Summary

Community detection problem has raised lots of interests of network study. With the proliferation of community detection algorithms, a problem appears that how can we choose an algorithm to output a better community solution without any knowledge of algorithms. As machine learning thinking: no one size fits all, we assume that each algorithm can outperform others in some certain problem space. This project considers the algorithms work on disjoint networks against those working on overlapping networks. Furthermore, fuzzy overlapping networks and crisp overlapping networks are also included in this paper. These are extensions compared with Leto Peel's study in [2] which has researched weighted and unweighted networks. We analyze those algorithms performances and then realize a method with which one can choose a class of algorithms. This method is feasible because study has shown that parameters of networks can affect algorithms performances, thus we can estimate parameters of networks and then in turn to estimate algorithm performances. Eventually, we adopt a simple classifier support vector machine to realize automatically selecting a class of algorithms.

Main contributions and achievements

- I found out the feature of networks *standard deviation of degree* can be used to estimate parameters of overlapping networks. See page 35, section 8.2.1.
- I wrote algorithms to compute mean value of clustering coefficient (CC) and standard deviation of degree (SD) then I plot scatters to analyze and confirm their effects on mixing parameter μ_t and overlapping parameter o_m such that we can realize parameter estimation of disjoint/ overlapping classification. See page 37-43, section 8.2.3, 8.2.4
- Another overlapping parameter o_n cannot be estimated for disjoint/ overlapping and fuzzy/ crisp overlapping classifications. See page 35-37, 43-44, section 8.2.2 and section 8.2.5
- Algorithms of this method have restrictions that for control comparison, we need to use an algorithm which has both class versions. On the contrary, an algorithm Infomap which works best for disjoint networks while bad in overlapping networks against an algorithm GCE suitable for overlapping networks rather than disjoint networks cannot be used as control groups. See page 45, section 8.3.1.
- I tested COPRA algorithm both disjoint version and overlapping version, results show that CC and SD can generate distinct algorithm performances, thus we can estimate class of algorithms. See page 46-50, section 8.3.3 and 8.4.1
- A classifier support vector machine (SVM) successfully predicted class of algorithms. See page 51-52, section 8.4.2

Acknowledgements

I would like to express my thanks to Dr. Steve Gregory for his precious and valuable guidance and advice during this project.

I would also want to thank Kirsten Cater and Julian Gough for supplying concrete and helpful feedback of my interim report.

Furthermore, I will thank my family for their support spiritually and materially.

Ke Shang, September 2011

Abstract

This project considers the problem that automatically selecting a class of community detection algorithms. Communities are those groups of nodes showing more connection in a network which can be detected by algorithms. However, recent study shows that although algorithms are designed in large scale, each of them is set to be fit for a specific problem space, as there is no unique definition of community. Often, networks are also generated with certain demands controlled by authors. This presents a great desire that one can choose a class of algorithms even not equipped with any prior knowledge of network community assignment and deep perceive of algorithms. By comparing disjoint network algorithms performance and those of overlapping network, we put forward a method to select class of algorithms by estimating parameters of artificial networks.

Table of Contents

1- Introduction

To better explore network communities, a lot of properties representing topology and features of networks have been developed by scientists. Among them, community structure is the one that has raised the most attentions of study. We assume that when groups of nodes show more connections by edges with those nodes also inside the groups than that with nodes between groups, community structure appears. This can be explained by a phenomenon *assortative mixing* in real world networks that is entities present a trend to interact with those similar entities. The objective of community detection algorithms is to find out the community assignments of a network.

However, along with the proliferation of community detection algorithms, choosing a proper algorithm becomes difficult. Since algorithms and their relative networks are designed by authors based on different understanding of community, each algorithm may only perform well in some specific range of network. The state of art to avoid this is by using benchmark networks and a comparative analysis [1]. Nevertheless, we still believe that for different class of algorithms, one class may outperform other classes in some certain regions based on machine learning "No one size fits all" [2].

Aims and objectives

According to recent study, structural properties can reveal algorithms performances and so do parameters of networks [2]. This project considers a method that by estimating parameters of benchmark, one can select a class of community detection algorithms automatically.

1. First, we define two classes of algorithms/ networks and then study and research to find out which features of observed networks can estimate parameters of these two classes of networks.
2. Confirming that parameters of networks can affect algorithm performance such that structural properties can in turn affect algorithm performance as well.
3. Using a classifier to predict class of algorithms.

Structure

Chapter 2 will give us the background of networks. The artificial network dataset is discussed in chapter 3 and algorithms working on the networks are detailed in chapter 4. The way to evaluate algorithm solutions and algorithm performances is described in chapter5. Machine learning classifier algorithm support vector machine is introduced in chapter 6. Chapter 7 discusses Peel's method that I need to extend. Chapter 8 is the experimental part of this project gives all the results. Finally, a classifier support vector machine is adopted to confirm the conclusion of this project.

2 - Background

In this chapter, we will first gain some concepts of simple graph theory and then get to know some network properties which are playing an important role in community detection area. To be more specifically, graph theory provides the basic study environment and network properties are some features of observed networks, through which, we can exploit community structure of networks and further realize some specific goals.

2.1- Basic Graph Theory

A network is a graph which is made of *vertices* (or *nodes*) linked by *edges* (or *links*) and thus can be visualized as sets of lines connecting points. If the end nodes of each edge are in order, we call the graph *directed graph* (or *digraphs*). What's more, edges can carry *weights*, a real number, representing some information in real world, such as in social network, the link weights of two people may reveal the frequency they contacted with each other. Directed graphs with loops, i.e. a vertex connect with itself by an edge, are called *multigraphs*. We also consider those graphs with more than two edges between a pair of vertices as *hypergraphs*. There is a naturally partitioned graph we are interested in: *bipartite graphs*, i.e. graphs made up by two types of nodes with edges linking only between different types of nodes [3]. See Figure 2.1 to get better understanding of graph:

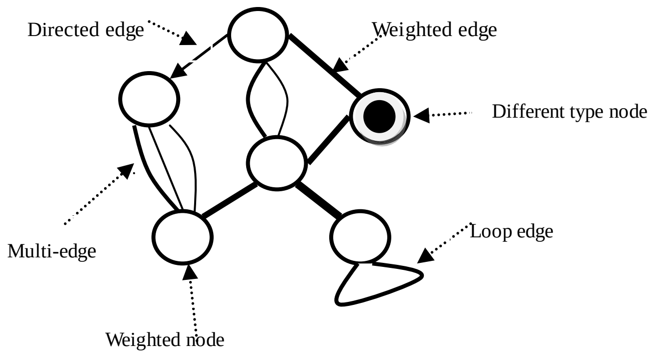


Figure 2.1: example of graph

We assume there are *n* vertices and *m* edges in a graph. The *maximum size* is defined as the total number of pairs of vertices, given the equation 1:

$$\text{Maximum size} = \frac{n(n-1)}{2} \tag{1}$$

Further, if two vertices are connected, they become *neighbors*, and the set of neighbors is called *neighborhood*. Besides, the total number of its neighbors is the *degree* k_v of the vertex v and thus we can get a *degree sequence:* $k_{v1,}k_{v2,} \dots, k_{vn,}$ by listing the degree of all the vertices. For weighted graph, the sum of linked edges is the *strength* of the vertex [3].

2.2 - Network properties

We assume the simplest network is a *random graph*. In this graph, *n* vertices and undirected edges are randomly placed, with probability *p* controlling $\frac{1}{2}$n(n-1) edges, i.e. maximum size edges are independently distributed by *p*. Meanwhile, degree distribution is according to a binomial distribution associate with large *n*.

However, real-world network is not a random graph. So we need to find other possible mechanisms and ways to help us obtain network structure and exploit its formation. Recent study of a wide range of networks has put forward some properties which have strong relationship with the behavior of the real networks, showing topology of those as well. In this part, we will describe *transitivity or clustering*, *degree distributions* and *community structure* [4].

2.2.1 - Transitivity or clustering

An obvious deviation can be seen between the real network and the random network mentioned above is called *network transitivity*, or in most time, we mention it *clustering*. In most cases, if vertices A and B are connected, B and C are connected, there exists a big chance that A and C are also connected. Translating this into context of social networks can be more meaningful that you and your friend's friend are likely to be also friends. Thus in the field of network topology, a *triangle* will presents in graph, i.e. three nodes make up a set in which each of them is connected by the others. *Clustering* expresses the number of triangles occurring in the graph. We define *Clustering Coefficient (CC)* as below equation 2 [4]:

$$\mathrm{CC} = \frac{3 * \text{number of triangles in the network}}{\text{number of connected triples of vertices}} \tag{2}$$

Where the connected triple refers to that a vertex with two edges connecting with two other vertices (see Figure 2.2). This formula CC computes the proportion of those triples which get the third edges to become triangles and thus the range of CC is [0, 1].

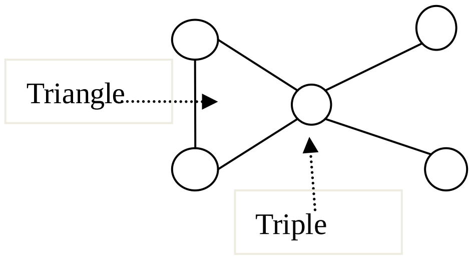


Figure 2.2: clustering coefficient CC, there are 8 triples and 1 triangle in this graph, thus CC = 3/8. [4]. Source: Ref. [4]

2.2.2 - Degree distribution

In random graph [5], each edge is present with same probability p_k, where p_k is the proportion of vertices which have degree k and thus random graph degree distribution presents binomial distribution. While in real world networks, most graphs' degree distributions are unlike binomial distribution, they normally will followed with a long right tail, i.e. right-skewed, we call the right tail part *power law* representing their degrees are higher than mean degree [4].

The way to measure this tail is to make a figure to show the *cumulative distribution* formula of p_k (equation 3):

$$P_k = \sum_{k'=k}^{\infty} P_k{}' \tag{3}$$

But we need to notice that this is not a best solution because the figure will not give a good visual effect of the degree distribution and also points on the figure are not good correlate to real data, since results after statistical computing will not independent any more. Below is a cumulative distribution example of a classical real network World Wide Web [6], there are 300 million vertices sets in the graph. It shows a power-law distribution and the straight-line between the horizontal axis and vertical axis represents vertex degree k and P_k respectively, is appears a doubly logarithmic scale (see Figure 2.3):

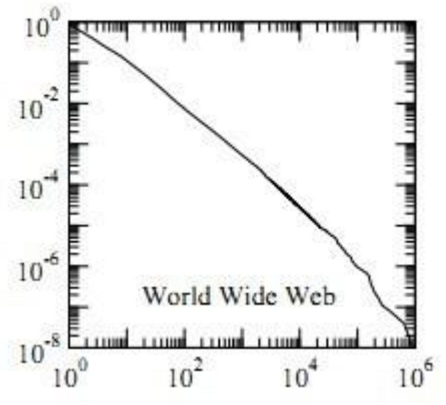


Figure 2.3: example of cumulative distribution. Source: Ref. [6]

Networks with power law in their tails are considered as *scale-free networks*: $p_k \sim k^{-\alpha}$ for exponent α. Scale-free means in a form, f(x) is fixed within a multiplicative factor under x, therefore power law is similar to scale-free, as f(ax) = bf(x) is the only solution.

2.2.3 - Community structure

In real world, most networks especially social networks show the property of *community structure*, which means the densities inside the sets of vertices are higher than that between them [4].

What causes community structure? The first origin of community structure is the *assortative mixing* pattern, i.e. vertices are tend to link with others who share same attributes, furthermore, the second possible reason is "*introduction*", according to clustering features, vertices connected to a same vertex are tend to become neighbors [4].

What is the relationship between community structure and clustering? Obviously, they are two different notations, but normally, if there is community structure, clustering coefficient is usually high, but not necessarily. Below is an example that there are three communities forming a community structure, while contains no clustering (see Figure 2.4):

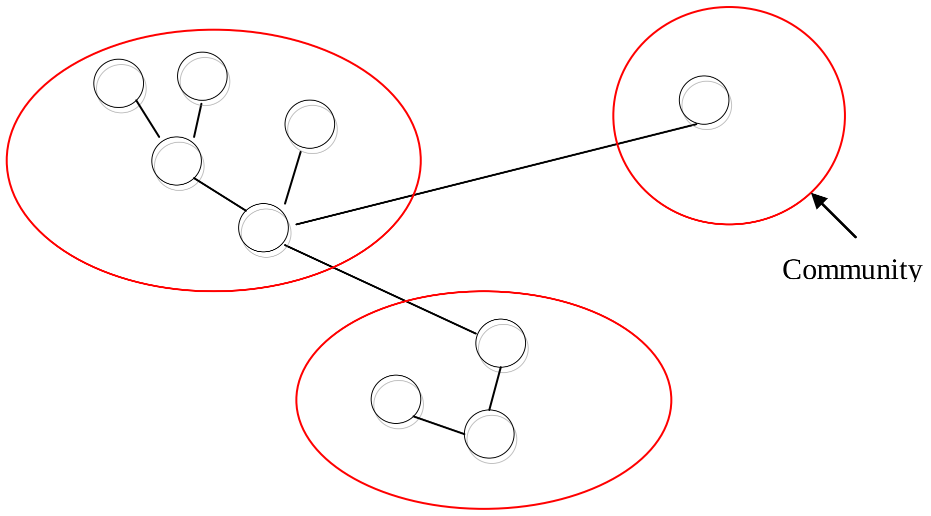


Figure 2.4: An example of a condition that there is no clustering but exists community structure.

2.3 - Types of network

There are different types of communities which can in turn make up different types of networks. The most common networks are *disjoint* network, by contrast, are *overlapping* networks. Furthermore, if we only focus on overlapping network, there are two types of overlapping networks: *fuzzy overlapping* networks and *crisp overlapping* networks.

2.3.1 - Disjoint network

A *partition* is with respect to disjoint type of community structure, in which each vertex is only assigned to one community [3]. An example of disjoint network called Zachary's karate club, different colors show different communities (see Figure 2.5):

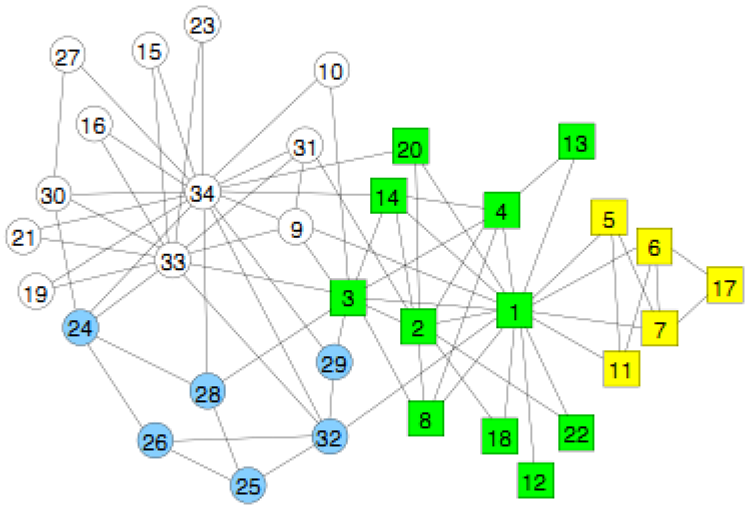


Figure 2.5: Zachary's karate club, a real network and disjoint as well. Source: Ref [7]

Most of algorithms designed to detect communities are assume that the networks are disjoint network. This can make sense in some conditions, such as in employee and employer relationship, many employees work for only one employer [8].

2.3.2 - Overlapping network

Conversely, we named a division of a overlapping communities *cover*. In overlapping networks, each vertex may belong to many communities. Below is an example of overlapping network of word association. Also, colors show different communities (see Figure 2.6).

Only a few algorithms are designed for overlapping networks. While overlapping networks are really important, for example, in social networks researcher normally belongs to more than one study groups [8].

2.3.2.1 - Crisp Overlapping network

In disjoint community of networks, if vertices i and j are in same community, the probability p_{ij} of the edge {i, j} is defined as p_{in}, or p_{out} vice versa. Further, if $p_{in} > p_{out}$, a community occurs. For crisp overlapping networks, the probability of an edge {i, j} depends on the number of groups that i and j are both in [10]. If i and j are belong to k communities, then $p_{ij} = p_k$ [11]. In [10], Gregory S. gave the simpler way to define p_{ij} (equation 4):

$$p_{ij} = p_1 \text{ if } \exists c \in C[i \in c \cap j \in c] \text{ else } p_0 \qquad (4)$$

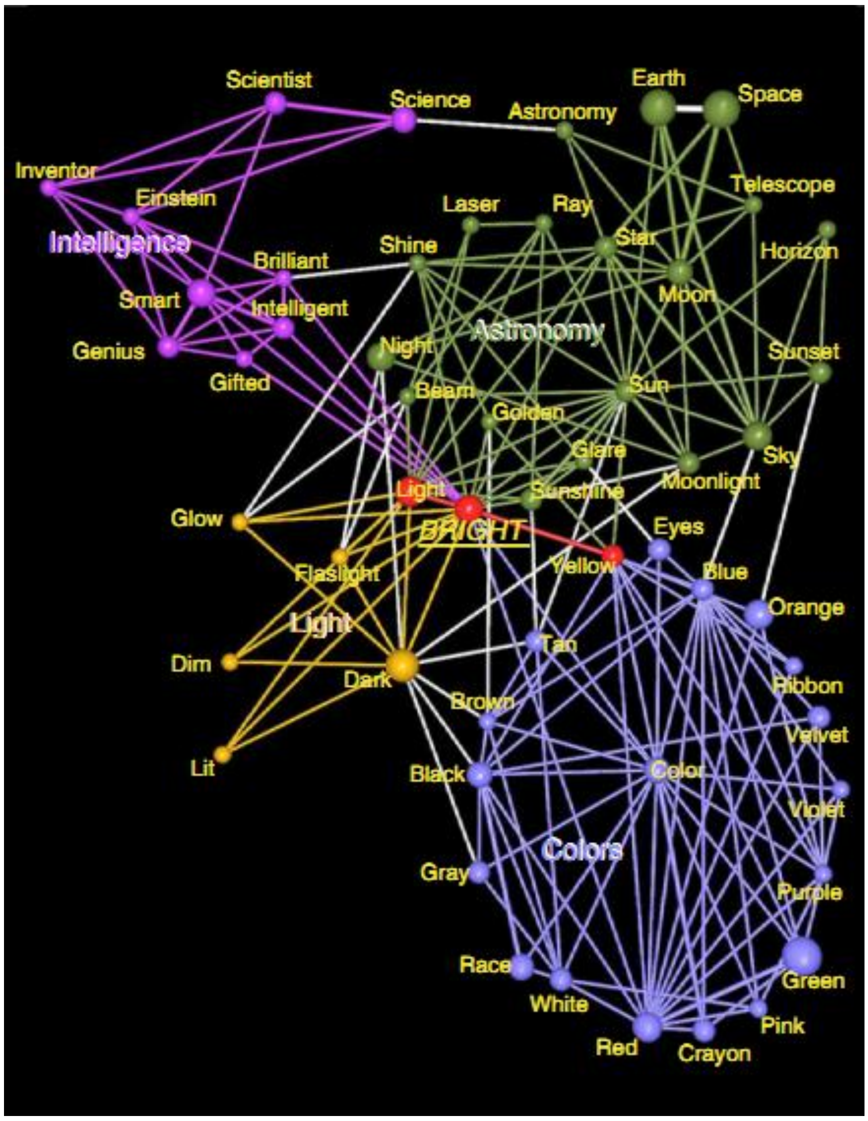


Figure 2.6: Overlapping network of word association. Four different colors represent Intelligence, Astronomy, Light and Colors respectively, of which, word "bright" belongs to all the communities. Source: Ref [9].

