Abstract

This project is to incorporate some instance-level constraints to recently-proposed community detection algorithms. The instance-level constraints specify whether two vertices belong to the same community or not, known as the must-link and cannot-link constraints.

The general framework of incorporating constraints is to find a mapping that will map the similarity data to a high dimension space (kernel), where two must-link nodes are mapped to be close enough and two cannot-link nodes are mapped to be far apart. In addition, in the new high space dimension, not only the constraints are satisfied, but also those vertices similar to the must-link vertices in the input space will be clustered into the same community and those similar to the cannot-link vertices will be clustered into the different communities.

Various techniques are used in all steps of implementation. In the step of formulating similarity matrix, the method based on the shortest path is selected after comparing with other methods. Moreover, the most challenging of this project is optimizing kernel matrix, which is addressed via Semidefinite Quadratic Linear Programming proposed by Wu et al [35].

Experimental results show that this algorithm can successfully detect community structure in famous benchmark networks (such as GN and LFR) and many real world networks (such as karate club and dolphins network).

Table of Contents

1: Introduction

Community detection has become an increasingly significant topic in machine learning and attracted considerable research interests in recent years. Generally speaking, community structure in networks is characterized by groups of vertices within which are densely connected and between which are sparsely connected, so it is commonly agreed that the vertices within a community are likely to share common properties and characters. Therefore, community detection has considerable significances in terms of identifying the implicit relationships between the nodes in the community.

There are all kinds of networks in the world, including Internet, social networks (with relation of friendship or colleagues), biological networks (such as gene network, neural network), and many other networks. Vertices of these networks are often grouped into communities or clusters which make community structure become a very common property in these networks. The vertices within a community are likely to share common properties and play similar roles within the network. For instance, people with similar hobbies often group together in the same community in social networks. Thus, the community structure is valuable and useful information for both commercial and scientific purpose. In addition, an efficient algorithm to detect and analyze community structures may be considerably useful in many fields.

Over the years, researchers have introduced a large number of algorithms towards community detections. Traditional methods of detecting communities were mainly from Computer Science and Social Network Analysis. In general, the methods for community detection are grouped as two categories: Optimization-based algorithms and Heuristic algorithms. Between optimization-based algorithms, two well-known algorithms are spectral algorithm and Kernighan-Lin algorithm. In heuristic algorithms, Girvan-Newman and Wu-Huberman are the typical algorithms.

Although these algorithms are effective for community detection in most cases, however in the real world networks there are all kinds of constraints, which cannot be addressed by most of recent community detection algorithms. In addition, in some cases, relations of nodes in networks are not clearly enough for community detection, especially the lack of background knowledge, constraints will be extremely useful as a kind of background knowledge.

1.1 Aims and objectives

This project in general is to incorporate some constraints (background information) on the most recently-proposed community detection (clustering) algorithms and evaluate

them in terms of accuracy and complexity by using some computer-generated and real world networks.

The aims are set to be:
1. Investigate the constraints and find the suitable community algorithm to incorporate constraints.
2. Design a general framework model to incorporate these constraints and implement the new algorithm.
3. Find a suitable benchmark and evaluate the new method in terms of accuracy and complexity.
4. Test the performances and compare with other community detection algorithms using the computer generated and real world networks, then improve the program.

1.2 Organization of dissertation

The dissertation is composed of following parts.
1. In chapter 2, some background information will be reviewed from some classical and prevalent community detection algorithms as well as some instance-level constraints. The analysis of how to incorporate constraints is also discussed in this part.
2. In chapter 3, the details of implementation will be listed as well as some problem will be specified.
3. In chapter 4, a description of the experimental performance and some comparisons with other algorithms will be given.
4. In chapter 5, a general evaluation as well as a final conclusion of this project will be made.
5. In chapter 6, some future suggestion and improvement will be discussed.

2: Background and Context

This section is intended as an introduction to some real world networks, community detection methods, instance-level constraints and well as an analysis of challenges of incorporating these constraints into community detection methods.

2.1 Networks

A network is an abstract structure that represents a system with a collection of nodes which is connected together by some pairs of lines. These nodes are referred to as vertices and these lines are usually called edges. A simple network can be seen in the figure 2.1.

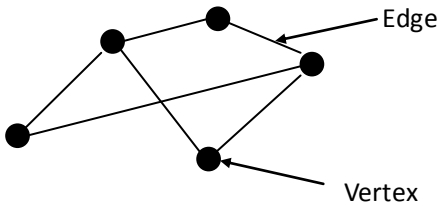


Figure 2.1: A small example network composed of five vertices and six edges.

Why are we so interested in Networks?
There are all kinds of networks in the world, including Internet, social networks with relation of friendship or colleagues, biological networks such as gene network, neural network, and many other networks.

One of the most important and widely used networks is Internet. The Internet is one kind of technological network, which connect millions of computers as vertices all over the world, and the edges are those physical connections between them, such as optical fiber cables.

Moving from the technological realm, another popular network which has been widely studied is the social network. A social network is usually a set or group of people with some relation between them, such as colleagues, relatives, classmates or business relations between companies. Figure 2.2 shows a network of mobile phone communications between users which is analyzed by Blondel et al [39].

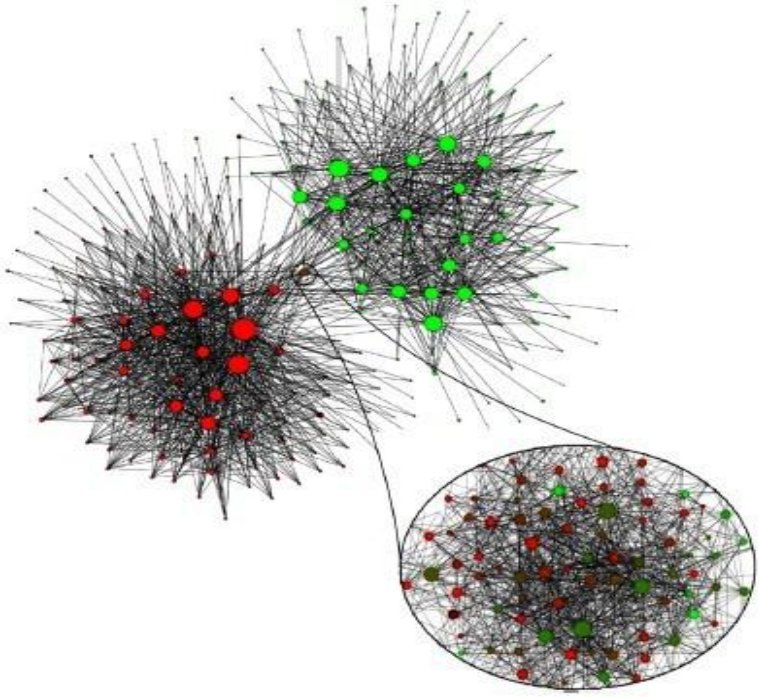


Figure 2.2: Community structure of a social Belgian mobile network. Vertices denote the subgroups with more than 100 customers. Colors represent the language spoken in the community (red for French and green for Dutch). Source: Ref. [39].

Social network, such as Myspace and Facebook are extremely popular these days and scientists in a wide variety of fields have done much research in these networks. Traud et al [40] survey the friendship of different American university students with some anonymous Facebook data and construct the structure of this network (figure 2.3).

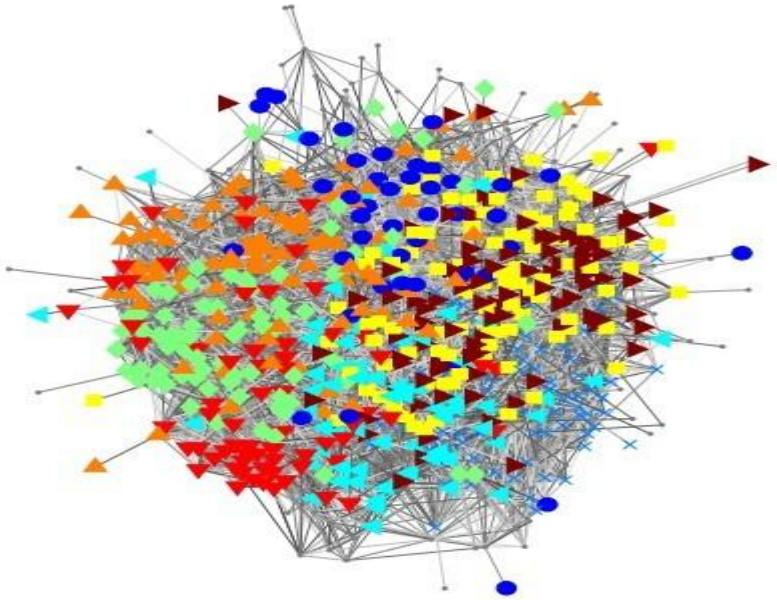


Figure 2.3: A social network of friendships between students at Caltech. The colors denote the dormitories of the students. Source: Ref. [40].

Another popular researched network is biological network. Actually a large quantity of biological systems can be regarded as networks. Some of them are highly researched, such as metabolic networks (Figure 2.4), Gene networks and neural networks. Generally, the communities in biological networks are characterized by functional organizations and the vertices between communities often have some functional relationship between them. There are many applications in biology using the community detection algorithm. For example, Holme et al [50] have successfully detected the hierarchical structure of metabolic networks using the algorithm proposed by Girvan and Newman [42].

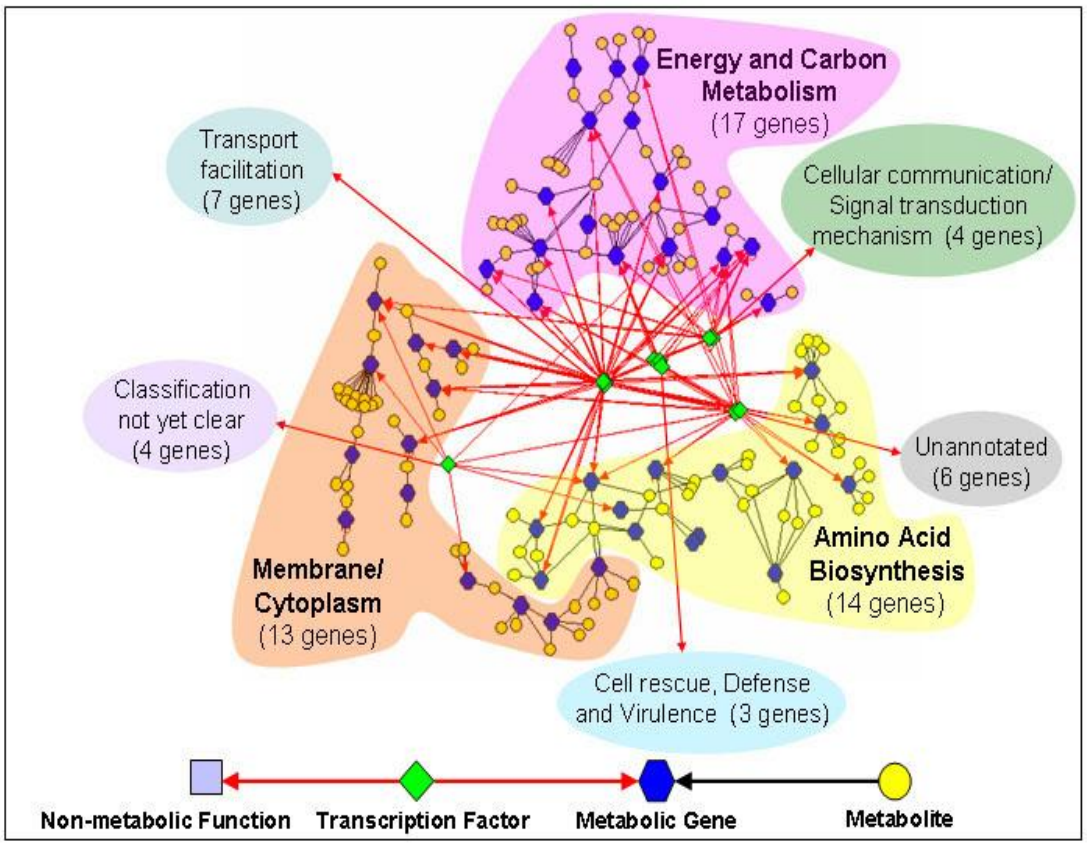


Figure 2.4: A metabolic network consists of metabolic genes involving in energy metabolism and amino acid biosynthesis, which is connected by a number of transcription factors. Source: Ref. [51].

2.2 Community detection

With increasing evidence that various systems in nature and society can be modeled as networks, community detection in networks becomes a hot research topic recently. Vertices of many real-world networks, such as Internet, social networks, biological networks, are often grouped into communities or clusters which make community structure become a very common property in these networks. The vertices within a community are likely to share common properties and play similar roles within the network. For instance, people with similar hobbies often group together with the type of community in social networks. Therefore, an efficient algorithm to detect and analyze community structures will be considerably useful in many fields.

In recent years, community detection has been an important topic in machine learning and has attracted considerable research interests from scientific communities, such as social network analysis, information retrieval. Community can have concrete applications. For example, identifying clusters of customers with similar interests in the networks of purchase relationships between customers and products of online retailers (like ebay, amazon) enables to set up efficient recommendation systems [10].

Motivated by the idea that community detection is in nature a kind of graph clustering, so we can detect community structure with the utilization of graph clustering algorithms. Over the years, researchers have introduced a large number of methods towards community detections based on clustering algorithms.

2.2.1 Traditional methods for community detection

Traditional methods of detecting communities were mainly from Computer Science and Social Network Analysis.

In computer science, the community detection task is usually called Graph Partitioning Problem. They divided the vertices of a network into some numbers of groups with roughly equal size and minimized the number of edges between different groups [22]. The number of edges between groups is called cut size. Graph partitioning is a fundamental issue in parallel computing, circuit partitioning. Majority of the graph partitioning problem are NP-hard and only several algorithms can do a good job. In practice, many algorithms of graph partitioning have been based on iterative bisection, which divided the graph into two groups and then further subdivided those two until it satisfied the specified number of groups [22]. Among these algorithms, two of them are very popular in previous: the spectral bisection method and Kernighan-Lin algorithm [22].

Spectral bisection method is based on the properties of the spectrum of the Laplacian matrix. The Laplacian matrix of an n-vertex undirected graph G is the $n \times n$ symmetric

matrix L whose element Lij is the similarity of vertex i and j, and the diagonal element Lii is the sum of Lij(j from 0 to n, not include i). Alternatively, L can also be written with L=D-A, where D is the diagonal matrix and A is the similarity matrix. Because all the rows and columns of matrix L are zero, therefore the vector **1**=(1,1,1,1,1,1…) (the number is n) is always an eigenvector with eigenvalue zero. The spectral bisection method is reasonably fast. However, the calculation of the eigenvectors of an n×n symmetric matrix generally takes nearly $O(n^3)$ which is rather slow [22]. But in most cases of practical network, the connection between networks is sparse, so the eigenvectors will be calculated more rapidly. The fundamental drawback of spectral bisection is that it only bisects graphs. The community partition of a network is usually achieved by repeated bisection, but this is always not a general optimization method.

Another popular method is Kernighan-Lin algorithm, which is a completely different approach to spectral bisection. Kernighan-Lin method is a greedy optimization method that optimizes a benefit function Q [22]. The benefit function Q represents the numbers of edges inside the group minus the edges between different groups. The size of the group will be specified in advance, which can be random or suggested by background information in the network structure. On sparse networks, a slightly different heuristic allows to lower the complexity to $O(n^2)$ [22]. The main drawback of this method is that it need specify the size of communities in advance. In addition, the detected communities are strongly dependent with initial configuration. Thus, a good start will result a more preferable division. Another problem of this method is that the run-time and storage costs will dramatically increase with the number of clusters [23]. Even if all of these problems can be overcome, the principal shortcoming is that it can only optimize the two groups. Although more subgroups can be division through iteration bisection, there is no guarantee about the general optimization.

