generated by the configuration model. The degrees of all vertices of the network are fixed, in the figure we have highlighted the degrees of C (m_C), of the vertex i at the center of the analysis (k_i) and of the rest of the graph G \ [C ∪ {i}] (M). These quantities are expressed as sums of contributions, which are internal to their own, set of vertices (as M∗) or related to subgraph C (in or out). This notation is used in the distribution of equation above [5]

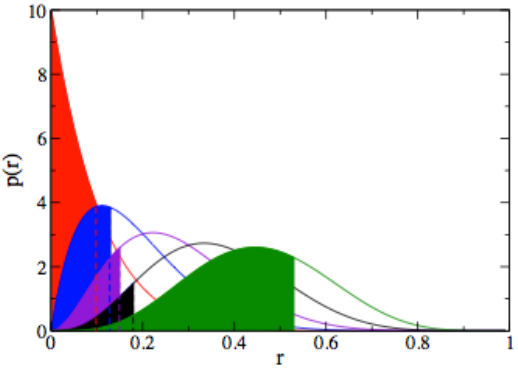


Figure 12 Probability distributions of the scores r of vertices external to a given subgraph C of the graph. The score rq is the q-th smallest score of the external vertices. In this particular case there are 10 external vertices. In the figure, we plot p(r1), p(r2), p(r3), p(r4), p(r5) (from left to right). As an example, the shaded areas show the cumulative probability Ωq for a few values of r that would correspond to the values estimated in a practical situation. In this case, the black area,q=4,isthesmallestandsocm =Ω4. If$\varphi(c_m)$<P, the vertices with scores r1,r2, r3 and r4 will be added to C.

$r(k_i^{in}) = \sum_{j=k_i^{in}}^{k_i} p(j|i,\mathcal{C},\mathcal{G})$ is used to rank the cumulative probability. From the variable r, it reveals the likelihood of the topological relation of each vertex with C

$$\Omega_q(r) = p(r_q < x) = \sum_{i=q}^{N-n_C} \binom{N-n_C}{i} x^i (1-x)^{N-n_C-i}$$ is to calculate the order statis- tic distributions

$\phi(c_m, N - n_C)$ gives the score of the cluster C.

B. Single Cluster Analysis

After a score to evaluate the statistical significance of the clusters has been obtained, afterwards, the score should be optimized across the network by dividing it into proper clusters. It has 2 steps to clean up cluster C, explore the possibility of adding external vertices to the subgraph C first, then prunch the non-significant vertices in C.

1. Compute r for each vertex i outside C and connected to it by at least one edge. Calculate Ω1(r) for the vertex with the smallest r by $\Omega_1(r) = P(r_1 < r) = 1 - (1 - r)^{N - n_C}$; If φ(Ω1(r),N−nC) < P, add the corresponding vertex to the subgraph. Otherwise, checks the next best vertex, until all the best vertices into subgraph C.
2. Pick vertex with the highest value of ri inside C to check for its significance as the first step does. It the worst vertex in C is turns out to be significant, we keep it inside C and the analysis of the cluster is completed. Otherwise, we move to next worst internal vertex in C and repeat the previous action, until make sure all the internal vertices are significant.

C. Network Analysis

In this phase, an algorithm will be introduced to enable full network analysis.

1. Start from a random single vertex, get group C with the initial vertex.
2. More vertices are added to C, considering the most significant among the neighbors of the cluster.
3. Perform the second phase.
4. Repeat the whole procedure starting from several vertices in order to explore different regions of the network. Hence the overlapping communities can be handled.

D. OSLOM

OSLOM is to assemble all the ingredients mentioned above together. It consist 3 phases,

1. Find significant clusters, until convergence
2. Analyzes the resulting set of clusters from previous actions, trying to detect their internal structure or possible unions
3. Detects the hierarchical structure of the clusters

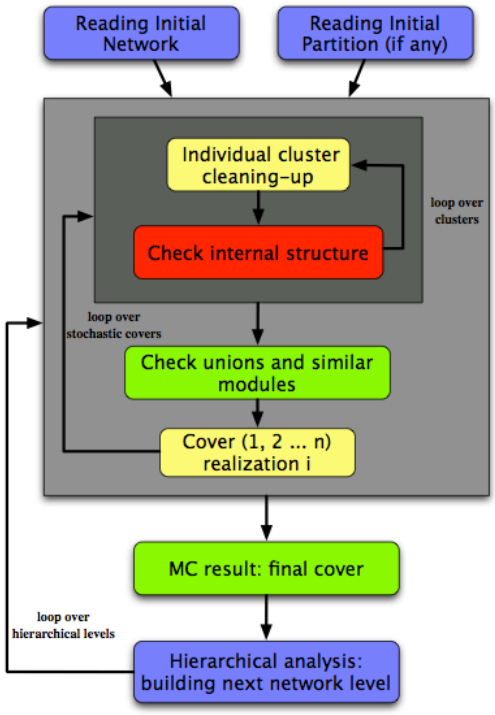


Figure 13 Flux diagram of OSLOM. The levels of grey of the squares represent different loop levels. One can provide an initial partition/cover as input, from which the algorithm starts operating, or no input, in which case the algorithm will build the clusters about individual vertices, chosen at random. OSLOM performs first a cleaning procedure of the clusters, followed by a check of their internal structure and by a decision on possible cluster unions. This is repeated with different choices of random numbers in order to obtain better statistics and a more reliable information. The final step is to generate a super-network for the next level of the hierarchical analysis.

Steps to handle hierarchy Structure:

1. Construct a new network formed by clusters obtained from the previous phase. Then turned each cluster into a supervertex. The edges between supervertices are superedges, which weighted by the number of edges between the initial clusters.
2. Once the supernetwork has been built, applies the method again, obtaining the second hierarchical level.
3. Iteration, until no clusters left.

2.3.3.5 Conclusion

From the research result, OSLOM seems can well handling all the problems that emphasized for communities detection on graphs. The problem that regard for OSLOM is the running time. It may cause too much iteration to slow down the process. Although it can cooperate with other algorithm to optimize the running time, when deal with the large network, it still be a problem. As mentioned in the research paper, the solution is using more processors to work together.

In the next stage of this project, an experiment will be deigned to verify if OSLOM is exactly better than the other methods.

2.4 Testing

2.4.2 Benchmarks

2.4.2.3 Planted l-partition model [25]

Planted l-partition is a computer-generated benchmark. It is a class of graphs. The model partitions a graph with $n = g \cdot l$ vertices in l groups, each group have g vertices. The probability of two vertices in the same group that link to each other is p, the

probability for the vertices link to each other from different group is q. Normally, $p > q$.

However, planted l-partition model is too simple to stand for a real network, in this model, the vertices are all in the same degree and all communities are in the same size.

2.4.2.4 Girvan and Newman Benchmark (GN) [15]

GN benchmark considered a particular case of l-partition model. It is regular used in literature. It set $l = 4$, $g = 32$ (so $n = g * l = 128$ vertices) and fixed the average total degree $\langle k \rangle$ to 16. This implies that probabilities of p and q are not independent parameters. But this benchmark is still not a good description of the real network.

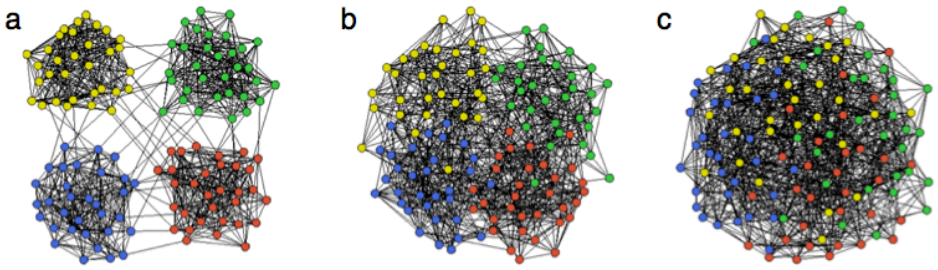


Figure 14 Benchmark of Girvan and Newman. The three pictures correspond to $z_{in} = 15$ (a), $z_{in} = 11$ (b) and $z_{in} = 8$ (c). In (c) the four groups are hardly visible [26]

2.4.2.5 LFR Benchmarks [27][28]

LFR is a type of benchmark graph, which was proposed by A. Lancichinetti, S. Fortunato and F. Radicchi that can be use for testing community detection algorithms. It is also based on planted l-partition model like GN benchmark. It assumes that the distributions of degree and community size are power laws. Research shows the complexity of LFR benchmark can be reach to O(m), where m is as usual the number of edges of the graph, so it can be used to create graphs much closer to reality. Moreover, now LFR have an extension version with overlapping communities.

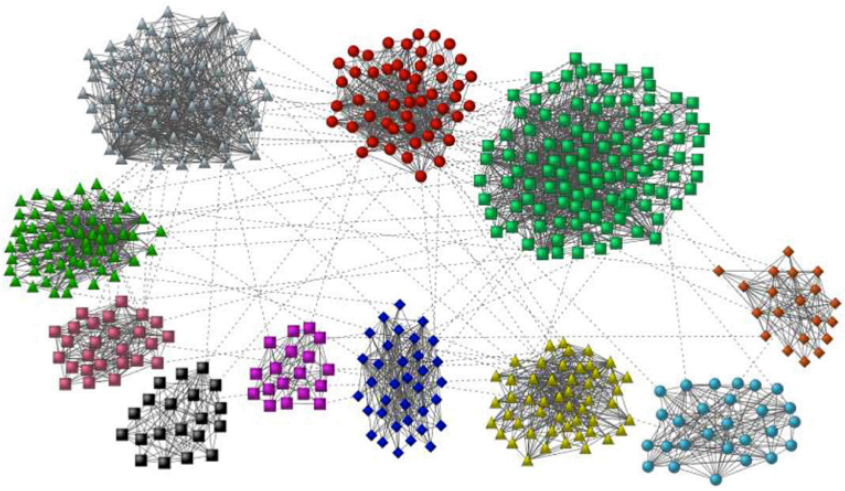


Figure 15 A realization of the LFR benchmark graphs [27], with 500 vertices. The distributions of the vertex degree and of the community size are both power laws. Such a benchmark is a more faithful approximation of real-world networks with community structure than simpler benchmarks like, e.g., that by Girvan and Newman [15].

The graph below shows another implementation of LFR using algorithm to make it have hierarchical structure as another extension for it.