2.3.2.2 - Fuzzy Overlapping network

For fuzzy overlapping communities, the probability p_{ij} (equation 4) is affected by both the number of communities *i* and *j* share, but also by the *belonging coefficient* ∂_{vc} [10].

A *belonging coefficient* ∂_{vc} represents the strength of each vertex *v* with respect to their neighbors in same community *c*. The total sum of belonging coefficient of a vertex is 1, i.e.

$$\sum_{\mathrm{c}} \partial_{\mathrm{vc}} = 1 \tag{5}$$

We can see an example as Figure 2.7:

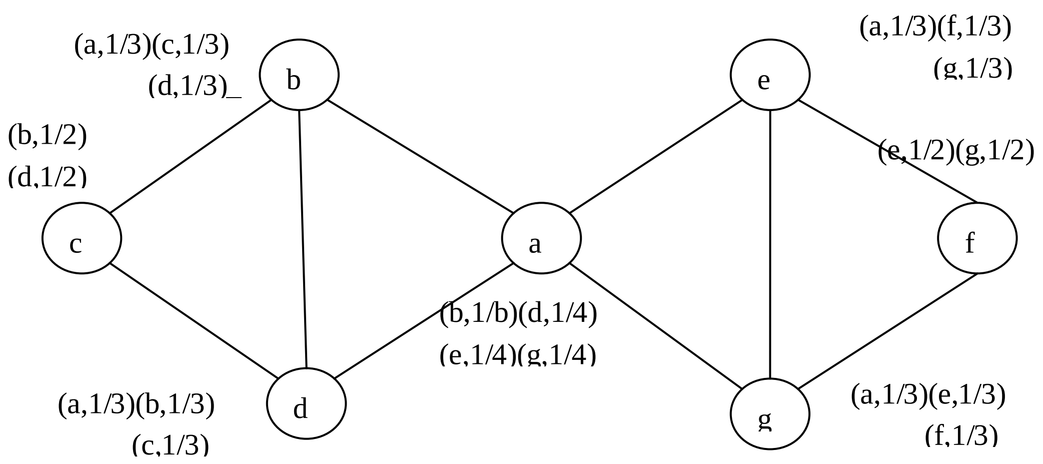


Figure 2.7: Fuzzy overlapping belonging coefficient [10].Source: Ref [10]

2.3.3 - Comparison among network types

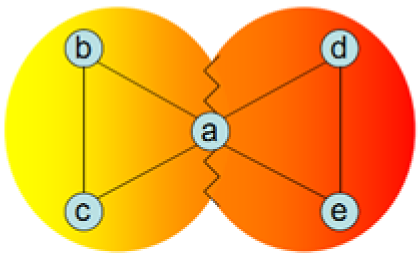


Figure 2.8: example graph for comparing different types of networks

For different types of networks, Figure 2.8 will be in different solutions. For disjoint networks, two partitions will be {{a, b, c}, {d, e}}. While for overlapping networks, node *a* should belong to two covers: {{a, b, c}, {a, d, e}}. Further, for fuzzy overlapping networks, node *a* is assigned to two communities with belonging coefficient weighting them {{(a, 1/2), (b, 1), (c, 1)}, {(a, 1/2), (d, 1), (e, 1)}}.

3 - Data

A good data set is a graph which can be drawn into a clear community structure, while real networks we got in hand are not enough to tackle all the community detection problems. Thus, we can generate artificial networks with some specific requirements which we should study about and also based on the basic notation of community [3].

Datasets are generated for testing algorithms performances. When an algorithm is designed, we should test its performance, while there are lots of applications with respect to different kinds of datasets, thus blindly rely on some algorithms and compare their performance will not make sense [3]. Due to this lack of standard and formal data, Girvan- Newman created a *benchmark* [12] based on a *planted l-partition model* [13]. However, this is still cannot reveal real networks' features. Thus, A.Lancichinetti enhanced the benchmark (*LFR benchmark*) [14] and then added some attribute controls so that the computer generated networks will be more like the real world networks [2, 3, 15], of which we will only mention *overlapping LFR benchmark* and *weighted LFR benchmark* [16].

3.1 - Planted l-partition model [13]

This is an algorithm which can generate artificial networks with community structure. The underlying idea of this model is the definition of community: If vertices i and j are in same community, the probability p_{ij} of the edge {i, j} is defined as p_{in}, or p_{out} vice versa. There will be a community structure when $p_{in} > p_{out}$ .i.e. intra-community density of edge is larger than inter-community edge density, [3]. On the country, $p_{in} \leq p_{out}$ means it is a random graph.

We listed parameters of this model as below:

n	number of vertices
l	number of groups
g	number of vertices in each group
p_{in}	vertices are linked with p_{in} probability in each group which forms a random graph with p_{in} connection probability.
p_{out}	vertices are linked with p_{out} probability between different groups
<k>	average degree of each vertex with $\langle k \rangle = p_{in}(g-1) + p_{out}\, g\,(l-1)$

Through this model, we can get a graph with $n= g*l$ vertices. However, we still need to discuss the drawback of this model: all the vertices in all the groups with same community size are of same degree, thus this model is not with respect to real world network community structures [15]. Researches show that normally the degree distributions are many vertices have low degree and a few number vertices have large degree affected by power law [15]. So the Girvan-Newman benchmark is considered.

3.2 - Girvan-Newman (GN) benchmark [12]

GN benchmark is designed based on planted l-partition model, the aim of GN is to enhance planted l-partition model and can make the artificial more like real world networks. Therefore, only two modified parameters are introduced into GN benchmark:

z_{in}	expected internal degree of a vertex with $z_{in} = p_{in}(g-1)$
z_{out}	expected external degree of a vertex with $z_{out} = p_{out}g(l-1)$

In GN benchmark, for those notations in planted l-partition model, they set $l = 4$, $g = 32$, thus $n = l*g = 128$, $<k> = 16$, Therefore, p_{in} and p_{out} are no longer independent parameters, their relationship can be indicated as below (equation 5):

$$p_{in} + 3p_{out} \approx 1/2 \tag{5}$$

After calculation, the modified notations z_{in} and z_{out} can be determined as equation 6:

$$z_{in} = p_{in}(g-1) = 31p_{in}$$
$$z_{out} = p_{out}g(l-1) = 96p_{out} \tag{6}$$

Now, we can get standard benchmark networks. Noticing that this vertex average degree $<k>$ is fixed at 16 which gives the disadvantages that all vertices are of same degree and community sizes are equal to each other. In addition, due to the simplicity of the benchmark, most algorithms perform well on this benchmark which is not a good result. Instead, we need more complicated and more realistic networks which will presence the heterogeneous distributions of degree and community sizes so that LFR benchmark is designed.

3.3 - LFR benchmark [14]

A.Lancichinetti *et al* added some attributes into GN benchmark to generate more realistic artificial networks, since normally real networks will carry weights and present overlapping feature. At the same time, power law is introduced into this benchmark. Let us see the notations of this benchmark:

τ_1	Power law exponent to generate degree distributions
τ_2	Power law exponent to generate community sizes
μ	Topological mixing parameter with range [0, 1]
$\frac{1}{1-\mu}$	Each vertex share this fraction of edge with others in same group, thus gives the equation: $k_{in} = (1-\mu)\,k$
$\frac{1}{\mu}$	Each vertex share this fraction of edge with others in different groups

Giving these power law attributes and mixing parameters, networks generated by LFR benchmark will be more realistic and more complex, thus presence the heterogeneous distributions of degree and community sizes such that can better lead to different algorithm performances. The complexity is linear in the link number, so one can test on large systems with small cost to analyze them [1].

LFR benchmark generated through the procedures below [3]:

1. By randomly pick a number from a power law distribution with exponent τ_2, we get a list of community sizes.
2. Using exponent τ_1 to extract degree distribution, each vertex *i* has an internal degree (1-μ) *k*.
3. All internal edges are randomly link to each other until no one is “alone”, thus we get the sequence of internal degree.
4. Add μk edges to each vertex i and then link them randomly with other edges that are not in the same group until no edge is “alone”.

3.4 - LFR benchmark with overlapping nodes [15]

A. Lancichinetti enhanced the LFR benchmark with overlapping attributes since real world network communities share vertices.

1. We first assign V_i *memberships* for each node i, i.e. *i* belongs to V_i communities. This is the different between the normal LFR benchmark in [14] in which V_i = 1 . Then we generate degree distribution as the LFR benchmark above using power law exponent τ_1
2. Community sizes $\{s_\epsilon\}$ are also generated by power law exponent τ_2. In this case, we can see $\sum_\varepsilon s_\epsilon = \sum_\mathrm{i} v_i$. At this point, we should consider which communities each node i belongs to. We can assign the nodes based on configuration model just like generating a bipartite network while with a constraint that the internal degree of each node cannot exceed the total number of nodes in the same group. To conclude, each community ε has s_ϵ links, and each node has v_i links.
3. Adding the external links. To randomly insert those links we first generate a new network G^{ext} of n nodes with degree distribution $\{\mu k_i\}$. Once two nodes are included in same two groups, we perform this rewire procedures to avoid more than one link will be set between the nodes:
   1) Assume A and B are in the same communities while they are assigned an external link such that they are included in G^{ext}
   2) We choose another node C which is not in the same community with A and B, in the meantime, C shares a edge with D which should not linked with B either.
   3) We cut the links between A and B, C and D while add links between A and C, B and D.

3.5 - LFR benchmark with weighted networks [15]

Links of real world communities are often carrying weights, so A. Lancichinetti also added the weight attributes in the normal version of LFR benchmark. This is also the benchmark used in L.Peel's paper [2].Because the weight parameter is playing an important role in his method.

First few steps are just like what we do in normal LFR benchmark, furthermore, we need to add a positive weight to each link. The procedures are as below:

1. Using power law exponent β to assign each node with strength, which is the sum of weights of links of each node.
2. To assign weights to each link we using a mixing parameter μ_w, works in the same way as topological mixing parameter μ in LFR benchmark.

4 - Algorithms

There are two basic lines of discovering groups. One way is *community structure detection* with respect to the real-world networks data, the number and size are depend on the network thus are not given artificially. The other method of detecting groups is *graph partitioning* which requires given the specific number and size of groups that we want the network split into [17].

In addition, traditional graph partitioning method is not good enough for it requires the specific number of communities as well as their sizes. Knowing all these details is based on the fact that we already know the community structures, while in normal cases, we just want algorithms themselves figure out that. Thus, we will focus on community detection algorithms to give the analysis of communities [3].

As mentioned before, there are huge amount of algorithms, each is designed on different assumptions of definition of network community, such that different techniques, such as label propagation [18], random walk [19] and fitness function [20, 21]. We will give details of those properties.

4.1 - Algorithms based on random walk

4.1.1 - Random walk definition

The most important property of random walk is *commute-time,* which is calculated by a random walker, starting from every vertex, to get to another vertex and then back to the starting vertex, *commute-time* is the average number of steps he needs. What the property shows is that inside communities, *commute-time* will becomes larger than that between communities, i.e. random walkers spend most time within communities.

4.1.2 - Infomap algorithm [22]

This algorithm is raised by Martin Rosvall and Carl T. Bergstrom in [22], the underlying idea is to combine the information theoretic approach with community detection, they adopted the random walk to represent the information flows and then analyze the graph into modules. This can be done by compressing the description of each node and the corresponding information flows. Eventually, we can get a map which briefly showing the communities regularities and their relationships.

Best network map can reveal important information but it needs to fulfill some requirements: minimal bandwidth, and better compressions. The information the map conveys can help us better understand network behavior, thus we need to detect the most efficient coarse-grained description of the flowing information.

The algorithm procedures can be described as below:

1. Describe a path of a network:

We need to find a code that can describe the information flows, which means we are not only aiming maximal compression that by assigning every node with a code can be achieved, but also a language that can express the network structure and real meaning of things in real world. To realize that, we can encode those groups whose nodes' information flow faster and easier with unique names. We call those groups as modules. We can see an example: (see Figure 4.1)

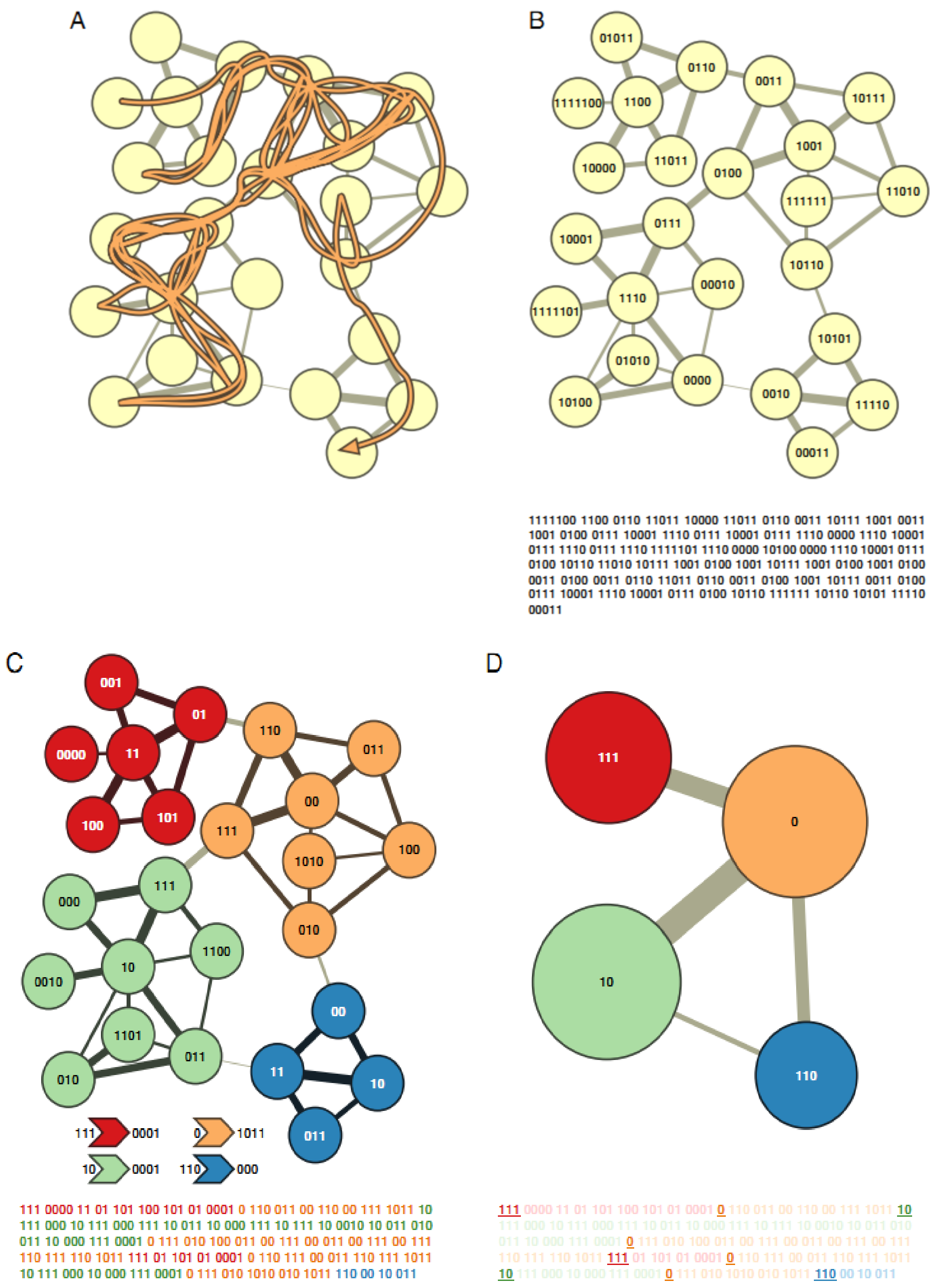


Figure 4.1: Infomap algorithm example with 25 nodes. Source: Ref: [22]

In Figure 4.1.A, the orange thickness line represents the random walk traverses the relative links. There are 71 steps in total to finish random walk such that we can get our information flows.

2. Huffman Coding:

   This is a method we can assign each node a unique name (code) with 314 bites in total. Those codes below Figure 4.1.B indicate the starting node and the end node, such as the information flow starts from 1111100 to 1100 and so on, finally get to 00011. Meanwhile, codeword lengths are computed from visiting frequencies, the higher frequency, the shorter codeword it will be. However, even if we find a good way to describe each node, we still cannot get the map, so we need to carry out the next step.

3. Highlighting Important Objects:

   To get the best map of the network, we need to further optimize by dividing network into two-level descriptions: assigning modules with unique names and reusing short node names inside each module [2]. We can make the analogy from U.S. city street names: Every city is assigned a unique name, while the street names are all the same. After this step, we successfully reduce the 314 bites code of former step to 243 bites as in Figure 4.1.C. Random walk starts from 111 bits indicates the information starts from red module, and adjacent 4 bits 0000 represent the starting node in that, etc.

4. Reporting module names:

   As we can see from Figure 4.1.D, a coarse-graining network can be achieved by only reporting the module names rather than detail locations inside the modules.

The results of Infomap algorithm are considered to be the best during the comparative analysis in [1]. Moreover, the speed of the algorithm is quite fast with a linear system size complexity such that is suitable for large networks.

4.2 - Algorithms based on fitness function maximization

4.2.1 - Fitness function [23]

Fitness function focuses on local structure, i.e. vertices themselves and their neighbors. Local structure reveals significant information in real world network. Such as in social network, groups tend to be local rather than covering the whole human being [23].

The fitness function formulation is defined as below (equation 7):

$$f_G = \frac{k_{in}^{G}}{(\,k_{in}^{G} + k_{out}^{G}\,)^{\alpha}} \tag{7}$$

Where k_{in}^{G} express the overall internal nodes degree of community G and k_{out}^{G} is the total external nodes degree of G. α is a parameter representing the size of the communities.

Since we know how to compute fitness, then what is *node fitness*? Assuming a fitness function is with respect to a node A of community G. we get the equation 8:

$$f_G^A = f_{G+\{A\}} - f_{G-\{A\}} \tag{8}$$

Where $f_{G+\{A\}}$ means the fitness with node A inside group G, and $f_{G-\{A\}}$ is the fitness with node A outside group G.

We can then conclude the *community average fitness* $\overline{f_P}$, the parameter α reveals the size of communities. Large value of α corresponds to small size community, and vice versa. From researches we conclude that when α equals 1, the cover found is relevant. While like modularity, if we stick to one value of α, there will be a weakness that resolution is constrained. Thus, we will change the value of α, to value the cover we found in various community sizes.

$$\overline{f_P} = \frac{1}{n_c} \sum_{i=1}^{n_c} f_{G_i}\ (\alpha = 1) \tag{9}$$

It is obvious that when a cover is *stable*, it will cover a large range of α. Under this circumstance, stable cover can resists being destroyed by subtly changing α.

4.2.2 - LFM algorithm---- finding hierarchical and overlapping community [23]

Based on *fitness function maximization*, we can detect the natural community which can be both hierarchical and overlapping. To realize detecting hierarchical community, we explore each level and size of network by modifying resolution parameter. At the same time, we perform the algorithm on each node, through this way, we get overlapping communities.

