Executive Summary

The aim of this project is to experiment with layout algorithms for both clustering and visualization. It has already been demonstrated in one of Andreas Noack's paper – *Modularity clustering is force-directed layout*, as a transformation of modularity into (a, r)-energy model is possible, layouts subsume clusterings. LayoutClustering was developed to achieve this goal. LayoutClustering is a Java application. The most interesting and exciting feature of LayoutClustering is that not only can it be used to get the communities of a network, but also display a network in a dialog. The followings are the bullet points for this dissertation.

- I implemented a version of the energy model proposed by Andreas Noack, and adapted it for generating coordinates, and then k-means is used to cluster the nodes into different communities, see pages 29-31.
- Experiments on the implementation are carried out and the results are compared with the ones got by GN and CNM algorithms, see pages 32-49.
- The evaluation of the program is given and come to a conclusion that layout can be used to cluster networks, see pages 50-52.
- In the implementation of LayoutClustering, some ideas come out and they are considered as future work that will make improvement to the program, see pages 53-54.

Contents

Instruction

Network researchers from Herbert Simon [31] to Mark Newman [32] have noticed that there is a common characteristic in many real-world network systems: The nodes in them are always grouped into different subsystem with dense intra-subsystem connections and relatively sparse inter-subsystem connections. Studying these subsystems are interesting, as they trend to have some similar attributes, for example, groups of friends in social networks, colleagues in different department of a company, related documents in hypertexts, regional economic organizations in international trade, and cohesive modules in some kind of software systems. The nodes in the subsystems potentially correspond to each other. If the elements in the network are modeled as nodes and the connections between them as edges, two widely used methods to study the community structure of a network are clustering, which are used for community detection, grouping nodes into different subsets, and layouts, which assign the nodes to the specific position in a metric space.

Actually, layout algorithms have different methods for drawing undirected graph: forced-directed and energy-based methods. The former one is a force system, which searches for a state of balance where the force on each node is zero. The other one are always based on an energy model and an algorithm that looking for a stable state with minimum energy. Because forces can be expressed as a negative gradient of energy, this meets the requirement of searching a local minimum of energy. Therefore, in my implementation, the coordinates of nodes are generated under the energy-based model.

Finding clusters in a network is a crucial problem in VLSI design [33], parallel computing [34], software engineering [35], and graph drawing [36]. However, most existing layout algorithms are not designed for doing this task, but generating readable visualizations. These popular force and energy models do not give the clusters clearly, especially when the graph has a small diameter.

Energy models enforce certain properties to make the drawing of networks easy to be viewed, and attributes of the drawn graph is inferred more efficient and the results of it is more valid. These properties that the drawing is required to have, include small and uniform edge lengths, well-distributed nodes, or well-separated clusters [37].

LayoutClustering applied in this paper will cluster the network using coordinates and give us drawings, which reveal the clusters of such networks. It also draws graph according to certain criteria (e.g. [38]).

Chapter 1: Aims and Objectives

The primary aim of this project is to use layout algorithms for community detection. As there are a lot of materials that discuss the theoretical relationship between layout algorithms and community detection, demonstrating the feasibility of my project is unnecessary. However, layout algorithms and clustering algorithms are computed separately. For example, Andreas Noack's *LinLogLayout*. However, in this project, I will use the coordinates of nodes in a network, which are generated by layout algorithms, for community detection. K-means will be my best option to cluster these nodes.

The objectives to meet this aim can be broken down as follows:

1. Study clustering algorithms and layout algorithms respectively and closely, identify useful studies in the literature relate to the relationship between clustering algorithms and layout algorithms.
2. Look for both clustering algorithms and layout algorithms, experiment with clustering algorithms and find out what will get by them. At the same time, use layouts to visualize the result get by the clustering algorithms.
3. Choose a suitable layout algorithm and study the difference between this layout algorithm and a community detection algorithm.
4. Implement the layout algorithm chosen in 3 and use k-means for community detection.
5. Based on the result of 4, test and improve the function implemented.
6. Analyze and discuss the communities detected by the layout algorithm. Make improvement if needed.
7. All the implementation will be done with Java programming.

Organization of this dissertation

This dissertation will first introduce some background knowledge and previous work done by the others. Basic theory used in my implementation will be given in Chapter 3. Chapter 4 will be the implementation of my program and Chapter 5 will be tests of my implementation. Evaluation is given in Chapter 6. Finally, a conclusion and future work is proposed in Chapter 7.