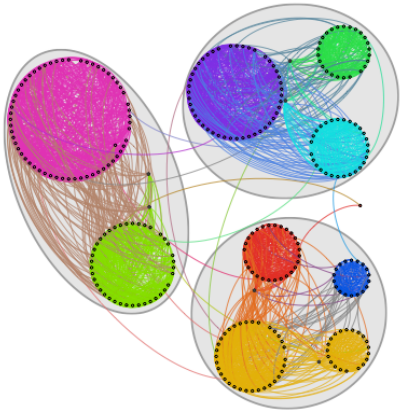


Figure 16 A realization of the hierarchical LFR benchmark with two levels. Stars indicate overlapping vertices. [5]

The code of LFR benchmark can be found and freely download at http://sites.google.com/site/andrealancichinetti/software

2.4.3 Measures

- **Normalized Mutual Information**

Normalized mutual information is currently very often used in tests of graph clustering algorithms. It use "1" and "0" to indicate the partitions are identical or independent [29]. This measure was originally defined for standard partitions that each vertex only belongs to a single cluster, but now it has an extension by Lancichinetti et al. that can also be applied on overlapping clusters. [8] The value of NMI more close to 1 means the communities that found more similar with the original partitions.

- **Modularity**

As mentioned above, modularity is a function that can be used to measure the partitions. John Kleinberg studied the earliest theorem [30], which cannot be extends to graph clustering. The most popular quality function is the modularity of Newman and Girvan, which based on the idea that the possible existence of clusters is revealed by the comparison between the actual density of edges in a subgraph and the density one would expect to have in the subgraph. However, it can just adapt to the disjoint communities [31][32]. Recently, Nicosia et al have extended it to the overlapping communities [33], which overlap modularith measures is defined as function,

$$f(x) = 60x\text{-}30 \text{ [34]}.$$

To evaluate the quality of the partition from the modularity, as the modularity with the value closer to 1, means the partitions have better quality.

Chapter III

3. Testing Options and Boundary

In this chapter, it will discuss what methods/algorithms have been used in the experiment. Those include not only the methods to detect communities, also the networks to be tested on and the measures to evaluate the methods. The experiment-running environment also needs to be specified in this section.

3.1 Methodology

- **Networks**

Depends on the genre of the network, it can be catalogue as real network and artificial network. To evaluate the performance of the community detection algorithms, it better to be ran the algorithms both on real-world network data and on the randomly generated synthetic network data.

However, there will be restrictions to evaluate algorithms by the real-world network. Because interpretation of the original data in the real network is normally un-revealable, so it is hard to make the judgment of communities that found by the algorithm. Although the original network data can be interpreted exactly as it is, it may not reflect the network structure.

On the opposite, by using the randomly generated synthetic networks, i.e. artificial network, which are built to evaluate algorithms, and the community structure is already known to compare with the detected communities, it can help to analyze the algorithms' behaviours in detail by vary the parameters of the networks. The shortage of the artificial network is might not show the properties of the real network, which means it made by conscious, sometimes too ideal to represent the real network.

- **Measures**

There are quite a few standard measures that can be used to compare the existed known communities and the detected communities, e.g. referenced in [35]-[37]. However, to work on overlapping communities, all that mentioned above are inappropriate. Despite of the measure that chose to be used in this case, which is normalized mutual information measure [8][29]. Omega index is an extended measure of adjusted Rand index, and a new variant of the mutual information measure to handle overlapping communities [38].

For real-world network, the can measured by modularity, which is evaluated by the relative density of edges within communities and between communities. The earliest modularity measure is proposed for only disjointed communities. However, recently it extended to support overlapping communities by adding variants to it. The value of overlap modularity (Q_{ov}) was used to measure the testing result depends on the number of communities that belongs to every vertex and the strength of its membership to each communities. However, for each vertex, it was assumed belongs to all its member communities equally.

3.2 Networks

3.2.1 Artificial Networks

LFR Benchmarks

- Potential problems

LFR benchmark is developed by the same team as OSLOM, is some cases, the developer known how to get better result on LFR benchmark. However, as in the previous research, almost all the recent developed and main stream algorithms was tested on LFR benchmark, it seemed to be the best synthetic network that can be suggested at this moment, so we still run the testing on LFR benchmark for all the artificial network testing cases.

- Parameters

Binary network, unweighted and undirected

-N	number of nodes
-k	average degree
-maxk	maximum degree
-mu	mixing parameter
-minc	minimum for the community sizes
-maxc	maximum for the community sizes
-on	number of overlapping nodes
-om	number of memberships of the overlapping nodes
-t1	minus exponent for the degree sequence
-t2	minus exponent for the community size distribution

- Settings

Nodes **N**: 1000 as constant;
Average degree **k**: 8 to 20
Maximum degree **maxk**: 16 to 40 (2*k)
Mixing parameter **mu**: 0.1 to 0.5 *(controls the proportion of random edges to total edges, which means from 90%, 80%, 70%, 60%, 50% of each nodes's edges end within that node's communities, and corresponded the remaining 10%, 20%, 30%, 40%, 50% end in some randomly selected community)*
Minimum of the community size **minc**: 10 to 200
Maximum of the community size **maxc**: 20 to 400 (2*minc)
Number of overlapping nodes **on**: 2 *(Each nodes can be belong to 2 communities)*
Number of memberships of the overlapping nodes **om**: 0 to 1000 *(each overlapping nodes belongs to the number 0, 500, 1,000 communities)*
Minus exponent for the degree sequence and community size distribution were left by default

As reviewed in [39] by Lancichinetti *et al*, it suggest maximum degree maxk equals 2.5*k and maximum community size maxc equals 5*minc. Each overlapping vertex belongs to two communities. The exponent of the power-law distribution of vertex degree t1 is -2 and community size t2 is -1.
In the experiments that done in this project, not all these settings been kept for the reason reduced the complicity of the benchmark to reduced the run time.

Ideally, the experiment should through mixing parameters from 0.1 to 0.8, and numbers of nodes should be set as both on 5,000 and 1,000 as most developer did to proving their own algorithms. However, in this project, it mean to be tested all the cases for the selected algorithms thoroughly, it takes time and required high performance of devices to achieve larger range of testing cases. For example, on the devices that used to ran the experiment in this project, the estimate time of running 5,000 nodes from mixing parameters 0.1 to 0.5 and keep the other parameters as stated above through over 5,400 cases is 1 or 2 months as minimum. The horribly long time consuming was not only because to produce the more complicated network, the core reason is the some of the target algorithms' own run time is unpredictable long. Reduced the number of nodes from 5,000 to 1,000 can reduced the testing cases from 5,000 to 1,000 level, which saved more time. However, the cases 5,000 nodes run with mixing parameters will be tested as the further research as the extension as the project in the future.

3.2.2 Real Networks

- **Coauthorship in Network Science**
  Source: http://www-personal.umich.edu/~mejn/netdata/netscience.zip
  This network is an interpretation of the coauthorship network of scientists working on network theory and experiment, which was compiled by M. [40] Newman, May 2006
  Nodes：1, 588
  Edges: 2, 742

- **Power Grid**
  Source: http://www-personal.umich.edu/~mejn/netdata/power.zip
  This is an undirected and unweighted network that representing the topology of the Western States Power Grid of US. [41]
  Nodes: 4, 940
  Edges: 6, 594

- **Internet**
  Source: http://www-personal.umich.edu/~mejn/netdata/as-22july06.zip
  This network symmetrised snapshot of the structure of the Internet at the level of autonomous systems, reconstructed from BGP tables posted by the University of Oregon Route Views Project. It created by M. Newman from data for 22nd July 2006 and is not previously published.
  Nodes: 22, 962
  Edges: 48, 436

These networks include directed/undirected and weighted/unweighted networks, which may inconsistent with the undirected and unweighted benchmarks that generated. Although all the methods that been tested are support directed and weighed network, this might still be a problem to judge the algorithms from their results. When choosing the real network model, there are several conditions limited the range of candidate networks. Firstly, due to the run time of the testing was very limited, the network that chose should not be too large. Secondly, ideally the network should be undirected and unweighted, but sometimes this attribute may unknown to the original model.

3.3 Communities Structure: Overlapping

- Why overlapping communities?

As stated above, the community structures can be appeared differently. The reason to test all the selected methods on overlapping communities' not disjoint or hierarchical communities is the OSLOM, which is the core algorithm that to be tested and compared can support overlapping communities well. Detection on disjoint communities is an ancient topics that ton of algorithms can do it very well. On the opposite, detecting hierarchical communities is not that popular and with lots of solutions can effectively detect the communities compared with OLSOM. So the target community structure is set only on the overlapping communities.

3.4 Methods

3.4.1 OSLOM

OSLOM is definitely the method that needs to be tested and compared with. It is the only known algorithm that base on statistically significance, which is the core of this project. It need to be compared with other selected algorithms on the same benchmarks or real networks and be observed of it performance on its ability of communities detection.

The default setting will be used for OSLOM. The code resourced from http://www.oslom.org, which developed by C++. There was not any coding requirement by calling this method.

The sample running command is was like,

```
/*        to compile the code              */
./compile all.sh


/*        to run the methods, which "example.dat" is the input network file to be
analyzed. It will automatically generated a folder of files (example.dat oslo files), but
the only useful file in this case than find useful is a plain text file called "tp" which
contains the grouping of the detected communities   */
./oslom undir -f example.dat –uw


/*        this can generate network graph with extension "*.net", however, it was
found not very useful in this project   */
./pajek write undir example.dat
```

Parameters

Like in "./oslom undir -f example.dat –uw", "undir" dedicated on undirected networks, "-uw " refers to unweighted network. Correspondingly, "dir" and "-w" refers to directed and weighted network. There has no other interested parameter that can play with in OSLOM.

3.4.2 Infomap

As from the previous research, Infomap shows its distinct abilities on detecting overlapping communities. It uses random walk to reveal communities. As the past researches showed, it almost found communities as good or even better than OSLOM. It is very sensible to compare it with OSLOM thoroughly to see the differences of its results and asked "why" and on "what" circumstances it shows the differences.

There are also not any requirements on coding for using Infomap. The source code can be found at http://www.tp.umu.se/~rosvall/code.html, which also developed by C++. In the implementation of OSLOM, it had embedded Infomap, COPRA and Louvain in it with a provided interface to call. Although it may cause some problems by using the code that been extended by someone, e.g. the run time may longer due to the generated extra files that may useful to the developer by OSLOM, but not interesting in this project. However, the original algorithm was not been modified, which will not affect the accuracy of its communities detection. For this reason, it is feasible to use the version of Infomap that coded with OSLOM.

The calling method just as the same with OSLOM, just will some extension commands,

```
/*      the command that executing Infomap in OSLOM package, because the
formatting output have coded to be the same with OSLOM, so the useful file also is
the plain text file "tp" with no extension */

./oslom undir -f example.dat -uw -infomap 1 -r 0
```

Parameters

"undir" and "-uw" means network is undirected and unweighted that same with in OSLOM. "-r" dedicate to the repeat times of Infomap. It was set to 2 in all the testing cases of the project for higher accuracy of the result. The valid range of –r is 0 to 10. "1" is just a running sequence number when the algorithms that embedded with OSLOM are running together iteratively. It make no differences when Infomap running alone.