Algorithm procedures are listed below and the worst-case computational complexity is $O(n^2 \log n)$.First, we detect natural community of node A:

1. Initially, group G includes node A with $k_{in}^{G} = 0$. We calculate all the fitness of A's neighbors, then include the one with largest fitness thus get a larger group G'
2. Recalculate fitness of all node in G'
3. If we get a negative fitness, the node is removed to form a new group G''
4. If step 3 happens, we iterate step 2 and step 3. Otherwise, we start from step 1 for group G''.

Algorithm stops when we iterate step 1 and get all negative number from nodes' fitness.

Second, we extended the algorithm to be suitable for overlapping communities.

1. Pick node A randomly
2. Detect natural community of node A using above algorithm
3. Pick node B randomly
4. Detect natural community of node B using above algorithm
5. Iterate from step 3 to step 4

4.2.3 - GCE algorithm ---- finding highly overlapping community [24]

This *Greedy Clique Expansion (GCE) algorithm* is raised by C. Lee, F. Reid, A. McDaid and N.Hurley in [24], based on the state of art that almost no algorithm performs well on highly overlapping networks. Considering community fitness, they made some modifies on LFM algorithm (see section 4.2.2), since they have found that an upper bound number of communities can make the algorithm performance poor, i.e. LFM algorithm is not suitable for the networks that a vertex may belong to large number of communities. The main innovative points in GCE algorithm are first detecting cliques to be seeds, and then by using greedy optimizing local fitness function (see section 4.2.1) they can expand the seeds identified before. In the end, the algorithm has been tested to be the only one that is suitable for artificial highly overlapping networks.

Algorithm procedures can be condensed as below:

1. Expanding an unique seed

   We assume G in the fitness function (see section 4.2.1) is a subgraph of S and can be considered as the core part of community C and we then can define G to be the single seed. More concretely, all the nodes in G are belonging to C, while not all nodes C contains are included in G. The next step is to expand G until all the nodes of C are embodied by G.

   This can be done by calculating fitness of each red node (see Figure 4.2), frontier nodes of the seed and the one which can contribute maximum fitness will be added.

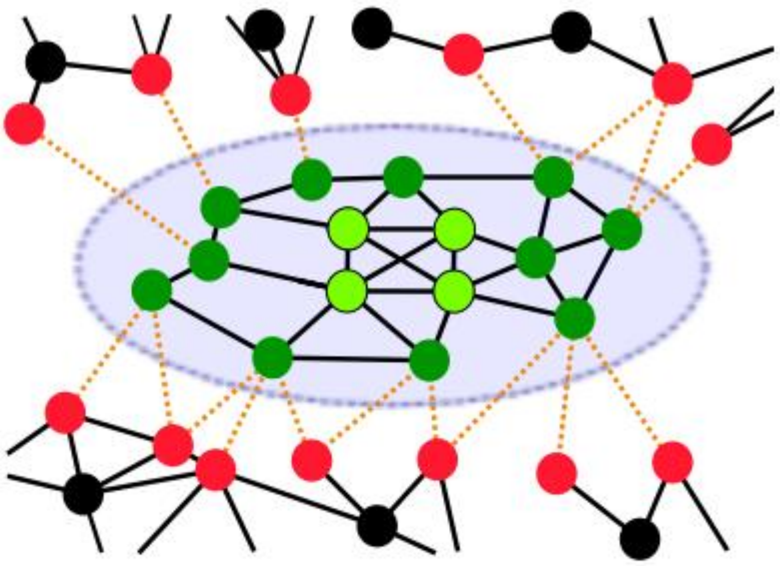


Figure 4.2: An ideal community with seed (green nodes) in the middle. Red nodes represent frontier nodes during the greedy expansion procedure. Source: Ref [24]

2. Choosing cliques as starting seeds

   This step is the innovative part comparing with LFM, since LFM chooses nodes randomly that are not have been assigned to any community before and keeping loop this step until all nodes are assigned to a community. However, an implicit drawback occurs, at the time when all node belong to a community, the algorithm will stop detecting communities, which is against the highly overlapping community structure.

   To avoid this defect, GCE identifies maximal cliques to be the starting seeds. Cliques are sets of nodes that are total connected while not embodied by any other sets of fully connected nodes (see Figure 4.2). By setting a major parameter *k*, clique minimum node number, we can ensure all possible seeds will be found. At the mean time, to confirm a good community structure generated, we need to keep *k* large enough. Therefore, *k* set as 3 or 4 are best choices.

3. Discarding duplicate communities

   Generating duplicate communities will happen to almost all the overlapping network algorithms, one way to solve that can be measure community distance, which can help identify what is near-duplicate communities, and then once assure, we just get rid of that result.

   We define the community distance as below:

$$\delta_E(G, G') = 1 - \frac{|G \cap G'|}{\min(|G|, |G'|)} \tag{10}$$

   Where G and G' are two given communities. This formula can reveal the proportion of smaller one's nodes which are not belonging to larger one. After comparing lots of communities, we can locate the near-duplicate community with the smallest community distance.

Applying to GCE algorithm, when we get a candidate community C' after the first 2 steps, we test whether C' is within the near-duplicate community distance. If so, we will discard C', or otherwise, we can accept C' as a community.

4. Loop step 2 and 3 until no seed can be detected.

Compared with other overlapping network algorithms, GCE works the best among them. We can conclude that from the comparative analysis in [31] (see Figure 4.3)

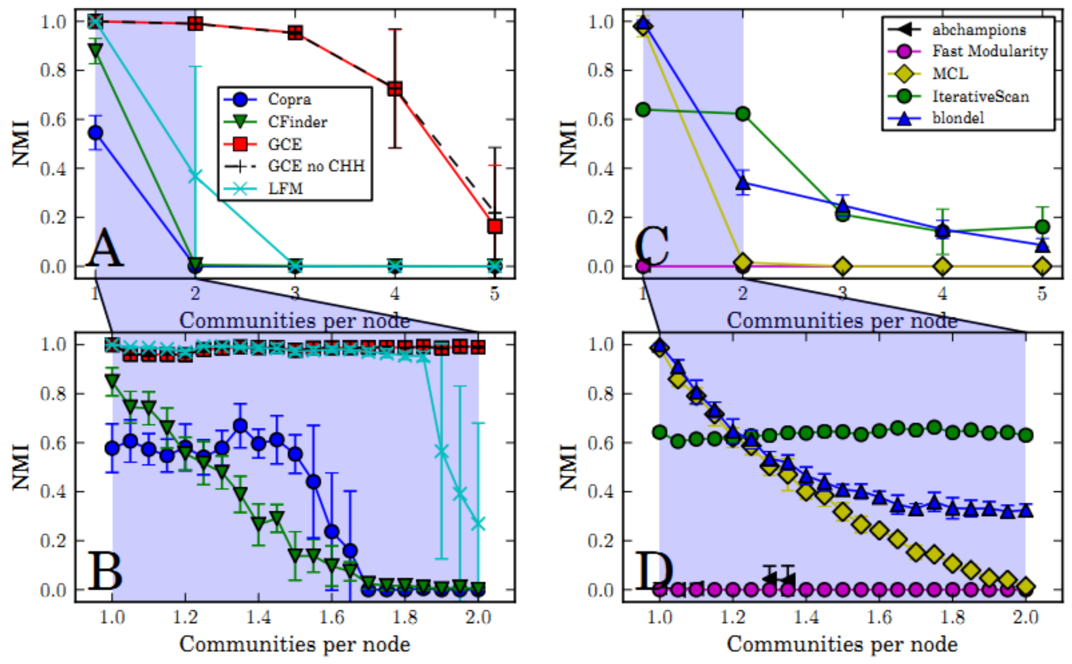


Figure 4.3: NMI values comparison among community detection algorithms on LFR benchmark when we set the overlapping community number increasing on the X axis. Source: Ref [24]

4.3 - Algorithms based on label propagation

4.3.1- Label propagation [18]

This is an algorithm raised by Raghavan in [18] which is also called RAK algorithm. It has no parameters thus can only detect disjoint communities. Also, it's a quite simple algorithm such that the complicity is near-linear.

The procedures of label propagation are concluded as below and we can see an example of that (see Figure 4.4):

1. We start with labeling all vertices a unique integer number as a label.

2. Update vertex x with the label of its neighbor whose label is used by most of its neighbors. If there are more than one most shared label, we choose randomly. We iterate this step until each vertex has the most shared label by its neighbors.

3. All members with the same label tend to become a community.

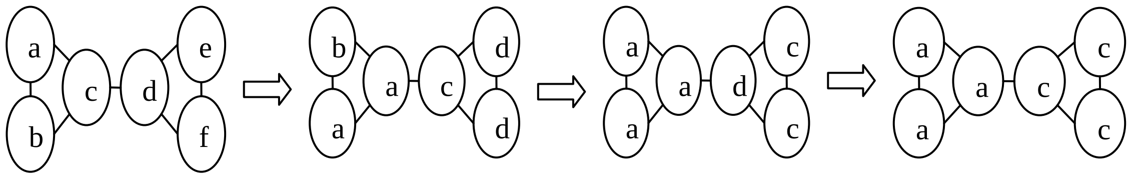


Figure 4.4: an example of label propagation

4.3.2 - Community Overlap PRopagation Algorithm (COPRA) [25]

S. Gregory Extended label propagation into finding overlapping communities by labeling each vertex with a set of labels and *belonging coefficients*. This is controlled by a parameter *v* which represents the maximum number of communities that a vertex can belong to. The reason we introduced belonging coefficient is, without that, we probably may copy all the neighbors of each vertex into their label sets which does not make any sense. Moreover, we need to ensure there are set of labels to guarantee that each vertex can be embodied by more than one community, we use those labels as *identifiers*.

A *belonging coefficient* represents the connecting strength of each vertex with respect to their neighbors in same community. The total sum of belonging coefficient of a vertex is 1. Thus during each propagation step, we need calculate the sum of belonging coefficients of a vertex and then normalize them according to the formula shown as below:

$$\mathrm{b_t(c,x)} = \frac{\sum_{\mathrm{y} \in \mathrm{N(x)}} \mathrm{b_{t-1}(c,y)}}{|\mathrm{N(x)}|} \tag{11}$$

Where $b_t(c,x)$ means in iteration step *t*, the belonging coefficient value of vertex *x* contributed by identifier *c*. We can see an example as Figure 4.5:

Figure 4.5 shows the result after first propagation. More precisely, community b contains 1/4 a, 1/2 c as well as d with weights 1/3. Likewise, community e contains a, f and g in the same way. However, we need to keep each vertex only remain some identifiers, not all of them. Belonging coefficient can realize this: when we get the label pairs as above, we screen them and only keep those who is larger than a threshold which is a reciprocal 1/*v*, of which *v* is the parameter of COPRA. When *v* = 1, COPRA is totally the same with RAK algorithm (see section 4.3.1), whereas *v* > 1, it can control how many identifier a vertex can be labeled. Results after deleting label pairs are shown as below.

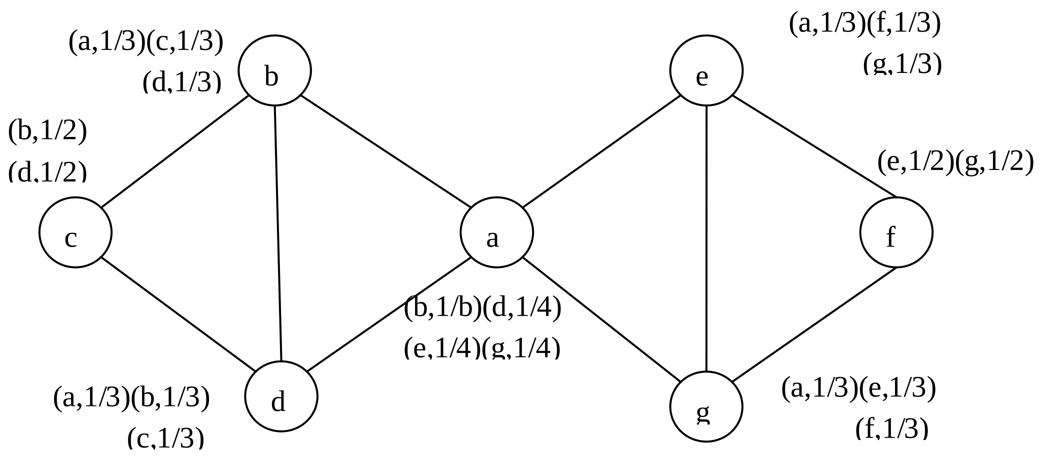


Figure 4.5: Propagation of labels [25].Source: Ref [25]

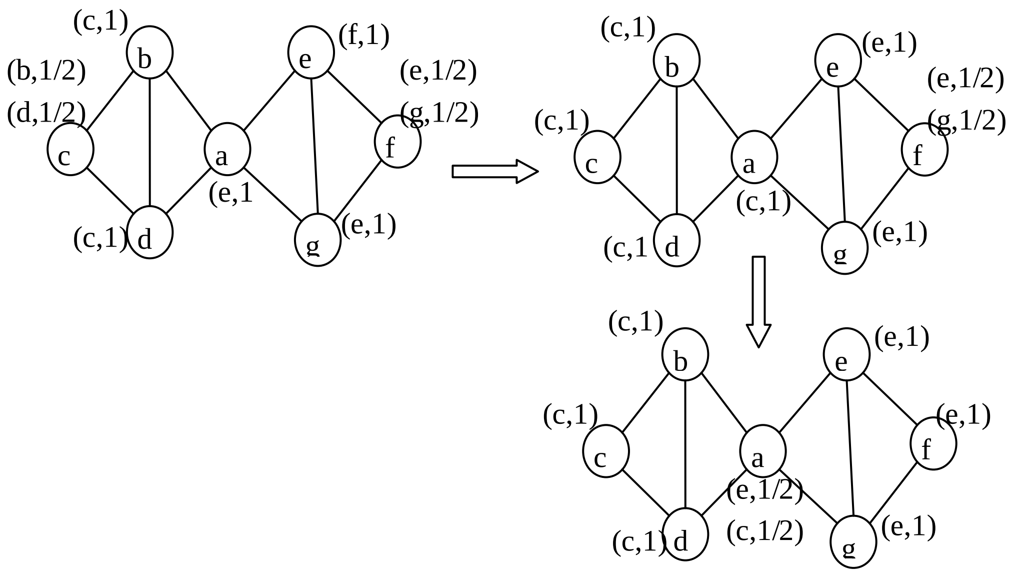


Figure 4.6: Propagation of labels with $v = 2$. [25].Source: Ref [25]

In Figure 4.6, we assume $v = 2$, thus threshold is 1/2. Vertex C has labels b and d, both are not lower than 1/2. Vertex b was supposed to belong to three communities: c, d and a, with weight 1/3, 1/3 and 1/3 respectively, while we randomly choose one, c as shown, therefore, we should normalize it to carry weight 1. Eventually, we get two communities: {a, b, c, d} and {a, e, f, g}, and obviously, they are overlapping communities corresponding to parameter $v = 2$.

Algorithm procedures are briefly described:

1. Initially, we label each vertex x with a community identifier c and a belonging coefficient b, we write these two parameter in a set (c, b).

2. We introduce a parameter v expresses the number of communities that a vertex can included in. By comparing the label sets and 1/ v, we delete those sets whose is smaller than the threshold. More specifically, when all the belonging coefficients are lower than

threshold, we remove all but the one with largest belonging coefficient. COPRA is identical refer to the circumstance under which *v* is less than 2.

3. We relabel each vertex's belonging coefficient such that the sum of that can remain 1.

5 – Algorithm evaluation method

5.1 - Comparing partitions: Normalized Mutual Information (NMI) metric

There are lots of metrics and functions that can value the performance of various community detection algorithms, of which *Normalized Mutual Information metric* is suitable for artificial networks.

This metric compare between an algorithm output division with the true network group assignment and then giving the value with respect to their similarity, of which 1 is the best [2].

Normalized mutual information formulation is as below:

$$I_{norm}(X:Y) = \frac{H(X)+H(Y)-H(X,Y)}{(H(X)+H(Y))/2} \tag{12}$$

Where X and Y are random variable of cover C' and cover C'' in the range from 0 to 1, where 1 means C' and C'' are totally matching with each other, and H(X) and H(Y) express the *entropy* of them respectively [30]. This formulation indicates that how much we can know about X if we are given Y.

However, there is no information given out from the NMI equation, so it only can be used as a quality marker to give the results whether the community structures found by algorithms are close to the real situations. While, this seems not to be problematic in Leto Peel's method [2], as information of the community structures is not required.

Let's see an example of using NMI to test an algorithm in [23] on an enhanced benchmark by Girvan and Newman [12]:

1. Data set—— hierarchical benchmark

First build the hierarchical network, which will be more like the real world network, we divided the network into two levels: (see Figure 5.1)

Micro-communities: There are 512 nodes divided into 16 partitions, each contains 32 nodes. We set k_1 as the average link number each node has with the others.

Macro-communities: we continue divide these 16 partitions into 4 super-groups, each contains 128 nodes and set k_2 as links with 96 nodes belong to three other groups while assigned to same super-group.

mixing parameter: K_3 represents the link number between each node with the rest of network. To test hard, we can change the ratio k_1/k_2, one special condition is when $k_1 = k_2$, a micro-communities presences, which we called "fuzzy" to indicate the situation that small communities mixed well [27].

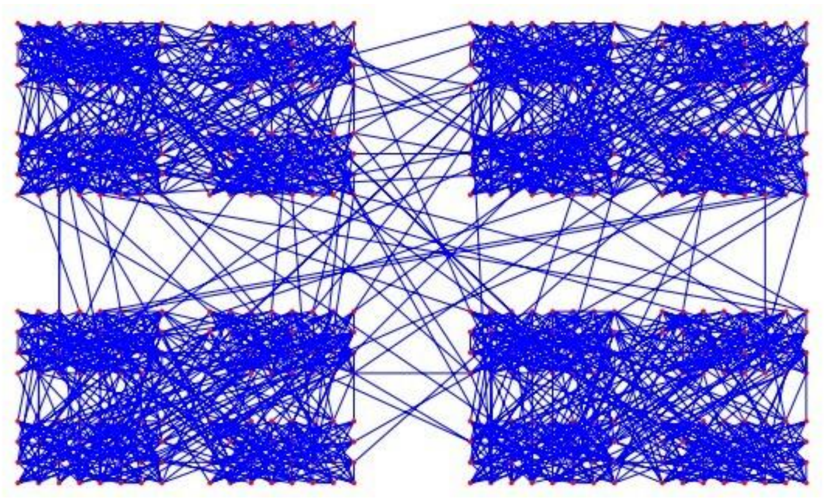


Figure 5.1: hierarchical network example [26]. Source: Ref. [26]

2. Algorithms based on fitness function maximization [20, 21]

See section 4.2.1 fitness function and section 4.2.2 LFM algorithm.

3. Results

Given the plot (see Figure 5.2) of average value of NMI on k_3, we can see from above that macro-communities performs pretty well before $k_3 = 24$, since it's NMI score keeps close to 1. While reaching $k_3 = 32$, when the internal edges number and external edges number are both 32, the NMI becomes unaccepted. In addition, micro-communities did great all the times.