Chapter 2: Background and Previous Work

2.1 Introduction

The community structure of many real-world networks is crucial for many studies, and has raised great interest. Among these, clustering and layout algorithms are two main streams in this trend. They are supposed to do a cooperative job when analyzing networks, such as groups of friends in social networks, cohesive modules in software systems, etc. Clustering algorithms will detect the communities and group the nodes of the network into different clusters; in the mean time layout algorithms will assign nodes to the relevant positions in a specific space. In this part, some background research of community and how to detect them will be introduced. The basic concept of clustering and layout algorithms will follow. Finally, some related work, which has been done before, would be shown.

2.2 Community

In sociological theory, people are the basic units to form the society. This can be applied to networks that exist nowadays. Network is made up of nodes. There can be a range of number of nodes. It can be up to million even billion, such as the World Wide Web [1,2], which is composed by the many hosts that are placed all over the world. It can be only several nodes such as a small network group in a family. Examples also include citation network [3, 4], food webs [5]. Nodes or vertices in these network, for instance, represent computers or routers on the internet, or animals in the food web, connected by edges or links, standing for the wire between the computers and other connection devices, the relationship between different kinds of animals.

2.2.1 What is community?

In a network, there is a feature that makes the network distributes according to some rules – community. Community is a group of nodes or vertices that are densely connected to each other in the same group, while having less connection to the nodes or vertices outside this community. The nodes in the same community share some common features or play similar role in the network. Clusters and modules also represent for community. In social network, people in the same community means that they may have same religions, base on the geographically nearby location, share the same interest or work for the same company and so on.

Community itself comes from social context as people tend to form groups with their family, workmates and friends. In Fig.1 a simple example of network with three communities is shown.

Community exists between the nodes and network. It can be called a small network itself, too, comparing to the large network it belongs to. Society provides a wide range of organizations, while the diversity on the Internet also led to the virtual groups. BBS is popular among the web users these days. It is also called a virtual

community. Users in a BBS can be divided into smaller groups according their interest. This is a community inside a community.

There is another welcome style among the open-source developers – mailing list. The core developers release the latest version of software by the mailing list. The subscribers will test the software they receive and they will send feedback to the core developers so that they can make some improvement to the software. Sometimes the subscribers can have their own solution to the problem they find and so do the core developers as they can send this solution to the other subscribers in the mailing list.

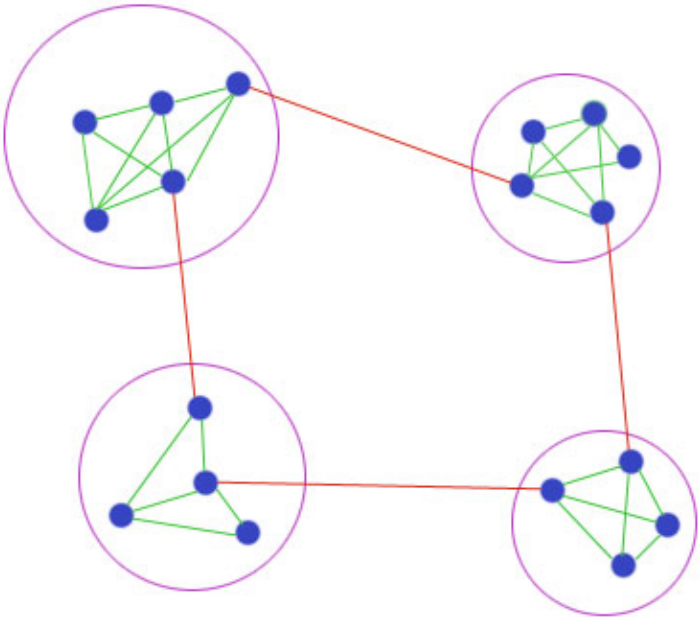

Fig.1 A simple graph with three communities.

2.2.2 Why we study community?

Community structure is common in the real network. It is useful to learn the layout of the network and study the "habit" of the nodes. In social network, it can help us learn about the people's common feature in the network extensively.

Nodes in a network can be grouped differently according to different dividing rules. Nodes in the same community have some common characteristics. Take the example of BBS again, in a BBS, people tend to have some common interests, such as sports and photography. When referring to sociological scope, people belong to the same community because they live in the nearby location, or they work together.

Because community has such features, studying community help people to learn the features of the whole network more easily. We can first learn the community and then to the whole network. People have used this method to study the human civilization. We can grasp the similarity of the small part of the network, and gradually the commonality of the whole network can be mastered more thoroughly, and it can provide insight into how the whole network functions.