3.4.3 COPRA

COPRA is another algorithms can provide great support on overlapping communities finding. It use label propagation techniques and showed extraordinary performance on most of the past researches. The developer of OSLOM also cited it as its strength on overlapping communities. So it was chose as well to use in the experiment to compare with OSLOM.

COPRA was an implementation of JAVA, which is not the same with the methods that chose above. However, it is also no extra coding needed for calling COPRA. The resources of COPRA can be found
http://www.cs.bris.ac.uk/~steve/networks/software/copra.html.

COPRA can run by following command in terminal,

/ *execute the method, example.dat is the input network, the result groupings will come into a plain text file named clusters-example.dat* */*

java -cp copra.jar COPRA example.dat

Parameters

There are lots of parameters can be set in COPRA, however, not all of those are useful.
-repeat r can be improve the accuracy of COPRA, but it will not be used in the testing, because of its effective can be not as much as another parameter.

-v v is the maximum number of communities per vertex. It default by v = 1, but it only appropriate to disjoint communities. Detect overlapping communities requires v greater or equal to 2. As discussed in [18], when v goes larger, it can obtain better result. But v = 4 is a bottleneck that when v greater than 4, the result will not doing better that obvious.

All the other parameters was kept as default.

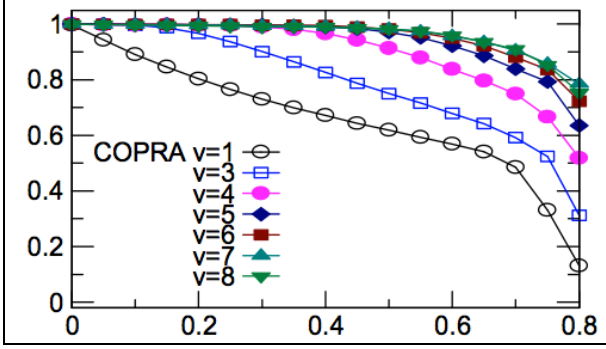


Figure 17 Showing when V grows, how the performance of COPRA will changes. Y axis is the Normalized Mutual Information[18].

3.4.4 GCE

The last method that was chose to be tested and compared with OSLOM is GCE, which was based on fitness function maximization. It was known as its ability to detect overlapping community efficiently.

The implementation of GCE algorithms is called GCE communities finder, it was also coded by C++. To run GCE community finder, it use command,

/ *this command is the sample to call GCE coomunity finder, "example.dat" in the input network, ">result.dat" specified the output file's path and name* */*

./GCECommunityFinder example.dat> result.dat

Parameters

All the parameters of GCE were been kept as default.

3.5 Measures

3.5.1 Normalized Mutual Information

There are couples of implementation that can compute NMI value accurately. One implementation is from *Lancichinetti et al* who extended the algorithm to let it appropriate to overlapping communities, which developed by C++, called "mutual". Another is from S. Gregory, which carried by JAVA. These two implementations both return the same result. Finally, "mutual" was decided to use.

The command to run mutual is,

```
/*      the command that runs "mutual, which required two input file, one specified the original community and the other is result groupings from the methods", a result value will feedback on screen by default      */

./mutual community.dat result.dat
```

Parameters

There have no parameters that can be use in mutual.

3.5.2 Modularity

This modularity algorithm that been used was proposed by Nicosia *et al*, the implementation were carried out by S. Gregory, which the source code that used was called *Modularity.java*. It compared the modularity between the original network and the result communities from the algorithms. Modularity.java shared a function that used in CONGA, which also proposed and developed be S. Gregory, to run Modularity.java, it should be put with CONGA.jar and compiled together.

The command to run Modularity.java is like,

```
/*              this command uses the network data and the result communities from the algorithm as the input      */

java Modularity network.dat result.dat
```

3.6 Tools and Techniques

- **Coding Requirements**

Basically, all the methods, measures and other core algorithms have been provided the implementations to be used directly, there have to needs to code for the delivered methods/measures. However, when connect the whole progresses of experiment go together, there are quite a lot manual input of commands and parameters, which is trivial, time consuming and really nothing technical. The solution for this problem is to do something to let the experiment going automatically. It requires a little bit coding to realize it.

One more coding requirement is to reformat all the input and output file in the form as the algorithms required to avoid inaccurate result or any possible error. It also needs a short program to read the result and write to the valid format.

After all the result had come out from the experiment, it also needs to be organized into an understandable and analyzable way, which also requires a small program to do it.

All these program that mentioned above were developed by **C#** as its own convenience and the familiarity to the developer.

- **Development and Experiment Environment**

Normally, all the referred methods/algorithms preferred to be compiled and executed under LINUX OS, which uses UNIX shell that similar as MAC OS X can support. The problem for running the experiment under LINUX which can be accessed in the Lab is it cannot support .NET framework, i.e. C#, because developer do not have such permission to do such change to the devices in the lab. Ubuntu is another choice that can support UNIX shell and easy to deployed to support .NET, however, when the partial testing was in progress, the GCE community finder was have difficulty to work normally and provide the expected result. But MAC OS X finally showed it compatibility to all the tools and techniques was have used in the experiment.

As to the convenience to the developer's own devices. MAC OS X was chose as the environment to run the experiment. Mono Develop is an application, which can support the development of .NET framework under MAC OS X and with the runtime to execute the solution.

3.7 Summary

- Methods: OSLOM, Infomap, COPRA, GCE
- Networks: LFR benchmark, real-world network models
- Measures: NMI, Modularity
- Coding Language: C#
- Experiment Environment: MAC OS X; LINUX as alternative
- Software: Mono Develop

Chapter IV

4. Implementation / Experiment

Chapter 4 will go through the experiment progress and point out the problems that had met during the experiment. The experiment flow will be listed and discussed.

4.1 Experiment Flow

4.1.1 On Artificial Networks

Firstly, all the parameters to generate the benchmarks were wrote in a *.xls as shown in the example below.

N	k	maxk	mu	minc	maxc	On	Om
1000	8	16	0.1	10	20	0	2
1000	8	16	0.1	10	20	500	2
1000	8	16	0.1	10	20	1000	2
1000	8	16	0.1	30	60	0	2
…	…	…	…	…	…	…	…
1000	20	40	0.5	200	400	500	2
1000	20	40	0.5	200	400	1000	2

After the excel had made, a file of all the commands that need to be ran for the experiment can be generated, the parameters of benchmarks read from the excel corresponded respectively. Then the commands to complete the rest of the experiment will write in the same executable file in sequence.

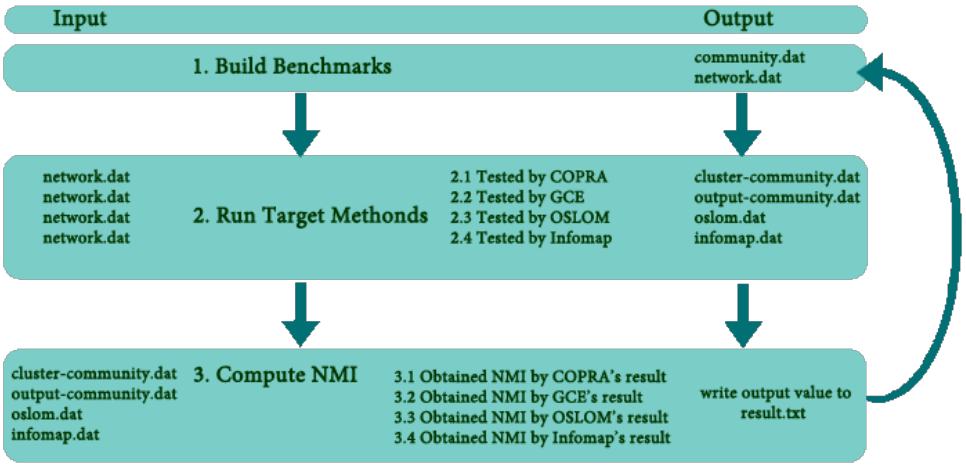


Figure 18 This graph shows the basic flow of the experiment on artificial communities. The iterations repeat until all the script had been executed.

The generated bash executable file is huge that with thousands commands in it. At the beginning, the approximate run time to finish all the commands on the target environment is about 100 hours, which is on very high risk. Once the device go down or the script be terminated unexpectedly, there has not so much time that afford to repeat everything again. So by using the "split" command in shell, the commands was divided in 4 files depends on the number of rows and manually adjusted the head and end of each executable file. Then the run time for each file will be in about one day,

which is reasonable. However, it took about 260 hours, which over two and half day to went through each file.

The table below shows the first iteration of the progress and explained how the script works in details.

#!/bin/bash	// Bash execute shell script
cd /Users/cloche1482/Desktop/design/bn	// Enter to the target folder where is the LFR benchmark generator
./benchmark -N 1000 -k 8 -maxk 16 -mu 0.2 -minc 50 -maxc 100 -on 1000 -om 2	// Build the benchmark with the planed parameters
cp community.dat ../data/community.dat	// Copy community.dat which generated by LFR benchmark to the folder data
cp network.dat ../data/network.dat	// Copy network.dat which generated by LFR benchmark to the folder data
rm community.dat network.dat	// Remove community.dat and network.dat from current folder
cd ..	// Back to last level directory
cd COPRA	// Enter the folder contains COPRA
java -cp copra.jar COPRA ../data/network.dat	// Compile and run COPRA with the input of generated network that saved in folder data
cp clusters-network.dat ../data/clusters-network.dat	// Copy the result of COPRA which is clusters-network.dat to data
rm clusters-network.dat	// Remove clusters-network.dat in current directory
cd ..	// Back to last level directory
cd gce	// Enter the folder where GCE Finder locates
./GCECommunityFinderUbuntu910 ../data/network.dat>../data/xxx.dat	// Run GCE community finder with input network graph network.dat and specified the output path (data) and name of the file that contains the result from GCE community finder
cd ..	// Back to last level directory
cd OSLOM2	// Enter the folder that OSLOM locates
./oslom_undir -f ../data/network.dat -uw	// Run OSLOM with input network.dat
cp ../data/network.dat_oslo_files/tp ../data/oslom	// Copy output file tp from generated directories to folder data and rename file to oslom
rm -rf ../data/network.dat_oslo_files	// Remove folder network.dat_oslo_files
./oslom_undir -f ../data/network.dat -uw -infomap 1	// Run Infomap in the same folder with same input
cp ../data/network.dat_oslo_files/tp ../data/infomap	// Copy output file tp from generated directories to folder data and rename file to infomap
rm -rf ../data/network.dat_oslo_files	// Remove folder network.dat_oslo_files
cd ..	// Back to last level directory
cd format	// Enter folder format which contains the executable file to reformatting output result that from the algorithms
mono ./converter.exe ../data/community.dat ../data/new_community.dat community	// Use mono runtime to execute converter.exe to reformatting community.dat to adjust it as a valid input for mutual