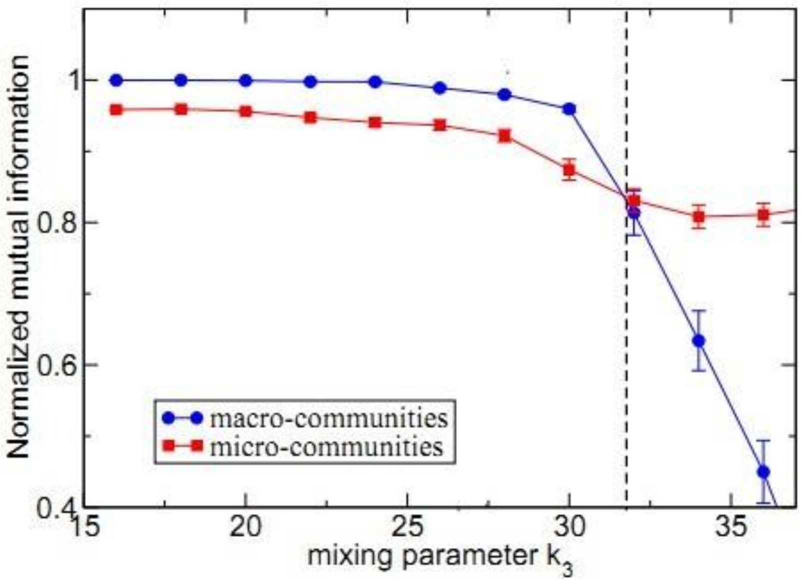


Figure 5.2: NMI score on a hierarchical benchmark. Source: Ref. [23]

Thus, we give the conclusion that the division given by algorithm is as same quality as the natural community until $k_3 < 32$ [23].

5.2 - Comparing Algorithms: Comparative Analysis [1]

Lancichinetti and Fortunato have proposed a method in [1] that how to comparative analysis algorithms' performances on different kinds of LFR benchmarks, like weighted benchmarks and overlapping benchmarks. Let us see an example of comparative analysis (see Figure 5.3):

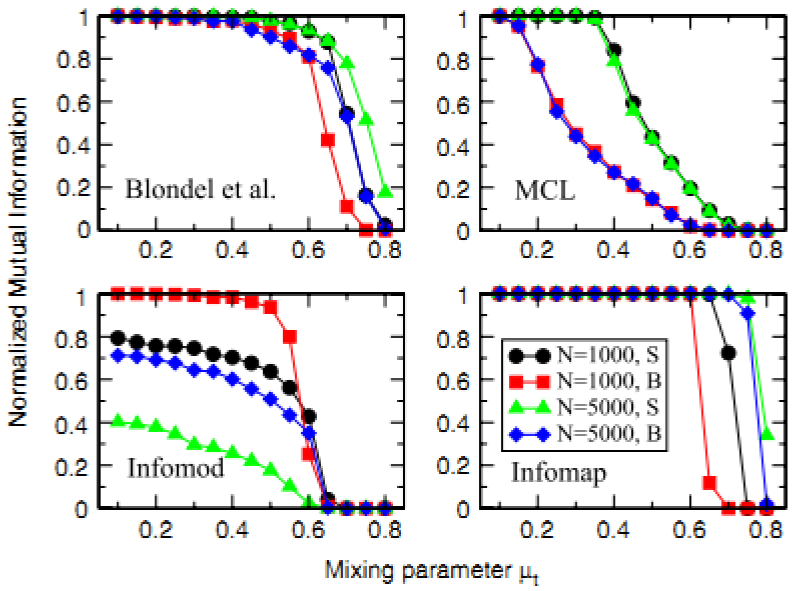


Figure 5.3: Algorithms' performances on LFR benchmark. We varying LFR mixing parameter μ_t on x axises with NMI value on the y axis measuring the performances. Different lines representing different sets of LFR benchmark parameters: node number N(1000 or 5000) and community sizes S[10,50] or B[20,100]. Source: Ref [1]

Through this result, we can see that Infomap algorithm performs the best and the most stable as well for even different setting of LFR benchmarks, almost all the NMI values are equals to 1 until μ_t is lower than 0.6. However, this happens not only to Infomap algorithm, almost will not make any sense when μ_t is higher than 0.6, since it is the boundary whether the LFR benchmarks can generate good partitions. From μ_t equals 0 to upper boundary, noise will be more obviously, thus the benchmark networks will no longer clearly detected, they will become even indistinguishable.

There are two critical elements of this method:

- Benchmark networks with different setting of parameters which can results different community structures.
- Community detection algorithms focusing on recovering the benchmark networks' community assignments.

Further, these two elements must corresponding to each other, since the community detection algorithms were designed based on different understanding of community structures. For example, most algorithms assume the networks are disjoint networks, such that we cannot use overlapping LFR benchmark [1].

6 - Classifier: Support Vector Machine (SVM)

Large scales of community detection algorithms have been raising a problem that we look forward to using intelligence analyst to predict the proper class of algorithms that should be chose. Since in most cases, we may not fully equip the knowledge of the algorithms and the underlying community structure, we desire there will be an automatically selecting system to predict the algorithm. This can be done by using a linear Support Vector Machines with Kernel-based methods.

6.1- Intelligence Analyst

There are two ways to solve problems: one is by considering all the required domain knowledge then after long time and big cost as well to construct a rule model which can provide resolutions. However, this method is not realistic; the second one is by learning from experiences. Nevertheless, since the quantity of data is growing, and data becomes vary and complex recently, we eagerly pursue automatic problem solver, also known as intelligence analyst. Finally, after three procedures: pre-processing, modeling and prediction, and eventually explaining, we can realize data mining [28].

6.2- Machine Learning

The reason why machine learning is employed in this project is the general concept of that, which is to find out one best fit data for the given previous knowledge from a large scale potential hypotheses space [29]. If we label data by category, it turns out to be *supervised learning* problems, and even *classification*. Applying to this project, a class of algorithms can be selected automatically.

6.2.1 - Supervised Learning

Supervised learning means that given some samples, one can predict specific properties. This can be fulfilled by taking those samples as learner, then study the relationship between sets of samples with characterized properties. What learner outputs is called hypothesis, this will be modified as samples change.

Supervised learning requires domain descriptions, which are the knowledge used to recognize factors of interest. Assume *joint probability density*

$$P(x,\ y) = P(x)P(y|\ x) \tag{13}$$

can reveal the relationship of *x* and *y*, where *x* refers to predictive variables and *y* is target variable, P(y| x) is *conditional density* and P(x) is the previous probability. The way they interact is what we care about. However, in reality, noise appears and can affect the function above. Supervised learning can solve this problem using the technique *risk minimization*:

$$R(f_s) = \int c(f_s(x), y)\, dP(x, y) \tag{14}$$

Where c represents the cost computing f_s. We label $y_i \in Y = \{1, \ldots, k\}$ and $x_i \in X$ such that a prediction function f_s (hypothesis) can learn from a training set $S = \{(x_i, y_i)\}$ and then do a classification by mapping :

$$f_s : X \rightarrow Y$$

$$f_s : x \longmapsto y \tag{15}$$

In this way, risk R can be minimized [31].

6.2.2 - Choosing Hypothesis

In fact, P(x, y) is unavailable from training data S, therefore, we need to compute *empirical risk* instead:

$$R_l(f_s) = \frac{1}{l}\sum_{i=1}^{l} \int c(f_s(x), y) \tag{16}$$

However, there exists a fact that even we minimize empirical risk, the output hypothesis still not meet our demand, as the hypothesis may suitable for both true data space and those data affected by noise. This *overfitting* phenomenon should be limited by *capacity control*: From a low capacity hypothesis space, i.e. from hypothesis space which contains small number of labels, we choose a hypothesis to realize minimal empirical risk can make a bigger chance that the true risk is still low [31].

6.2.3 - Statistical Learning Theory

We assume two- class classification and compute the 0/1-loss cost function as below:

$$c(f_s(x), y) = \begin{cases} 1 & \text{if } f_s(x) \neq y \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

This is error rate which can be minimized by lower a *upper bound*. If we calculate empirical risk in hypothesis spaces and choose the one can finally lower a upper bound, *structural risk minimization* is achieved [31].

6.3 - Principles of SVM

Support vector machine is designed on account of statistical learning theory, more specifically, SVM is based on structural risk minimization, using capacity control to avoid overfitting phenomenon and thus realize a goal that only solve certain problems rather than all of them.

There are two key elements of SVM:

- Mathematical programming: can find parameters with linear equality and inequality constraints [28].

- Kernel functions: these functions give SVM the ability to predict in a wide range of regularized non-linear hypothesis spaces.

Trivially, from geometrical interpretation aspect, support vector classification (SVC) is designed for finding out an optimal hyperplane which can separate surface so that the distance vector between the two classes is the same [28]. This is of linearly level. Further, Kernel function can extend SVC to non-linear cases [31].

7 – Peel’s method

Leto Peel has proposed an innovative method in [2] to choose algorithms with respect to their classes. It is well known that there are lots of algorithms designed to detect communities, while each algorithm focus on one specific problem space which is just against the theory of machine learning that is “ no one size fits all”. We assume that under different circumstances, one class of algorithms may outperform other classes even it is already pronounced this condition is not the problem space it excels in.

Recently researchers have already found that choosing a proper algorithm is a hard problem as there is no acknowledged absolute definition of community and evaluation methods. Therefore, the state of art is apply algorithms on benchmark networks and then compare their performance using the similar way of comparative analysis in [1]. Though this methodology has performed better, there should be more solutions for this condition: what if we do not know about the algorithms’ procedures and not required the knowledge of community assignment? Leto Peel put forward the opinion that is by estimating network structural properties which have shown major effect on algorithms’ performances [1] to predict which algorithm class is better. In the end, he successfully proposed a way through which we can automatically choose a class of algorithms after intelligence analyst of the known community structure.

First, Leto Peel classifies algorithms into two classes: weighted network algorithms and unweighted network algorithms, as linked weights can reveal important information (see section 2.1.). To generate a network with link weights, he adopts LFR benchmark with weighed networks’ attributes (see section 2.1). The reason to choose this artificial network is that it contains properties which can assign community and at the same time, normal real networks do not provide us bigger choice amount and also may be different among different authors. In this benchmark, μ_w and μ_t are two mixing parameters we most interested in, however, we cannot easily measure these two parameter values, since it would require basic community assignment [2]. Thus, we can use the observed features to estimate them.

To compare algorithms performance, we need to first estimate parameters of that benchmark, i.e. μ_w and μ_t topology mixing parameter. Among some metrics such as degree distribution, average diameter and some others which are related to parameters, Peel found the node measure weighted Clustering Coefficient and unweighted Clustering Coefficient can predict μ_w and μ_t respectively. The reason can be explained as nodes connected with each other are more likely to be assigned into one group [2].

After decided which data set to use, he chose two algorithms on purpose since algorithms in this method are constrained by two classes. I.e. algorithms should be available for both weighted and unweighted networks. Only in this way, can we clearly see the different performances of two classes such that we can decide which class is better. In Peel’s paper, he used Infomap algorithm and COPRA algorithm.

Once get the value of parameters and algorithms, we can compare algorithms' performances by computing their normalized mutual information scores by changing parameters. More specific, he first compare the algorithms' performance when μ_w is changed while μ_t is set to certain values. Through this way, he got the conclusion unweighted algorithms are not affected by μ_w and perform well when μ_t is low. On the other hand, weighted algorithms are affected by both parameters and performs well when μ_w is low. Thus he set μ_w is as low as 0.3, and get the important result (see Figure 7.1): not until μ_t is same as μ_w, unweighted algorithms did better than weighted algorithms. This proofs the fact that fewer inter-community links corresponding to lower inter-community weights. In conclusion, we can make a prediction of which class to choose.

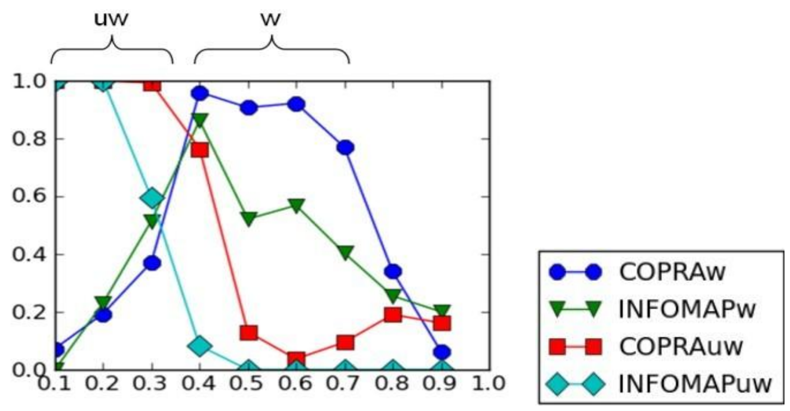


Figure 7.1: NMI scores for weighed and unweighed algorithms as μ_t varied. $\mu_w = 0.3$ [2]. Source: Ref [2]

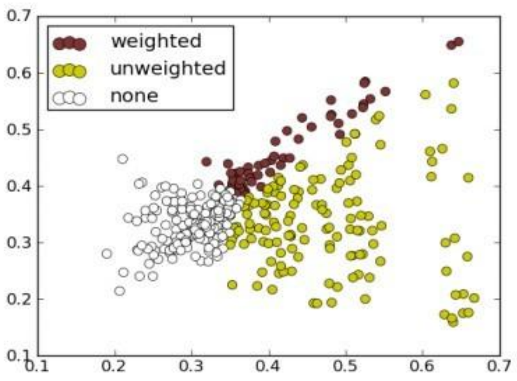
Finally, to confirm the correctness of the method Peel proposed, he used linear support vector machines (SVM) to classify and obtained a classifier confusion matrix. See Table 1.

Table 1: classifier confusion matrix [2]. Source: Ref [2]

		Predicted Class		
		Weighte d	Unweighted	None
True Class	Weighted	42	3	36
	Unweighted	4	125	18
	None	6	5	209

From this matrix, we can see most networks are assigned a proper class. The higher value we got in the diagonal, the better the model is.

To use SVM, first according to the best performance, each networks were assigned a class among the class set {unweighted, weighted, none}. None means the NMI score is below a threshold. From the results (see Figure 7.2), we are able to predict class of algorithms.



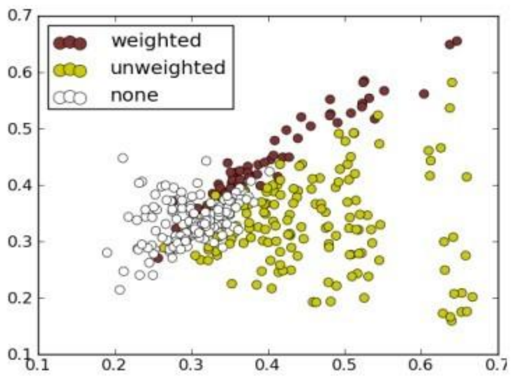


Figure 7.2: left: predicted classification right: true classification [2]. Source: Ref [2]

8 - Experiments

This section discusses the problem space of my project. My project is to extend Leto Peel's method (see chapter 7) into other classifications of network algorithms. Since the most popular classification of network algorithms is {disjoint, overlapping}, I will set this as the major new classification I explore. Furthermore, if some parameters not work on this classification, I will try more narrow classification which is only focusing on overlapping networks: {fuzzy overlapping, crisp overlapping} (section 8.2.5).

First I will discuss the methodology (section 8.1) of testing algorithms I adopt in my project. Then I will find out the possible properties of network (section 8.2) which can be estimated by features of known networks (section 8.2.1) in new classification of network algorithms. By comparative analyzing algorithms' performances (section 8.3), we can justify that a prediction of which class of algorithms is better can be made.

8.1 - Methodology

Methodology of testing algorithms includes three main parts (see chapter 3 and chapter 4, 5): first we need to specify dataset on which algorithm of different classes will detect communities, and then by the measurement we choose, we can get the performance of the algorithm.

8.1.1 - Dataset

8.1.1.1 - Overlapping LFR benchmark

In this project, since the main objective is to estimate parameters of artificial networks, and the first new classification of algorithms is disjoint/ overlapping, we can adopt Overlapping LFR benchmark (see section 3.4). The reason that LFR benchmark outperforms other benchmarks is that it is a network generator which can produce overlapping communities, with features or properties appearing in real networks, based on an observed community structure. What's more, parameters of the community structure can be varied under our control to help us analysis algorithm performances. Further, parameters are the most important elements in the project, since once they are estimated by features of community structure, there stands a chance that class of algorithms can be estimated either. However, even so, overlapping LFR benchmark vertices are all embodied by same amount of overlapping communities which is the only limitation [25].

8.1.1.2 - Parameters of Overlapping LFR benchmark [16]

Overlapping LFR benchmark can generate networks according to a known community structure, which is contributed by parameters cater to users. There are two main types of parameters:

- Community parameters

N	vertices number
c_{min}	minimum community size
c_{max}	maximum community size
τ_2	power-law distribution exponent controlling community size
o_m	overlapping communities number a vertex can embodied
o_n	overlapping vertices number

- Network parameters

<k>	average degree
k_{max}	maximum degree
μ_t	mixing parameter: a vertex share edges with a fraction of μ with other communities vertices
τ_1	power-law distribution exponent controlling vertex degree

Like discussed in [1] by Lancichinetti *et al*, during the experiment part of the project, we keep some of the parameters fixed: $<k>$ is 25, thus k_{max} should be 2.5*$<k>$, which is 50. Similar, c_{max} would better around 5*c_{min}, so we set 50 and 15, respectively. Finally, power-law exponents τ_1 and τ_2 could have the value of -1 and -2. Normally, N should higher than 1000, otherwise the community structure will not make too much sense, but considering the computing time by running algorithms, we just take N as 1000. However, mixing parameter μ_t, overlapping parameters o_m and o_n are not fixed and need testing instead.

8.1.1.3 - Fuzzy generator based on Overlapping LFR benchmark [10]

For the second classification of classes of algorithms, fuzzy/ crisp overlapping networks, overlapping LFR benchmark seems insufficient, since it can only generating normal crisp networks. Therefore, Steve Gregory proposed a method in [10] that by using a fuzzy network generator based on LFR benchmark, we can get fuzzy overlapping networks and a set of communities. The fuzzy generator's procedure flow chart is showed as Figure 8.1:

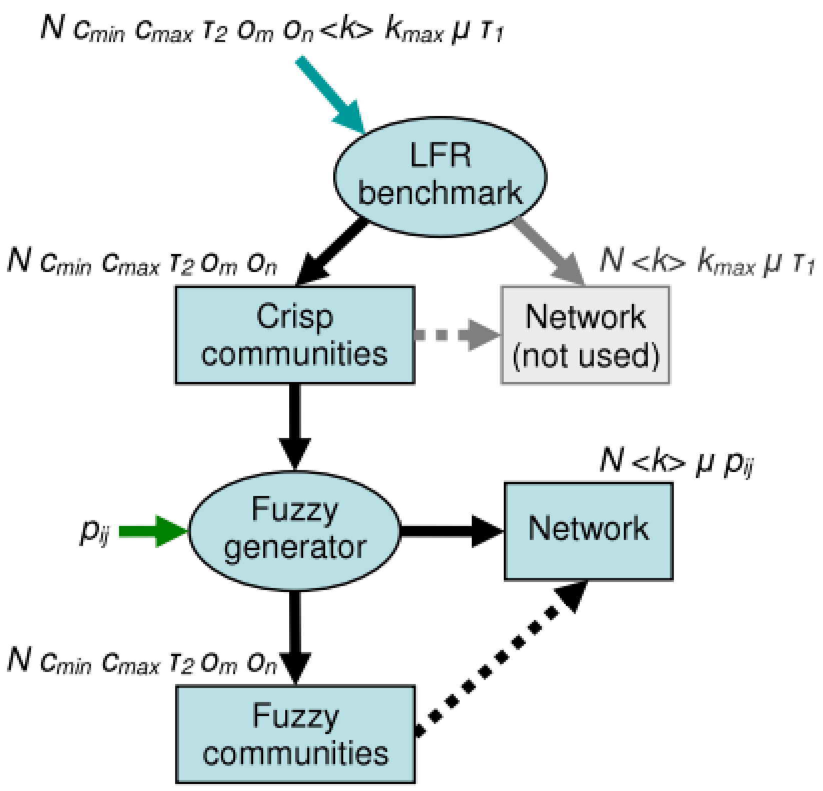


Figure 8.1: fuzzy network generator. Source: Ref [9]

Crisp communities can be changed into fuzzy communities by a belonging coefficient ∂, which is randomly selected and added to each vertex (see section 2.3.2.2). More specifically, vertex v belongs to two communities: s and g, thus by a uniform distribution, we can compute ∂_{vs} in range [0, 1] and $\partial_{vg} = 1 - \partial_{vs}$ as well [10].