Nodes in different communities behave differently to each other. Some nodes act more importantly than the others in the same community.

2.2.3 Community in real-world network

In this part, I will present some example of real-world network with community, so we can see what it looks like.

In Fig.2, this is the example of Zachary's network of karate club members [7], which is well known as it is regularly used to test the community detection algorithm. It is made up of 34 nodes, standing for the members of a karate club in the United States, who were examined during a time of 3 years. The edges connect the persons who were found to have a relationship outside the activities of the club. Assume that vertex 1 and 34 is the main member in this club, a conflict that happens between them may divide all the members into two groups, supporting 34 and 1 respectively. It is shown in squares and circles.

Fig.3 is the network of collaborations of scientists working at the Santa Fe Institute (SFI). The vertices stand for the scientists at SFI while the edges connect those who worked together for at least one paper. In this graph, authors of one paper link to each other. Little connection links those that are not in the same groups.

Protein network in an animal is an increasing important subject in biology and bioinformatics, because how the cells perform for an animal's daily life is depended on the interaction between the proteins. The proteins are divided into groups according to some identical or similar functions, as different colors shown in the graph. Some of the groups have no interaction to the rest of the protein network. As shown in the figure, most of the communities are related to cancer and metastasis, which makes the study of the protein network important.

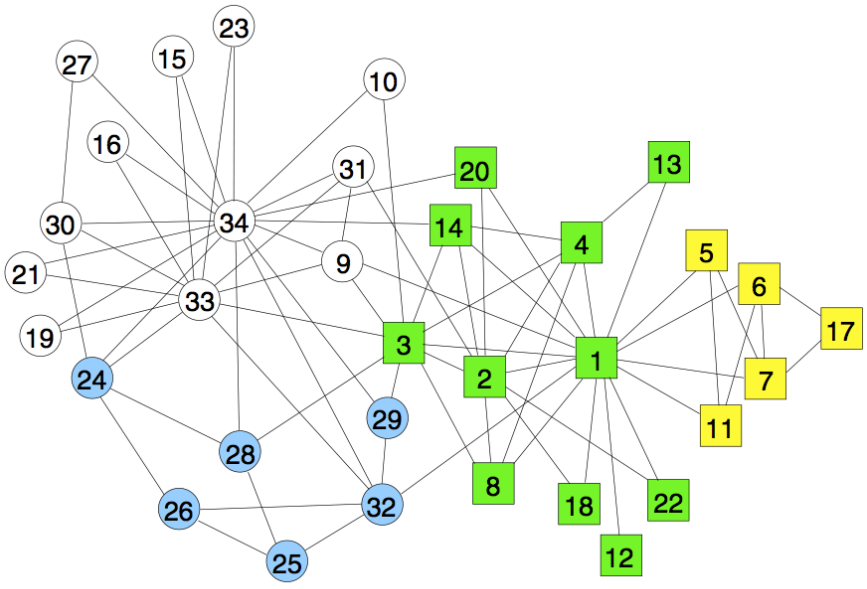

Fig.2 Zachary's karate club. This is the best partition found by the algorithm of Newman and Girvan [8]. This picture is copied from Ref.[15].

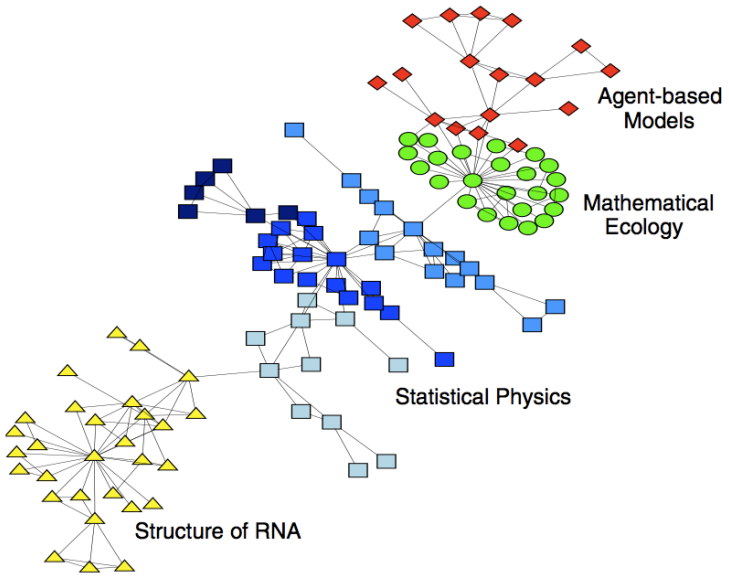

Fig.3 Collaboration network between scientists who work at the Santa Fe Institute [9]. This picture is copied from Ref. [15].