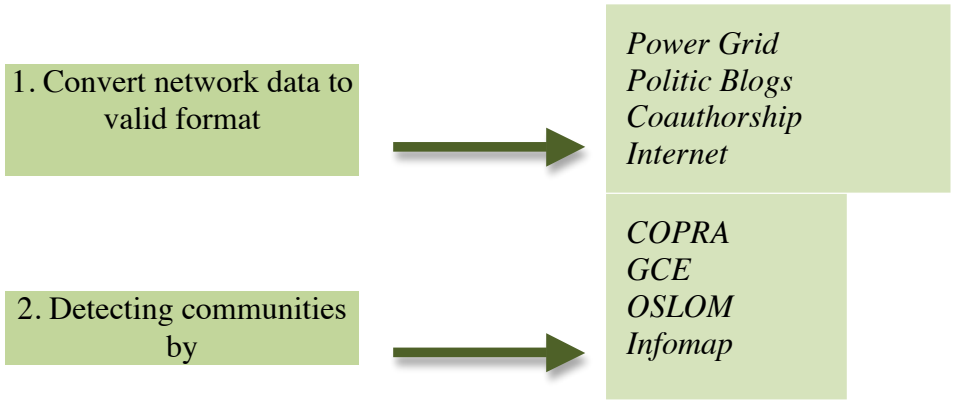
mono ./converter.exe ../data/oslom ../data/new_oslom.dat tp	// Reformatting oslom to adjust it as a valid input for mutual
mono ./converter.exe ../data/infomap ../data/new_infomap.dat tp	// Reformatting infomap to adjust it as a valid input for mutual
rm ../data/community.dat	// Remove community.dat from data
rm ../data/oslom	// Remove oslom from data
rm ../data/infomap	// Remove infomap from data
cd ..	// Back to last level directory
cd mutual3	// Enter folder mutual3 which with the application mutual to compute NMI
./mutual ../data/new_community.dat ../data/clusters-network.dat>>../data/res.txt	// Use mutual to compute the NMI between the original communities file and the result from COPRA
./mutual ../data/new_community.dat ../data/xxx.dat>>../data/res.txt	// Use mutual to compute the NMI between the original communities file and the result from GCE
./mutual ../data/new_community.dat ../data/new_oslom.dat>>../data/res.txt	// Use mutual to compute the NMI between the original communities file and the result from OSLOM
./mutual ../data/new_community.dat ../data/new_infomap.dat>>../data/res.txt	// Use mutual to compute the NMI between the original communities file and the result from Infomap
cd ..	// Back to last level directory
cd data	Enter folder data
*rm *.dat*	// Remove all the file with extension .dat in folder data

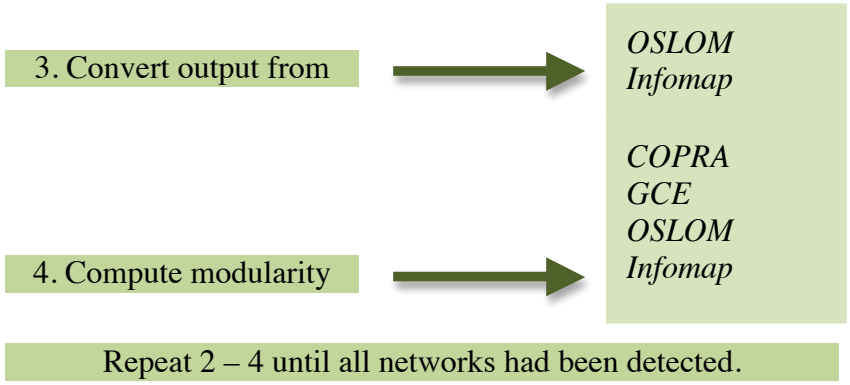
After the commands had all executed, there are 4 plain text file with result NMI value was generated. Then the result will be processed to correspond to the parameters of benchmarks like followed table which as easier to be analyzed.

N	k	maxk	mu	minc	maxc	on	om	COPRA	GCE	OSLOM	Infomap
1000	8	16	0.2	50	100	0	2	0.653495	0.533333	0.966948	0.967052
1000	8	16	0.2	50	100	500	2	0	2.22045e-16	0.134047	0.128917
1000	8	16	0.2	50	100	1000	2	0	2.77556e-16	0.354126	0.346458

4.1.2 On Real Networks

- **Flows**





4.2 Codes

As all the methods/algorithms that referenced in this project can be used directly, there is not any requirement to develop any code for those methods in this project. So the coding section is not necessarily to be emphasized, as it is not that important to the weight of this project. However, to fluent the progress of the experiment and let it goes automatically and smoothly with accuracy and high efficiency, there are still some program should be done to assist the experiment.

- **Write Commands in File**

In this method, it use file stream from library to do read and write file. The data that read were saved in the array list.

```
private static void SaveRawData(string path, ArrayList res, int cnt)
{
    if (!File.Exists(path))
    {
        //create file
        using (FileStream fs = File.Create(path)     { }
    }
    string N = " -N ";
    string K = " -k ";
    string MAXK = " -maxk ";
    string MU = " -mu ";
    string MINC = " -minc ";
    string MAXC = " -maxc ";
    string OM = " -om ";
    string ON = " -on ";
    using (System.IO.StreamWriter file = new System.IO.StreamWriter(path))
    {
        string line = "./benchmark";
        for (int c = 0; c < cnt/8; c++)
        {
            line += N + res[8*c];
            line += K + res[8 * c + 1];
            line += MAXK + res[8 * c + 2];
            line += MU + res[8 * c +3];
            line += MINC + res[8 * c +4];
```

```
                line += MAXC + res[8 * c+5];
                line += ON + res[8 * c+6];
                line += OM + res[8 * c+7];
                file.WriteLine("cd /home/cloche1482/Desktop/design/bn");
                file.WriteLine(line);
                file.WriteLine(@"cp community.dat ../data/community.dat
cp network.dat ../data/network.dat
rm community.dat network.dat
cd ..
cd COPRA
java -cp copra.jar COPRA ../data/network.dat>../data/clusters-network.dat
rm clusters-network.dat
cd ..
cd gce
./GCECommunityFinderUbuntu910 ../data/network.dat>../data/xxx.dat
cd ..
cd OSLOM2
./oslom_undir -f ../data/network.dat -uw
cp ../data/network.dat_oslo_files/tp ../data/oslom
rm -rf ../data/network.dat_oslo_files
./oslom_undir -f ../data/network.dat -uw -infomap 2
cp ../data/network.dat_oslo_files/tp ../data/infomap
rm -rf ../data/network.dat_oslo_files
cd ..
cd format
./converter.exe ../data/community.dat ../data/new_community.dat community
./converter.exe ../data/oslom ../data/new_oslom.dat tp
./converter.exe ../data/infomap ../data/new_infomap.dat tp
rm ../data/community.dat
rm ../data/oslom
rm ../data/infomap
cd ..
cd mutual3
./mutual ../data/new_community.dat ../data/clusters-network.dat>>../data/res.txt
./mutual ../data/new_community.dat ../data/xxx.dat>>../data/res.txt
./mutual ../data/new_community.dat ../data/new_oslom.dat>>../data/res.txt
./mutual ../data/new_community.dat ../data/new_infomap.dat>>../data/res.txt
cd ..
cd data
rm *.dat");
line = "./benchmark";
                        }
            }
        }
```

- **Convert Result Format**

The reason to convert the format of the text is because the valid format to *mutual*, which will give the NMI value that compared with the detected communities and the original communities is fixed as,

```
23 12 2 45
89 11 78 24 51 88
90 91 25 67 36
…
```

However, the original output from benchmarks and some algorithms does not look like this. So the conversion must be done to let the file become a valid and correct input to *mutual*.

The results from OSLOM, Infomap and the original communities output by benchmarks are not valid to *mutual* that need to be converted.

a. Reformat the result from OSLOM/Infomap

The output file from OSLOM and Infomap are easy to be processed, basically, they just have one more header line for each community. The idea is to trim the header off from the result like followed.

```
#module 0 size: 100 bs: 9.80166e-101

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
98 99 100

#module 1 size: 100 bs: 7.43383e-101

101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166
167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188
189 190 191 192 193 194 195 196 197 198 199 200
```

To realize this, the function read the original file but escape each line started with "#module" and write to a new file.

```
private string[] ProcessTpData(ref int c)
{
        var dat = from string n in TpData where !n.StartsWith("#module") select n;
        string[] res = new string[TpData.Count];
        c=0;
       foreach (var g in dat)
      {
         //Console.WriteLine("Group {0} Contains :", g.Cata);
         res[c] = g.ToString();  //.Add(w.Key);
            //Console.Write(w.Key + " ");
         c++;
         //Console.WriteLine();
      }
                        return res;
                }
```

b. Reformat *community.dat* to groupings from benchmark

The original communities from benchmarks are a little bit more complicated compares to the previous case. In this case, the original file just list which node belongs to which communities. As in these setting of the experiment, each node can belong to as much two communities. And what will be done is to allocate every node to the community it belongs.

```
1       110 265
2       62 363
3       27 336
4       61 170
5       396
6       355
7       168 320
8       36 188
9       79 261
10      71 134
11      68 98
12      10 376
13      103 416
14      80 107
15      196 295
```

To solve this problem, every node that read from the file was sort into an array until the file had read completely, and then writes each array as a line into a new file.

```
private string[] ProcessCommunityData(ref int c)
{
        var dat = from n in CommunityData group n by n.Value into g select new { Cata =
g.Key, Numbers = g };
        string[] res = new string[CommunityData.Count];
       c=0;
       foreach (var g in dat)
       {
          foreach (var w in g.Numbers)
          {
             res[c] += w.Key.ToString() + " ";  //.Add(w.Key);
             //Console.Write(w.Key + " ");
          }
          c++;
          //Console.WriteLine();
       }
                        return res;
              }
```

- **Convert Network Format**

The format of a real-world network model can be vary, in this project, all the selected network are in *.gml format, which need to be converted to the valid input to the algorithms.