In study of social networks, sociologists developed hierarchical clustering methods for detecting communities in networks [22]. Hierarchical clustering is very common and often used in many real networks, such as social network, biology etc. These networks often have a hierarchical network structure. Hierarchical clustering method is used to identify groups of vertices with high similarity. Thus, the first step of hierarchical clustering is to define a similarity measure between vertices based on the network structure. In previous works, many similarity measure are proposed, several of them will be discussed below. In general, hierarchical clustering method can be divided into two categories [32]: agglomerative and divisive. The first strategy is a "bottom up" approach, starting from an empty network with n vertices, but no edges, then vertices will are gradually connected by edges to merge into communities. The second strategy is a "top down" approach, starting from the complete network, then edges are gradually removed to merge individual communities. The process of these methods could be terminated at any step and gets a corresponding community structure of networks. The method of hierarchical clustering is shown in figure 2.5.

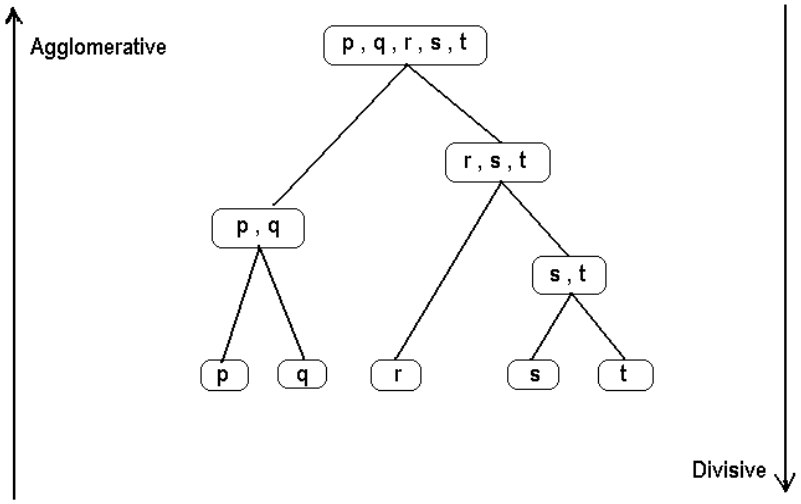


Figure 2.5: Hierarchical Clustering. Source: Ref. [32].

There are a couple of different ways to extract community based on hierarchical clustering methods [32]. The most common method is called single linkage clustering [32], which is also known as the nearest neighbor technique. The similarity between two communities is the minimum element xij with i in one community and j in the other community. The complete linkage clustering is also called farthest neighbor, which is on the opposite of linkage clustering. The similarity between two communities is the maximum element xij with i in one community and j in the other community. Another method is called average linkage clustering. The similarity between two communities is the average similarity element xij with i in one community and j in the other community. The following figure 2.6 is an example of a small hierarchical tree [19].

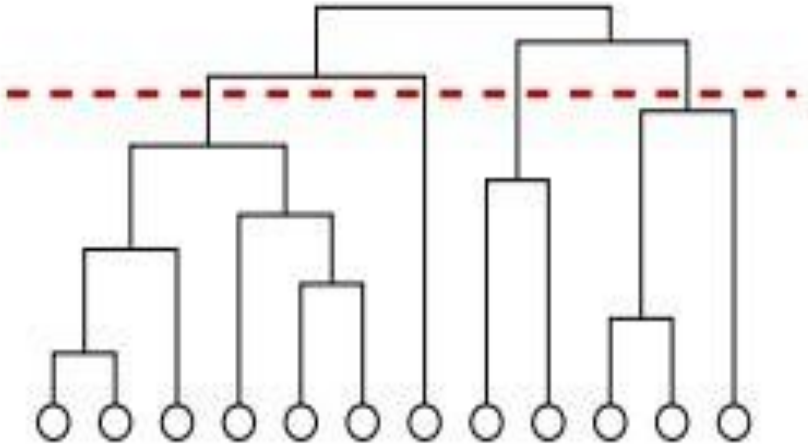


Figure 2.6: An example of a small hierarchical tree. Source: Ref. [19].

2.2.2 Modern methods for community detection

Although some previous traditional methods are useful for community detection, there are some obvious shortcomings as mentioned before. Recently, many algorithms are proposed, such as Girvan-Newman algorithm, spectral clustering, modularity optimization based algorithm. Some of them will be introduced in the following part.

Girvan-Newman algorithm is one of most popular community detection algorithms. Instead of finding the most central edge of the community, it tries to find the edge which is the least central of the community. Through continuously removing these edges, they can find the community structure from the original graph.

The method proposed by Girvan and Newman is focused on edge betweenness, which is generalized from the concept by Freeman [41][42]. Freeman constructed a concept of vertex betweenness, which means the number of shortest paths between all vertex pairs that run along that vertex. It can measure the influence of the vertex to other vertices in the communities. They generalized this idea from vertex to edge and proposed the edge betweenness to measure the edge most "between" the communities [42]. So the edge betweenness of an edge is the number of shortest paths between all vertex pairs that run along this edge (Figure 2.7). It is intuitive that the edges between communities will have a large number of edge betweenness, because all the shortest paths between communities must run along these few edges. Therefore, the community structure can be found by removing these edges.

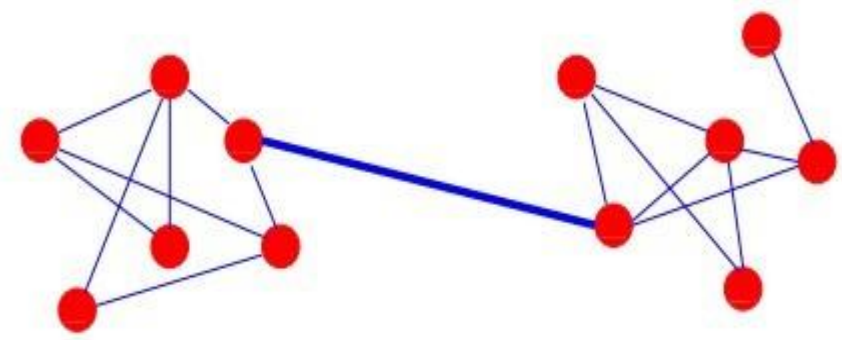


Figure 2.7: The edge between communities has the highest edge betweenness. In this figure, the edge with bold blue color is has the highest edge betweenness in the network. Two communities will be found by removing this edge. Source: Ref. [23].

The steps of this algorithm are in the following [42]:
1. Compute all the edge betweennesses in the network
2. Remove the edge with the highest edge betweenness
3. Recalculate all the edge betweennesses in the network after the removal of the highest one
4. Repeat previous two steps until there are not edges remaining between communities

As shown by their research, this algorithm can successfully detect network community structure. Although the step 3 in the algorithm is essential to detect the community, the calculation of all the edge betweennesses greatly increases the complexity [23]. As author noted, the algorithm run with complexity of $O(n^3)$ in sparse networks, where n is the number of vertices in the network. In addition, there is not standard benchmark to judge which partition they detected is the best one. Later they proposed a popular benchmark called modularity and it was widely used to judge the goodness of the partition. In general, the partition with the largest value of modularity will be regarded as the best partition. Although the method of Girvan Newman has been widely used in practical application, it is still impractical for large networks because of this complexity. Thus many other modifications of the Girvan-Newman algorithms are proposed to improve the speed of the algorithm. Tyler et al [43] proposed a similar algorithm with the same step of algorithm of Girvan-Newman, but it can improve the speed of calculation substantially. Instead of calculating all the edge betweennesses, they selected a limited number of vertices which is chosen at random. They applied it to a network of email and achieve a great result [23] [43]. In order to improve the speed of community identification, Radicchi et al [48] proposed another algorithm by iteratively removing edges between communities similar as the algorithm of Girvan and Newman. Instead of defining edge betweenness, they calculate the short loops of edges (for example loops of triangles) in the network and defined edge clustering coefficient C_{ij} (i,j are two vertices). As Radicchi et al [48] showed that value C_{ij} of edges between communities will be small and this value is quite negatively correlated with edge betweenness proposed by Girvan and Newman. In their algorithm, the edges with small value C_{ij} are removed and then all the remaining edges are recalculated. Because the measure in this algorithm is a local measure, it can be calculated quickly and the whole algorithm runs faster with complexity $O(m^4/n^2)$ on a graph (m edges and n vertices).

Modularity optimization algorithm is another popular and promising community detection algorithm. Modularity proposed by Girvan-Newman is a kind of quality function as benchmark for measuring the goodness of partitions. The partition with the largest value of modularity should be the best partition, or at least a very good one. Unfortunately, it has been proved that the modularity optimization is a NP-hard problem [24], so it is impossible to optimize the modularity directly. Many algorithms are proposed to find the approximation of modularity maximum in a reasonable time. Some of these are widely used, such as greedy method, simulated annealing, extremal optimization and genetic algorithm [23].

The first popular algorithm based on modularity optimization (Q function) is proposed by Newman [49]. In general, it is an agglomerative hierarchical greedy method. Given a network with n vertices, one starts from n communities with each community containing a single vertex. The communities are joined together by

agglomerating pair of edges, the edges are selected by calculating in the greatest increase in Q (ΔQ) at each iteration step. The result of the algorithm can be represented as a "dendrogram" which displays the order of merging (figure 2.8 is an example of "dendrogram" of communities in karate club network). Since ΔQ can be calculated in constant time, each step of this algorithm will take worst time O(n+m).There are a maximum of n-1 join operations to construct the "dendorogram", so the worst time is nearly $O(n^2)$ on a sparse graph (n is the number of vertices and m is the number of community). This algorithm runs much faster than GN algorithm and can be extended to more complex networks.

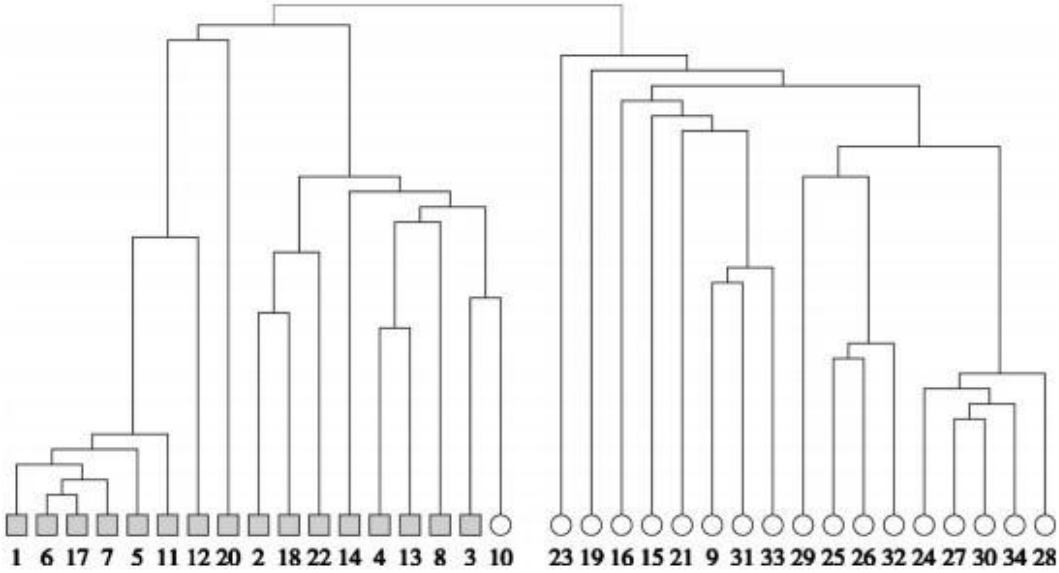


Figure 2.8: The "dendrogram" of communities in karate club network of Zachary by Newman's algorithm. Source: Ref. [49].

2.2.3 Quality function

Reliable community detection algorithms are supposed to identify good partition of networks. But what is a good clustering? In order to distinguish quality of clustering, quality function is created to evaluate the "goodness" of a partition of a network into communities.

The most popular quality function is the modularity of Newman and Girvan [19]. The definition of it has been mention before and will be shown in details in the following part. Suppose we are given a particular division of a network with g communities. Let us define a $g \times g$ matrix e whose component eij is the fraction of the edges in the original network that a vertex in community i connect with a vertex in community j. The sum of i_{th} row $a_i = \sum_j e_{ij}$ indicates all the fraction of edges connected to community i. The modularity Q is defined in the following [19].

$$Q=\sum_i\left(e_{ii}-a_i^2\right)=\mathrm{Tr}(e)-||e^2|| \quad (1)$$

where ||x|| indicates the sum of components in the matrix x. The modularity measures the fraction of the edges that fall within the given groups minus the expected such fraction if edges were divided at random [22]. Thus, we will get Q=0 if the number of within-community edges is no better than random. With the maximum value Q=1 if the division of community are strong community structure. As is reported in [19], the value of Q will nearly from 0.3 to 0.7 in the real world network. Modularity optimization seems to be an effective way to qualify the community detection in both real and artificially generated networks. Based on the modularity, several algorithms are created to maximize the modularity function. However, optimization of modularity is a challenging task, and it has been proved that finding the optimal Q value is an NP-hard problem [24]. In addition, in the survey of Fortunato et al [25], modularity function is useful to compare partition of communities with the same number of modules, whereas the comparison of partition of communities with different number of modules is not straightforward and may result in confusion [25]. The most important of all, the modularity has exposed the resolution limits [23] [26].

Another quality function (D function) called modularity density was proposed by Li et al [26]. It is a quality measure for evaluating the partition of a network based on average modularity degree. Given a network G= (V, E), V denotes the vertices and E denotes the edges. If V_1 and V_2 are two disjoint partitions of a network, A is similarity matrix of this network. So L (V_1, V_2) = $\sum_{i\in V_1, j\in V_2} A_{ij}$. Given a network with n partitions $V_1, V_2 \ldots V_n$. The equation in (1) can be rewritten in the following: Source: Ref. [26].

$$Q=\sum_{i=1}^{n}\left[\frac{L(V_i,V_i)}{L(V,V)}-\left(\frac{L(V_i,V)}{L(V,V)}\right)^2\right] \quad (2)$$

The modularity density function is the average modularity degree and can be written as follows:

$$d(V_i)=d_{in}(V_i)-d_{out}(V_i) \quad (3)$$

where $d_{in}(G_i)$ denotes the average degree within the partition V_i, $d_{out}(G_i)$ denotes the average outer degree of the partition V_i [26]. Thus the above equation can be written as:

$$d(V_i)=\frac{L(V_i,V_i)-L(V_i,\bar{V}_i)}{|V_i|} \quad (4)$$

where $\bar{V}_i$ is $V-V_i$ and $|V_i|$ is the number of vertices in V_i. Thus the D function is as follows:

$$D=\sum_{i=1}^{n} d(V_i) = \sum_{i=1}^{n} \frac{L(V_i,V_i)- L(V_i,\bar{V}_i)}{|V_i|} \qquad (5)$$

They noted that the partitions will be best with the largest value of D, so the community detection can be regarded as a problem of D function optimization. Unfortunately the optimization of D function is also a NP-hard problem, so they proved the equivalence of kernel k-means and optimization of D function. Then kernel k-means can be used as a method to optimize the D function. In addition, the modularity density can overcome the resolution limit which is the main problem of modularity [26]. To prove the reliability of modularity density, Li et al [26] discussed it from the following steps.