8.1.1.4 - Drawbacks of real-world network

Still, we have to discuss the reason why we don't evaluate algorithms' performances on real-world networks. First of all, the communities of the initial real networks are hard to know thus we can not one hundred percent measure algorithms' outputs. Second, for those communities we already found in real-world networks perform sometimes not clearly in the network structure [25].

8.1.2 - Measure

For artificial and overlapping networks, among all the standard measures, NMI (section 5.1) performs the best and has become a standard. Therefore, I adopt this metric to compare performances of algorithms.

We will use the term "NMI of algorithm X on network Y" refer to the meaning that the NMI value compared between the true community assignment of network Y and the results of algorithm X detected.

8.2- Focusing on estimating properties of networks

Among all the metrics that can measure community topology, and thus are called properties of networks, degree distribution and clustering coefficient can be considered to estimate parameters of LFR networks. For the reason that they are both node measures and therefore, the mean value of local clustering coefficient and mean value of degree distribution can represent the whole network. However, in the project, we made a little modify on degree distribution: we calculate its standard deviation value. We mention Standard Deviation (SD) instead of standard deviation of degree distribution in this paper. We will plot figures showing the relationship between CC, SD and parameters of networks: μ_t, o_m or o_n. For the reason that both o_m and o_n are overlapping parameters, we need to find out which one performs the most obvious influence by CC and SD after critical testing on the whole range of them.

8.2.1 - Reasons of choices of CC and SD as the observed features

Since in Leto Peel's method, we have already successfully estimated μ_t by CC, thus we still keep this part. However, for the overlapping part, we need to do some research:

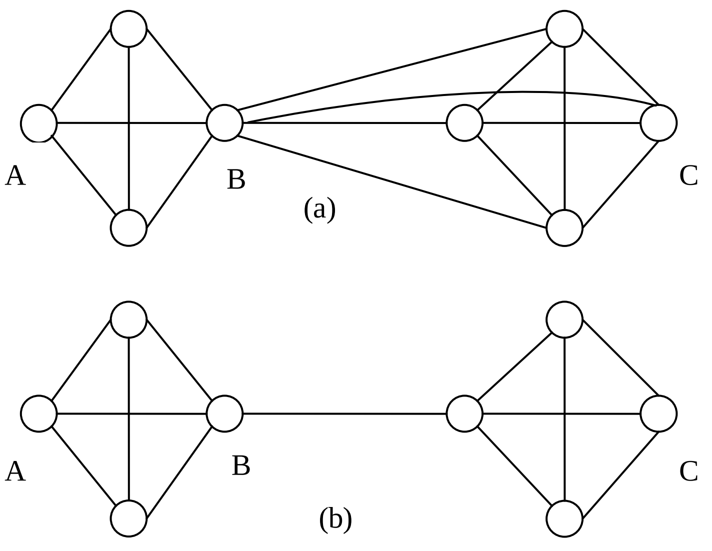


Figure 8.2: SD values are different in overlapping network (a) and disjoint network (b).

For Figure 8.2.(a), degree of node B is 7 and degree of node A and node C are both 3. Thus, SD of (a) will be higher. For Figure 8.2.(b), degree of node B is 4 and degree of node A and C are both 3, therefore degree deviation of disjoint networks will normally lower than that of overlapping networks.

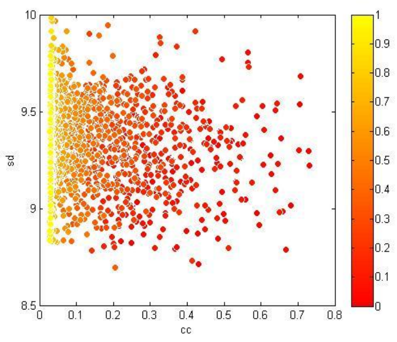
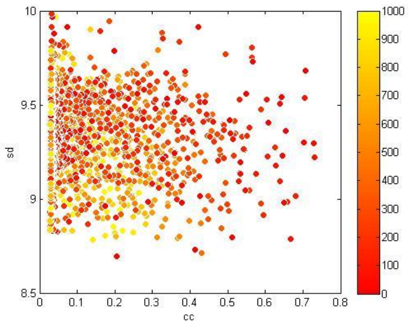
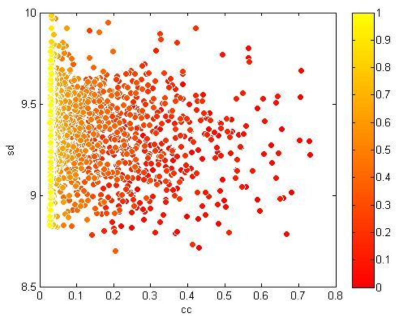
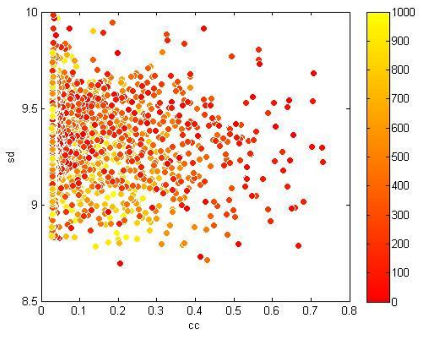
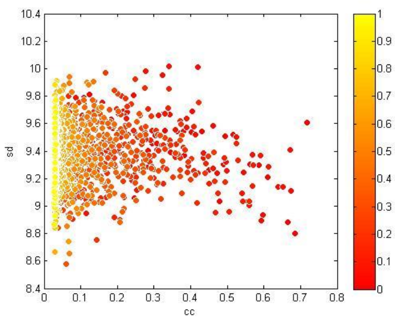
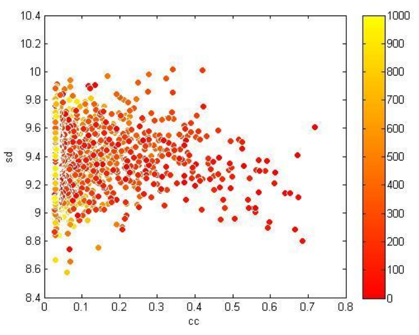
8.2.2 - Results of parameter o_n of disjoint/ overlapping networks

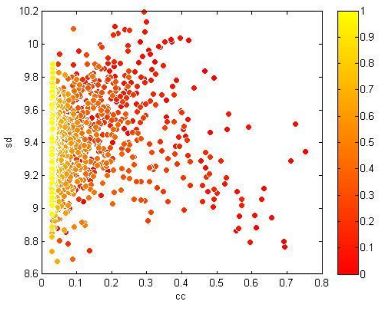
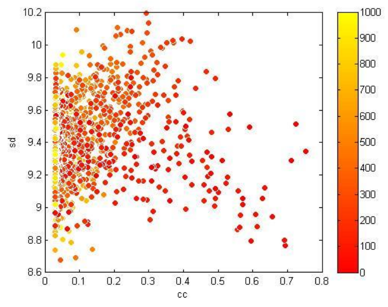
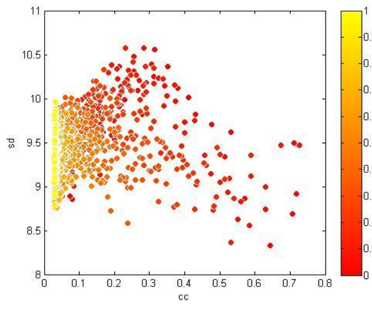
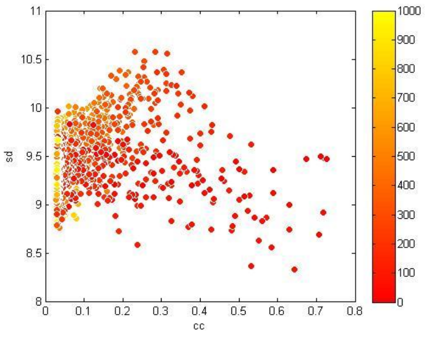
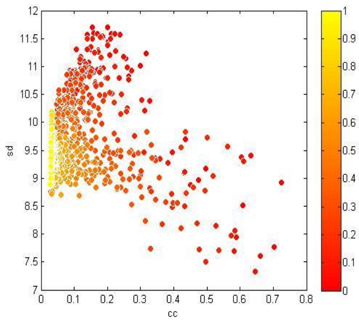
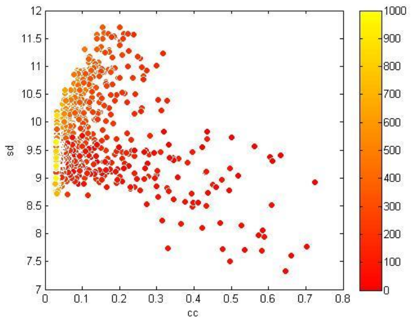
Since o_n represents the number of overlapping vertices, we should test the whole range of o_n, which should be [0,1000]. Of which, $o_n = 0$ in fact is disjoint networks, since no vertex belongs to more than one community. Instead, when $o_n > 0$, there will be overlapping networks.

At the same time, we should keep o_m on a fixed value and μ_t on the overall range of itself [0, 1]. By adding μ_t by 0.025 and o_n by 100 each iteration, we can finally generate 1600 networks. After calculating CC and SD value of each network, we can plot a figure to show their the influence on μ_t and o_n, respectively.

Results are shown as below:

Table 2: Influence of CC and SD on μ_t and o_n

Value of o_m	Influence of CC and SD on μ_t	Influence of CC and SD on o_n
$o_m = 2$		
$o_m = 3$		
$o_m = 4$		

$o_m = 5$		
$o_m = 6$		
$o_m = 10$		

In each plot, there are 1600 dots; each represents a mean value of μ_t or o_n of a network. As the color bar on the right hand side, from red color upwards to yellow color, μ_t increases from 0.0 to 1.0 and o_n increases from 0 to 1000, simultaneously. CC and SD are x-axis and y-axis, respectively. Those figures are plot by software MATLAB.

We can conclude that when CC increases, values of μ_t become larger, since colors change from yellow to red if we observe those plots horizontally. This is not influenced by value of o_m so much. However, when o_m is lower than 5, there almost no impact of SD on μ_t, for the fact that they all stay in the range of [8.8, 10] of y-axis. In contrast, when o_m is higher than 5, figures become curvilinear, we need to analysis them accompany with CC's impact: When CC is lower and SD is lower, μ_t values normally higher. Inversely, when CC increases and SD increases at the same time, μ_t values appear lower.

From Table 2 above, it is obvious that CC cannot affect values of o_n, since each color scatters from the 0 to higher horizontally. However, SD has little impact only when o_m is higher than 6, that is, when SD is higher, o_m is lower. These results are not as what we expected before, without clearly and orderly changes of colors, it will be difficult for us to estimate parameters of networks.

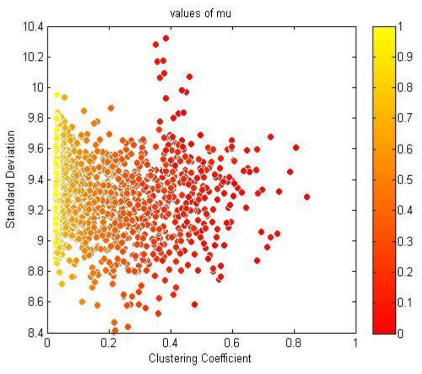
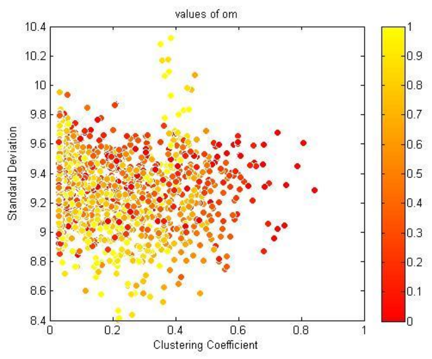
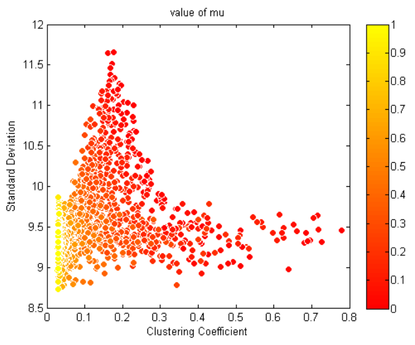
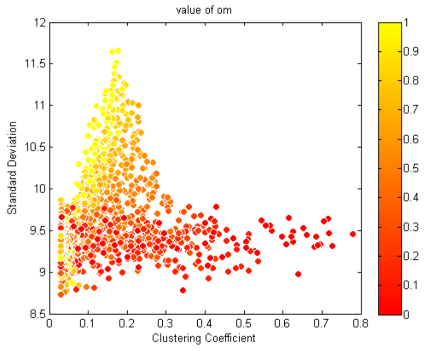
8.2.3 - Results of parameter o_m of disjoint/ overlapping networks

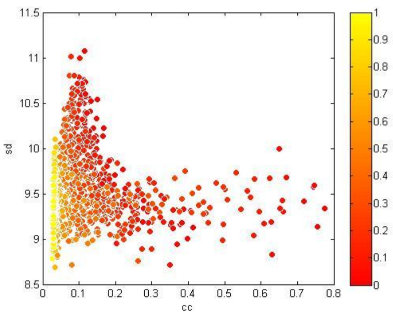
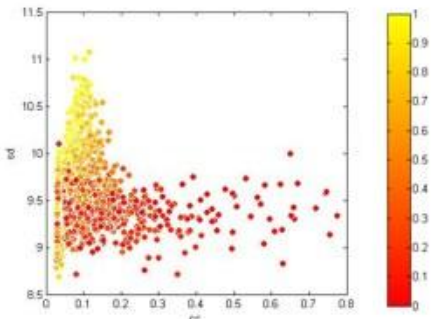
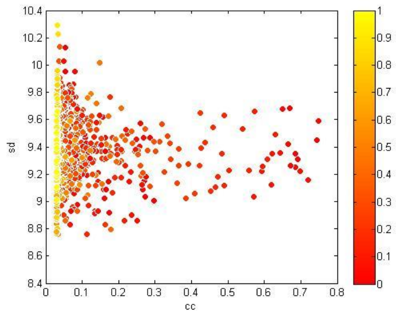
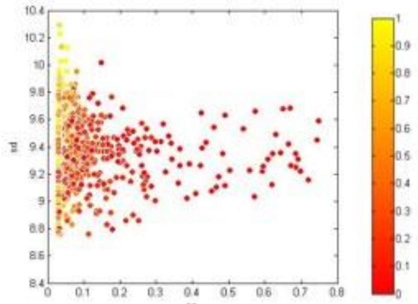
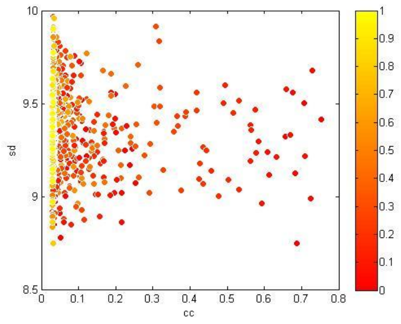
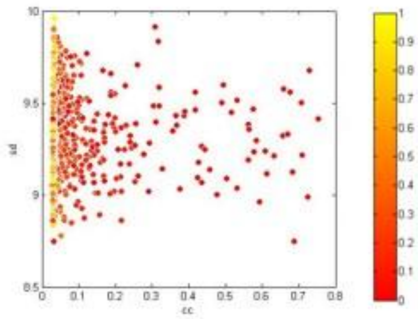
Overlapping parameter o_m refers to the number of communities that a vertex can belong to, thus its range should be [1, 10]. More specifically, when $o_m = 1$, the networks generated are disjoint networks. Instead, when $o_m > 1$, overlapping networks they are. However, recent study shows that almost all the overlapping network algorithms are useless when $o_m > 10$, therefore, we set the upper boundary of o_m to be 10.

To make control comparisons, we can vary o_n by adding 100 each time to plot different figures of most possible kinds of overlapping networks. Like the test on parameter $\mathrm{o_n}$, we still need to construct 40 values of μ_t in the range of [0.0, 1.0], for the purpose that we need to plot 1600 networks in one figure. Similar, CC and SD are x-axis and y-axis, constituting plots which can represent impact of CC and SD on overlapping parameter o_m.

Results are shown as below:

Table 3: Influence of CC and SD on μ_t and o_m

Value of $\mathrm{o_n}$	Influence of CC and SD on μ_t	Influence of CC and SD on o_m
$\mathrm{o_n} = 100$		
$\mathrm{o_n} = 300$		

$o_n =$ 500		
o_n = 800		
o_n = 1000		

In each plot, there are 1600 dots; each represents a mean value of u_t or o_m of a network. As the color bar on the right hand side, from red color upwards to yellow color, u_t increases from 0.0 to 1.0 and o_m increases from 0.0 to 1.0, simultaneously. Notice that o_m values are no longer in the range [1, 10], as we normalized those to fit software drawing system. CC and SD are x-axis and y-axis, respectively. Those figures are plot by software MATLAB.

By carefully analysis those results we can get that when o_n is higher than 300 and lower than 500, figures appear to be best from the visual aspect. Otherwise, figures are not making any sense. Generally speaking, in the range [300,500] of o_n, the effect of CC and SD on μ_t can be summarized as that when CC and SD are higher, μ_t values will be lower.

Like results of u_t, when o_n is lower than 300 or higher than 500, plots are hardly conveying any useful messages. However, when o_n is 300 and 500, especially 300, plots seem really meet our prospect. To be more precisely, when SD is higher while CC is lower, values of o_m become higher. On the contrary, when SD is lower, o_m

becomes lower.