The original *.gml data is like,

```
Creator "Lada Adamic on Tue Aug 15 2006"
graph [
  directed 1
  node [
    id 1
    label "100monkeystyping.com"
    value 0
    source "Blogarama"
  ]
  node [
    id 2
    label "12thharmonic.com/wordpress"
    value 0
    source "BlogCatalog"
```

```
 ]
……
node [
   id 1490
   label "zeph1z.tripod.com/blog"
   value 1
   source "Blogarama"
 ]
 edge [
   source 267
   target 1394
 ]
……
edge [
   source 1133
   target 1408
 ]
 edge [
   source 1133
   target 1152
 ]
]
```

In this file, the first part about the list of nodes is useless. The useful data that need to concern is the second part, which lists the connection of the edges. To reformatting the network data, the nodes section was skipped and the id of nodes that listed in edges section was fetched then write in a new text file as “1133 1152”. After the whole document has been processed, the fetched two columns of data has been swapped then write in the text again and sort in ascendant order according to the id of in first column.

```
        private int[,] LoadEdge(string path,ref long counter)
        {
                int[,] data = new int[100000,2];
                if (!File.Exists(path))
      {
        // Error
        Console.WriteLine("Error, file not found.");
        return data;
      }
      // Open the stream and read it back.
      counter = 0;
      string line;
      // Read the file and display it line by line.
      System.IO.StreamReader file = new System.IO.StreamReader(path);
                        int st = -1;
      while ((line = file.ReadLine()) != null)
      {
        string em = line.Trim();
        if(em=="edge")
        {       st=1;     }
        else      {
                        if(st>0)  {
                                if(st==2){
                                string[] arr = em.Split(new string[] { " " }, StringSplitOptions.None);
                                data[counter,0] = Convert.ToInt32(arr[1]);
                                }
                                if(st==3){
                                data[counter,1] = Convert.ToInt32(em.Split(new string[] { " " },
StringSplitOptions.None)[1]);
                                counter++;
                                st=-1;
                                        }
                                        st++;     }          }
```

```
        }
        file.Close();
                        return data;
                }
                private int[,] ProcessEdgeData(ref int[,] c)
                {
                        int lt,rt;
                        long t=counter;
                        for(long i =0;i<counter; i++,t++)
                        {
                                c[t,0] = c[i,1];
                                c[t,1] = c[i,0];
                        }
                        counter*=2;
                        for(long i=0;i<counter-1;i++){
                                for(long j=i+1;j<counter;j++)
                                {
                                        if(c[i,0]>c[j,0]){
                                                //swap
                                                lt = c[j,0];
                                                rt = c[j,1];
                                                c[j,0] = c[i,0];
                                                c[j,1] = c[i,1];
                                                c[i,0] = lt;
                                                c[i,1] = rt; }
        }       }
                        return c;       }
        private void SaveEdgeData(string path,int[,] res,long c)
        {
        if (!File.Exists(path))
         {
        //create file
        using (FileStream fs = File.Create(path))
        { }
        }
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(path))
        {
           for(long i =0;i<c; i++)
              file.WriteLine( res[i,0] + "\t" + res[i,1] );
        }       }
```

- **Format Result**

The result that out put from the experiment was like,

```
mutual3:        0.503733        // From COPRA
mutual3:        0.617242        // From GCE
mutual3:        0.899531        // From OSLOM
mutual3:        0.929498        // From Infomap
```

The raw data of the result is just simply the screen output of *mutual*. To enable these data be analyzed as the judgement for the algorithms, it need to be match with parameters of the corresponded benchmarks in the desired format.

N	k	maxk	mu	minc	maxc	on	om	COPRA	GCE	OSLOM	Infomap
1000	8	16	0.2	50	100	0	2	0.653495	0.533333	0.966948	0.967052

This part of codes followed showed the progress how the result that load from the text and be parsed.

```
private List<ResultsDataType> LoadResult(string path)
                {
```

```
                        int c=0;
                        RawData = LoadRawData("data.txt");
List<RawDataType> res = ProcessRawData(ref c);
                        List<ResultsDataType> dic = new List<ResultsDataType>();

    if (!File.Exists(path))
    {
       // Error
       Console.WriteLine("Error, file not found.");
       return dic;
    }
    // Open the stream and read it back.
    string line;
                        long cnt=0;
                        //Console.WriteLine("community load start ");
                        // Read the file and display it line by line.
    System.IO.StreamReader file = new System.IO.StreamReader(path);
                        ResultsDataType t = new ResultsDataType();;
                        //Console.WriteLine("community read file ");
                        while ((line = file.ReadLine()) != null)
    {
       string em = line.Trim();
       string[] restr = em.Split(new string[] { "\t" }, StringSplitOptions.None);
                                if((int)(cnt%4) == 0) t.GCE =Convert.ToSingle(restr[1]);
                                if((int)(cnt%4) == 1) t.CORPA =Convert.ToSingle(restr[1]);
                                if((int)(cnt%4) == 2) t.OSLOM =Convert.ToSingle(restr[1]);
                                if((int)(cnt%4) == 3) t.INFOMAP =Convert.ToSingle(restr[1]);
                                if((cnt+1)%4==0)
                                {
                                        dic.Add(t);
                                         t = new ResultsDataType();
                                }
                                cnt++;
                        }
                        line="";
                        Console.WriteLine(dic.Count());
                        Console.WriteLine(res.Count());
                        Console.ReadLine();
                        for (c = 0; c < res.Count(); c++)
    {
       line += res[c].N+ "\t";
       line += res[c].K+ "\t";
       line += res[c].Maxk+ "\t";
       line += res[c].Mu+ "\t";
       line += res[c].Minc+ "\t";
       line += res[c].Maxc+ "\t";
       line += res[c].Om+ "\t";
       line += res[c].On+ "\t";
                                dic[c].Parameter = line;
                        }
                        return dic;
                }
```

4.3 Problems

- Unexpected long run time that takes to finish all testing cases on ran on benchmarks, which seriously delayed the progress of the project to begin next stage experiment and analysis.
- The experiment went on real network was not good enough. There are tons of real network model that can be found on Internet. However, as the limitation of the deadline of the project and the speed of the algorithms, the amount of the available network model can be chose became very small. It was because, firstly, most of the real network data, e.g. social network, it normally have thousands nodes and millions edges, which can be estimated will take another whole month wait for the result come out, which is not affordable to the time limitation of this project. Although there are some smaller network model exist, but some of them are obviously too small with just under thousand or even hundred nodes, which are have less persuasion to proof the algorithms. Ideally, the large social or community networks still are preferred. But it seemed within the range of this project, it will not possible to be tested through, however, it will be considered as the further extension work for this topic.
- The experiment has just run single round. The result was not been double-checked. This problem was found incidentally when the same benchmark ran by the same algorithm with same setting on MAC OS X and LINUX but the result came out very differently. Because this problem was found at very late stage of the project unexpectedly. So the experiment has not been ran the second time to check if the results are all correct. It has planed to run the experiment again on a different environment to check the result as the further work.
- Another problem for this project is how deep the algorithms have been learned by the researcher of this project. As there are almost no requirements to the researcher to code the methods for use, there is no necessity to know exactly how the algorithms work mathematically. Furthermore, that is not easy to learn all the algorithms that related to this project comprehensively in very short time. The wide but shallow knowledge to the algorithms may cause difficulties when analyse the results. E.g. the different from the result may found but the reason to cause the differences may fail to found.

Chapter V

5. Results & Analysis

In this chapter, the result will be listed and analyzed. From this chapter, the objectives of the project are expected to show that have been achieved. The results will be discussed in two sections, which come from artificial networks model and real network model.

5.1 Artificial Networks

In this section, it will focus on the comparison between OSLOM and other traditional algorithms. As OSLOM already known as its great ability on detect communities. It will pay more attention on the cases that OSLOM failed detect communities while other algorithms did. It also aimed to show on what circumstance that OSLOM trend to failed the detection.

- **GCE vs. OSLOM**

Table 1 This table shows the cases that when NMI of GCE greater than 0.4, which means GCE detected the communities with appropriately groupings but OSLOM did worse than GCE. The cases in block letters showed the cases GCE had proper groupings of communities while OSLOM seems totally failed the detection.

Mixing Param.	GCE	OSLOM	*Infomap*	*COPRA*
0.1	0.97792	0.952765	1	0
0.1	**0.848243**	**0.486395**	1	0
0.1	**0.62616**	**0.1519**	0.15356	0
0.1	**0.565582**	**0.33727**	0.347373	0.365731
0.1	**0.529707**	**0.217652**	0.196345	0.397905
0.1	**0.426687**	**0.226388**	0.252548	0.686758
0.1	**0.423476**	**0.148915**	0.182258	0.499666
0.1	**0.417623**	**0.150982**	0.14811	0.274489
0.2	1	0.998776	0.998776	1
0.2	0.85456	0.84773	0.974194	0
0.2	0.674936	0.610423	0.935933	0
0.2	0.627762	0.255608	0.231836	0.340486
0.2	**0.576885**	**0.136567**	0.14597	0.459579
0.2	**0.571892**	**0.193316**	0.192379	0.524751
0.2	**0.451316**	**0.161511**	0.166416	0
0.2	**0.433109**	**0.210089**	0.206088	0
0.3	**0.487294**	**0.30692**	0.326661	0
0.3	**0.467023**	**0.231842**	0.253414	0
0.3	**0.407077**	**0.268033**	0.272096	0.318303
0.4	1	0.997728	0.997728	0
0.4	**0.623653**	**0.159593**	0.118931	0
0.4	**0.569821**	**0.235264**	0.237477	0.596383
0.4	**0.496965**	**0.213693**	0.277776	0
0.5	0.996412	0.983759	0.983759	0.873933
0.5	0.948714	0.894958	0.884135	0.833286
0.5	0.770656	0.720687	0.881076	0
0.5	0.521397	0.435558	0.541999	0
0.5	**0.415191**	**0.270483**	0.291576	0

As seen from the table, while the mixing parameters become bigger, OSLOM may likely to detect the communities appropriately. There are about ***27%*** cases shows GCE had better result than OSLOM. There are ***18*** cases found that GCE successfully detected the communities but OSLOM failed, which is about ***6.5%*** cases found significant out of all the testing cases shows GCE obtained better NMI than OSLOM.

- **COPRA vs. OSLOM**

Table 2 This table shows the comparison between COPRA and OSLOM. The cases showed in blocked font are that cases that found COPRA has detected communities successfully but OSLOM failed. In this table, it just listed the cases that with NMI's value greater than 0.4 of COPRA. The rest of the results can be found in the appendix.