1. A clique will not be separated into two parts with modularity density.
A clique is a community in which every vertex is connected by edge. Given a clique with n vertices, the optimization of modularity density will not separate the clique into two parts.

Li et al [26] proved it by contradiction. If p is a clique divided into two cliques G_1 and G_2 with the number of nodes are n_1 and n_2 respectively, so the number of edges between G_1 and G_2 is $n_1 \times n_2$. Let D_0 be the modularity density of G and D_1 be the modularity density of p, Source: Ref. [26].

$$D_0 = n - 1,$$

$$D_1 = \frac{n_1(n_1-1)-n_1n_2}{n_1} + \frac{n_2(n_2-1)-n_1n_2}{n_2} = -2$$

Since $D_0 > D_1$, a clique will not be divided into two parts by optimization of modularity density.

2. Most modular networks can be solved correctly by modularity density
Li et al [25] followed the way of Fortunato et al [23] to test the quality of modularity density using the schematic example which is a network composed of a ring of cliques connected by single links (Figure 7). In this network, each clique is a complete small network K_n with n nodes ($n \geq 3$) and $n(n-1)/2$ edges. Suppose that there are m cliques in a ring, so the network has mn nodes and $mn(n-1)/2 + m$ edges (figure 2.9).

This network has a clear community structure with m communities, but the method of optimizing modularity cannot get the correct result. Li et al [26] proved that the approach of optimizing modularity density can effectively detect the community structure of this network. The modularity density of partition which can detect m communities is calculated as: Source: Ref. [26].

$$D_m = m\frac{n(n-1)-2}{n} = m(n-1-\frac{2}{n})$$

The modularity of the partition in which every k cliques merge to be a single community is:.

$$D_k = \frac{m}{k} \times \frac{kn(n-1)+2(k-3)}{kn}$$

Where $k \geq 2, n \geq 3,$ and $m \geq 2$.

$$\begin{aligned} D_m - D_k &= m\left(n-1-\frac{2}{n}\right) - \frac{m}{k} \times \frac{kn(n-1)+2(k-3)}{kn} \\ &= m\left[(n-1) - \frac{2}{n} - \frac{n-1}{k} - \frac{2(k-3)}{k^2 n}\right] \\ &> m\left[(n-1) - \frac{2}{n} - \frac{n-1}{k} - \frac{2}{kn}\right] \\ &\geq m\left[(n-1) - \frac{2}{n} - \frac{n-1}{2} - \frac{1}{n}\right] \\ &= m\left[\frac{n-1}{2} - \frac{3}{n}\right] > 0 \end{aligned}$$

Therefore, it has been proved that modularity density can resolve this network correctly. Although this is a special network, Li et al [26] noted that this result is also valid for any other similar problems. Along with the first prove that modularity density will not divide the clique into two communities, is has been proved that optimization of modularity density can correctly detect the community detection (with each single clique as a community) [26].

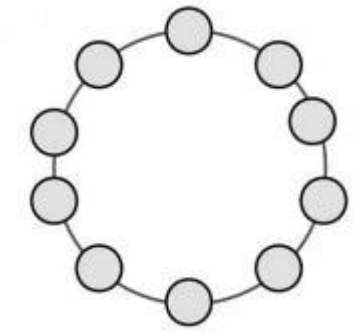


Figure 2.9: A ring of cliques with n vertices in each clique. Source: Ref. [26].

3. Modularity density can detect communities with different sizes

In [23], Fortunato et al have proposed that modularity optimization may merge small

modules. Then Li et at [26] prove that optimization of Modularity Density does not have such a problem.

Assuming that there is a network composed of four cliques, two cliques are represented by K_p and other two cliques are K_n, for $n \geq p \geq 3$(Figure 2.10). The modularity density of the partition in which all the cliques are separated as different communities are: Source: Ref. [26].

$$D_{seperate} = \frac{n(n-1)-1}{n} + \frac{n(n-1)-3}{n} + 2\frac{p(p-1)-2}{p}$$

$$= \frac{n(n-1)-1}{n} + \frac{n(n-1)-3}{n} + 2(p-1) - \frac{4}{p}$$

The modularity of the partition in which the two K_p cliques are merged together are: Source: Ref. [26].

$$D_{merge} = \frac{n(n-1)-1}{n} + \frac{n(n-1)-3}{n} + \frac{2p(p-1)}{2p}$$

$$= \frac{n(n-1)-1}{n} + \frac{n(n-1)-3}{n} + (p-1)$$

when $p \leq 3$, it can be verified that:

$$D_{seperate} - D_{merge} = (p-1) - \frac{4}{p} > 0$$

Therefore, combining with two above proves, Li et al [26] proved that optimization of modularity density can correctly detect the community structure. In addition, there is no resolution limit problems compared with optimization of modularity algorithms.

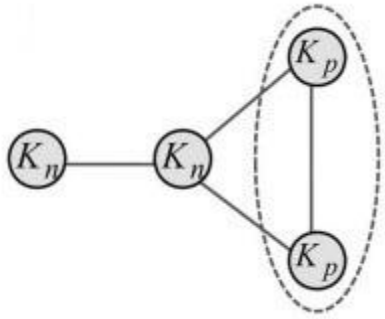


Figure 2.10: A network consisting of two pairs of identical cliques, in which one pair of cliques have n nodes and the other pair of cliques has p nodes. Source: Ref. [26].

2.2.4 Challenges of community detection

There are three main challenges for community detection.

1. Precise definition for community: The precise definition of community is the first step for community detection, however no such precise definition exists yet. Each algorithm makes different assumptions consistent with intuitive concept of a community [17]. The majority of algorithms assume that a network consisting of disjoint communities and this makes sense in many networks. Many previous works [18] revealed that complex network models displayed an overlapping community structure, which is also called fuzzy community. Other algorithms proved the existence of the hierarchical organization of modularity in networks. These complicated community structure make it more difficult to cluster communities.

2. Measure of community: There is no consensus to measure the efficiency of the division of community, which is a main shortcoming in many community detection algorithms. In 2004, Newman et al [19] introduced quality function Q which can measure the strength of a given partition of a network and it can also be used to select an optimal numbers of communities. Based on this feature, many algorithms are proposed to optimize this Q function, which is proved an NP-hard problem although.

3. Topology structure of complex networks [21]: Although there are various similarity measures presented [13][14][15], measurement of network topology structure is difficult. Several similarity measures, such as kernel similarity, shortest path based similarity, neighborhood based similarity are well studied. Each community detection algorithm will make different similarity matrix (or dissimilarity matrix) to change the clustering algorithm to community detection. Therefore, it is significant to select a suitable similarity measure (dissimilarity measure) to detect community in networks.

2.3 Instance-level constraint

Constraint-based community detection (network clustering) is an innovation recently in machine learning and data mining. In traditional, clustering is a method of unsupervised learning and a common technique in data analysis. However, in this project, constraints which are something regarded as priori knowledge will be incorporated into community detection (network clustering) algorithms, therefore constraint-based network clustering is a semi-supervised learning method. Generally, there are two directions to combine the constraints with network clustering algorithms. The first one is to add some unlabelled data as a kind of priori knowledge in initialization phase, and the second one is to add some constraints (supervision) in the process of clustering. Some people have done a deep research in the former one [1]. However, much attention have been focused on the area of adding some constraints (supervision) in the process of clustering [2][3][4][5].

2.3.1 Types of constraints

Two types of basic instance-level constraints are first present by Wagstaff: must-link (pairs of points that should belong in the same cluster) and cannot-link (pairs of points that should belong in different clusters) [2]. Although Must-link and cannot-link constraints are simple, they are very important to network clustering. ML(x,y) means x and y are must link in the same cluster. CL(x,y) means x,y are cannot-link in the same cluster. There are some important features about must-link and cannot-link [6].

The transitivity of must-link constraints [7]: 1) If ML(a,b) and ML(a,c), then ML(b,c) is true. Therefore it could also be regarded that three vertices are grouped into a clique. 2) There are two sets of vertices X and Y, with the vertices of which are completely must-link. If there are vertices $a \in X, b \in Y$ and ML(a,b), so any vertice $x1 \in X, yi \in Y$, ML(x1,y1) is true.

The entailment of cannot-link constraints [7]: 1) If ML(a,b), ML(c,d), and CL(a,c), so CL(a,d),CL(b,c) and CL(b,d) are true. 2) There are two sets of vertices X and Y, with the vertices of which are completely must-link. If there are vertices $a \in X$, $b \in Y$ and CL(a,b), so any vertice $x1 \in X, yi \in Y$,CL(x1,y1) is true.

From the research of Davison [6], there are other two spatial constraints: such as δ-constraints and ε-constraints shown in figure 2.11. δ-constraints means the distance of two separate clusters are greater thanδ, so it is equal to the combination of must-link constraints and δ-distance. ε-constraint means that the diameter of cluster is more than ε-distance, so it is equal to the combination of cannot-link constraints and ε-distance.

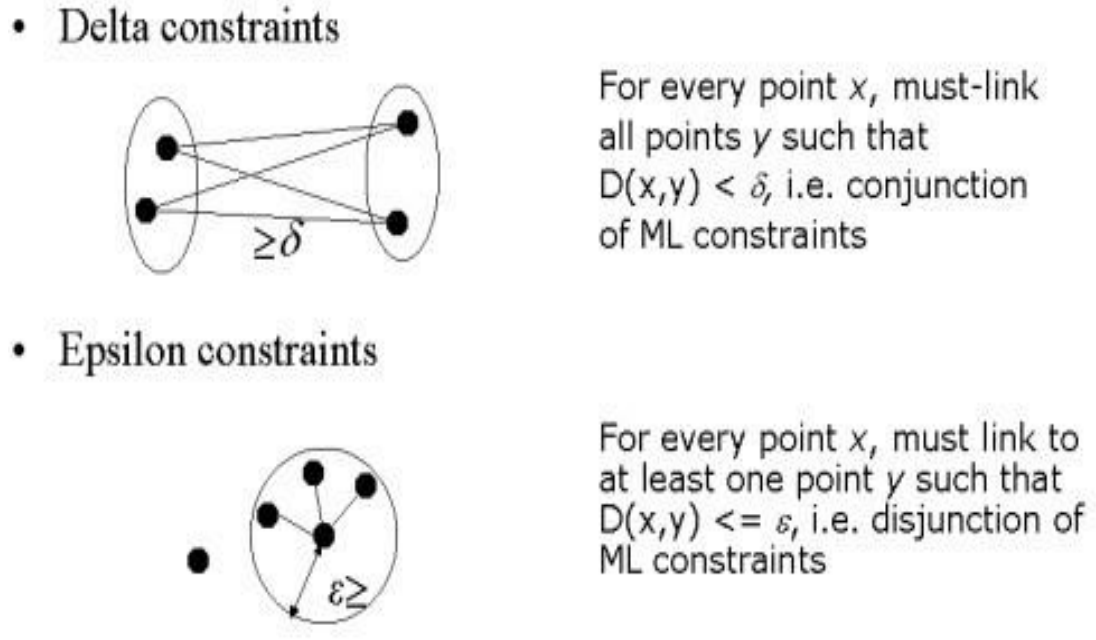


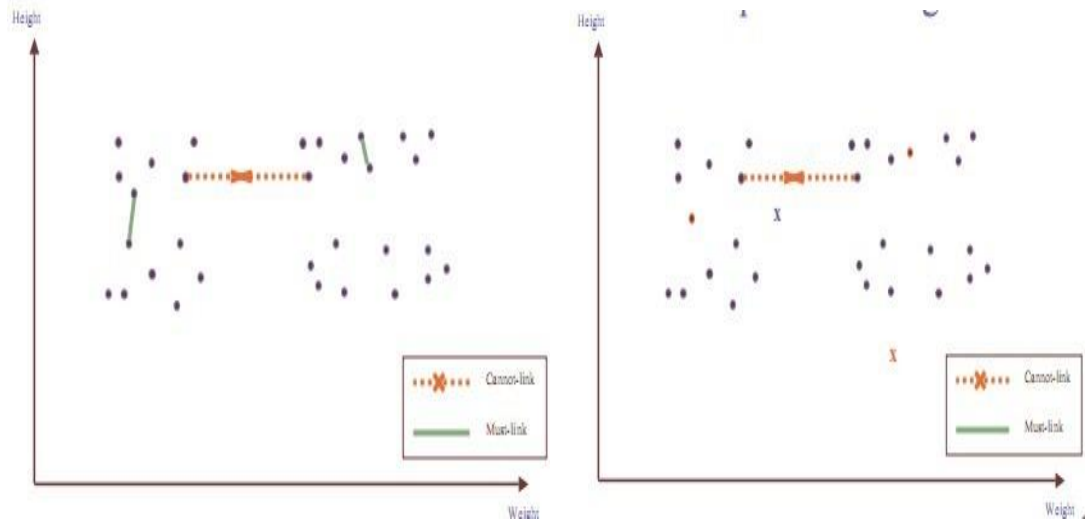
Figure 2.11: δ-constraints andε-constraints. Source: Ref. [7].

2.3.2 Utilization of constraints

The typical clustering methods are unsupervised learning, which is very useful in data clustering with no other background information. However in many domains, some constraints are very important as some background knowledge, they can greatly improve the speed and efficiency of clustering. For instance, in a user interactive system, the feedback information from users with the type of must-link and cannot-link constraints is very important and effective for system to optimize the choice for users.

In general, constraints can be used in two types. The first type is to incorporate constraints in the initialization phase, where the initial clusters are formulated by must-link and cannot-link constraints. The second type is that constraints are incorporated in the process of clustering, such as change the algorithm to satisfy these constraints as much as possible.

The former type of constraints is mainly used in the initialization phase. For instance, in the cop-kmeans clustering algorithms [6], must-link and cannot-link constraints should be incorporated in the first stage. If there is ML(a,b), a and b (vertex) should merge together to c, which is the average point of a and b. This kind of constraint is shown in figure 2.12.



(a) (b)

Figure 2.12: (a) is the initial constraints (b) describe the combination of must-link constraints. Source: Ref. [6].

The second kind of constraint is used to modify the clustering algorithms so that it can bias the search in the process of clustering. The pairwise constraints specify which instances are in the same clusters (must-link) and which in the different clusters (cannot-link). Much research has been down in this area, one of them is to add a penalty function to the objective function, so it will make the algorithms satisfy the constraints [4] [8].

2.3.3 Challenges of constraints

Although constraints are important as a kind of background knowledge, some limitation and challenges need to be discussed. The challenges of constraints are feasibility and complexity, which are totally different based on the available clustering algorithms we choose.

1. Feasibility
In this part, the problem of feasibility will be discussed in this part. Since the constraints are must-link and cannot-link, so the feasibility of constraints will be discussed individually.

Feasibility under must-link constraints [7]: From the research of Klein [9] and Davison[7], ML-constraints problem can be solved in polynomial time. From the research of Klein, they modified a feasibility of must-link to satisfy the triangle inequality in a new distance function. However, in the research of Davison, he successfully utilized the transitive feature of must-link constraints to solve the feasibility problem of must-link. Figure 3 will show their research.
Feasibility under cannot-link constraints [7]: Both Klein and Davison show the

conclusion of feasibility of CL-constraints problem. In Davison's research, he used a straightforward reduction from the Graph K-Colorability problem (K-COLOR) to prove the feasibility. Table 2.13 will show their research.

Table 2.13: Results for feasibility of non-hierarchical clustering (Given K) and hierarchical clustering (unspecified K). Source: Ref. [7].