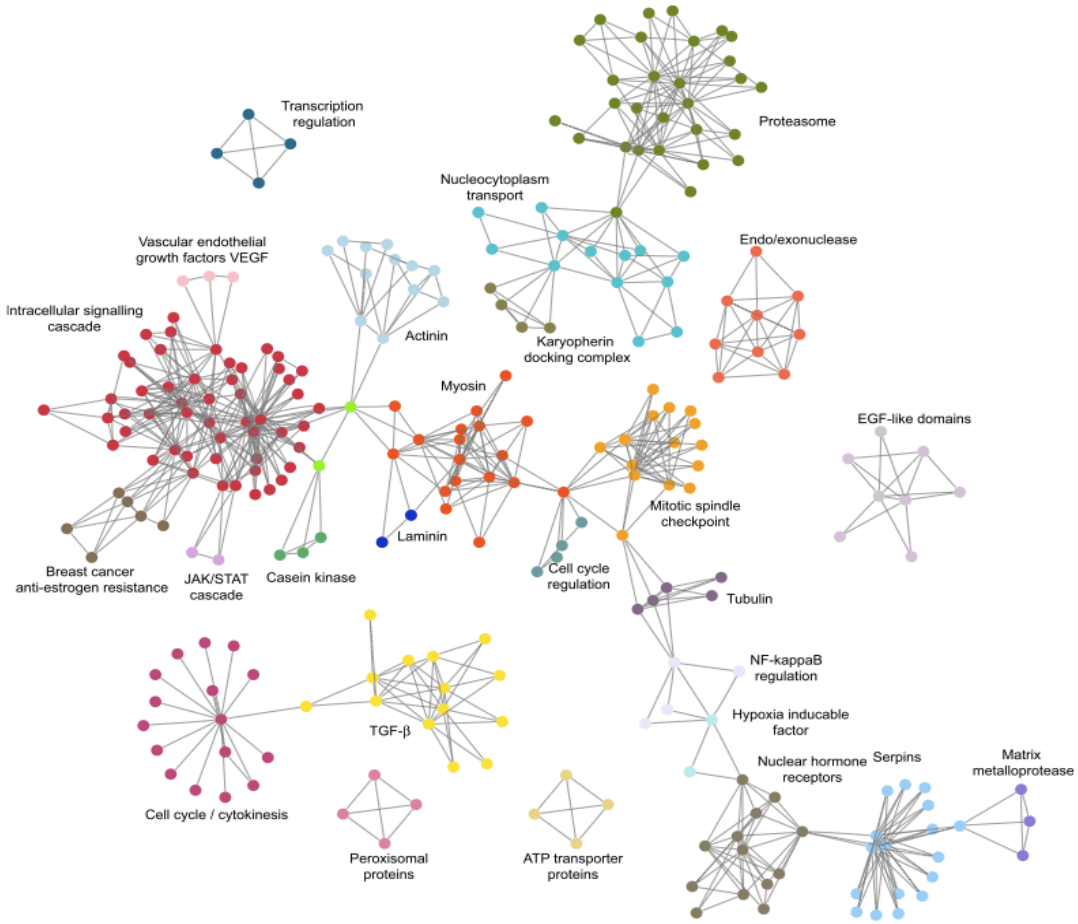

Fig.4 Community structure in the protein network of rat [10]. This picture is copied from Ref. [15].

In many cases of the networks, the relationship or interaction between elements of a system may have a definite direction. Predator-prey relationship in food webs is a distinct example. In Fig.5, a directed network is shown. The nodes in the network represent for the web pages in the World Wide Web, and the directional edges between them stand for the hyperlinks, which enable the users to move from one page to another one. Sometimes, page A has the hyperlink to page B, but it does not mean that page B has a link back to page A. So some edges in Fig.5 is unidirectional.