Mixing Params	*GCE*	COPRA	OSLOM	*Infomap*
0.1	0.277912	**1**	**0.249337**	0.231443
0.1	0.269015	**1**	**0.249963**	0.297934
0.1	0.170213	**1**	**0.139419**	0.173994
0.1	0.168098	**1**	**0.138695**	0.200693
0.1	0.164675	**1**	**0.146389**	0.16921
0.1	0.147226	**1**	**0.180323**	0.187026
0.1	0.0150658	**1**	**0.0922252**	0.100882
0.1	0.0142835	**1**	**0.128992**	0.0816002
0.1	1.11E-16	**1**	**0.114064**	0.121942
0.1	0	**1**	**0.103921**	0.147148
0.1	0	**1**	**0.105565**	0.107284
0.1	0	**1**	**0.0956342**	0.170081
0.1	0	**1**	**0.130602**	0.125534
0.1	0	**1**	**0.128805**	0.132596
0.1	0	**1**	**0.0582265**	0.0660639
0.1	0	**1**	**0.147834**	0.17394
0.1	0	**1**	**0.152964**	0.147788
0.1	0	**1**	**0.190651**	0.186848
0.1	0	**1**	**0.153855**	0.16163
0.1	0.426687	**0.686758**	**0.226388**	0.252548
0.1	0.3499	**0.627424**	**0.193669**	0.173447
0.1	0	**0.507276**	**0.112498**	0.145518
0.1	0.423476	**0.499666**	**0.148915**	0.182258
0.1	0.273948	**0.499598**	**0.197999**	0.210033
0.1	0.0940876	**0.494104**	**0.179435**	0.168662
0.1	0	**0.485728**	**0.00612646**	0.0174881
0.1	0	**0.43037**	**0.156606**	0.148099
0.1	0.269616	**0.429261**	**0.257995**	0.287692
0.1	0.172151	**0.40866**	**0.185478**	0.212902
0.1	0.363779	**0.408398**	**0.146956**	0.159664
0.2	1	1	0.998776	0.998776
0.2	0.0135236	**1**	**0.10088**	0.104981
0.2	0.0120103	**1**	**0.372291**	0.499254
0.2	0.00770728	**1**	**0.0695588**	0.0784132
0.2	2.22E-16	**1**	**0.118718**	0.105391
0.2	1.11E-16	**1**	**0.101867**	0.15906
0.2	0	**1**	**0.136452**	0.124347
0.2	0	**1**	**0.0585005**	0.0602391
0.2	0	**1**	**0.0834665**	0.0759773
0.2	0	**0.995863**	**0.0710875**	0.0835288
0.2	0	**0.991319**	**0.0485505**	0.0115239

0.2	0.000697382	**0.991159**	**0.113158**	0.10877
0.2	0	**0.989498**	**0.0486043**	0.0572192
0.2	0.336935	**0.985167**	**0.27223**	0.283685
0.2	1.11E-16	**0.970895**	**0.137805**	0.114966
0.2	0.0734419	**0.966922**	**0.10814**	0.156972
0.2	0	**0.895825**	**0.049865**	0.079167
0.2	0.28829	**0.840602**	**0.238512**	0.234068
0.2	1.11E-16	**0.824357**	**0.266692**	0.21826
0.2	0	**0.81264**	**0.024524**	0.047366
0.2	0.00132405	**0.779376**	**0.0384133**	0.0351094
0.2	0.251899	**0.741993**	**0.146437**	0.181407
0.2	0	**0.714705**	**0.135498**	0.112693
0.2	0	**0.702421**	**0.0909688**	0.0818788
0.2	0.150391	**0.689759**	**0.105135**	0.136163
0.2	0.00335918	**0.630373**	**0.158327**	0.176901
0.2	0.0656017	**0.568527**	**0.0499964**	0.0406265
0.2	0.571892	**0.524751**	**0.193316**	0.192379
0.2	1.11E-16	**0.460067**	**0.0551481**	0.116449
0.2	0.576885	**0.459579**	**0.136567**	0.14597
0.2	0.247935	**0.434221**	**0.174486**	0.19396
0.3	0.0322516	**0.9982**	**0.124294**	0.107735
0.3	0.179253	**0.98858**	**0.102463**	0.187387
0.3	2.22E-16	**0.974858**	**0.0164102**	0
0.3	0.266226	**0.966673**	**0.283848**	0.282147
0.3	1.11E-16	**0.911528**	**0.0858994**	0.0701861
0.3	3.33E-16	**0.857941**	**0**	0
0.3	0	**0.845544**	**0**	0
0.3	0.206267	**0.837047**	**0.215923**	0.221353
0.3	0	**0.831105**	**0.00383005**	0.00351093
0.3	0.120539	**0.770249**	**0.152991**	0.120866
0.3	3.33E-16	**0.76765**	**0.00982346**	0
0.3	0.504804	**0.690303**	**0.51942**	0.575792
0.3	0	**0.509352**	**0.000312229**	0.000878968
0.4	0.867021	0.971677	0.975394	1
0.4	0.174996	**0.829872**	**0.185761**	0.173993
0.4	0.319613	**0.761403**	**0.248308**	0.271433
0.4	0.569821	**0.596383**	**0.235264**	0.237477
0.4	0.233091	**0.563461**	**0.139169**	0.124275
0.5	0.937879	1	0.974934	0.974934
0.5	0.346958	**0.55814**	**0.151633**	0.117998

Similar to the results from GCE, as the mixing parameter grows, the results of OSLOM appear better than COPRA. When

There are about ***12%*** cases found that COPRA detected the communities better than OSLOM. There's ***82*** cases found that COPRA detected the communities appropriately but OSLOM totally failed, it is about ***64%*** cases that found significant out of all the cases that found COPRA did better than OSLOM, which this percentage is much higher than GCE that found 6.5% cases significant.

- **Infomap vs. OSLOM**

Table 3 This table shows the comparison between Infomap and OSLOM. It listed the cases that NMI f Infomap greater than 0.4. The cases than showed in blocked font are the cases that found Infomap detected the communities much better than OSLOM. There were not a lot cases found that Infomap detected the communities successful but OSLOM failed.

Mixing Param.	*GCE*	**Infomap**	**OSLOM**	*COPRA*
0.1	0.97792	1	0.952765	0
0.1	0.664138	**1**	**0.766306**	0
0.1	0.848243	**1**	**0.486395**	0
0.1	0.261302	**0.656135**	**0.173919**	0
0.2	0.614557	**1**	**0.771006**	0
0.2	0.386822	**1**	**0.769163**	0
0.2	0.783748	0.997009	0.923884	0
0.2	0.85456	0.974194	0.84773	0
0.2	0.674936	**0.935933**	**0.610423**	0
0.2	0.164707	**0.828928**	**0.538731**	0
0.2	0	**0.742888**	**0.387302**	0
0.2	0.376877	**0.717**	**0.583258**	0
0.2	0.494891	0.625855	0.550894	0
0.2	0.401048	**0.621337**	**0.493373**	0
0.2	0.0120103	0.499254	0.372291	1
0.3	0.2414	1	0.926556	0
0.3	0.548249	**1**	**0.575647**	0
0.3	0.259245	0.807702	0.743181	0
0.3	0	**0.743114**	**0.452756**	0
0.3	0.386933	0.704584	0.587915	0
0.3	0	**0.677754**	**0.376277**	0
0.3	2.22E-16	**0.600538**	**0.189272**	0
0.3	3.33E-16	0.596387	0.456721	0
0.3	0.305588	**0.582423**	**0.187404**	0
0.3	0.504804	0.575792	0.51942	0.690303
0.4	0.867021	1	0.975394	0.971677
0.4	0.290302	1	0.928571	0
0.4	0.532709	0.974349	0.853501	0
0.4	0.845541	0.950353	0.948643	0
0.4	0.287949	0.842159	0.755903	0
0.4	0	**0.828637**	**0.625429**	0
0.4	0	**0.828605**	**0.518724**	0
0.4	0.361348	0.713688	0.62825	0
0.4	1.11E-16	0.713624	0.553646	0
0.4	0.287053	0.569726	0.391225	0
0.4	0.424431	0.507133	0.438059	0
0.4	0	0.483393	0.391859	0
0.5	0.996412	0.983759	0.983759	0.873933
0.5	0.822623	0.93692	0.90431	0.872401
0.5	0.770656	0.881076	0.720687	0
0.5	0.346247	0.778307	0.618963	0
0.5	0.530246	0.762857	0.746451	0
0.5	0.462883	0.759013	0.754768	0
0.5	0.521397	0.541999	0.435558	0
0.5	1.11E-16	0.492858	0.368961	0

There are about ***35%*** cases found that Infomap detected communities better than OSLOM. However, different to COPRA's great differences with the result from

OSLOM, the NMI from Infomap and OSLOM showed very similar values. In most of the cases, the results from OSLOM and Infomap obtained same or very similar values. From the 35% cases that found of Infomap better than OSLOM, there were just 17 cases, which means lower than 5% cases showed significant different than OSLOM.

Similar to other algorithms that discussed above, with the comparison between Infomap and OSLOM, OSLOM also seems likely to fail the cases with the mixing parameter trends to be smaller. That was apparently can be seen that OSLOM failed most cases with mu = 0.1 than mu = 0.5.

- **OSLOM vs. Other Algorithms**

Table 4 This table shows the comparison between OSLOM and Infomap. Different from table 3, this table shows the cases that OSLOM detected communities better than Infomap. This data segment shows the cases OSLOM's NMI greater than Infomap. The cases in blocked font are the cases that found OSLOM had great advantages than Infomap.

Mixing Param.	*OSLOM*	*GCE*	**Infomap**	*COPRA*
0.1	**0.546299**	0.0368472	**0.285787**	0
0.2	0.996642	0.702767	0.766208	0
0.3	0.962591	0.693979	0.901814	0
0.3	0.9142	0.413185	0.881254	0
0.3	0.544404	1.67E-16	0.526317	0
0.4	1	0.758424	0.997428	0.812837
0.4	1	0.383547	0.868992	0
0.4	**1**	0.515031	**0.47615**	0
0.4	0.914309	0.503609	0.853134	0
0.4	0.778303	0	0.776519	0
0.4	0.59439	0.191331	0.517173	0
0.5	1	0.86385	0.964286	0
0.5	0.983759	0.996412	0.983759	0.873933
0.5	0.740165	0.279943	0.668197	0
0.5	0.698217	0.280893	0.683238	0
0.5	0.514512	1.11E-16	0.489632	0
0.5	0.424317	0	0.371016	0
0.5	0.421987	0.123582	0.407766	0
0.5	0.401287	0.154733	0.388709	0

There were about 21% cases that found OSLOM obtained better result than Infomap, however, there were just very few cases showed great difference between the results from these two algorithms. There are no cases that found as Infomap completely failed to detect the communities but OSLOM did. It is same as the comparison in table 3.

As showed in table 4, it proof the assumption that made from the other tables, which OSLOM shows better performance when mixing parameter grows bigger, which means OSLOM can detect communities better than traditional algorithms when the amount of edges with random nodes on one side in higher percentage.

There were 12% cases of all testing cases that found OSLOM successfully detected the communities but COPRA failed. And there were about 8% cases of all testing cases that found OSLOM detected the communities successfully but GCE failed. The threshold to judge if the algorithms failed to detect the communities here was when NMI lower than 0.7.

5.2 Real Networks

There were three real network model had been tested as planed. However, the results of testing seem not very persuasive to propose any notion about the algorithms. The measure to evaluate also appeared some problems when evaluating the partitions from the algorithms as the operation to execute the programs were all correct.

- **Coauthorship in Network Science**

This network model has 1588 nodes and 2742 edges. It is a directed and weighted network. GCE was failed to be computed the modularity by its results for some reason that unknown at the moment. From this case it can be found that Infomap showed obtained the best quality of community partitions, followed by OSLOM and COPRA.