Constraint	Given K	Unspecified K
Must-link	P	P
Cannot-link	NP-complete	P

From the research of Davidson [7], they conclude that 1)Since the CL constraints (NP-complete) make the feasibility of problem intractable, hence the combination of CL constraints are intractable. Therefore, it is difficult to find available clustering algorithms to satisfy all the constraints. For a simple example, there are constraints ML(x,y) and CL(x,y), regardless of K, it is impossible to find clusters which satisfy both of these constraints.

In hierarchical clustering, the number of clusters (K) is not specified, so the feasibility problem is significantly different from non-hierarchical clustering. From the research of Davidson [7], all the must-link and cannot-link constraints can find a feasible solution in hierarchical clustering. However, another issue should be raised: Dead-ends. In the worse case, the traditional hierarchical clustering algorithms will raise premature dead-ends, which will make the dendrogram reach a stage where no pairs of clusters can be merged without destroying one or more constraints [7].

2. Not all constraints are useful

The assumption made in constraint based clustering algorithms is that these constraints can give the algorithm some desirable background information. But the research from Davidson et al [7] show that it is still possible for constraints sets to decrease clustering accuracy even if the constraints are got from the background truth. A survey from Davidson et al [7] is shown in table 2.14.

Table 2.14: Average performance of four different constraint based clustering algorithm. Source: Ref. [7].

	Algorithm							
	CKM		PKM		MKM		MPKM	
	None	Const	None	Const	None	Const	None	Const
Glass	69.0	69.4	43.4	68.8	39.5	56.6	39.5	67.8
Ionosphere	58.6	58.7	58.8	58.9	58.9	58.9	58.9	58.9
Iris	84.7	87.8	84.3	88.3	88.0	93.6	88.0	91.8
Wine	70.2	70.9	71.7	72.0	93.3	91.3	93.3	90.6

2.4 The Feasibility of kernel k-means

Community detection (graph partitioning) has become increasingly popular recently for representing a wide variety of systems in real world. Several algorithms including spectral clustering, optimized-based clustering, have recently been proposed to detect community structure successfully. In this part kernel k-means algorithm will be shown to be effective to detect community detection through the equivalence with these algorithms such as spectral and optimize-based clustering. Therefore, kernel k-means algorithm is selected to incorporate constraints in this project.

2.4.1 Spectral clustering

Spectral clustering algorithms use information contained in the eigenvectors of affinity matrix (in network clustering, it is similarity matrix). Spectral clustering algorithms have been proved to be effective on many tasks, including web search and image segmentation [3]. The basic steps of spectral graph clustering will be described below and figure 3.1 show several types of normalizations in spectral clustering.

Form spectral representation [3]:

1. Given a graph G=(V,E), form the affinity matrix(similarity matrix) $A \in R^{n\times n}$;
2. Define D to be the diagonal matrix with $D_{ii} = \sum_j A_{ij}$, and compute the Laplacians matrix L which can be seen in table 2.15.
3. There are k groups (k is specified in advance). Find $x_1, x_2, x_{3\ldots\ldots} x_k$, the k largest eigenvectors of L, which will be chosen to orthogonal. Then form the matrix $X=[x_1\ x_2 x_{3\ldots} x_k] \in R^{n\times k}$ by putting the eigenvectors in the columns.
4. From the matrix Y from X by normalizing the rows of X to be unit length. (i.e. $Y_{ij} = X_{ij}/(\sum_j X_{ij}^2)^{1/2}$)

For clustering

5. Cluster each row of X (as a point in R^k) into k clusters using k-means or any other sensible clustering algorithm.
6. Finally, assign the original point x_i to cluster j if and only if row i of X was assigned to cluster j.

Table 2.15: All types of normalizations are used by spectral clustering. Source: Ref. [8]

Normalization	L(A,D)
None	L=A
Divisive	$L=D^{-1}A$
Symmetric Divisive	$L=D^{-1/2}AD^{-1/2}$
Normalized Additive	$L=(A+d_{max}I - D)/d_{max}$

2.4.2 Optimized-based network clustering

In optimization based Network clustering (community detection), the goal is to optimize the objective function [8]. Let us assume a graph G=(V,E,A), where V is the set of vertices, E is the set of all the edges of the connected vertices, A is the affinity matrix(it is similarity matrix in community detection), where Aij is the similarity value of vertex i and vertex j. $\text{link}(X,Y)=\sum_{i\in X,j\in Y} A_{ij}$ and degree(X)=link(X,V). The method aims to find a k disjoint partitioning $\{V_c\}_{c=1}^{k}$ to optimize the objective function in Table 2.16.

Table 2.16: Several graph clustering objectives. Source: Ref. [8].

Name	Objective Function
Ratio Association	$\text{maximize} \sum_{c=1}^{k} \frac{\text{link}(V_c,V_c)}{\lvert V_c\rvert}$
Ratio Cut	$\text{minimize} \sum_{c=1}^{k} \frac{\text{link}(V_c,V/V_c)}{\lvert V_c\rvert}$
Normalized Cut	$\text{minimize} \sum_{c=1}^{k} \frac{\text{link}(V_c,V/V_c)}{\text{degree}(V_c)}$

The objective functions in the table above can be generalized with the utilization of general weighted variants. Thus the objective functions can be reformulated as below. Source: Ref. [8]:

$$\text{maximize} \sum_{c=1}^{k} \frac{\text{link}(V_c,V_c)}{w(V_c)} \quad \text{or} \quad \text{maximize} \sum_{c=1}^{k} \frac{\text{link}(V_c,V/V_c)}{w(V_c)}$$

where $w(V_c)=\sum_{i\in V_c} w_i$ (w_i is the weight of vertex i).

2.4.3 Kernel k-means

Kernel k-means is used to map the data to a higher dimensional feature space using a nonlinear function, then using the k-means to divide the linear separator in the new space [8]. Given a set of vectors $X=\{x_i\}_{i=1}^{n}$ (x_i represents the vector i), weighted kernel k-means aims to find k disjoint partitioning $\{v_c\}_{c=1}^{k}$ of data (v_c represents the cluster c), so the objective function is in the following. Source: Ref. [8]:

$$D(\{v_c\}_{c=1}^{k})=\sum_{c=1}^{k} \sum_{x_i\in v_c} w_i ||\varphi(x_i) - m_c|| \qquad (6)$$

Where

$$m_c = \frac{\sum_{x_i \in v_c} w_i \varphi(x_i)}{\sum_{x_i \in v_c} w_i} \quad (7)$$

w_i is the non-negative weight of vector I, and $\varphi(x_i)$ is a non-linear function mapping the vector i to higher-dimension space. If all the weights are set to be 1 and φ is the plain function, then this equation is a standard k-means algorithm. If put (7) into (6), we can get the following equation.

$$\left|\left|\varphi(x_i) - m_c\right|\right| = \varphi(x_i) \cdot \varphi(x_i) - \frac{2\sum_{x_j \in v_c} w_i \cdot \varphi(x_i) \cdot \varphi(x_j)}{\sum_{x_j \in v_c} w_i} + \frac{\sum_{x_j, x_l \in v_c} w_j \cdot w_l \varphi(x_j) \cdot \varphi(x_l)}{\left(\sum_{x_j \in v_c} w_j\right)^2} \quad (8)$$

As we can see in the equation, the above $\varphi(x_i) \cdot \varphi(x_j)$ can be computed in advance and we call it $k_{ij} = \varphi(x_i) \cdot \varphi(x_j)$. Thus, the distance function in the higher-dimension space can be computed through the kernel matrix with the its component k_{ij}. Using the kernel matrix, the equation (8) can be rewritten as below.

$$k_{ii} - \frac{2\sum_{x_j \in v_c} w_i \cdot k_{ij}}{\sum_{x_j \in v_c} w_i} + \frac{\sum_{x_j, x_l \in v_c} w_j \cdot w_l \cdot k_{jl}}{\left(\sum_{x_j \in v_c} w_j\right)^2} \quad (9)$$

2.4.4 Equivalence between kernel k-means, spectral and optimized clustering

To prove the equivalence of kernel k-means, optimization based network clustering and spectral clustering, Dhillon [30] et al transformed them individually to a trace maximization problem.

For the weighted kernel k-means as trace maximization, the minimization of the objective function equals to. Source: Ref. [30]

$$\max \text{trace}(Y^T W^{-1/2} K W^{-1/2} Y) \quad (10)$$

where Y is an orthonormal n×k matrix that is proportional to the square root of the weight matrix W.

For the optimization based graph clustering, the maximization of the graph partitioning objectives can also be written as trace maximization, through some transformation, the objective function equals to.

$$\max \text{trace}(Y^T W^{-1/2} A W^{-1/2} Y) \quad (11)$$

Therefore, this equation (11) is exactly the same with (10).

2.5 Summary

This section has reviewed some traditional and recent popular community detection approaches as well as some instance-level constraints. In addition, some quality functions for measuring the partitions of network are also introduced. What's more, kernel k-means algorithm is proved to be effective for community detection through the equivalence with other popular algorithms such as spectral and optimized algorithms. Therefore, kernel k-means is selected to incorporate these constraints in this project.

The next section will describe the implementation of my project in details including the problem specification and proposed solution.

3: Implementation

In this part, the steps of implementation in this project will be illustrated in details and some problems in the project will be analyzed and specified.

As mention in the above section, some other forms of constraints can be transformed into must-link and cannot-link constraints, so the constraints we assumed here are two categories: must-link and cannot-link. In addition, constraints can be used in two types within clustering as discussed in the above section. In the first type, constraints are incorporated in the initialization phase, where the initial clusters are formulated by must-link and cannot-link constraints. The second type is to incorporate constraints in the process of clustering, such as change the algorithm to satisfy these constraints. In this project, these constraints are mainly used in the second type.

In 2004, Basu [4] et al proposed a probabilistic framework for constraint-based clustering. Although this method can not be directly used in community detection, it proposed a direction of how to incorporate the constraint to the clustering. Recently, Dhillon [8] et al proposed a new semi-supervised network clustering (community detection) through kernel k-means method and they also proved the equivalence of optimization based network clustering and kernel k-means method. This method is mainly based on the method of Basu [4] to incorporate constraints in the objective function of clustering. Therefore, they declared a new constraint-based kernel k-means clustering method which was proved to be efficient in network clustering. However, their ways of using constraints are not well justified too. They only change the similarity to be 0 with cannot-link constraints vertices in the input similarity matrix, which does not really mean that the two vertices will tend to belong to different clusters.

In this project, these constraints are incorporated to construct a special kernel matrix. After that this new kernel will be used to cluster the high dimension space data using k-means algorithm.

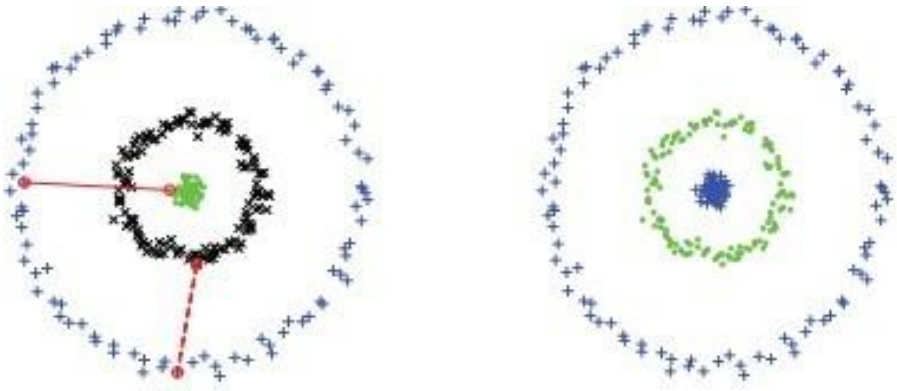
3.1 Problem specification

As mentioned before, constraints will be incorporated in the process of clustering, such as change the algorithm to satisfy these constraints. Therefore, the most challenging problem of this project is how to incorporate constraints to get a desirable kernel matrix.

Typically, it can be divided into two main components. The first one is how to the general framework of this kernel matrix, and second is how to find the kernel matrix that has certain desirable properties.

The general framework is to find a mapping that will map the similarity data to a high dimension space, where two must-link nodes are mapped to be close enough and two cannot-link nodes are mapped to far apart [33]. Then in the new kernel, not only the constraints are satisfied, but also those vertices similar to the must-link vertices in the input space will be classified into the same cluster and those similar to the cannot-link vertices will be classified into the different clusters [33].

To better illustrate the motivation of this project, let us see how to cluster these data in figure 3.3. Although the input data are not vertices, it can demonstrate the same meaning. As can be seen in the following figures, there are three separate classes in figure 3.1(a) and two classes in figure 3.1(b) (with different colors) [33]. The outer circle merge with inner circle to be one class because of the must-link constraint, not just merge the two must-link nodes. Similarly, the middle circle and outer circle will be in different classes due to the cannot-link constraint, not just the two cannot-link nodes. Therefore, the desirable output is shown in figure b. The detail of how to optimize and get this kernel will be illustrated in the next part.



(a) (b)

Figure 3.1: (a) a data network with one must-link constraint (solid red line) and one cannot-link constraint (dashed red line). (b) Final kernel network incorporating the constraints. Source: Ref. [33].

3.2 General steps of implementation

3.2.1 Construct similarity matrix

Because most of clustering algorithms aims to deal with vector based data, and can't be directly adopted on graph based data. So it's crucial to find appropriate methods for transforming community detection to a clustering problem.

Vertex similarity is an important concept in network analysis and data mining [11]. Take social network for example, two people might be considered similar if they have similar hobbies, professions or same friends.

In a network structure, the vertices have high similarity within the community and have sparse similarity between communities. Therefore the first step of community detection is to form the similarity matrix with the relation of edges. In a general community detection problem, similarity are measured based on the structure of original graph, which is sometimes called structural similarity [12]. In this part several similarity measure are introduced and the one with best performance is used in this project.

1. Neighbor based similarity
In previous, the most common approach has been to focus on structural equivalence. In networks, two vertices are regarded to be similar if most of their neighbors are same [11]. Let Ni be all the neighborhood vertices of vertex i in a network, Nj be all the neighborhood vertices of vertex j in a network. So the number of common friends of i and j is:

$$Snaive(i,j)=|Ni \cap Nj|$$

where |x| denotes the number of vertices in set x. Snaive can be one measure of similarity. However, this similarity is not satisfactory for it may take values for vertices with high degree even if they share small number of neighbors. There are at least three similarity metrics are proposed in previous work. Source: Ref. [13-15].

$$Sjacccard(i,j)=\frac{|Ni \cap Nj|}{|Ni \cup Nj|}$$

$$Scosine(i,j)=\frac{|Ni \cap Nj|}{\sqrt{|Ni||Nj|}}$$

$$Smin(i,j)=\frac{|Ni \cap Nj|}{\min(|Ni|,|Nj|)}$$

All these three metrics could get the similar value with the maximum s=1 when

Ni = Nj. However, these similarity metrics do not take the connectivity of vertices into account, which is a very important condition in network. It is apparent that two connected vertices are more similar compared with two vertices with no connection.

2. Shortest path based similarity

The shortest path has been comprehensively used in many fields as a feature matrix. Gustafsson et al [34] formulate a distance matrix based on the shortest path and define the Euclidian distance measure. Let define Pij is the shortest path between vertices vi and vj, then the distance matrix is $P=[Pij]\,n \times n$, With this matrix, the distance between vertices vi and vj is measured as

$$dij=\sqrt{\sum_{k=1}^{n}(P_{ik}-P_{jk})^2}$$

So the similarity matrix can be defined as

$$S(vi\,,\,vj)=\frac{1}{dij}$$

Here we propose another popular shortest based similarity measure.

$$S(vi\,,\,vj)=\frac{1}{P_{ij}^{k}}$$

Where k is a constant and can be used to control the role that the shortest paths play in the similarity matrix. For i=j, we set the $P_{ij}^{k}=1$. When $k\rightarrow\infty$, $S(vi\,,\,vj)\rightarrow A$,where A is adjacency matrix, so this can be also regarded as an adjacency matrix based similarity measure. In the project, we set the k=1.