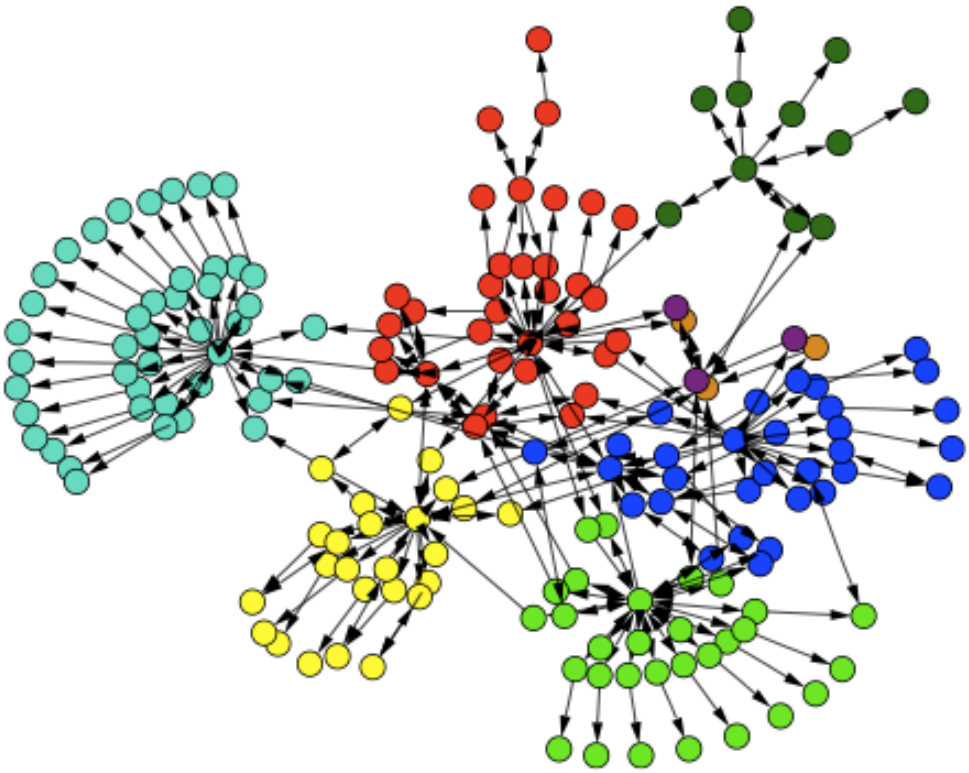

Fig.5 Sample of web pages of a web site and their hyperlinks. Communities were detected with the algorithm of Girvan and Newman [11]. This picture is copied from Ref. [15].

2.3 Elements of community detection

In fact, the problem of graph clustering is not clearly defined. The main element of this issue is that the concepts of community and partition are not strictly defined. However, some of the ambiguities conceal and there are many effective ways to solve them. Therefore, it does not have to have shared definitions for this problem.

Meanwhile, there is another problem. Structural clusters make sense when the vertices in the network are sparsely connected. For example, if the number of edges is m while the one of nodes is n, and $m >> n$, in this case, detecting community for this network may be a waste of time, as the vertices in this network interact with each other so intensively that it is better to treat as an integral community, rather than several ones.

For this problem, we need to find another solution, similar to data clustering [12], which requires a different concepts and modus. While the communities in graphs are obvious because of the density of the edges inside and outside the community, in data clustering, communities are groups of nodes that are closed to each other in a metric space.

2.3.1 Community complexity

The large amount of data in the real-world networks makes the efficiency of the clustering algorithms critical. The computational complexity of an algorithm is the probable resources and time an algorithm needs to finish a task. For community detection, the complexity of an algorithm is determined by the quantity of nodes and edges in a network. However, the exact complexity of an algorithm is hard or even impossible to be calculated, all we can do is to figure out the worst situation for an algorithm to perform a task and get the complexity in this case.

In the notation of $O(n^{å}m^{ß})$, n and m stands for the number of vertices and the number of edges respectively. This notation means that the complexity grows with the increase of the n and m. å and ß as exponents also have some influence on the computational time. Most of the time, lowest possible values of these two exponents are preferred, which means the lowest cost for this task to perform under a given algorithm. Sometimes, an algorithm with complexity of $O(n^{å}m^{ß})$ cannot handle some over large network, because its running time grows too much faster than $O(n)$ or $O(m)$.

Many clustering algorithms can deal with all the clustering problems. Even so, it may be too slow to handle large network, and an approximation algorithm is needed in this case, which does not give an exact solution to the problem but an approximate one, with the advantage of lower complexity. It may also give different solutions to the same problem, with different initial conditions or parameters of itself. No matter what case it is, the user should give probable bounds on the goodness of the approximate solution, so that the algorithms can feedback with respect to this kind of bounds. However, there is no absolute ways to quantify an algorithm is better than the others.