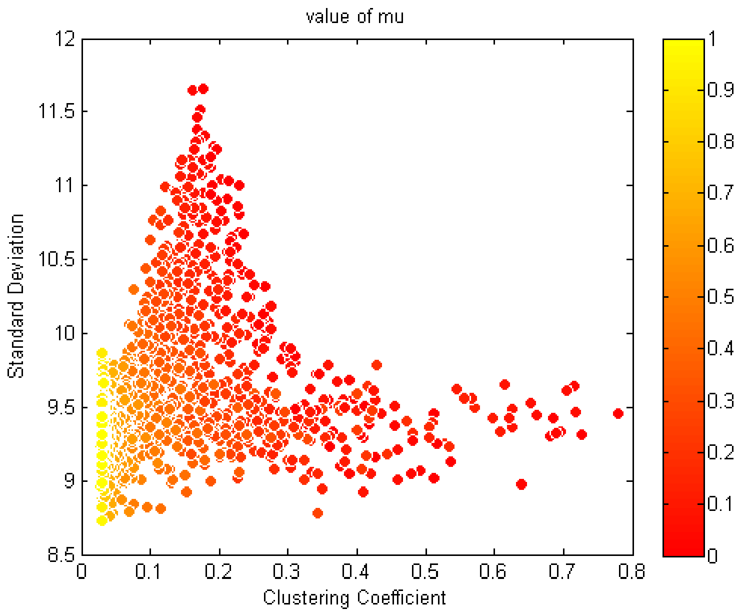
8.2.4 - Best parameter setting of disjoint/ overlapping networks

After comparison, we can take the best pair of figures' parameter setting as ours in the experiment part. Which is N = 1000, <k> = 25, k_{max} = 50, τ_1 = -2, τ_2 = -1, c_{min} = 15, c_{max} = 50, $o_n = 300$, o_m varies from 1 to 10 (adds 1 each time), μ_t varies from 0.0 to 1.0 (adds 0.025 each time).

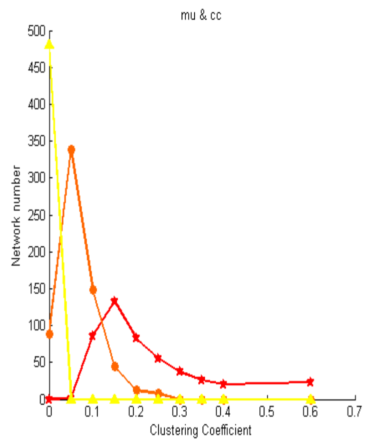
Between overlapping parameters o_m and o_n, we choose o_m to be the parameter that can be estimated by CC and SD, as it shows more clearly and colors of results of o_m are gradually change with an order. However, this is concluded from experiments, we need to analyze it from theoretical respect: Since o_n is the number of overlapping nodes, while o_m is controlling how many communities the overlapping nodes can belong to. Thus, even if o_n is as high as the entire nodes number, if o_m is set to be very small, the whole network structure will tend to be like disjoint networks. Thus, highly overlapping community structure is mainly constructed by o_m.

Based on above, $o_n = 300$ and $o_n = 500$ are two preferable choices that we can take. Between these two, $o_n = 300$ parameter setting contributes a better scatter, since 1600 dots are more sparsely assigned, which is easier to observe.

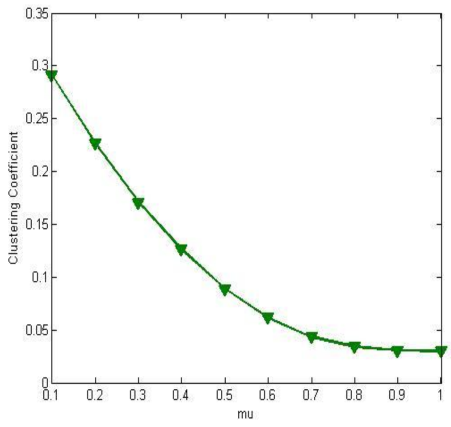
Let us plot some curve lines to intuitively summarize the influence of CC and SD on mixing parameter μ_t and on overlapping parameter o_m. Parameter setting is same as in section 8.2.3. First, let us see how CC and SD affect mixing parameter μ_t. We first have a glance at the original figure representing their effects. Then we plot relationship line to express the way they interact.



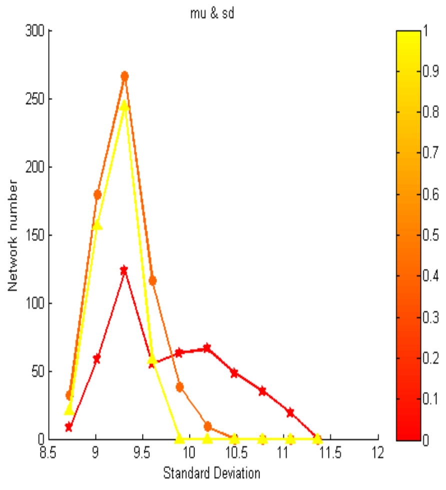
(a)



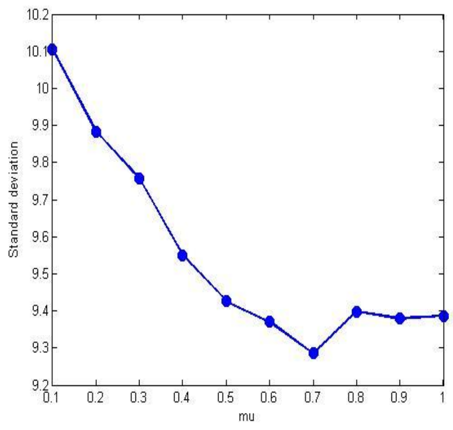
(b)



(c)



(d)

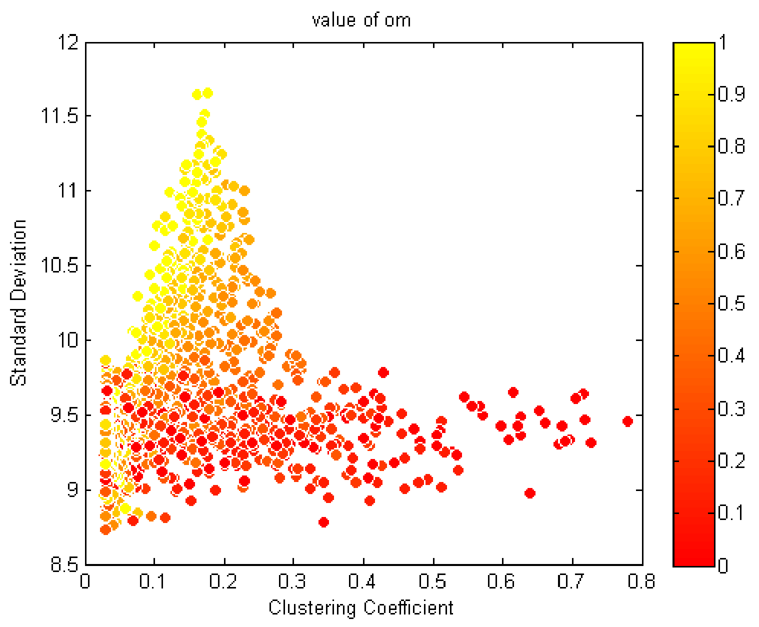


(e)

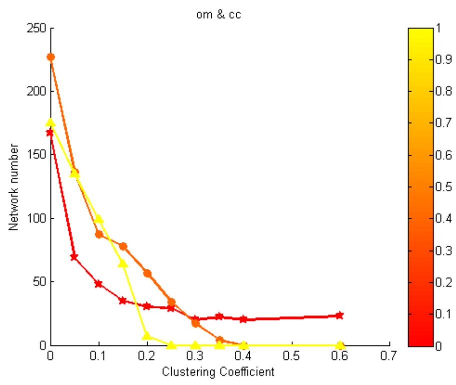
Figure 8.3: Analysis of relationship between CC and SD with mixing parameter μ_t, (a) is the initial figure of them, (b) shows when CC changes (x-axis), the number of networks (y-axis) change, and

curve lines with colors referring to the value of μ_t. (c) Relationship between CC and μ_t. (d) Shows when SD increase (x-axis), network numbers (y-axis) with different values of μ_t (different colors) change. (e) SD's impact on μ_t.

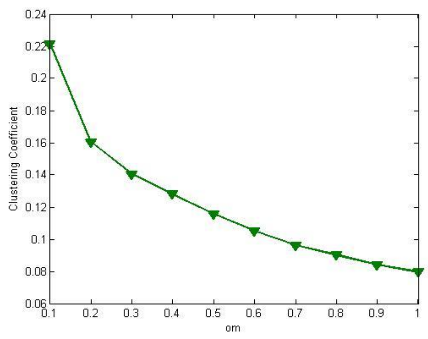
Second, we explore affect of CC and SD on o_m in the similar way:



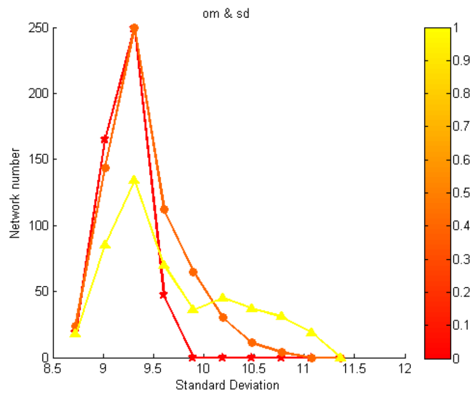
(a)



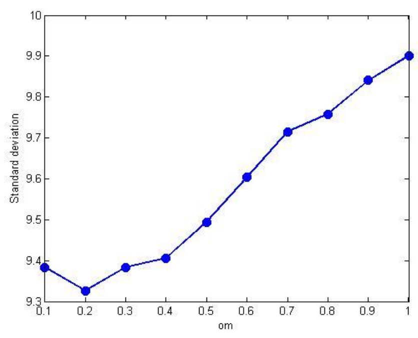
(b)



(c)



(d)



(e)

Figure 8.4: Analysis of relationship between CC and SD with overlapping parameter o_m. (a) is the initial figure of them, (b) shows when CC changes (x-axis), the number of networks (y-axis) change,

and curve lines with colors referring to the value of o_m. (c) Relationship between CC and o_m. (d) Shows when SD increase (x-axis), network numbers (y-axis) with different values of o_m (different colors) change. (e) SD's impact on o_m.

We can conclude the way that how we can estimate parameter only by observed CC and SD of disjoint or overlapping networks.

- For mixing parameter μ_t

  When CC value is higher, we will get lower mixing parameter μ_t value

  When CC value is lower, mixing parameter μ_t value becomes higher.

- For overlapping parameter o_m

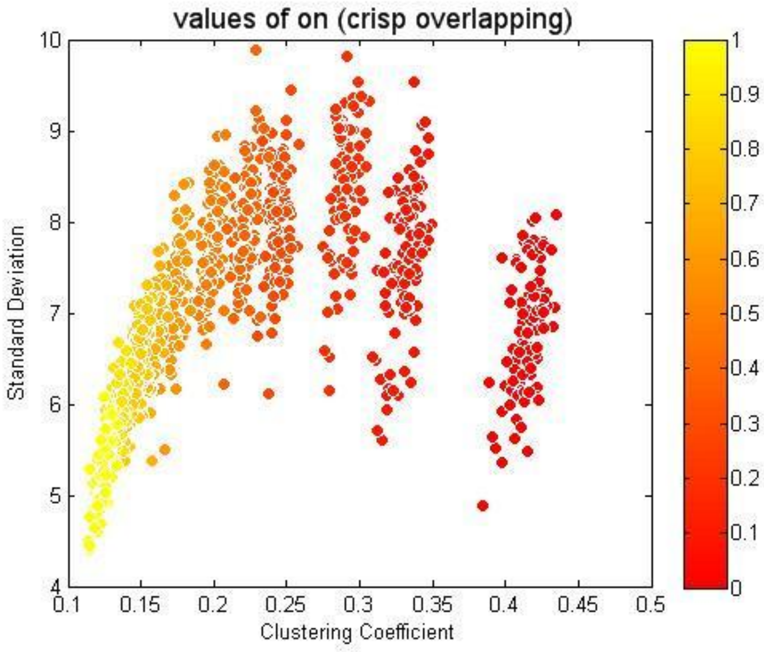
  When SD value is higher, we can get higher values of o_m

  When SD value is lower, overlapping parameter o_m will be lower.

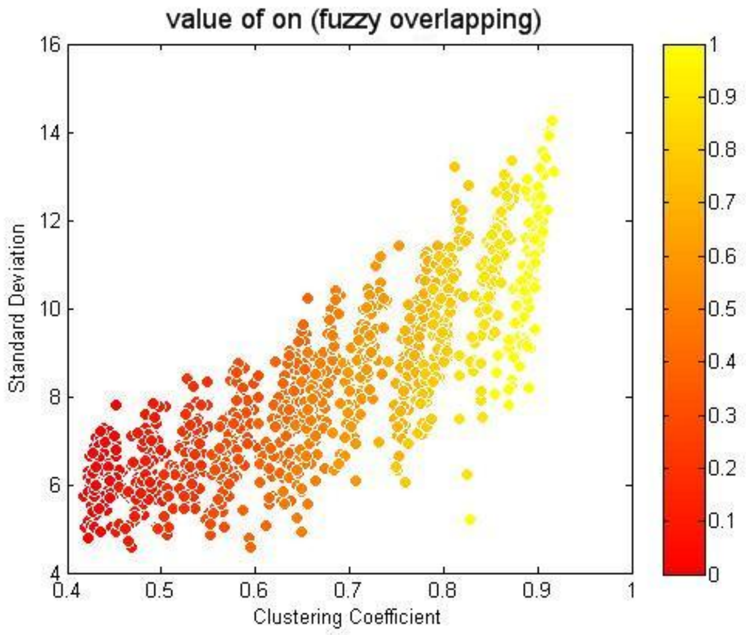
Those can be explained in a more theoretically point of view, which is, when CC value is lower, community structure is not so obviously. This kind of networks can be generated by us if we set μ_t to be higher, thus amount of edges between communities will no longer less than edges inside communities, which means community structure is hard to detect. Meanwhile, overlapping parameter o_m can also lead to this kind of networks as our need, when we set o_m to be higher. On the other hand, low value of SD represents good community structure.

8.2.5 - Results of parameter $\mathbf{o_n}$ of fuzzy/ crisp overlapping networks

Overlapping parameter o_n fails to be estimated in the disjoint/ overlapping classification, we can give it a try in the second classification: fuzzy overlapping networks and crisp overlapping networks. We adopt the same way to plot figures to test whether CC and SD can estimate o_n.



(a)



(b)

Figure 8.5: Influence of CC and SD on o_n of crisp overlapping networks (a) and fuzzy overlapping networks (b).

Unfortunately, o_n cannot be estimated by SD in crisp overlapping networks. As shown from Figure 8.5.(a), when SD is around range [5, 8], o_n can be small or large, which is really indeterminate. In conclusion, overlapping parameter o_n is not suitable for this estimating parameter method not only in disjoint/overlapping networks but also in fuzzy/crisp overlapping networks.

8.3- Algorithm performance

8.3.1 - Choices of algorithms

Leto Peel's method in [2] has a restriction that algorithms should available for either class of the classification, for the purpose of taking a controlled comparison between the two classes of algorithms' performances. Thus, COPRA is considered.

However, I also tested algorithm performance in another direction: to produce controlled comparison, I tested on an algorithm which is good at disjoint networks but not overlapping networks ------ Infomap, and also on an algorithm which is good at overlapping networks but not disjoint networks ------ GCE algorithm. Results are not optimistic, for almost all the performances are same between these two algorithms.

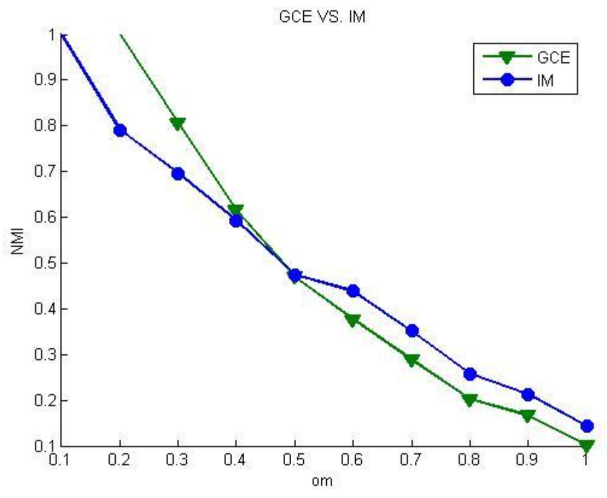


Figure 8.6: NMI scores represent performances of GCE and Infomap (IM) as o_m varies. μ_t = 0.1.

8.3.2 - Parameter of algorithm COPRA [25]

As discussed in section 4.3.2, the only parameter of COPRA is v, which refers to maximum number of communities that a vertex can belong to. Steve Gregory has found that when v is 6, COPRA can produce a near-perfect solution for small networks [25].

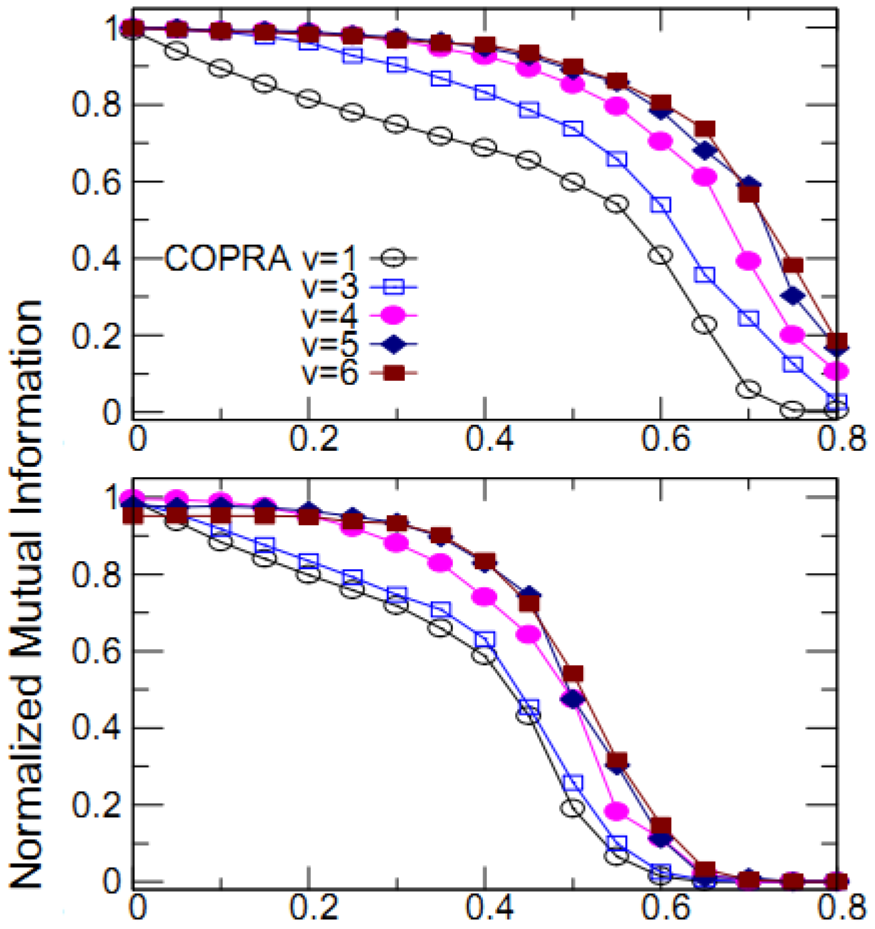
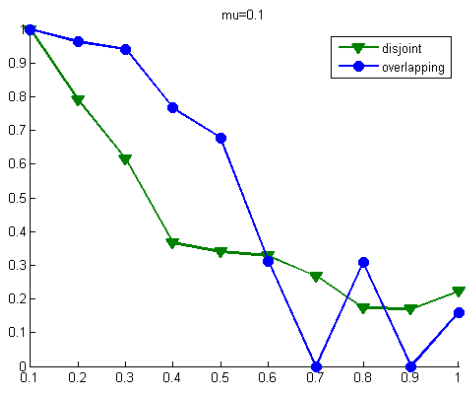


Figure 8.7: NMI scores (y-axis) of COPRA on small networks as fraction of overlapping vertices changes (x-axis).Parameter setting: N = 1000, $<k>$ = 20, $cmin$ = 20. Top: μ_t = 0.1, Bottom: μ_t = 0.3. Source: Ref [25].