3. Edge betweenness based similarity

Edge betweenness is another popular similarity measure, which is defined as the number of shortest paths between pairs of nodes that run along it in the graph. If a network contains loose communities that are only connected by a few inter-group edges, then shortest paths between these communities must go along one of these edges. Thus, inter-group edges will have high edge betweenness. Let define bij is the shortest path between vertices vi and vj, then the distance matrix is $b=[bij]\,n \times n$. So the similarity metrics is. Source: Ref. [12].

$$S(vi\,,\,vj)=\frac{1}{bij}$$

After the test of all similarity above, the similarity base on shortest path are selected to compute the similarity matrix.

3.2.2 Construct kernel and incorporate constraints

1. Construct the kernel matrix

As mentioned before, the general framework is to find a mapping that will map the similarity data to a high dimension space, where two must-link nodes are mapped to be close enough and two cannot-link nodes are mapped to be far apart [33]. The most important of all, in the new kernel, not only the constraints are satisfied, but also those vertices similar to the must-link vertices in the input space will be classified into the same cluster and those similar to the cannot-link vertices will be classified into the different clusters.

Let $\emptyset$ be a mapping from X (similarity space) to F (high dimension),

$$x_i \in X \rightarrow \emptyset(x_i) \epsilon F$$

The above analysis can be regarded as an optimization framework: Source: Ref. [33].

$$\min_{\emptyset} S(\emptyset) \tag{1}$$

$$s.t||\emptyset(x_i) - \emptyset(x_j)||_F < \varepsilon, \varphi \ \forall(x_i, x_j) \in M, \tag{2}$$

$$||\emptyset(x_i) - \emptyset(x_j)||_F < \beta, \varphi \forall(x_i, x_j) \in C, \tag{3}$$

where ε is a small positive number, and β is a relatively big positive number. M is defined as must-link constraints and C is cannot-link constraints. The inequalities in (2) and (3) make the must-link vertices close enough and cannot-link vertices far apart. The minimum value of $S(\emptyset)$ in (1) make the vertices similar to two must-link vertices close and vertices similar to two cannot-link vertices far enough.

Li et al [33] proposed a method to map the similarity data to a unit hypersphere with the utilization of constraints. The general idea is to map the two must-link vertices to the same point and two cannot-link vertices to be orthogonal in this unit hypersphere. So they can get the following equations.

$$< \emptyset(x_i), \emptyset(x_i) >_F = 1, i = 1,2 \ldots\ldots n$$

$$< \emptyset(x_i), \emptyset(x_j) >_F = 1, \forall(x_i, x_j) \in M$$

$$< \emptyset(x_i), \emptyset(x_j) >_F = 0, \forall(x_i, x_j) \in C$$

Where $< \cdot, \cdot >$ is defined as the dot product in this high dimension space.

Through some mathematically transformation, the above equations can be rewritten in the following optimization problem.

$$\min_K: \quad L \cdot K \tag{4}$$
$$Kii=1, i=1,2\ldots n$$

$$Kij=1, \forall (x_i, x_j) \in M \tag{5}$$

$$Kij=0, \ \forall (x_i, x_j) \in C \tag{6}$$

2.Optimize the kernel matrix through incorporating constratints
Because the complexity of the kernel is $n \times n$ (n is the number of vertices), it is rather difficult to solve this SDP problem effectively in previous part, especially a complex network with many vertices. A useful method is to use low-rank kernel approximation to reduce complexity. There are generally two ways to optimize the kernel matrix.

1). The first method is to use the spectral regularization to optimize the kernel.
The general idea is to factorize the kernel matrix K to a much small matrix and then optimize this small matrix. For example, the kernel matrix $K=QYQ^T$, where Q and Y are of sizes $n\times m$ and $m \times m$ respectively (n is the number of vertices, m is a much small number can be set). The value of Q can be computed priori to the learning of K, so if we can compute the value of Y then we could get the value of K. The efficient choice of Q is very important in this method. Firstly, we will review some background information about spectral graph theory and some related notations.

We assume the graph is G=(V,S), it is a undirected and weighted graph. V and S denote a set of vertices and the similarity of these vertices respectively. The similarity matrix is symmetric and the value is positive from (0,1]. Let D be a $n \times n$ diagonal matrix whose entry di satisfies that$d_i = \sum_{j=1}^{n} S_{ij}$. So the graph Laplacian matrix is L=D-S. The normalized graph Laplacian of G is $\bar{L} = D^{-1/2}LD^{-1/2}$. $\bar{L}$ is a symmetric and positive semidefinite matrix. Let (λ_i, V_i)(i=1,2..n) denote the eigenvalues and associated eigenvectors of $\bar{L}$, where $0 \le \lambda_1 \le \lambda_2 .. \le \lambda_n$. Let define

$$F=(V_1, V_2 \ldots \ldots V_i)$$

F is the eigenvector matrix of $\bar{L}$. So the $Q=F_m$, where $F_m = (V_1, V_2 \ldots \ldots V_m) = (u_1, u_2 \ldots u_n)^T$,where u_i^T denotes the i row of F_m . With some mathematical manipulations, Li et al [36] turn the (3) into

$$\min_{Y \ge 0} \sum_{(i,j,t_{ij}) \in S} \left((QYQ^T)_{ij} - t_{ij}\right)^2 \tag{7}$$

Where S={ $(i, j, t_{ij}) | \text{if } i = j \text{ or } (i,j) \in M, t_{ij} = 1.$ $\text{if}(i,j) \in C, t_{ij} = 0$ }. The above problem can be rewritten in the following form.

$$\min_y y^T B y + b^T y + c \tag{8}$$

Where $y=vec(Y)\in R^{m^2}$ denotes the vector concatenating all the column of Y.

$$B = \sum_{(i,j,t_{ij})\in S} x_{ij}x_{ij}^T \ , b = -2 \sum_{(i,j,t_{ij})\in S} t_{ij}x_{ij} \tag{9}$$

$$c = \sum_{(i,j,t_{ij})\in S} t_{ij}^2 , x_{ij} = vec(u_j u_i^T) \tag{10}$$

From above equation, we can see that B, b and c can be calculated in advanced. Using the Schur Complement[37][36], the above optimization problem can be rewritten as a SDP problem and then use some software package, such as CSDP [38] to solve it. Source: Ref. [33].

$$\min_{y,v} v + b^T y \tag{11}$$

$$s.t\ Y \geq 0 \text{ and } \begin{bmatrix} I_{m^2} & B^{\frac{1}{2}}y \\ (B^{\frac{1}{2}}y)^T & v \end{bmatrix} \tag{12}$$

where $I_{m^2} \in R^{m^2\times m^2}$ is a identity matrix and v is a slack variable. The second constraint equals to $(B^{\frac{1}{2}}y)^T(B^{\frac{1}{2}}y)^T \leq v$ [35][36].

2). The second method is to transform the SDP problem to SQLP problem
The previous method can get the kernel matrix via solving the Semidefinite Programming problem (SDP), but the constraints in (18) is of size $(m^2+1)\times(m^2+1)$. So such a SDP is not efficient enough. As noted by author Li et al [36], m is set to be a small value.

The second method proposed by author Wu et al[35], is to transform a SDP to a SQLP problem (Semidefinite-Quadratic-Linear-Programming). As mentioned before, B is a known matrix with size of $m^2 \times m^2$. Let r is the rank of B. With the matrix factorization, B can be factorized to $B=A^TA$. Now let z=Ay, so the (11) and (12) can be rewritten as: Source: Ref. [35].

$$\min_{y,z,u} \ u + b^T y, \tag{13}$$

$$s.t.\ z = By \tag{14}$$

$$z^T z \leq u, \tag{15}$$

$$Y \geq 0, \tag{16}$$

With the problem (13-16), Wu et al [35] change it to be the following problem

$$\min_{y,z} \ (e_1 - e_2)^T \mathbf{u} + b^T y \ , \tag{17}$$

$$\text{s.t. } (e_1 + e_2)^T \mathbf{u} = 1, \quad (18)$$

$$\text{By-C}\mathbf{u} = 0, \quad (19)$$

$$\mathbf{u} \in K_{r+2}, \quad (20)$$

$$Y \geq 0, \quad (21)$$

Where $K_n = \{(x_0; x) \in R^n : x_0 \geq ||x||\}$. In the previous problems (17-19). They transform the problem (12) with sizes of$(m^2 + 1) \times (m^2 + 1)$constraints to be one second-order cone constraint of size r+2 and one linear constraint of size r+1. As the author noted, SQLP have greatly improve the efficiency in complexity.

3.2.3 K-means algorithm

In this part we will review some background information of k-means algorithm and show the implementation of clustering high space data using k-means. In machine learning, K-means is one of the most popular unsupervised learning algorithms that solve the well known clustering problem.

The procedure follows a simple way to partition a given data set with the known number of clusters. Randomly given a set of cluster centers (m1,m2…mk), the algorithm proceeds by iteratively optimizing two steps.

In the assignment step: Compute the values of each node to all the cluster centers, and assign the node to the nearest cluster.

In the update step: Recompute the center of the nodes in the cluster by calculating the means.

The algorithm is mainly in the following steps (figure 3.2):

1. Set the number of clusters.
2. Randomly select the k nodes as initial cluster centers
3. Calculate the value of each node to all the cluster centers and assign them to the nearest cluster
4. When all nodes have been assigned, recalculate the positions of the K centers.
5. Repeat the previous two steps until centers do not move.

The following figure is shown to be the standard k-means algorithm.

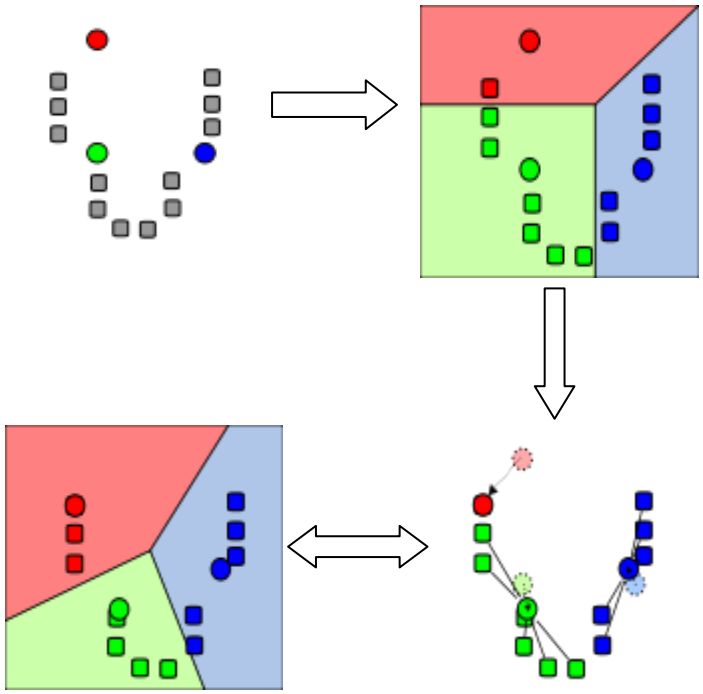


Figure 3.2: (a) k cluster center are randomly selected. (b) Assign each nodes to the nearest cluster centers. (c) Recalculate position of k cluster centers (d) Step 3 and 4 are repeated until cluster centers do not move. Source: Ref. [54].

3.2.4 Measure for partition

To measure the performance of our algorithm, we should compare it with the given real 'correct' partition. There are many criteria researched before, some of them are widely used. Normalized mutual information is a measure that is widely used. In this project, the extension of normalized mutual information is used to evaluate our performance and is proved to be reliable [46][47]. So we will introduce this in the following part in details.

Normalized mutual information is based on the framework of information theory which is defined as:

$$I_{norm}(X:Y) = \frac{H(X)+H(Y)-H(X,Y)}{(H(X)+H(Y)/2} \qquad (22)$$

Where H(X)(H(Y)) is the entropy of the random variable X from the partition $C'(C'')$. The range of this value is from 0 to 1, and will take maximum value 1 when two variables are exactly the same.

Normalized mutual information can compare overlapping partitions, so every note in the network may belong to more than one cluster. In a partition C', if the node i is in the cluster k of partition C' then $(x_i)_k = 1$, otherwise $(x_i)_k = 0$. If X is a random

variable from partition C', it can be regarded as a array with the kth entry of this array is $X_k = (X)_k$, the probability is: Source: Ref. [46]

$$P(X_k = 1) = \frac{n_k}{N}, P(X_k = 0) = 1 - \frac{n_k}{N},$$

Where n_k is the number of nodes in cluster k of partition C'. Given a partition C' and C'', the X and Y is the random variables of these two partitions respectively and k, m is the kth and mth cluster in these two partitions. So the following equation is

$$P(X_k = 1, Y_m = 1) = \frac{|C'_k \cap C''_m|}{N} \quad (23)$$

$$P(X_k = 1, Y_m = 0) = \frac{|C'_k| - |C'_k \cap C''_m|}{N} \quad (24)$$

$$P(X_k = 1, Y_m = 0) = \frac{|C''_l| - |C'_k \cap C''_m|}{N} \quad (25)$$

$$P(X_k = 0, Y_m = 1) = \frac{N - |C'_k \cap C''_m|}{N} \quad (26)$$

As Lancichinetti et al [46] noted, to infer the information lacked in changing from C' to C'', especially the information to infer X_k given Y_m, they defined it as:

$$H(X_k|Y_m) = H(X_k, Y_m) - H(Y_m) \quad (27)$$

From the above equation, we can see that if there is a cluster C''_d that is the same as cluster C'_k in C', so $H(X_k|Y_b) = 0$. Then, they extended this equation to all the components of Y.

$$H(X_k|Y) = \min_{m \in \{1,2\ldots\}} H(X_k|Y_m) \quad (28)$$

The normalized of $H(X_k|Y)$ is:

$$H(X_k \,|\, Y)_{norm} = \frac{H(X_k|Y)}{H(X_k)} \quad (29)$$

Combing all the cluster together, so the final definition of the normalized conditional entropy is:

$$H(X \,|\, Y)_{norm} = \frac{1}{|C'|} \sum_k \frac{H(X_k|Y)}{H(X_k)} \quad (30)$$

The computation of $H(Y|X)_{norm}$ follows the same procedure, then: Source: Ref. [46]

$$N(X|Y) = 1 - \frac{1}{2}[H(X|Y)_{norm} + H(Y|X)_{norm}] \quad (31)$$

They finally added some constraints in equation (28) to avoid some problem. For example, the Y_m should satisfy the following equation. Source: Ref. [33].

$$h[p(1,1)] + h[p(0,0)] > h[p(1,0)] + h[p(0,1)] \quad (32)$$

where $P(X_k = 1, Y_m = 1)$ is denoted as $p(0,0)$, and $h(p) = -p \cdot \log(p)$. If none of Y_m satisfy the equation (31),

$$H(X_k|Y) = H(X_k) \tag{33}$$

In totally, the method is in the following steps:
1.Compute the equation from (23) to (25) for the known k.
2.Compute the equation (28) and the constraints (32), if none of Y_m satisfy the equation (32) then compute the equation (33).
3.For every k, repeat the previous steps and compute the $H(X|Y)_{norm}$ in equation (30)
4.Compute the equation (31) and get the final performance.