2.3.2 Community

(1) Basics

The first problem for community detection is what the community is, what is the standard to divide the nodes into groups. Unfortunately, there is no definition to be acceptable in any case. Actually, the definition is depended on what the specific network has in mind.

The basic for all definitions is that there are more edges in a community than the edges linking nodes between communities.

(2) Local definition

Communities are sets of nodes that have few connections with the rest of the network, so at certain conditions, we can treat them as a separate entity with specific property. And it makes sense to focus on the sub-networks, which are under examined.

Clique is an important concept in community. It is a group that each vertex in it has more connections with the rest part of a clique. Triangles are the most common clique, and are normal to see in a real network, while large cliques are less frequent.

However, if we divide communities according to this principle, clique maybe a problem to some study. All the nodes in a clique are obviously symmetrical, with no difference between them. But in many practical studies, we need to know the difference of nodes in a clique and, more importantly, which one is the crucial member that may have influence over the other ones. For example, a captain in a football team. However, it is possible to define the sub-networks, which are still clique-like without the notion of clique. For example, the properties that are not only common but also available in this group.

There is another standard that the vertex must have a minimum number of connections to the other ones in the same community. Cohesion is important in this criterion. More and more attention is put on the comparison of the internal and external degree of each node. Different definition is given according to the scope that needs it to be.

(3) Global definition

Communities can be also defined with regard to the whole network. That is because the sub-network is essential to be part of the network; it would be a huge damage to the whole if it were taken away. Take an understandable example, if a doctor want to study how the brain of an animal functions, he cannot take the brain away from the animal, he must keep the animal complete, if he did it on the opposite way, only will he get some bloody organs without any further cognition. So in most cases, definitions of communities are indirect ones, in which some global attributes of the network are used to generate the communities.

(4) Definition based on the similarity of nodes

Vertices in a community are normally similar to each other. In this notion of community, vertices are grouped together based on the common properties they have and regardless of whether they are connected to each other or not. This measure is the basic of most traditional method.

The number of edge- (or vertex-) independent paths between two vertices is another method worth considering. There is no shared edges or vertices, and their number is depended on the maximum flow that can be conveyed between these two vertices with some conditions [15]. For this method, there can be a problem as the number of the path may be uncountable, but using a weighted sum of number of paths can solve this.

2.3.3 Partitions

(1) Basics

Partition is a division of network that is divided into clusters, and each node belongs to at least one cluster [15]. In the real networks, there are many vertices that belong to different communities.

The number of possible partition in k clusters of a network with n vertices is the Stirling number of the second kind $S(n, k)$ [13]. The total number of possible partitions is the *n-th* Bell number $B_n = \sum_{k=0}^{n} S(n, k)$ [13]. In the limit of large n, B_n has the asymptotic form [14]

$$B_n \sim \frac{1}{\sqrt{n}}[\lambda(n)]^{n+1/2} e^{\lambda(n)-n-1},$$

where $\lambda(n) = eW(n) = n/W(n)$, $W(n)$ being the Lambert W function [39]. Hence, with the graph size of n, B_n grows more rapidly than exponentially. Also, listing and evaluating of all sub-network is impossible, only if there are few vertices in the network.

Partitions can be ordered hierarchically, when there are different levels of community at different scales in a network. In such a situation, a network always has a community inside the other bigger community, and so on (Fig.6). For example, we can divide a crowd of children as they attend different school. When we come into a school, they should belong to different grades or classes. It is common to have this kind of network in the real world.