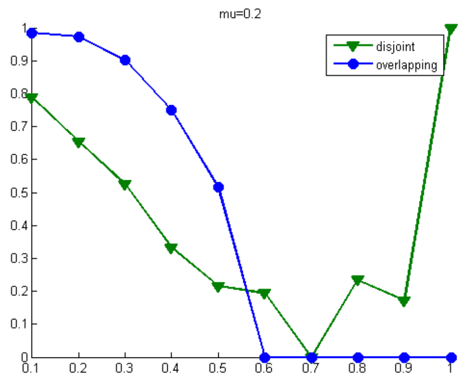
Based on the results above, since the networks parameter setting in this project is also a small networks, parameter *v* of COPRA should set as 6 like recommended.

8.3.3 - Performance affected by parameter of network

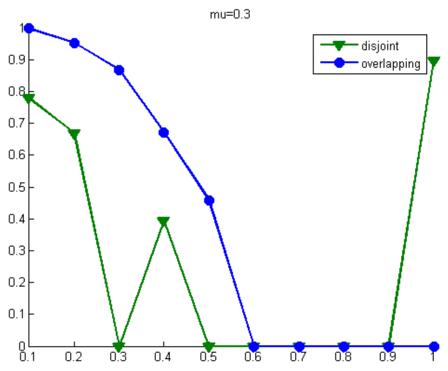
It has already been found that parameters of network can affect performance of algorithms in [1]. Therefore, we can estimate classes of algorithms instead of estimating parameters, since choice of class of algorithms is our final goal. To describe algorithm performance, NMI metric is adopted. We tested algorithms on the same networks with parameter setting listed in section 8.2.4.



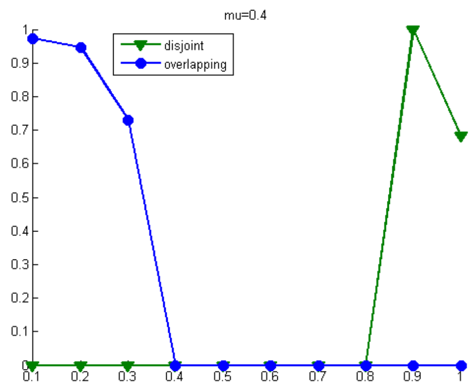
(a)



(b)



(c)

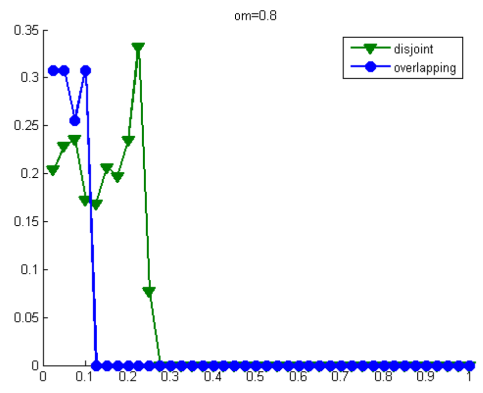


(d)

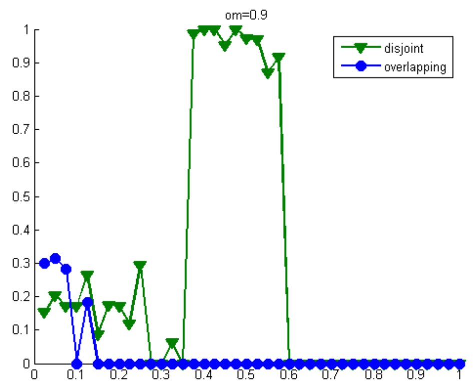
Figure 8.8: NMI scores (y-axis) change as o_m varies (x-axis). (a) plot shows results when μ_t is fixed on 0.1. Similar, subplot (b) is for the $\mu_t = 0.2$. Also, (c) and (d) refer to outputs when μ_t is 0.3 and 0.4, respectively.

Figure 8.8 shows the performances of disjoint COPRA (disjoint) and overlapping COPRA (overlapping) by calculating their NMI scores for different setting of μ_t. A comparative analysis method can help us figure out which class of algorithms performs better. By varying o_m we can concluded that, when o_m is lower than 0.6, overlapping class algorithm works better than disjoint class algorithm. Therefore, for each value of μ_t and $o_m < 0.6$ is the region of which overlapping class outperforms disjoint class algorithms. Notice that all the o_m values have been normalized.

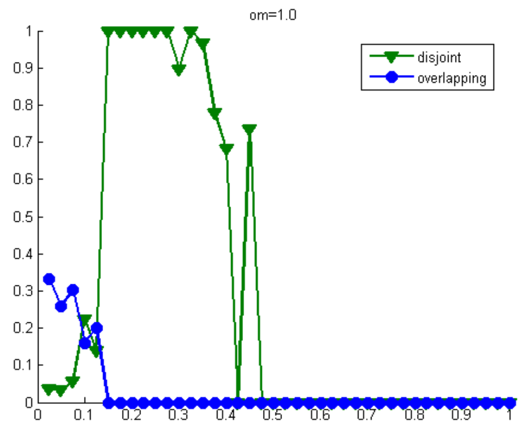
Furthermore, the region of which disjoint class of algorithms can be chose is still uncertain. From above results, we can assume until o_m is as high as 0.8, disjoint algorithm COPRA performs weak. Therefore, we observe those performances which values of o_m are higher than 0.8.



(a)



(b)

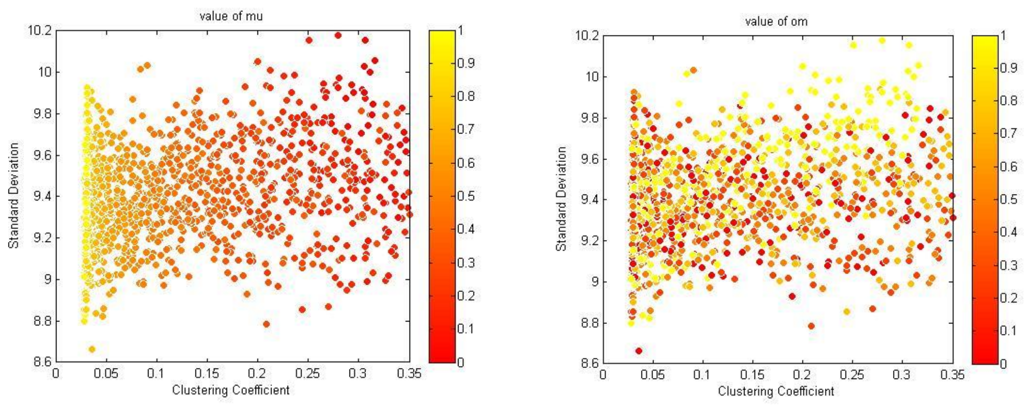


(c)

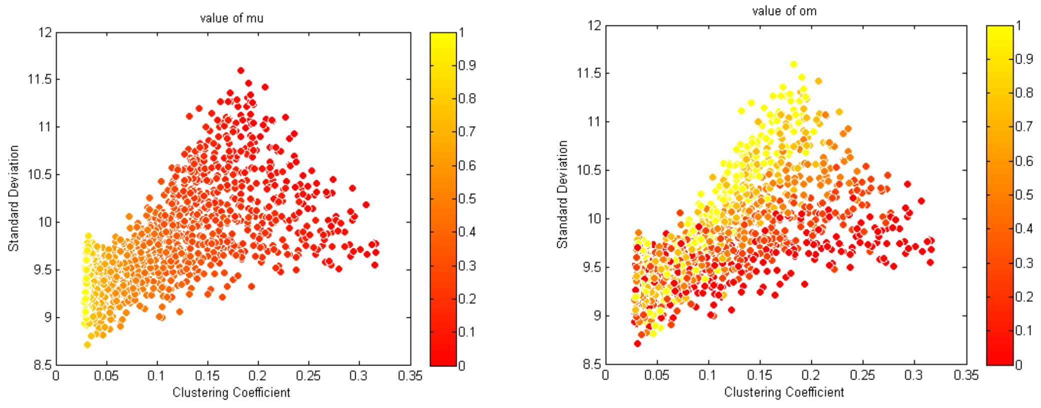
Figure 8.9: NMI scores (y-axis) of disjoint COPRA (disjoint) and overlapping COPRA (overlapping) as μ_t changes (x-axis) on different settings of o_m. (a) subplot is when $o_m = 0.8$. (b) and (c) are results when $o_m = 0.9$ and $o_m = 1.0$.

It can be seen from Figure 8.9 that as in different settings of o_m, when μ_t is higher than 0.1 and lower than 0.3, disjoint class algorithm shows superiority. Thus, if networks are generated by setting μ_t in range [0.1, 0.3] and o_m in the range [0.8, 1.0], we can select disjoint class algorithms to better detect communities. In contrast, overlapping class algorithms are preferable.

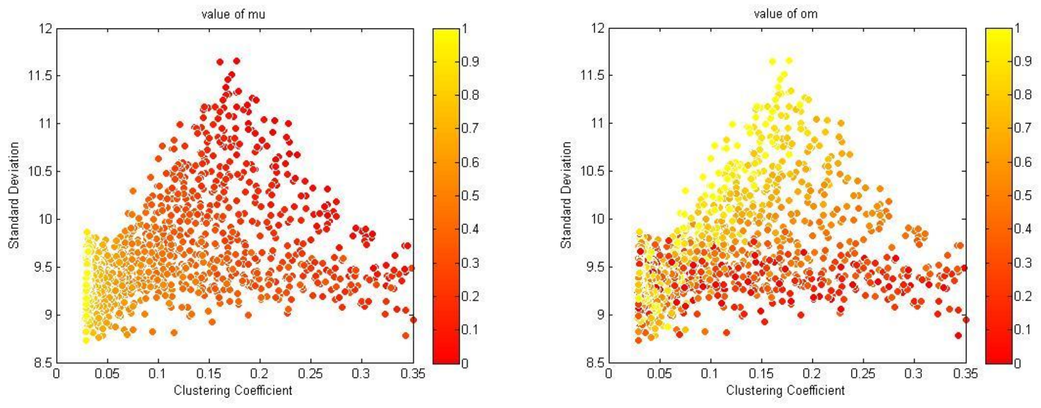
The results concluded from above need to be further proved that why $\mathrm{o_m}$ needs to be higher. We have analyzed that o_m should be in the range of [1, 10]. However, how much the higher part of range [5, 10] can contribute the overall results? A comparison need to be made to test the differences of CC and SD's impact between higher range of o_m and lower range which is [1, 5]. Parameter setting should remain the same as in section 8.2.4.



(a) $o_m = 1 - 5$



(b) $o_m = 5 - 10$



(c) $o_m = 1 - 10$

Figure 8.10: Influence of CC and SD on different range of o_m. Left: mixing parameter μ_t. Right: overlapping parameter o_m.

Scatter plots above (see Figure 8.10) confirm that when o_m is higher within the overall range, CC and SD can better estimate parameters which indicate that CC and SD can better predict the better class of algorithms.

8.4 - Results analysis

8.4.1 Estimation results

Tests prove the idea that extending Leto Peel's method into disjoint and overlapping classification can get good results. Between two overlapping parameters o_m and o_n, the former accompany with μ_t can be estimated by features of known networks CC and SD. However, tests on parameter o_n did not result in good solution. Based on this, we further tested whether parameter o_n can make sense in the classification: fuzzy overlapping algorithms and crisp overlapping algorithms, while the outcome turns out to be still not working. Therefore, we can assure that by CC and SD, we can estimate o_m and μ_t.

Knowing o_m and μ_t can influence algorithm performances, we can instead using CC and SD to estimate algorithm performances but not parameters any more. Like suggested in Leto Peel's paper, algorithm should be suitable for both classification of networks. For disjoint and overlapping classification, COPRA can meet our demand. While, another try has been carried out, which is choosing two algorithms and each one excels in one class of classification of networks. To approach this, algorithm Infomap for disjoint networks and algorithm GCE for overlapping networks are considered. Nevertheless, this appears to be fault.

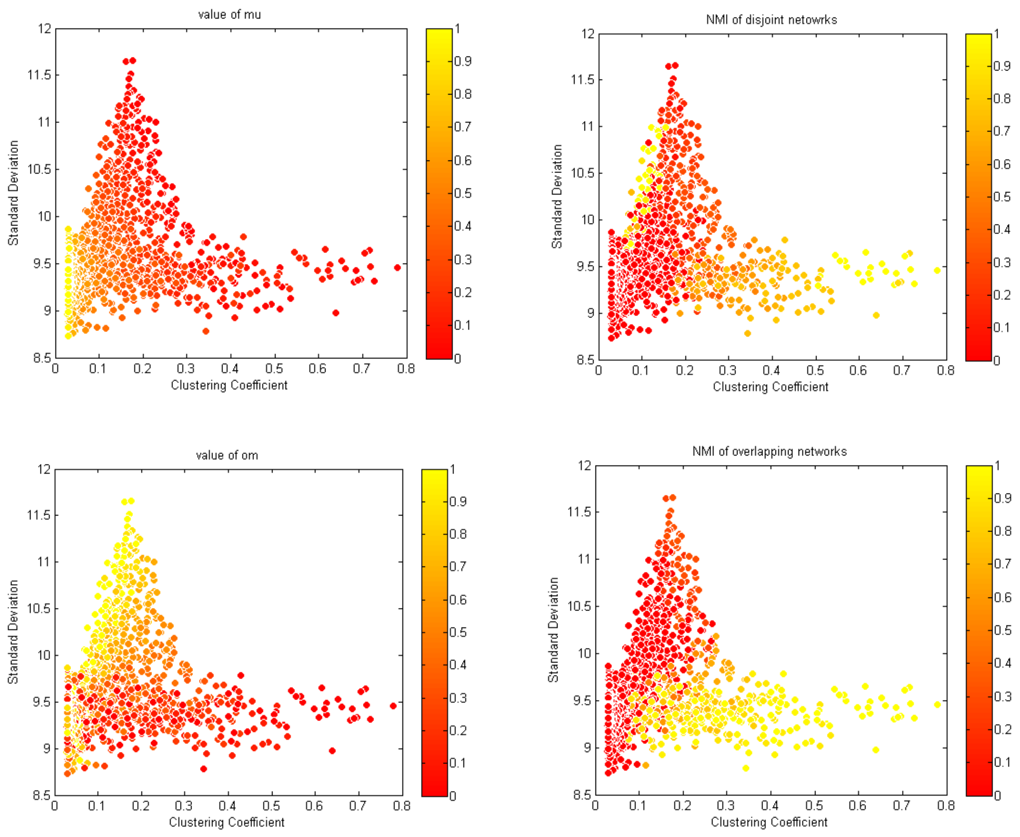


Figure8.11: CC and SD can estimate both parameters of networks (Left) and algorithm class (Right)

To be more specifically, if CC is higher and SD is lower, algorithm performance is better. This raise a question: How can we know which part is which class of algorithms. This can be done by a simple classifier: Support Vector Machine.

8.4.2 SVM prediction

Since SVM is a two-class classifier, we first need to assign each network a class {disjoint, overlapping}. In this project, SVM is trained on 4000 networks and randomly chose 2000 as tests. It take CC and SD as inputs, we stick to our original way to plot scatter figures to show how a SVM can make a prediction:

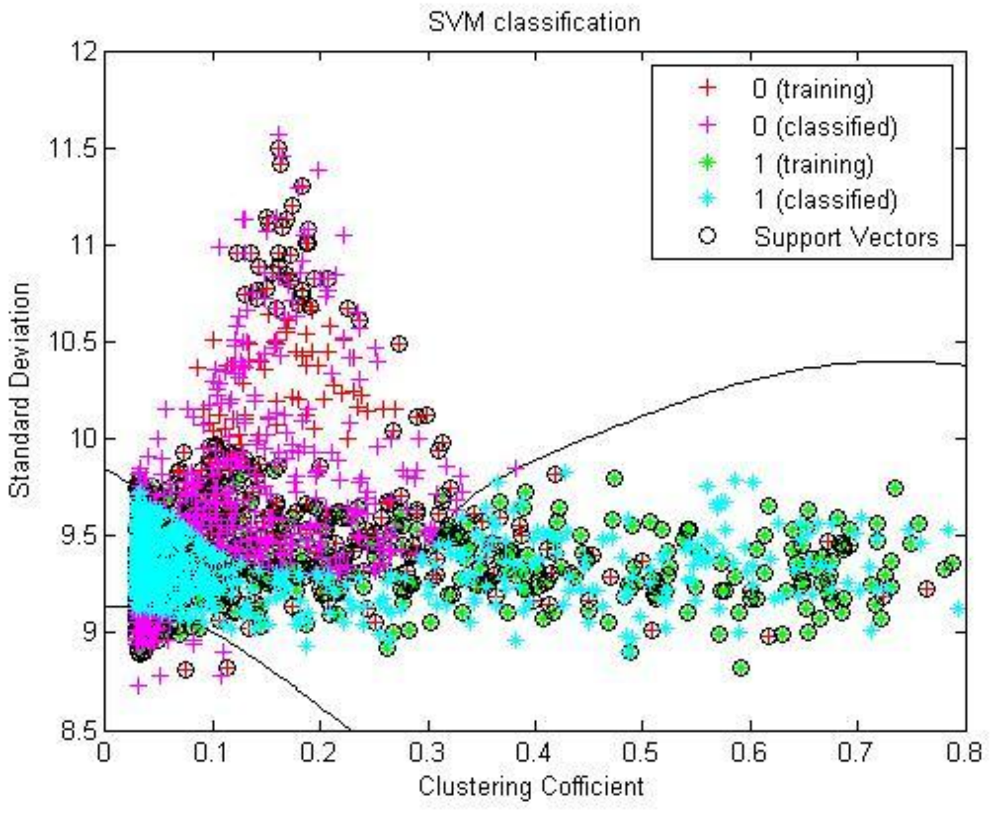


Figure 8.12: SVM classification. Two black lines show the classify results.

Figure 8.12 shows the good result that SVM can make a right prediction with accuracy rate higher than 0.6, such that confirm the former sections of discussion.

Table 4: Classifier SVM output

		Ture class	
		disjoint	overlapping
Predicted class	disjoint	209	79
	overlapping	291	421

The first row indicates the number of samples that were classified as positive, with the number of true positives in the first column, and the number of false positives in the second column. The second row indicates the number of samples that were classified as negative, with the number of false negatives in the first column, and the number of true negatives in the second column.

Correct classifications appear in the diagonal elements, and errors appear in the off-diagonal elements. Inconclusive results are considered errors and counted in the off-diagonal elements.

Table 4 affirms the prediction is valuable and can be adopted since off-diagonal elements are less than the numbers in the same row.

9 – Evaluation

There are three main objectives in this project. For the first objective which we need to find out features of known networks that separated into two classes by us to estimate possible parameters of those networks. This project successfully completed this aim. Let discuss it in detail:

- Parameter setting of artificial networks

The parameter setting of overlapping LFR benchmark in this project is based on existing study in [15]. We chose the most preferable parameter setting as parameters.