3.2.5 Kernel k-means algorithm

Based on the previous analysis, the overview of Kernel K-means algorithm will be listed in the following part (figure 3.3):

Algorithm: Kernel K-means Community Detection

Input: The network graph G(V,E),V denotes the vertices, E denotes the edges, pairwise constraints sets M and C, and the number of clusters k.

Output: k Communities of the networks

1: Calculate the symmetric similarity matrix $S=[S_{ij}]$ though the shortest path.

2. Form the normalized graph Laplacian $\bar{L} = I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$, where $I \in R^{n \times n}$ is identity matrix, D is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} S_{ij}$, n is the number of vertices

3.Compute the eigenvalues and corresponding eigenvectors, then get the first eigenvectors $v_1, v_2 \ldots v_n$ from the first m eigenvalues.

4.Solve the SQLP (Semidefinite-Quadratic-Linear Programming) problems from equation (18) to (22) to get a kernel matrix K.

5.Apply K-means algorithm in kernel matrix K to get the final k clusters.

Figure 3.3: The kernel k-means algorithm for community detection.

3.3 Summary

This section described the general framework of this project and the whole implementation in details from the formulation of similarity matrix to detecting the k network communities finally. Especially the general framework we tried to incorporate the constraints to community detection algorithms. There are generally two methods to get the final kernel matrix, we explained all these approaches in details and selected a more efficient one to the kernel matrix. Besides these, we also introduced other problems, such as the k-means algorithm. All these problems have been implemented and tested.

In next section, we will test this algorithm with some artificial and real world networks with known structure and compare it with some other popular algorithms.

4. Experimental results

In this section, the new constraint based Kernel k-means algorithm is applied to a class of comprehensively used computer-generated and real world networks with known community structure to evaluate the performance in accuracy and efficiency. Firstly, we will focus on whether the algorithm can correctly extract the community structure. Then, the tolerance of resolution limit problems will be tested. The constraints incorporated in the following networks account for nearly one fifth of all the edges.

4.1 Performance on discovering community structure

4.1.1 Computer-generated graphs

The common way to evaluate the algorithm is to generate artificial network with known community structure and compare the communities found by the algorithm with known communities. The GN benchmark network was originally designed by Girvan and Newman [42] and has been widely used in many other community detection algorithms[21][26][42]. This network has 128 vertices and divided into 4 communities with 32 vertices in each community. Each vertex has 16 edges with a constant probability Z_{in} to be the number of inter-community edges and $Z_{out} = 16 - Z_{in}$ to be the number of between-community edges. The community structure will be more vague with the increase of Z_{out}. Figure 4.1 shows the network of Benchmark of Girvan and Newman with $Z_{out} = 5$.

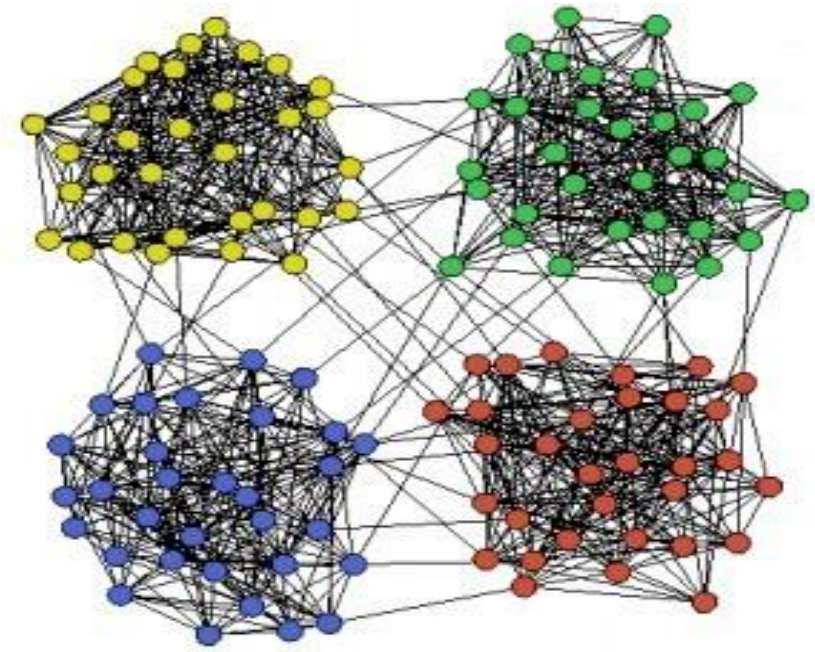


Figure 4.1: Benchmark of Girvan and Newman with $Z_{out} = 5$. Source: Ref. [23].

In this experiment, each Z_{out} generated 10 networks and the average results are summarized in figure 4.2. The vertical coordinates in figure is the accuracy of partition with the known community structure and horizontal coordinate is the value of Z_{out}. It can be seen that all the algorithms have good performance when $Z_{out} \leq 6$, our algorithm performs better than GN algorithm when $Z_{out} \geq 7$.

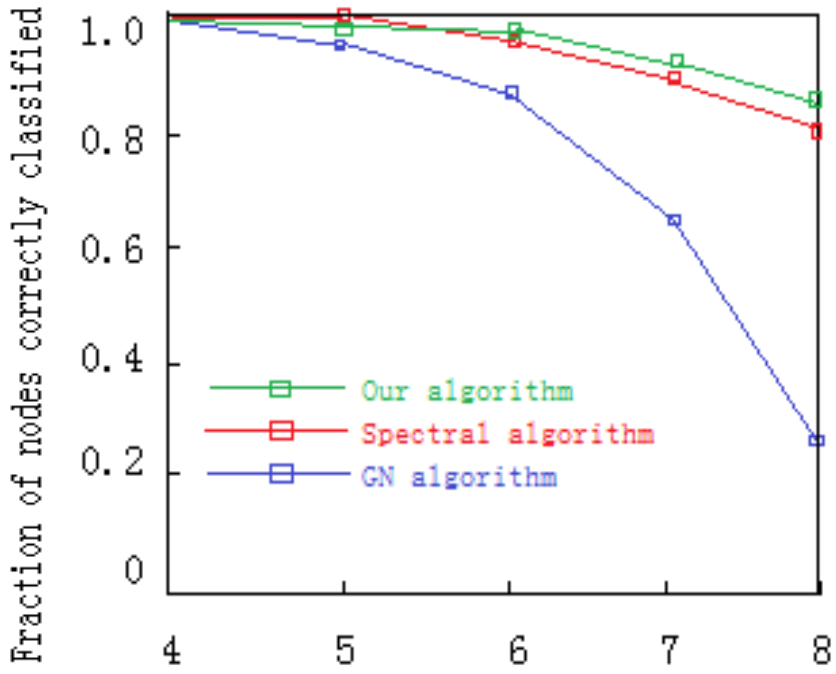


Figure 4.2: Test of three different algorithms on artificial networks (GN) with known community number.

To test the robustness of this kernel k-means algorithm, we followed the approach of Rosvall et al [44] to formulate an asymmetric network. For example, in the above networks, 3 communities (each with 32 vertices) are merged together to be a large communities with 96 vertices, so there are new two communities with 96 vertices and 32 vertices respectively. These asymmetric networks are difficult for GN, spectral algorithm and kernel k-means algorithm to detect the community structure. But our algorithm has better performance than other two algorithms due to incorporating the background information. The performances of each algorithm are shown in the figure 3. Finally, we test our algorithm on another set of benchmark networks called link asymmetric networks which are constructed by Rosvall et al [44]. These networks consist of two communities with 64 vertices in each communities, the vertices in the network has different average degrees ranging from 8 to 24. For these networks, we use $Z_{out} = 2,3,4$. Table 4.3 shows the experimental performance.

Table 4.3: Performance of three algorithms on three benchmark networks (Symmetric, Node Asymmetric, link Asymmetric) measured as fraction of nodes correctly classified.

Network	Z_{out}	D [26]	Q [22]	Our method
Symmetric	6	0.99	0.99	0.99
	7	0.97	0.97	0.98
	8	0.91	0.89	0.90
Node Asymmetric	6	0.99	0.85	0.98
	7	0.98	0.80	0.96
	8	0.94	0.74	0.92
Link Asymmetric	2	1	1	1
	3	1	0.96	0.99
	4	0.99	0.74	0.96

4.1.2 Real world networks

1. Karate club network

The first of the real world networks is from the well known karate club study of Zachary [45], which is widely used as a test example for identifying community structures in networks [12][21][22][26][42]. Zachary observed 34 members of all karate clubs over two years. During this period, the karate club broke because of the disagreement between the administrator of the club and club's instructor. Finally, the instructor left and started a new club with half of the original club members. A network of friendships between club members was constructed by Zachary, which is shown in figure 4.4. The 'square' and 'circle' of vertices represent two communities, led separately by two leading vertices 1 and 34. The vertices with green color are often misclassified. Our kernel k-means algorithm can perfectly detect the two communities in this network. However, the GN algorithm and spectral algorithm misclassified one vertex (vertices 10 and 3 respectively).

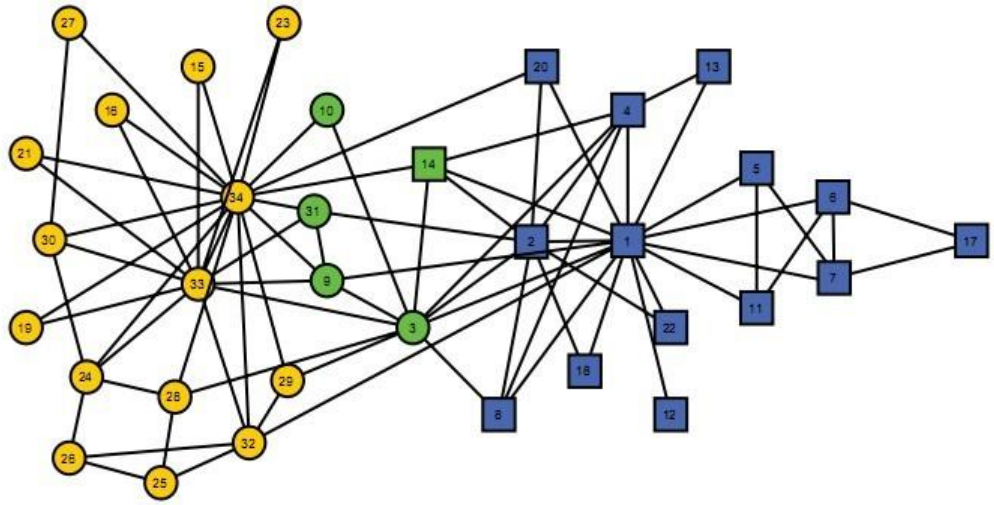


Figure 4.4: The Zachary's karate club. The 'circle' and 'square' represent the two communities. The vertices with green color which are shared between two communities are often misclassified. Source: Ref. [8].

As mentioned before, the community structure in this network is known, so we adopt normalized mutual information to measure the similarity of partitions. The procedure of normalized mutual information has been introduced in previous section. If two partitions are totally the same, the value will achieve 1. In contrast, the value will be 0 if two partitions are totally different. The following table 4.5 is the comparison of three algorithms in karate club network.

Table 4.5: The performance of three algorithms in karate club network with 34 vertices and 2 communities.

	GN	Spectral	Our method
Accuracy	97%	97%	100%
NMI	0.833265	0.833265	1

2. Football team network

The third real network is the network of US college football which is commonly used as a workbench for community detection testing in [25][26][42]. This football network is a representation of the schedule of Division I games for the 2000 season [26] in which there are 115 nodes (teams) and613 edges (games) played in the course of the year. The teams are generally divided into 12 conferences and each of them have around 8-12 teams. Games in the same conference (community) are more frequently than between conferences and each team play an average of seven games in the same conference and four games between conferences in 2000 season. The games are not uniformly distributed and teams geographically close to one another but belong to different conference are more likely to plays more games compared with teams geographically far apart.

The network can be seen in the figure 4.6. There are 12 colors displaying 12

conferences (community) and our algorithm can successfully detect the community structure of this network.

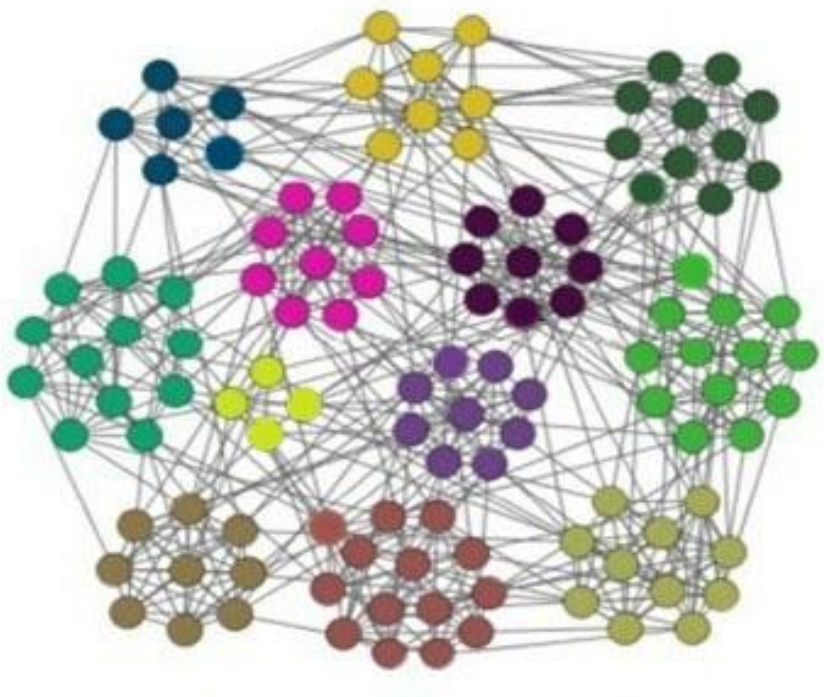


Figure 4.6: Community structure of the football team network with 12 communities and 115 vertices. Source: Ref. [26].

3. Other networks

Two more networks are listed as follows: The first network is a social network of 62 bottlenose dolphins living in Doubtful Sound, New Zealand. This network was researched by the biologist David Lusseau [53] for seven years. From the observation of frequent association between dolphins, he divided the network into two communities (circles and squares) which can be seen in the figure 4.7. Our method can successfully find the natural split of two communities. In addition, we can also partition the network into five communities, which is the same result with the method proposed by Newman et al [19] through the modularity optimization.

Another network is journal index network which is constructed by Rosvall and Bergstrom [44]. It is composed of 40 journals (nodes) from 4 different fields: physics, chemistry ecology and biology and 189 edges connecting journals if there is at least one article from one journal cites another article in other journals during 2004. As it can be seen in the figure 4.8, our method can successfully detect four communities except that one journal (Physical Review Letters) is classified into chemistry group. The most important reason for this mistake is that this journal is closely linked to journals in chemistry group. If must-link constraint is imposed on this journal, the physic and chemistry groups tend to merge together.