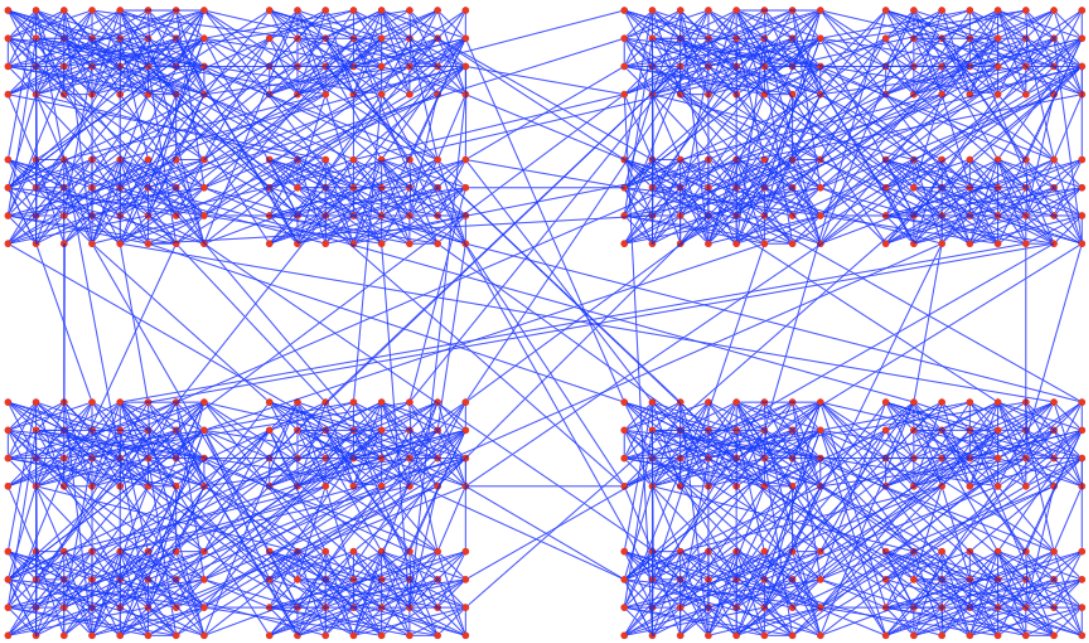

Fig.6 A simple example of a hierarchical graph. Sixteen clusters with 32 nodes, and every four of them form a larger community [16]. This picture is copied from Ref. [15].

There is another kind of hierarchical structure of a network – dendrogram, Fig.7 is an example. It looks like a tree in computer science. All the nodes are at the bottom, and as going upward, they gradually form groups. The mergers of them are shown by a horizontal line. The top level indicates the whole network as a single community. Cutting a graph at some height will result in a partition of this network, and a community in a lower level in completely included in a community at higher level. It

is used generally, as it can show the relationship of affiliation and inheritance between clusters.

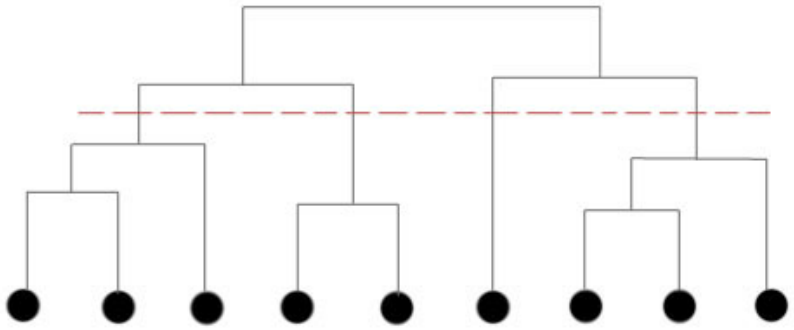

Fig.7 Dendrogram. The red horizontal line cut the graph shows a partition in communities.

(2) Quality function: Modularity

Good partitions need to be identified, that is what a reliable algorithm is supposed to do. But how can an algorithm know what is a good partition? There should be some rules to do this task. In the field of data clustering, Jon Kleinberg has given his own theory [17].

With a set *S* of points, and a distance function *d* is defined, clusters can be found based on the distance between the points. Kleinberg demonstrated that no clustering could satisfy all the following attributes at the same time:

1. Scale-invariance: Given any constant *a*, multiplying any distance function *d* by *a* provides the same clustering [17].

2. Richness: if a decent distance function *d* is given, recovering a partition of the given point set is possible [17].

3. Consistency: given a partition, any change to the distance function does not effect the clusters that decrease the distance between two points of different cluster or increase the distance between two points of the same cluster [17].

However, this theorem cannot be used everywhere especially in graph clustering, because general defined distance function is not suitable for an incomplete network [15]. In a generic network, the last two properties are always well defined except the first property, as the distance function is crucial in this case. Richness means that if the partition is the resulting network's natural outcome, for example, edges can be set between the vertices in the same cluster.

Nowadays, many algorithms are able to distinguish useful partitions, but it does not mean that the partition found in this situation is good as well. Therefore, a quantitative standard is helpful in this case. It can assign a score to the partition of a network, and users can know which partition is best based on the score given by the quantitative function. However, user should still bear in mind that how a partition is good in a specific condition.