- Classes of networks/ algorithms

In Leto Peel's method (see chapter 7), he experimented on weighted and unweighted networks. What we need to notice is that these two classes can cover all networks since each network has to be weighted otherwise unweighted. According to this criterion, we set the classes to be {disjoint, overlapping}, since these two classes also represent the entire network. However, considering may be this classification, we set {fuzzy overlapping, crisp overlapping} to be the backup classification which only focuses on overlapping networks. This is what Leto Peel did not mention about. Finally, results show that the first classification can work and outputs are pretty optimistic (see section 8.2.3). On the contrary, one overlapping parameter o_n cannot be estimated even in {fuzzy, crisp} classification not to mention {disjoint, overlapping} classes (see section 8.2.2 and section 8.2.5). Even so, the critical experiments are still meaningful, since we can assure that only one overlapping o_m can be estimated.

- Features of network

There are lots of properties can represent network topology and its behavior. Among them, we chose clustering coefficient (CC) and standard deviation of degree (SD) as our features of networks. The reason of picking them is that CC is already tested in Leto Peel's method and showed positive results that it can estimate mixing parameter u_t, what's more, SD can refer overlapping attribute and we analyzed it in section 8.2.1. Another reason that CC and SD are selected is that the mean value of these two properties can represent each whole network. While other properties only focus on some certain vertices not the entire network, such as edge betweeness or centrality. However, there still exists a selection, average shortest path, but considering its computing time and network size, we deprecated that. Eventually, results show that CC and SD can estimate mixing parameter and overlapping parameter of artificial networks (section 8.2.3).

- Overlapping parameter

There are two overlapping parameters in overlapping LFR benchmark, so we need to find out the one shows best influence affected by CC and SD. We plot scatters to compare them in section 8.2.3 and section 8.2.4. Figures indicate that o_m gives out clearer and better distinct results. Therefore, o_m is the one we trying to estimate. The value of fixed o_n is decided after testing and analysis in section 8.2.3. We finally chose the one that $o_n=300$ for better plots and results.

For the second objective which requires that we need to confirm that CC and SD can further estimate algorithm performances. This objective also successfully achieved in section 8.3. However, there are some issues we need to do some critical thinking and research.

- Restriction of algorithm

In Leto Peel' method, algorithms have to be suitable for each class of networks for control comparison. For this respect, COPRA is considered in this project, since it has both disjoint network version and overlapping network version controlled a parameter v. However, control comparison can be built in other direction: two algorithms and each excel in only one class of networks. This is omitted by Leto Peel while we did the test in section 8.3.1, of which algorithm Infomap which experts in disjoint networks and algorithm GCE which overwhelming in overlapping networks are adopted. However, result is not optimistic for these two algorithms not showing distinct performances (see 8.3.1). Therefore, COPRA is one we take into consideration. Actually, we cannot fully ascertain a conclusion only based on one set of data, thus further work testing on other algorithm is desirable. This project only got one set of positive results is because of the one month time limit. Finally, good results are obtained and analysis is in section 8.4.1, where disjoint COPRA and overlapping COPRA show distinct performances controlled by CC and SD, thus we can estimate class of algorithms.

- Region that algorithm experts

COPRA algorithm performances analysis presents a result that when o_m is higher that 0.8 (after normalization), there will be regions that different class of COPRA outperforms each other. We did some analysis to find out the reason why o_m has to be high in section 8.3.3. It can be seen that when o_m is higher, CC and SD can better estimate parameters and future estimate class of algorithms.

For the third objective, we adopted a simple classifier support vector machine (SVM) to classify algorithms and then successfully got good predictions (see section 8.4.2). Until now, the whole aim of this project has been completed and realized successfully, what's more, we also did lots of critical thinking and evaluation to confirm our results. If time allows, future work can be done to go on explore this area of community detection of network.

10 – Future work

This project explores a method that by estimating parameters of network, one can choose a class of community detection algorithms. We first tested on a classification: disjoint and overlapping network algorithms, if this classification doesn't work, we in turn focus on another two classes: fuzzy overlapping and crisp overlapping network algorithms.

The parameters of networks are mixing parameter and overlapping parameter of the parameter setting for generating artificial networks. We adopt LFR benchmark for both two classifications. Since there are two overlapping parameters, we need to find out the one shows more relationship with features that can estimate them. Between overlapping parameters o_m and o_n, o_m presented more obvious interaction with clustering coefficient CC and standard deviation degree SD, which are two structural parameter of networks that can be calculated even without explicitly getting the community structural assignments. Therefore, overlapping parameter o_m and mixing parameter μ_t can be estimated by CC and SD. Further, we tested overlapping parameter o_n for the second classification, however, no clear relationship shown between o_n and SD on fuzzy overlapping and crisp overlapping networks.

Study shows that mixing parameter and overlap parameter can affect algorithm performance. We use the NMI measure to score COPRA algorithm performances on both disjoint networks and overlapping networks. Results show that even for overlapping networks, disjoint version of COPRA can still outperform overlapping version COPRA in certain regions, which confirms our assumption that for a class of networks, there exists a choice between two classes of algorithms. Using CC and SD to estimate class of algorithms instead of parameters of networks can give us good results.

Finally, SVM is a machine learning algorithm which can help us classify algorithm classes and make a prediction automatically. Thus, selecting community detection algorithms by estimating parameters can be realized.

Automatically choosing community detection algorithms is a new area of network study, the state of art is the achievement by Leto Peel in [2]. More specifically, Leto Peel realized a method that by estimating parameters of networks, one can choose the class of algorithms. However, this only has been carried out for the classification: weighted network algorithms and unweighted network algorithms. This project successfully extended this innovative method on other classification: disjoint network algorithms and overlapping network algorithms. Nevertheless, this still leave a large scale of problem space to explore: different classes of algorithms and even future can apply this method on real networks. To approach this, artificial network generator parameter community size should be set to a specific value, which requires estimation.

Towards this project results: by using SVM, one can choose a class of algorithm with the accuracy rate 0.63. This can be improved by setting a new input of SVM, which is a threshold of the NMI of algorithms. This can be explained that when NMI is lower than a certain value, algorithm results seem to be not making too much sense, such that can affect SVM prediction. However, SVM is a two-class classifier, which restrict the input parameters to be only two of them. We can solve that by setting the input pairs to be {disjoint, overlapping}, {disjoint, none}, {overlapping, none}, of which *none* represents those algorithms with NMI scores lower than a threshold. Finally, a prediction can be selected by a voting scheme like recommended in [2]. Since computing all the training set around 4000 networks is really time-consuming, we left calculating the threshold of disjoint and overlapping classification as future work.

Reference

[1] A. Lancichinetti and S. Fortunato, "Community detection algorithms: A comparative analysis," Physical Review E, vol. 80, 2009.

[2] L. Peel, "Estimating Network Parameters for Selecting Community Detection Algorithms", BAE systems, Bristol, UK

[3] S. Fortunato, "Community detection in graphs", Physics Reports 486 (2010) 75-174

[4] M. E. J. Newman, "The structure and function of complex networks", SIAM Review 45, 167-256 (2003)

[5] Erdos, P. and Renyi, A., "On random graphs", Publicationes Mathematicae 6, 290-297 (1959).

[6] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J., " Graph structure in the web", Computer Networks 33,309{320 (2000).

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

[8] S. Gregory, "Finding overlapping communities using disjoint community detection algorithms", in: S. Fortunato, R. Menezes, G. Mangioni, V. Nicosia

[9] G. Palla, I. Derényi, I. Farkas, T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society", Nature 435(2005) 814818.

[10] S. Gregory, "Fuzzy overlapping communities in networks", (2011), J. Stat. Mech. P02017

[11] Sawardecker E N, Sales-Pardo M and Amaral L A N 2009 Detection of node group membership in networks with group overlap Eur. Phys. J. B 67 277

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

[13] A. Condon, R.M. Karp, "Algorithms for graph partitioning on the planted partition model", Random Struct. Algor. 18 (2001) 116140.

[14] A. Lancichinetti, S. Fortunato, F. Radicchi, "Benchmark graphs for testing community detection algorithms" , Phys. Rev. E 78 (4) (2008) 046110.

[15] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," Physical Review E, vol. 80, (2009), p. 16118.

[16] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," Physical Review E, vol. 80, 2009, p. 16118.

[17] M.E.J. Newman, "Modularity and community structure in networks", Proc. Nat. Acad. Sci. USA 103 (2006) 8577-8582

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

[19] Ziv E, Middendorf M, Wiggins CH,"Information-theoretic approach to network modularity"(2005) Phys Rev E 71:046117.

[20] J. Baumes, M. Goldberg, M. S. Krishnamoorthy, M. M. Ismail, N. Preston, "Finding communities by clustering a graph into overlapping subgraphs", N. Guimaraes, P. T. Isaias (Eds.) ,IADIS AC, IADIS, 2005, 97- 104

[21] J. Baumes, M. Goldberg, M. M. Ismail, “Efficient identification of overlapping communities”, IEEE International Conference on Intelligence and Security Informatics, ISI, 2005, 27- 36

[22] M. Rosvall, C.T. Bergstrom,” Maps of random walks on complex networks reveal community structure” , Proc. Natl. Acad. Sci. USA 105 (2008) 11181123.

[23] A. Lancichinetti, S. Fortunato, and J. Kertész, “Detecting the overlapping and hierarchical community structure of complex networks,” New J. Phys, vol. 11, 2009, p. 033015.

[24] C.Lee, F.Reid, A. McDaid, and N. Hurley, “Detecting highly overlapping community structure by greedy clique expansion”, in SNA-KDD 2010 (ACM, Washington, DC,2010) pp. 33- 42.

[25] S. Gregory, “Finding overlapping communities in networks by label propagation” , New J.Phys. 12 103018 (2010)

[26] M. E. J. Newman, “The structure and function of complex networks”, SIAM Review 45, 167-256 (2003)

[27] 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. 91102.

[28] N. Cristianini and J. Shawe-Taylor, “An introduction to support vector machines”, Cambridge University Press, (2000).

[29] T. Mitchell.” Machine Learning.” McGraw-Hill International, 1997.

[30] Danon L, D´ıaz-Guilera A, Duch J and Arenas A, ”Comparing community structure identification”, (2005) Journal of Statistical Mechanics, P09008

[31] Burbidge, R., Buxton, B.F.: An introduction to support vector machines for data mining. In Sheppee, M., ed.: Keynote Papers, Young OR12, University of Nottingham, Operational Research Society, Operational Research Society (2001) 3–15

Appendix: Source code

Computing SD and CC:

```
public static void readGraph(String filename) throws FileNotFoundException {
            int i;
            int filelength = 0;
            int numberpoint = 0;
            Iterator<Integer> it;
            ArrayList<HashSet<Integer>> edges = new ArrayList<HashSet<Integer>>();
            ArrayList<HashSet<Integer>> edges2 = new ArrayList<HashSet<Integer>>();
            File filepath = new File(filename);
             if (filepath == null){
                  throw new IllegalArgumentException("File should not be null.");
             }
             if (!filepath.exists()){
                 throw new IllegalArgumentException("File does not exist: "+filepath);
             }
            Scanner scanFile1 = new Scanner(filepath);
             try {
                 while (scanFile1.hasNextLine()){
                 //processLine(scanFile.nextLine());
                 String oneLine = scanFile1.nextLine();
                 Scanner scanLine = new Scanner(oneLine);
                   if (scanLine.hasNext()){
                     filelength++;
                     } else {
                    }
               scanLine.close();
               }
             } finally {
             scanFile1.close();
            }
            System.out.println("filelength value is2:"+filelength);
            for (i=0; i<filelength; i++) {
                edges.add(new HashSet<Integer>());      //store in the hashset
                edges2.add(new HashSet<Integer>());      //store in the hashset
             }

             Scanner scanFile = new Scanner(filepath);
             int index=0;
             try {
                 while (scanFile.hasNextLine()){
                 //processLine(scanFile.nextLine());
            String oneLine = scanFile.nextLine();
            Scanner scanLine = new Scanner(oneLine);
              if (scanLine.hasNext()){
                    int nodes1 = Integer.parseInt(scanLine.next());
                    int nodes2 = Integer.parseInt(scanLine.next());
                    edges.get(nodes1).add(nodes2);
                    edges.get(nodes2).add(nodes1)
                    edges2.get(index).add(nodes1);
                    edges2.get(index).add(nodes2);
                    index++;
                     System.out.println(""+nodes1+""+nodes2);
                     } else {
                          }
             scanLine.close();
            }
          } finally {
               scanFile.close();
        }
      System.out.println("index value is:"+index);
      for(HashSet<Integer> ns : edges2){
            for(Integer n1 : ns) {
                for(Integer n2 : ns) {
                     if (n1 < n2) {
                        numberpoint = n2;
                          }
                     }
                 }
            }
      numberpoint++;
      System.out.println("numberpoint value is2:"+numberpoint);
      int triples = 0;
      int closed = 0;
      int[] degreenumber = new int[numberpoint];
      index=0;
     for(HashSet<Integer> ns : edges2){
      for(Integer n1 : ns) {
           for(Integer n2 : ns) {
                if (n1 < n2) {
                     if (edges2.get(index).contains(n2)) {
                          index++;
                      degreenumber[n1]++;
                      degreenumber[n2]++;
                          }
                    }
```

```
                  }
              }
   }
  System.out.println("index value is2:"+index);
  for(HashSet<Integer> ns : edges){
         for(Integer n1 : ns) {
                for(Integer n2 : ns) {
                        if (n1 < n2) {
                               triples++;
                               System.out.print("Triple "+n1+" "+n2);
                                 if (edges.get(n1).contains(n2)) {
                                        closed++;
                                        System.out.print(" closed");
                                      }
                                 System.out.println();
                                      }
                               }
                        }
                 }


  double totaldegree = 0;
  double averageDegree = 0;
  double variance = 0;
  double totaldegreesquare = 0;
  double[] degreesquare= new double[numberpoint];
          for(i=0;i<numberpoint;i++)
  {
   degreenumber[i] /=2;
     totaldegree += degreenumber[i];
  }
  System.out.println("totaldegree value is2:"+totaldegree);
  averageDegree = totaldegree/numberpoint;
  System.out.println("averageDegree value is2:"+averageDegree);
  for(i=0;i<numberpoint;i++)
  {
degreesquare[i]=(degreenumber[i] - averageDegree)*(degreenumber[i] - averageDegree);
     totaldegreesquare += degreesquare[i];
  }
  System.out.println("totaldegreesquare value is2:"+totaldegreesquare);
  variance =Math.sqrt( totaldegreesquare/numberpoint);
  System.out.println("The variance is:"+" "+variance);

  double cc = (double)closed/triples;
```

```
            System.out.println("closed"+closed);
            System.out.println("triples"+triples);
            System.out.println("cc "+cc);
        }
}
```

Matlab plot:

Main function:

```
clear;
cd('D:\study\matlab')
filep='on.xlsx';
cd('D:\study\matlab'
data=xlsread(filep);
x=[data(:,3) data(:,4) data(:,1)];
y=[data(:,3) data(:,4) data(:,2)];
%%%%%%%%%%%%%%%%%%%
    x(:,3)=(x(:,3)-min(min(x(:,3))))/(max(max(x(:,3)))-min(min(x(:,3))));
    y(:,3)=(y(:,3)-min(min(y(:,3))))/(max(max(y(:,3)))-min(min(y(:,3))));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure
skplot(x);
figure
number(x);
figure
number_y(x);
figure
skplot(y);
figure
number(y);
figure
number_y(y);
```

skplot(x) function:

```
function skplot(x)
[a b]=size(x);
rgb=[0 0 0];
for i=1:a
y=[x(i,1) x(i,2)];
%xlim([0.00,0.35]);
%%rgb=[x(i,3) x(i,4) x(i,5)];
rgb=[1 x(i,3) 0];
```

```
plot(y(:,1),y(:,2),'MarkerFaceColor',rgb,...
    'MarkerEdgeColor',[1 1 1],...
    'Marker','o',...
    'LineStyle','none');%'DisplayName','b(:,2)'
xlabel('Clustering Coefficient');
ylabel('Standard Deviation');
%xlim([0.0,0.35]);
hold on
end
hold off

end
```

number(x):

```
function number(x)
[a b]=size(x);
A=[0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0];


for i=1:a       %read in every line of matrix x

    if 0<=x(i,3)&&x(i,3)<=0.3     %rgb is red
        j=1;
        else if 0.3<x(i,3)&&x(i,3)<=0.7      %rgb is orange
                j=2;
            else if 0.7<x(i,3)&&x(i,3)<=1.0      %rgb is yellow
                    j=3;

                end
            end
    end
        if 0<=x(i,1)&&x(i,1)<0.05
            A(j,1)=A(j,1)+1;
        else if 0.05<=x(i,1)&&x(i,1)<0.10
                A(j,2)=A(j,2)+1;
            else if 0.10<=x(i,1)&&x(i,1)<0.15
                A(j,3)=A(j,3)+1;
                 else if 0.15<=x(i,1)&&x(i,1)<0.20
                         A(j,4)=A(j,4)+1;
                      else if 0.20<=x(i,1)&&x(i,1)<0.25
                              A(j,5)=A(j,5)+1;
                           else if 0.25<=x(i,1)&&x(i,1)<0.30
                                   A(j,6)=A(j,6)+1;
                                   else if 0.30<=x(i,1)&&x(i,1)<0.35
```

```
                                                       A(j,7)=A(j,7)+1;
                                                       else if 0.35<=x(i,1)&&x(i,1)<0.40
                                                               A(j,8)=A(j,8)+1;
                                                               else if 0.40<=x(i,1)&&x(i,1)<0.45
                                                                         A(j,9)=A(j,9)+1;
                                                                         else if
0.40<=x(i,1)&&x(i,1)<0.60

A(j,10)=A(j,10)+1;
                                                                              end
                                                                    end
                                                           end
                                                  end
                                         end
                                end
                      end
                end
            end
        end
end
  z=1;
   y1=[0.00 A(z,1);0.05 A(z,2);0.10 A(z,3);0.15 A(z,4);0.20 A(z,5);0.25 A(z,6);0.30 A(z,7);0.35 A(z,8);0.40
A(z,9);0.60 A(z,10)];
  zz=2;
  y2=[0.00 A(zz,1);0.05 A(zz,2);0.10 A(zz,3);0.15 A(zz,4);0.20 A(zz,5);0.25 A(zz,6);0.30 A(zz,7);0.35
A(zz,8);0.40 A(zz,9);0.60 A(zz,10)];
  zzz=3;
  y3=[0.00 A(zzz,1);0.05 A(zzz,2);0.10 A(zzz,3);0.15 A(zzz,4);0.20 A(zzz,5);0.25 A(zzz,6);0.30 A(zzz,7);0.35
A(zzz,8);0.40 A(zzz,9);0.60 A(zzz,10)];
  hold on
  plot(y1(:,1),y1(:,2),'Marker','pentagram',...
       'LineStyle','-','MarkerFaceColor','red','Color','red','LineWidth',2);
  plot(y2(:,1),y2(:,2),'Marker','o',...
       'LineStyle','-','MarkerFaceColor',[1 0.4 0],'Color',[1 0.4 0],'LineWidth',2);
  plot(y3(:,1),y3(:,2),'Marker','^',...
       'LineStyle','-','MarkerFaceColor','yellow','Color','yellow','LineWidth',2);
xlabel('Clustering Coefficient');
ylabel('Network number');
title('om & cc');

end
```