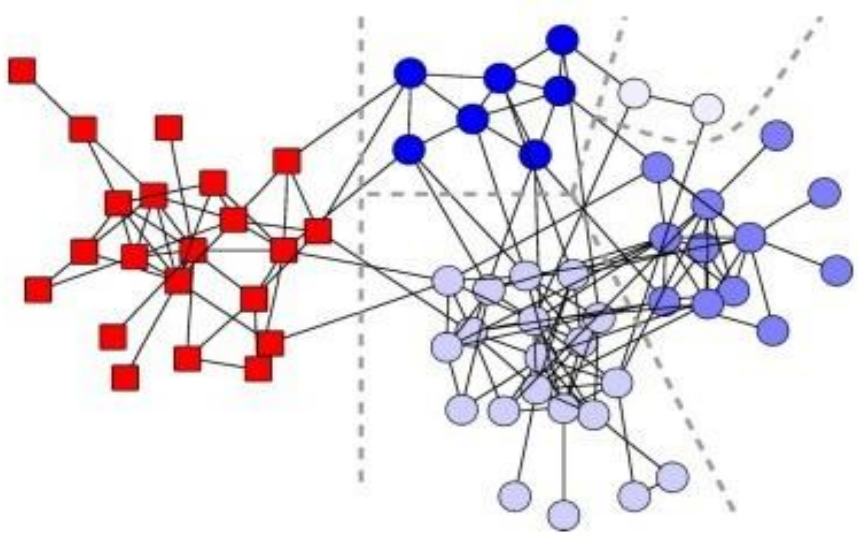


Figure 4.7: Social network of 62 bottlenose dolphins. The square and circle represent the two natural community of this network. The circle is split into four subgroups through the modularity optimization by Newman et al. Source: Ref. [19].

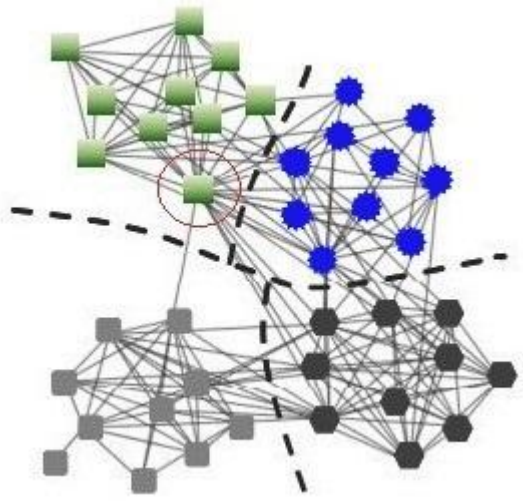


Figure 4.8: The network of journal index consisting 40 nodes form 4 groups: physics(green), chemistry(blue), biology (black) and ecology (grey). The journal with red circle is misclassified. Source: Ref. [44].

4.2 Performance on tolerating resolution limit problem

As mentioned before, the modularity optimization algorithms are criticized for the serious resolution limit problem [20], where small communities are likely to join to be large communities. Therefore, it has been a significant aspect for community detection algorithms to escape the resolution [20][21][25]. The following experiments are set to test the performance of our algorithm in terms of tolerating resolution limit.

Although the GN benchmark is one of the most popular graphs for community detection, it is not appropriate to test the resolution limit analysis due to the several problems, such as all the vertices in the network have essentially the same degree, all communities in the network are the same sizes. Because of the above reasons, the GN benchmark graph can hardly be regarded as the proxy of the real world network, thus the LFR benchmark graph proposed by Lancichinetti et al [52] is adopted for this experiment. The LFR benchmark is similar but more realistic compared with GN benchmark in the following three parts. First of all, it allows users to specify the node degree distribution. Secondly, user can set the fraction (u) of its links with other vertices in the network and $1-u$ is the fraction with the vertices in the same community. Finally, the size of communities in the network can also be set with minimum and maximal seizes s_{min} and s_{max} respectively [52]. Then LFR generator will produce vertices and communities by sampling those distributions. U is the

As mentioned above, u is the links with other vertices in the network. So if u=0, all the links are intra-community. Based on the method proposed in [52], we generate 10 samples for each of LFR benchmark networks with 200 vertices and the value of u varied from 0.1 to 0.5 (figure 4.9). In these networks, size of network is from 5 to maximum 20, the minimum and maximum degrees of nodes are 2 and 20 respectively with average degree 15. An example of this network is in figure 6. The exponent of the degree distribution and community size is 2 and 1 respectively.

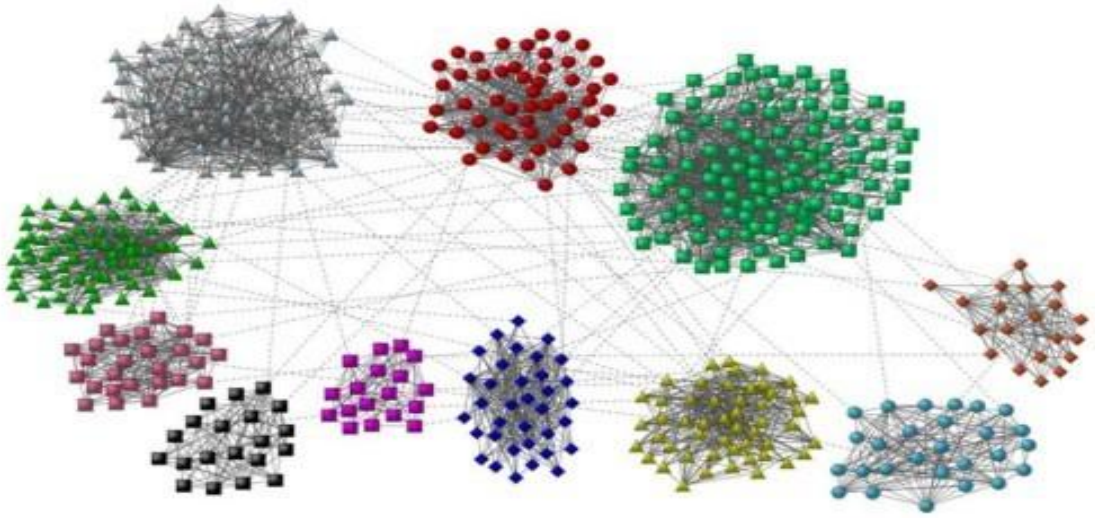


Figure 4.9: An example of LFR benchmark graphs with 500 vertices and 10 communities. Source: Ref. [52].

The average results are illustrated in the figure 4.10 with our algorithm as well as other three algorithms, and the u value of each node is predefined. From the comparison of figure 1 and figure 7 in terms of spectral algorithm and our algorithm, it can be seen that there is better performance in GN benchmark graph than LFR benchmark. The most important reason is that the number of community and the size of each community in GN benchmark are fixed, which becomes easily for most algorithms.

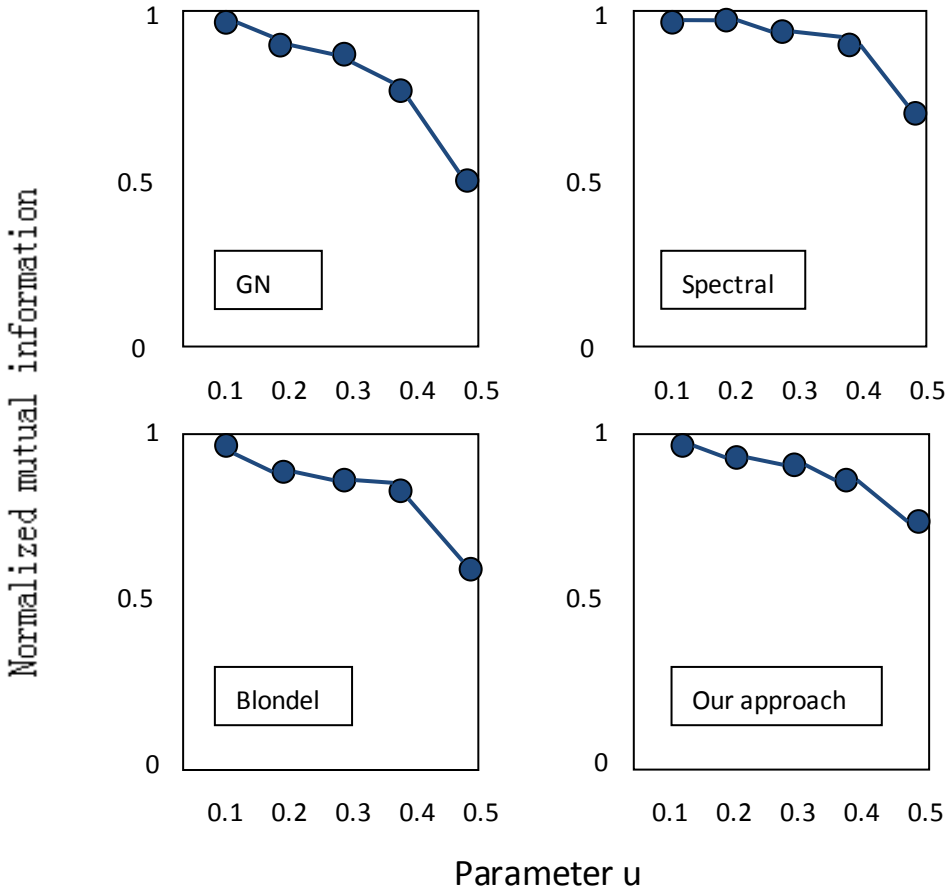


Figure 4.10: Test performance of four algorithms on the undirected and unweighted LFR benchmark graphs, each point is an average of 10 instances.

From figure 4.10, it can be seen that spectral algorithm have better performance than other algorithms when the value $u \leq 0.4$. Particularly, when value $u \leq 0.2$ it can detect community structure nearly 100%. Algorithms based on modularity optimization (such as Blondel algorithm [39] in the figure above) do not have good performance over most of natural communities due to the resolution limit problem mentioned above. From figure 7, it can also be seen that our algorithm has steady performance compared with other algorithms and achieve nearly 0.7 at u=0.5, and the spectral algorithm nearly 0.65. Because u denotes the links with other vertices in the network, it becomes increasingly harder to detect community structures correctly when the value u increases. When the network structure is not so fuzzy ($u \leq 0.4$), spectral algorithm performs great, but it declines greatly when the network becomes fuzzy.

4.3 Summary

In this section, the performance of our algorithm is tested on some famous computer generated benchmark graphs as well as some classical real world networks and the results are compared with other popular algorithms (such as GN algorithm and spectral algorithm). In addition, normalized mutual information is applied to compare the partition we detect with the known solution. After lots of experiment, we can see that our algorithm has a good performance in most of the computer-generated and real world networks with enough constraint. In particular, our algorithm does not have the resolution limit problem compared with modularity optimization based algorithm (such as Blondel algorithm [39]).

5: Conclusion

Community detection (graph clustering) is an important topic in machine learning and it has attracted considerable research interests in recent years. Majority of recent research for community detection are unsupervised machine learning clustering algorithms, however they are not always suitable because there are all kinds of constraints or restrictions in real world network, which are particularly useful for applications in domains as a kind of Priori knowledge. Therefore in this project a new kernel method incorporated with some instance-level constraints is adopted to detect community structure in networks. It is based on the method of Wu et al [35] to optimize the kernel using the constraints through Semidefinite Quadratic Linear Programming.

In order to detect community structure in networks, we need to know how it works and the feasibility of incorporating instance-level constraints. In chapter 2, some traditional and nowadays popular community detection methods are introduced as well as the analysis of instance-level constraints and the challenges of incorporating constraints. From the research, it can be seen that there are various instance-level constraints such as ε-constraints and δ-constraints, but most of them can be transformed into two categories: must-link (pairs of vertices must be in the same cluster) and cannot-link (pairs of vertices can not be in the same cluster) constraints, thus the constraints we incorporated in this project are these two constraints.

In chapter 3, the general framework of incorporating constraints is specified as well as the detailed implementation of this project is discussed including mainly three steps. The general framework is to find a mapping that will map the similarity data in input space to a high dimension space (kernel), where two must-link nodes are mapped to be close enough and two cannot-link nodes are mapped to be far enough. In addition, those vertices similar to the must-link vertices in the input space will be clustered into the same community and those similar to the cannot-link vertices will be clustered into the different communities. As for the general steps of implementation, the first step is to formulate the similarity matrix of the network. As mentioned before, community detection is a special kind of clustering which clusters the vertices with high similarity in the same community and low similarity between the different communities, however the only information we know about the network is some edge relations, so the first step is to formulate the similarity from these edge relations. In this project, the shortest path based similarity measure is selected after comparing with other methods. As listed in chapter 3, the second step is divided into two parts: construct and optimize the kernel matrix. To construct kernel matrix, we should find a mapping that will incorporate must-link and cannot-link constraints. For example, the method in this project is to map the two must-link vertices to the same point and two cannot-link vertices to be orthogonal in high kernel hypersphere. To optimize kernel matrix, we adopt the method through Semidefinite Quadratic Linear Programming

through some mathematical manipulation which is proposed by Wu et al [35]. The last step is to cluster the kernel matrix through k-means algorithm.

Chapter 4 demonstrates the performance of method we proposed through some famous computer generated and popular real world networks. Finally, the results are compared with other popular algorithms (such as GN algorithm and spectral algorithm). The experiments are totally divided into two parts. Firstly, we focus on whether the algorithm can correctly detect the community structure. Then, we pay attention to the resolution limit problems. Normalized mutual information is adopted for comparison of partition.

Although the method we proposed can successfully detect community structure in most cases, there are some limit and should make an improvement. Thus next section we will conclude some suggestion to the future work.

6: Future work

Although our method has a good performance in community structure detection, there is still some parts need to be improved. In this chapter, we should analysis the existing problems and conclude some suggestion to the future work.

6.1 Number of communities

In this project, we assume that umber of clusters is known. However, in most cases, the number of communities in networks is really difficult to be known in advance. This may be the biggest limitation in our project.

There are various methods to determine the number of communities (m) in network. Each method has its advantage and shortcoming. One of the most widely used is to compute the quality function (Q function or D function). To get the number of community (m), the general procedure we suggest is as followed: firstly, we need to define a measure (Q function or D function), then start from the smallest value (for example m=1) and gradually increase this number until the quality function achieve its maximum value. The value corresponding to the maximum value of the measure is selected as the number of community.

6.2 Sufficient constraints

After the research, we know that most of previous constraint based community detection algorithm will only incorporate constraints through easy approaches, for example, if vertices are must-link, so their approach is to change the similarity to1, and if vertices are cannot-link, they will define the similarity of these two vertices to be 0. However, the similarity of 0 between two cannot-link vertices does not really mean they will be clustered in different communities. Therefore, in this project I follow the method proposed by Wu et al [35] to find a mapping that the similarity data will be mapped to a unit hypersphere with the utilization of constraints. Two must-link vertices are mapped to the same point and two cannot-link vertices will be orthogonal in this unit hypersphere. In addition, those vertices similar to the must-link vertices in the input space will be clustered into the same community and those similar to the cannot-link vertices will be clustered into the different communities.

Although this method can sufficiently incorporate all the constraints and have a good performance in experiment, there is a limit that we need sufficient constraints to optimize the kernel matrix. This is the reason why the constraints we incorporate in experiment account for nearly one fifth of all the edges. The suggestion of future development is to change the optimization solution and maximally use the similarity information between vertices as well as improve the accuracy with minimum

constraints.

6.3 Directed graph and overlapping graph

The vast majority of networks are undirected and non-overlapping graphs. But in real world, they are many directed graphs and communities in these networks overlap to some extent. For example, in social networks, the acquaintance of a teacher to a student may not be equal to the acquaintance of the student to a teacher. Thus, it seems essential to extend algorithm to detect directed and overlapping graph.

To detect community structure in directed graph, one suggestion to the future work is to compute the asymmetric similarity matrix using the method such as edge betweenness. To find overlapping community structure in networks, one possible suggestion in this project is to replace the last step (k-means algorithm) with some other algorithms, such as fuzzy c-means.

After addressing all above problems, we believe that the huge improvement will be made and this method will be applied to more real networks for community detection.