Modularity			
COPRA	***GCE***	***OSLOM***	***Infomap***
0.8100043439146104		0.8969609755267102	0.9079448548951625

- **Power Grid**

This network is an undirected and unweighted network that has 4940 nodes and 6594 edges. In this case, OSLOM showed the best partition quality that slightly better than Infomap while COPRA almost failed the detection.

Modularity			
COPRA	***GCE***	***OSLOM***	***Infomap***
0.1443308150744903		0.5073863000242289	0.5008300668368072

- **Internet**

This network is a directed and weighted network with 22962 nodes and 48436 edges. The modularity of OSLOM and Infomap were failed to compute, COPRA also failed to detect the communities as can be seen from the modularity.

Modularity			
COPRA	***GCE***	***OSLOM***	***Infomap***
0.08986423704950432			

5.3 Conclusion

As the discussion that made above from the comparison of the data, there are a brief summary can be made,

1. OSLOM is a good method which used statistically significance to detect communities, compares to Infomap, COPRA and GCE on their ability to detect overlapping communities, OSLOM showed its advantages that better than COPRA and GCE, but not better than Infomap. From the total cases that been successfully detected by each algorithms, according to the average NMI for each algorithms, the rank of the four algorithms should be Infomap > OSLOM > GCE > COPRA.
2. As can be seen from the data, OSLOM have advantages on detecting communities when the mixing parameters show higher percentage of random edges.

3. COPRA showed its advantages on detecting communities when the random edges in lower percentage.
4. OSLOM is a good algorithm in average, but it is not the best one, at least on the field of detect overlapping communities with lower percentage random edges.

Chapter VI

6. Critical Review / Evaluation

In this chapter, it will provide the critical reviews from different aspects of that relevant with this project. The evaluations with covers the techniques and tools that used in this project, which had discussed too much in the previous section. An overview evaluation to this project also will be seen in this section correspond to the success criteria that made in the interim report. In the end of this chapter, it will be a self-evaluation of the personal effort that made to this project.

6.1 Techniques and Tools

6.1.1 Networks

- **Synthetic Network – LFR benchmark**

LFR benchmark is very popular to use as a tool to generate the testing network graphs for the algorithms that aim to detect communities. It can generate overlapping communities with weight and direction. However, the directed and weighted attribute had not been used in this project because the network was set to be undirected and unweighted binary network.

The importance of testing the algorithms on the benchmark is, the original communities are accurately known by the researchers. The result from the detection algorithms can be compared with original communities with its similarity. The communities on a real network are difficult to known and controlled, it is harder to judge the algorithms by the result from a real world network. Furthermore, the parameters of the benchmark can be changed to discover how the detection algorithms work in details. The algorithms can be tested thoroughly by changing every parameter with a specific range, which is impossible to the real network.

The problem for the synthetic network is, it is made by purpose, programmed by algorithms, it is too perfect compared to real network, which cannot always use to replace the testing that run on real network.

In this project, LFR benchmark had showed its reliability to provide the well-generated artificial networks data that used to be tested.

- **Real Network**

There were three real network model had been selected to be tested. The characteristics of these three networks can be seen as the number of their nodes, which on the level of 1,000, 5,000 and 20,000, which are stand for the different size of the network. However, these networks are not all unweighted and undirected, which is not consistent with the benchmarks that all set to be unweighted and undirected.

These network models are originally in *.gml format, however, after produced to the valid format, which same to the network data generated by LFR, it can be well adapted by the detection algorithms.

However, the real network that chose for testing was not very appropriate that failed to obtain results for every algorithm, which is a problems need to be recorded and fixed as further work.

6.1.2 Algorithms

- **OSLOM**

From the research and experiment we can see, OSLOM is a very good algorithm, which detect communities by statistically significance on network graphs. It is quite powerful that widely adapt to most of the community structures. It showed high reliability and accuracy compare to most of the other detection algorithms. In the experiment that dedicated to overlapping communities, it showed its stability on detection. Even it failed to detect the communities on some cases but other algorithms achieved. But as showed from NMI values, it normally works better than other algorithms except Infomap, which showed better average NMI than OSLOM. The biggest problem for OSLOM is it speed, which is extremely slow compares to the algorithms like COPRA and GCE etc. It was found appropriate with the cases with higher value of mixing parameters as OSLOM can reveal its advantages.

- **Infomap**

Infomap is another very powerful algorithm like OSLOM, which uses random walk different from OSLOM uses statistically significance. Infomap also can detect on most of the community structure but it cannot support the detection on random graph and cannot support to detect homeless vertex compared to OSLOM. Infomap shared the same problem with OSLOM. Although the communities grouping from it averagely are better than other algorithms and about similar with or better than OSLOM, it speed in some cases especially on the large graph with thousands nodes is too slow that required very high performance on the testing devices.

- **COPRA**

COPRA is a very fast algorithm that uses label propagation. It can support the detection to disjoint and overlapping communities. The result from COPRA as showed in the experiment is not as good as OSLOM and Infomap. However, it showed great advantage on the network with lower mixing parameter. By increase the times of repetition and the value of v, which stand for each vertex can belong to v communities as maximum. The result can be better. However, as the effect is not very obvious to change the times of repetition, the parameter r, which stand for repetition was left as default. Also because the concentration of this project was focus on OSLOM, which stand for the algorithm that uses statistically significance, COPRA was not keen to put too much effort and times to let it did it best on detection.

- **GCE**

Similar to COPRA, GCE is not the centre of this project. It was just interested by its advantages on detecting overlapping communities. But it was not expect to waste too much time on it. GCE uses fitness function maximization, which also is an algorithm focus on overlapping communities as well as disjoint communities. It is not as good as OSLOM and Infomap as shown by NMI by average.

6.1.3 Measures

- **NMI**

NMI is short for normalized modularity information, which is the most used measure to evaluate the communities detected by the algorithms compared with the original communities generated in the synthetic network. It is not the only properties can be used to judge an algorithms, however it is the most important property. The other criterion like run time of the algorithm is not appropriate to apply in this project. Because it is too obvious that COPRA and GCE are much faster then OSLOM and Infomap, it is not comparable on the run time.

- **Modularity**

Modularity is the measure of the quality of partition. It compares the original network data with the grouping of communities result from the algorithms. It applied in the cases that run on real network, which the original communities are unknown, that it not possible to compute the NMI value.

6.1.4 Programming Languages

- **C#**

Although C# is such a powerful programming language that support by Microsoft's .NET framework which same the energy on programming by call functions in library, however, in this case, it is not very convenient due to the incompatible with Linux, which limited the testing execute on better devices.

6.1.5 Environment

- **MAC OS X on a 2.53GHz Intel Core 2 Duo MacBook Pro**

The OS is not the problem, it well supported all the implementation of the algorithms and C# with plug-in. However, the device/hardware is not powerful enough to support the experiment execute in very ideal status. In another word, the experiment went too slow than it run on a better device with high configuration of hardware, e.g. the desktop in the lab MVB2.11. This restrict the experiment result be verified by run the testing the second time or add extra testing cases to approve the comparison with much persuasive data.

6.2 Project

6.2.1 Success Criteria

According to the success criteria that list in the interim report as followed, an evaluation of this project can be made.

- If the aim of the project has been achieved. Can statistically significant based communities detecting can be proved as a better solution compared to other method.
  Yes, from the previous research, experiment and analysis progress, a conclusion can be given to the objectives of this project.

  First of all, of the four algorithms that tested, OSLOM had showed its advantages

that equals or over other algorithms in most of the testing cases. We can see on most of the time, OSLOM is a very good algorithm. However, it did failed in some specific circumstances. From the experiment we can draw this conclusion, detect statistically significant communities on network is a good approach that better than most existed algorithms. However, depends on the condition of the network, this approach is not definitely the best choice. It had found the cases OSLOM failed to detect communities but some other algorithms did it appropriately or at least did better than OSLOM depends on the result of NMI.

- As OSLOM is the only existed method that used to stand for statistically significant based communities detecting method, so it should be appropriate to represent the whole class of method that base on this theory. If it is not appropriate, what else that can use to help to achieve the aim of this project.

  From the research that made in the previous stage, OSLOM is the only existed method using the concept statistically significance. From the testing result from OSLOM, it shows better performance than the algorithms that using other techniques to detect communities. The only problem for OSLOM is its speed, which is terribly slow compares to some algorithms might be not as powerful as it, but absolutely much faster than it. However, OSLOM is a good algorithm that can well represented the algorithm that based on statistically significance.

- If the testing results are valuable to make comparison to the performance of the detecting methods. In the worst case, the results will be obtained, but that is not useful to judge whether detecting based on statistic significant can provide better implementation of detecting methods.

  The experiment has successfully obtained plenty of data from the algorithms that can support the analysis to the result and able to make comparison to the algorithms. It is believed that all the testing results are useful and valid. The only problem is the testing cases are not set to very large range due to the time issue. More testing cases may provide more information about the algorithms, and corresponded consume more time.

- In this case, does the result of the testing cases clearly showed available data to judge the detecting methods. This is to measure if the testing cases are well designed or not.

  Yes, it was believed that the testing cases are well designed and the results are valuable to use to judge the algorithms. From the graphs and analysis showed in above section, we can say the experiment had support this project finished successfully.

- If this project is complete in time, that means followed the instruction of the project management documents. Every process is working under control as it been expected.

  This project had been completed in time strictly followed the project management documents as planed and expected. All the planed tasks had been finished. There were some compromises on the time when for a particular stage was not finished

in time, however, this kind of delay had not effect the progress of the project. All the tasks went through smoothly without too much struggling, this could be seen as benefit from a successful management to the whole progress of the project.

6.3 Self-performance

As a critical review, firstly, the problems needed to be pointed out. After the whole project finished, there are several points can be emphasized,

1. Lack of understanding to the algorithms that researched on from the technical prospect.
2. The reading and noting to the papers that researched on were to rough, which missed the important points sometimes.
3. Lack of researcher's sprit that required the effort to check over the experiment over and over again to guarantee the reliability of a proposed opinion.
4. Lack of awareness to cite other researchers' work all the time.
5. Haven't put extra time in this project to make it better as possible.
6. Made wrong decision to run the experiment on a laptop that it cannot not support very well, which finally found was a disaster that seriously restrict the flexibility of the experiment.
7. Not enough communication with supervisor for instruction and supervision.

Despite of the problems, there are also some efforts that made to support the succeed of this project,

1. Strictly followed the plan and finished the project in time.
2. Specific research on the relevant techniques to accumulate enough information and resources that can support this project.
3. Made an appropriate experiment plan with the suggestion from supervisor.
4. Appropriate analysis to the experiment and achieved the objectives of this project.
5. Adjust the plan when unexpected situation happens, e.g. system down, lost of testing result and experiment interrupted etc, to help the project finally finished successfully.