7.References

1. Zhu, X. Semi-supervised learning literature survey. Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison (2005).
2. Wagstaff, K. & Cardie, C., Rogers, S. Constrained k-means clustering with background knowledge. Proc. 18th Intl. Conf. on Machine Learning (2002).
3. Kamvar, S & Klein, D. Spectral learning. Proc. 17th Intl. Joint Conf. on Artificial Intelligence (2003).
4. Basu, S & Bilenko, M. A probabilistic framework for semi-supervised clustering. Proc. 10th Intl. Conf. on Knowledge Discovery and Data Mining (2004).
5. Davidson, I. & RAVI, S. Hierarchical clustering with constraints: Theory and practice. Knowledge Discovery and Data Mining 14, 1(2005).
6. Davidson, I & Basu, S. Clustering with Constraints (2007).
7. Davidson, I & Basu,S. A survey of Clustering with Instance Level Constraints (2007).
8. Dhillon, I & Kulis, B. Semi-supervised Graph Clustering: A Kernel Approach (2004).
9. Klein, D & Kamvar, S. From Instance-Level Constraints to Space-Level Constraints: Making the Most of Prior Knowledge in Data Clustering, Proc. 19th Intl. Conf. on Machine Learning, Sydney, Australia, July 2002, pp. 307–314 (2002).
10. Reddy, K. & Kitsuregawa, M. Proceedings of the Second International Workshop on Databases in Networked Information Systems (Springer-Verlag, London, UK), pp. 188-200 (2002).
11. Leicht, E & Holme, P. Vertex similarity in networks. Physical Review E 73(2) 026120 (2006).
12. Liu, Z & Li, P. Community detection by affinity Propagation (2008).
13. Jaccard, P. Distribution be la flore alpine dans le basin des dranses et dans quelques regions voisines. Bulletin del la Socit Vaudoise des Scieces Naturalles 37, 241-272(1901).
14. Salton, G. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley, Reading, MA (1989).
15. Ravasz, E. & Somera, A.L.: Hierarchical Organization of Modularity in Metabolic Networks. Science 297(5586), 1551-1555 (2002).
16. Burt, R.S. Position in networks. Social Forces 55,93-122(1976).
17. Gregory S. Finding overlapping communities in networks by label propagation. University of Bristol, England (2009).
18. Palla,G & Farkas, I. Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (2005), pp. 814-818.
19. Newman, M & Girvan, M. Finding and evaluating community structure in networks. PHYSICAL REVIEW E 69,02613(2004).
20. Fortunato, S & Barthelemy, M. Resolution limit in community detection (2006).
21. Ma, X & Gao, L. Semi-supervised clustering algorithm for community structure

detection in complex networks(2009).
22. Newman, M. Detecting community structure in networks. Department of physics and center for the study of Complex Systems, University of Michigan, Ann Arbor, MI 48109-1120 (2004).
23. Fortunato, S. Community detection in graphs. Complex Networks and Systems Lagrange Laboratory, ISI Foundation, Viale S. Severo 65, 10133, Torino(2009).
24. Brandes, U & Delling, D. Maximizing modularity is hard. Physics/0608255 30 August (2006).
25. Fortunato, S. Quality function in community detection. Complex Networks lagrange laboratory(CNLL), ISI Foundation, Torino, Italy (2007).
26. Li, Z & Zhang, S. Quantitative function for community detection. PHYSICAL REVIEW E 77, 036109 (2008).
27. Wagsraff, K & Cardie,C. Clustering with Instance-level Constraints. Proceedings of the Seventeenth International Conference on Machine Learning (2000).
28. Klein, D & Kamvar, D. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. Proc. 19th Intl. Conf. On Machine Learning (2002).
29. Andrew, Y & Michael, I. On spectral clustering: Analysis and an algorithm. In Proceedings of Advances in Neural Information Processing Systems (NIPS 14)(2002).
30. Dhillon, I & Guan, Y. A unified view of kernel k-means, spectral clustering and graph cuts (Technical Report TR-04-25). University of Texas at Austin (2004).
31. Duda, R & Hart, P. Pattern classification and scene analysis(1973).
32. Hierarchical Clustering. Located at : http://www.resample.com/xlminer/help/HClst/HClst_intro.htm. [Accessed 9/15/2010].
33. Li, Z&Liu, J. Pairwise Constraints Propagation by Semidefinite Programming for Semi-Supervised Classification. In ICML(2008).
34. Gustafsson, M & Hornquista, M. Comparison and validation of community structures in complex networks, Physica A 367, 559-576(2006).
35. Wu, Xiao&Li, Zhen. Fast Graph Laplacian Regularized Kernel Learning via Semidefinite -Quadratic-Linear Programming. NIPS (2009).
36. Li, Zhen&Liu jian. Constrained Clustering via Spectral Regularization. CVPR (2009).
37. Boyd, B & Vandenberghe, L. Convex Optimization. Cambridge University Press, (2004).
38. Brochers,B.CSDP, a C Library for Semidefinite Programming. Optimization Methods and Software, 613-623.
39. Blondel, V &Guillaume, J. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment.P10008 (2009).
40. Traud, A &Kelsic, E. Comparing community structure to characteristics in online collegiate social networks. eprint arXir:0809.0690 (2008).
41. Freeman, L .A set of measures of centrality based upon betweenness. Sociometry 40, 35-41(1977).

42. Girvan, M & Nerman, M. Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA99,7821-7826(2002).
43. Tyler, J &Wilkinson, D. Email as Specroscopy: Automatd Discovery of Community Structure with Organizations(2005).
44. Rosvall, M &Bergstron, C. An information theoretic framework for resolving community structure in complex networks, Proc. Natl.Acad. Sci. USA 104(2007).
45. Zachary, W. An information flow model for conflict and fission in small groups. J. Anthropol. Res. 33(1977).
46. Lancichinetti, A &Fortunato, S. Detecting the overlapping and hierarchical community structure in complex networks. New Journal of Physics 11, 033015(2009).
47. Dannon, L & Diaz-Guilera, A. Comparing community structure dentification. J. Stat. Mech. Theory Exp (2005).
48. Radicchi, F & Castellano, F. Defining and identifying communities in networks Preprint cond-mat/0309488 (2003).
49. Newman, M. Fast algorithm for detecting community structure in networks. PHYSICAL REVIEW E 69, 066133(2004).
50. Holme, P & Huss M. Subnetwork hierarchies of biochemical pathways. Bioinformatics Vol. 19 no. 532 (2003).
51. Gifford, Miriam. Modeling the Virtual Plant: A systems Approach to Nitrogen-Regulatory Gene Networks (2006). Located at: http://4e.plantphys.net/article.php?ch=&id=352. [Accessed 9/23/2010]
52. Lancichinetti, A & Fortunato, S. Benchmark graphs for testing community detection algorithms. PHYSICAL REVIEW E 78, 046110(2008).
53. Lusseau, D. The emergent properties of a dolphin social network. Proc. R. Soc. London B (suppl.) 270, S186-188 (2003).
54. Http://en.wikipedia.org/wiki/K-means_clustering#cite_note-10. [Accessed 9/23/2010]

Appendix A: Source Code

Based on the limitation of space, only main and some functions are listed here.

Main function:

```
clc,clear,close all;

addpath('.\csdp6.1.0winp4\matlab');
addpath('.\Solver\');
addpath('.\Solver\Mexfun');

load SimiMatrix;
load M_link;
load C_link;
%load dolphin_simi;
%load dolphin_M_link;
%load dolphin_C_link;
D=C;
n=size(D,1);
G=zeros(n);
for i=1:n
      for j=1:n
        if (i==j)
            G(i,i)=1;
        else G(i,j)=1/D(i,j);
        end
      end
end
d=zeros(n,1);
for i=1:n
      d(i)=sum(G(i,:));
end
D=diag(d);
L=D-G;
H=sqrtm(D);
Lsym=pinv(H)*L*pinv(H);

[E,B]=eig(Lsym);

diagonal=diag(B);

[diagonal,index]=sort(diagonal);
for i=1:size(B,1)
   new_E(:,i)=E(:,index(i,1));
end
m=15;
QE=new_E(:,1:m);

% formulate the convex quadratic semidefinite program

[A, b] = coquad(QE,M_L,C_L);

% symmetrize b
b = reshape(b,[m,m]);
b = (b+b')/2;
b = b(:);

%%%%%%%%%%
SQLP %%%%%%%%%%%%%%%%
[blk,At,C1,b1] = clcPr(A, b);

tic;

[obj1,X1,y1,Z1,info1] =
sqlp(blk,At,C1,b1);
toc;

K_final=QE*X1{1,1}*QE';

c=2;

ind=round(n*rand(1,c));
nc=K_final(ind,:);
ty=zeros(1,n);
nr=zeros(1,c);

maxiter=1000;
iter=1;
while iter<maxiter
```

```
for i=1:n

dist=sum((repmat(K_final(i,:),c,1)-nc).^2,2)
;
            [m,ind]=min(dist);
            ty(i)=ind;
      end
      for i=1:c
            ind=find(ty==i);
            nc(i,:)=mean(K_final(ind,:));
            nr(i)=length(ind);
      end
      iter=iter+1;
end

position=zeros(n,1);
for i=1:c
      str=['The nodes in cluster ' num2str(i) '
are'];
      disp(str);
      ind=find(ty==i);
      fprintf('%d ',ind);
      fprintf('\n');
      position(ind)=i;
end
position;
```

Coquad function:

```
function [B, b] = coquad(Q,M,C)

[Npts m] = size(Q);
B = zeros(m^2);
b = zeros(m^2,1);

for i = 1:Npts
      U = Q(i,:)'*Q(i,:);
      s = U(:);
      B = B + s*s';
      b = b + s;
end

for k = 1 : size(M,1)
      i = M(k,1);
      j = M(k,2);
      U = Q(j,:)'*Q(i,:);
      s = U(:);
      B = B + s*s';
      b = b + s;
end

for k = 1 : size(C,1)
      i = C(k,1);
      j = C(k,2);
      U = Q(j,:)'*Q(i,:);
      s = U(:);
      B = B + s*s';
end

b = -2 * b;
```

Localformulate function:

```
function    [F0, FI, c] =
localformulateSDP(S, D, b)
% formulate SDP problem
% each FI that corresponds to the LMI
for the quadratic cost function has
% precisely 2*D^2 nonzero elements.
But we need only D^2 storage for
% indexing these elements since the FI
are symmetric
tempFidx = zeros(D^2, 3);
dimF = (D^2+1) + D;
idx= 0;
for col=1:D
      for row=col:D
            idx = idx+1;
            lindx1 = sub2ind([D D], row,
col);
            lindx2 = sub2ind([D D], col,
row);
            tempFidx(:,1) = [1:D^2]';
            tempFidx(:,2) = D^2+1;
```

```
if col==row
                tempFidx(:,3) = S(:,
lindx1) ;
                FI{idx} =
sparse([tempFidx(:,1);
tempFidx(:,2);
row+D^2+1; ],

[tempFidx(:,2);
tempFidx(:,1);
row+D^2+1; ],

[tempFidx(:,3);
tempFidx(:,3);
1;], dimF, dimF);
          else
                tempFidx(:,3) = S(:, lindx1)
+ S(:, lindx2);
                FI{idx} =
sparse([tempFidx(:,1);

tempFidx(:,2);

row+D^2+1;

col+D^2+1; ], ...

[tempFidx(:,2);
tempFidx(:,1);
col+D^2+1;
row+D^2+1; ],

[tempFidx(:,3);
tempFidx(:,3);
 1;1;    ], dimF, dimF);
          end
     end
end

idx=idx+1;
```

```
% for the F matrix corresponding to t
FI{idx} = sparse(D^2+1, D^2+1, 1,
dimF, dimF);

% now for F0
F0 = sparse( [[1:D^2]], [[1:D^2]],
[ones(1, D^2)], dimF, dimF);

% now for c
b = reshape(-b, D, D);
b = b*2 - diag(diag(b));
c = zeros(idx-1,1);
kdx=0;
%keyboard;
for col=1:D
    for row=col:D
      kdx = kdx+1;
      c(kdx) = b(row, col);
    end
end
%keyboard;
c = [c; 1];
return;
```

sdpToSeDuMi function:

```
function [A, b, c] = sdpToSeDuMi(F0,
FI, cc)
if nargin < 3
    error('Cannot convert SDP
formulation to SeDuMi formulation in
sdpToSeDumi!');
end

[m, n] = size(F0);
if m ~= n
    error('F0 matrix must be squared
matrix in sdpToSeDumi(F0, FI, b)');
end
p = length(cc);
```

```
if p ~= length(FI)
    error('FI matrix cellarray must have the same length as b in sdpToSeDumi(F0,FI,b)');
end


c = reshape(F0', n*n,1);

% converting equality constraints of the canonical SDP
zz= 0;
for idx=1:length(FI)
    zz= zz + nnz(FI{idx});
end
A = spalloc( n*n, p, zz);
for idx = 1:p
    temp = reshape(FI{idx}, n*n,1);
    lst = find(temp~=0);
    A(lst, idx) = temp(lst);
end
% The SeDuMi solver actually expects the transpose of A as in following
% dual problem
% max b'y
% s.t. c - A'y is positive definite
% Therefore, we transpose A
% A = A';

% b doesn't need to be changed
b = cc;
return;
```

Chop function

```
function [R, P, I] = cholp(A, pivot)

if nargin == 2, piv = pivot; else piv = 1; end

[n, n] = size(A);
pp = 1:n;
I = [];

for k = 1:n

    if piv
        d = diag(A);
        [big, m] = max( d(k:n) );
        m = m+k-1;
    else
        big = A(k,k);    m = k;
    end
    if big <= 0, I = k-1; break, end

    if m ~= k
        j = 1:k-1;
        if j
            temp = A(j,k); A(j,k) = A(j,m); A(j,m) = temp;
        end
        temp = A(k,k); A(k,k) = A(m,m); A(m,m) = temp;
        A(k,m) = conj(A(k,m));
        j = k+1:m-1;
        if j
            temp = A(k,j)'; A(k,j) = A(j,m)' ; A(j,m) = temp;
        end
        j = m+1:n;
        if j
            temp = A(k,j); A(k,j) = A(m,j); A(m,j) = temp;
        end
        pp( [k m] ) = pp( [m k] );
    end

    A(k,k) = sqrt( A(k,k) );
    if k == n, break, end
    A(k, k+1:n) = A(k, k+1:n) / A(k,k);

%    For simplicity update the whole of the remaining submatrix (rather
%    than just the upper triangle).
    j = k+1;
```

```
j = k+1:n;
      A(j,j) = A(j,j) - A(k,j)'*A(k,j);

end    % Main k loop

R = triu(A);
if isempty(I), I = n; end
if nargout == 2 & ~piv
    P = eye(n);
elseif nargout >= 2
    P = eye(n); P = P(:,pp);
end
```

clcPr function:

```
function [blk,At,C,bt] = clcPr(A, b)
    % Cholesky decomposition
    B = cholpp(A);

    q = size(B,1);              % the rank of A
    n = sqrt(size(B,2));
    s = n*(n+1)/2;

    % collect parameters for semidefinite
block
    blk{1,1} = 's'; blk{1,2} = n;

    for k=1:q;
        tp = reshape(B(k,:), [n,n]);
        tp = (tp+tp')/2;
        AA{1,k} = tp;
    end;

    matrepdiag = svec(blk(1,:), AA);
    At{1,1} = [zeros(s,1), matrepdiag{1}];

% collect parameters for quadratic block
blk{2,1} = 'q'; blk{2,2} = q+2;
At{2,1} = [1,1, zeros(1, q); zeros(q,2),
-1*speye(q)];
bt = [1; zeros(q,1)];

    % collect parameters for the objective
function
    tmp = reshape(b, [n,n]);
    tmp = (tmp + tmp')/2;
    C{1,1} = tmp;
    C{2,1} = [1; -1; zeros(q,1)];
```