Chapter VII

7. Further Improvement

The objectives of this project have been believed that have achieved according to the very specific discussion that stated above. However, it is very hard to say this topic had been done perfectly, there are still lots of details and further works can be done to enhance this project and to verify the results from the experiment to make it more persuasive. We had mentioned some points that need to be considered previously. In this section, we will list all the possible enhancements that need to be done in the future as the extension of this project.

7.1 In Research

- The research width of the related techniques was enough. What can improve in the future is try to enhance the depth of the research to comprehensively understand how and why the algorithms were been designed like that, instead of just know how to call the methods and the concepts of the algorithms. It will be very helpful to give much deeper analysis and comparison to the algorithms.

- As stated above, LFR benchmark was developed by the same group of people with OSLOM, which the results on LFR benchmark may benefits to OSLOM, as the developer knows how to get better result on it. For this reason, the research need to focus on any benchmark / synthetic network that proper to all the algorithms that been tested.

- As to the real-world network, currently, the resources of the network model that found are second-hand from Internet, which collected and used by other people. It is preferred to collect and make first-hand data to model some real network to be tested. It is believed that the first-hand data can be understood and used much appropriate.

7.2 In Experiment

- The experiment needs to be run the second time on a different environment such as Linux or Ubuntu to clarify the doubt on the correctness of the result communities. Some coding works possibly need to be re-done to adapt to the new environment. E.g. Linux in lab cannot run .NET framework, so it may try to use JAVA or C to re-program the functions to let it work in the lab.

- The setting of benchmarks can be changed to run the testing on larger range of parameters, which means more testing cases. Like more (5000) nodes on a bigger graph, less (1000) nodes on a smaller graph, more nodes on a smaller graph or less nodes on a bigger graph etc. The range of mixing parameter can be increase from 0.1-0.5 to 0.1-0.8, etc. That will be helpful to observe the performance of the algorithms.

- More community structures with more algorithms can be tested. In the previous work, the experiment was just focus on overlapping communities. However, OSLOM is an algorithm that widely adapt to almost all type of community structures. The hierarchical communities also can be tested although currently there are not too much algorithms are available to hierarchical structure, however, Infomap is just an instant that can be use to detect hierarchical communities.

- As the testing on the real network went not very well, it should be tested again.

7.3 In Analysis

- The analysis in the previous work was kind of shallow that just stated the obvious feedback from the data. In the further work, the analysis is expected to go much deeper to dig out the reasons that cause the problems/differences from the theory and implementation of the algorithms.

- It is expected finally from the analysis the strength and weakness of detecting statistically significant communities can be dug out, but not only gave a judgment of if statistically significant communities can always work better than other algorithms.

- In the previous analysis, OSLOM was found be affect by mixing parameter. In the future research, it hoped could be found out if it can be affected by other parameters, e.g. average degree.

- It also expected to researched the reason why OSLOM trends to fail to detected the cases with lower mixing parameter.

Chapter VIII

8. Conclusion

The whole progresses of this project from chose the project topic to final submission of the thesis has gone through about 8 months. From this over half-year time, there are lots of experiences that had been accumulated from different dimension.

Firstly, my sight to the filed of communities detection was widely opened. To be honestly, before this project, I have no idea about what is vertex and edge in network. During the period of doing this project, I gained comprehensive understanding of communities on network and the techniques to detect the communities. My project also required research on the relevant prospect of this topic. I also need to know how to judge these approaches that can detect the communities, and what else that can be done after the communities have been found. E.G. colouring the communities or hide them in advance etc. After all, I realized communities' detection is a wide area can be studied on. There are tons of techniques can be applied in this area. And it developed very fast. Most of the techniques I was used in my project were published after 2010. Hence I found this area still had great possibilities for discovery in the future.

Secondly, despite of the technical knowledge that relevant to my degree, my project management skill also been improved. I proudly found when I did my master degree project, everything went more smoothly than when I was doing the BSc project, it seemed most of my schedule was been finished in time without exception or unexpected technical problems or delay. Although the progress was not faultless that still showed several problems like lack of instant communication with supervisors, research sometimes went to roughly that need to be re-work on it again, the notes haven't been take very well thus forget where to cite from when doing the report etc.

As it is my first research project ever before since my last development project, I found there are lot of differences between these two types of projects. In a development project, I put most effort on coding. But this time, my concentration was mostly set on testing and analysis apart from the research section. I was put lot of time on consider how to support a successfully experiment, which may similar as a development progress. However, the most excited part of a research project is to analyze the obtained data from the experiment and propose a persuasive idea from the analysis. In this point, the researcher is required to be honest with the data and with intelligent to interpret as much from the data.

Finally, appreciations and respects need to be given to Steve Gregory, who is my supervisor who supports me a lot on this project. I am also very grateful to Bowen Yan, who provides me lot of technical support to me that saved my time and save me from wrong research direction. This project would not be success without their help.

Bibliography

[1] M.E.J. Newman, Detecting community structure in networks, Eur. Phys. J. B 38 (2004) 321–330.

[2] L. Danon, J. Duch, A. Arenas, A. Díaz-Guilera, in: C. G., V. A. (Eds.), Large Scale Structure and Dynamics of Complex Networks: From Information Technology to Finance and Natural Science, World Scientific, Singapore, 2007, pp. 93–114.

[3] S.E. Schaeffer, Graph clustering, Comput. Sci. Rev. 1 (1) (2007) 27–64. [16] S. Fortunato, C. Castellano, Community structure in graphs, in: R.A. Meyers (Ed.), Encyclopedia of Complexity and Systems Science, vol. 1, Springer, Berlin, Germany, 2009, eprint arXiv:0712.2716.

[4] M.A. Porter, J.-P. Onnela, P.J. Mucha, Communities in networks, Notices of the American Mathematical Society 56 (2009) 1082–1166.

[5] A. Lancichinetti, F. Radicchi, J.J.Ramasco and A. Fortunato, Finding statistically significant communities in network, (2010) arXiv:1012.2363v2 at www.arXiv.org.
[6] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113

[7] L. Donetti, M.A. Muñoz, Detecting network communities: a new systematic and efficient algorithm, J. Stat. Mech. P10012 (2004).

[8] A. Lancichinetti, S. Fortunato, J. Kertész, Detecting the overlapping and hierarchical community structure in complex networks, New J. Phys. 11 (3) (2009) 033015.

[9] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large network, Phys. Rev. E 70 (6) (2004) 066111

[10] B. Yan, S. Gregory (2009) Detecting communities in networks by merging cliques, 2009 IEEE International Conference on Intelligent System (ICIS 2009), pp 832

[11] S. Fortunato, M. Barthelemy, Resolution limit in community detection, Proc. Natl. Acad. Sci. USA 104 (2007) 36-41

[12] B. H. Good, Y-A. de Montjoye, and A. Clauset, Performance of modularity maximization in practical contexts, Phys. Rev. E 81, 046106 (2010)

[13] S. Fortunato, C. Castellano, Community structure in graphs, in: R.A. Meyers (Ed.), Encyclopedia of Complexity and Systems Science, vol. 1, Springer, Berlin, Germany, 2009, eprint arXiv:0712.2716.

[14] L.C. Freeman, A set of measures of centrality based on betweenness, Sociometry 40 (1977) 35–41.

[15] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99 (12) (2002) 7821–7826.

[16] S. Gregory, An algorithm to find overlapping community structure in networks, in: Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD 2007, Springer-Verlag, Berlin, Germany, 2007, pp. 91–102.

[17] U.N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (3) (2007) 036106.

[18] S. Gregory (2010) Finding overlapping communities in large network by label propagation, New J. Phys. 12 103018

[19] B.D. Hughes, Random Walks and Random Environments: Random Walks, vol. 1, Clarendon Press, Oxford, UK, 1995.

[20] M. Latapy, P. Pons, Computing communities in large networks using random walks, Lect. Notes Comput. Sci. 3733 (2005) 284–293.

[21] M. Rosvall and C. T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proc. Natl. Acad. Sci. USA, 105, 1118 (2008).

[22] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of

complex networks in nature and society, Nature 435 (2005) 814–818.

[23] A. McDaid and N. J. Hurley, Using Model-based Overlapping Seed Expansion to detect highly overlapping community structure, in ASONAM 2010 (2010).

[24] C. Lee, F. Reid, A. McDaid, and N. Hurley, Detecting Highly Overlapping Community Structure by Greedy Clique Expansion, in SNA-KDD (ACM, Washington, DC, 2010) pp. 33-42 (2010)

[25] A. Condon, R.M. Karp, Algorithms for graph partitioning on the planted partition model, Random Struct. Algor. 18 (2001) 116–140.

[26] R. Guimerà, L.A.N. Amaral, Functional cartography of complex metabolic networks, Nature 433 (2005) 895–900.

[27] A. Lancichinetti, S. Fortunato, and F. Radicchi, Benchmark graphs for testing community detection algorithms, Phys. Rev. E, 78, 046110 (2008).

[28] A. Lancichinetti, S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities, Phys. Rev. E 80 (1) (2009) 016118.

[29] L. Danon, A. Díaz-Guilera, J. Duch, A. Arenas, Comparing community structure identification, J. Stat. Mech. P09008 (2005).

[30] J. Kleinberg, An impossibility theorem for clustering, in: Advances in NIPS 15, MIT Press, Boston, USA, 2002, pp. 446–453.

[31] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks Phys. Rev. E69 026113 (2004)

[32] M. E. J. Newman, Modularity and community structure in networks Proc. Natl Acad. Sci. USA 103 8577 (2006)

[33] V. Nicosia, G. Mangioni, V. Carchiolo and M. Malgeri, Extending the definition of modularity to directed graphs with overlapping communities J. Stat. Mech. P03024 (2009)

[34] V. Nicosia, G. Mangioni, V. Carchiolo and M. Malgeri, Extending modularity definition for directed graphs with overlapping communities arXiv:0801.1647 (2008)

[35] W. M. Rand, (1971) Objective criteria for the evaluation of clustering methods J. Am. Stat. Assoc. 66 846

[36] L. Hubert and P. Arabie, Comparing partitions J. Classif., 2 193 (1985)

[37] A. L. N. Fred and A. K. Jain, Robust data clustering Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition pp 128–33(2003)

[38] L. M. Collins and C. W. Dent, Omega: A general formulation of the Rand index of cluster recovery suitable for non-disjoint solutions Multivar. Behav. Res. 23 231 (1988)

[39] A. Lancichinetti, S. Fortunato, Community detection algorithms: a comparative analysis Phys. Rev. E80 056117 (2009)

[40] M. E. J. Newman, Phys. Rev. E 74, 036104 (2006).

[41] D. J. Watts and S. H. Strogatz, Nature 393, 440-442 (1998)