The maximum modularity of a network increases as the scale of it grows [18]. Therefore, modularity should be used for comparison of the quality of community structure of those networks that are similar in size. Modularity is zero if the whole network is taken as a single community. Modularity is always not larger than one, not include one, and it can be negative as well [15]. Modularity can also indicate that if all the partition with a negative number, there is no community structure in the network. On the contrary, large negative modularity numbers means that there should be sub-network that has few internal edges and many external edges between them [19].

Modularity is widely used in many community detection algorithms. It enables users to evaluate the stability of partitions [20], can be used in visualization for designing layouts [21], and implements renormalization of a network, by conversing a network into a smaller one without changing the community structure [22].

2.4 Clustering Algorithms

Clustering

In community detection, clustering is commonly used. It assigns a set of nodes to subset, and the nodes in the same subset are similar to some extent, which help to study the whole set of nodes.

The followings are the three main kinds of clustering.

A. Traditional methods

1) Hierarchical algorithm

It is used to find clusters in the previously established clusters. It can be either bottom-up or top-down. The bottom-up algorithm treats all the nodes as one cluster itself at first and gradually merge them into a large one, while the top-down algorithm does in the opposite way, it treats the whole set of nodes as a cluster and then divides it into subsets.

In computer science, the Huffman tree is a typical example of the hierarchical algorithm. It always chooses *n* smallest nodes in the whole set to group into sub-tree, until all the vertices are grouped.

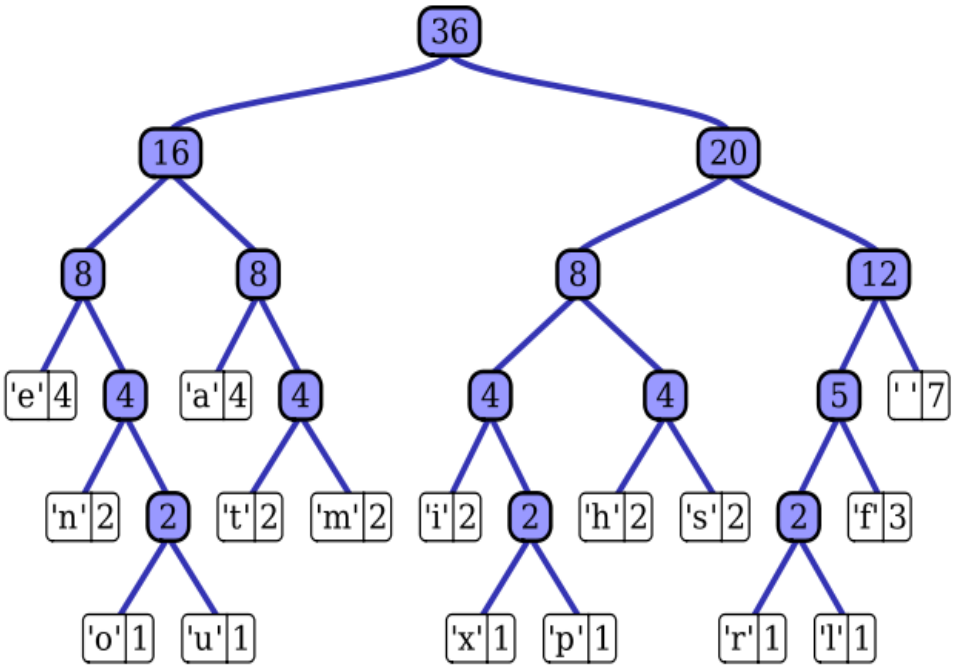

Fig.8 An Example of Huffman tree. Taken from http://en.wikipedia.org/wiki/Huffman_tree.

2) Partition algorithm

This kind of algorithm determines all clusters at the very beginning time, but it is so similar to the top-down algorithm in the hierarchical algorithm. The number of clusters in this kind of algorithms is always assigned previously, for example, *k*. Every node in the network is placed in a specific metric space, so the dissimilarity between nodes is always measured by the distance between them. And the initial *k* centroids are very important to get a good partition of the network.

K-means is famous in this kind of clustering. It assigns every vertex to the cluster whose center is the closest to it. And the center of the cluster is the average of all the points in it, like a center of gravity in an inerratic object.

The simple steps of k-means is shown:

1. Choose the number of cluster, *k*.
2. Randomly generate *k* clusters or choose *k* points to be the center of clusters.
3. Assign the other point to its nearest cluster center.
4. Recomputed the new centers of clusters.
5. Repeat the previous two steps until all the nodes in the network are assigned to a cluster and the network is in a state of equilibrium.

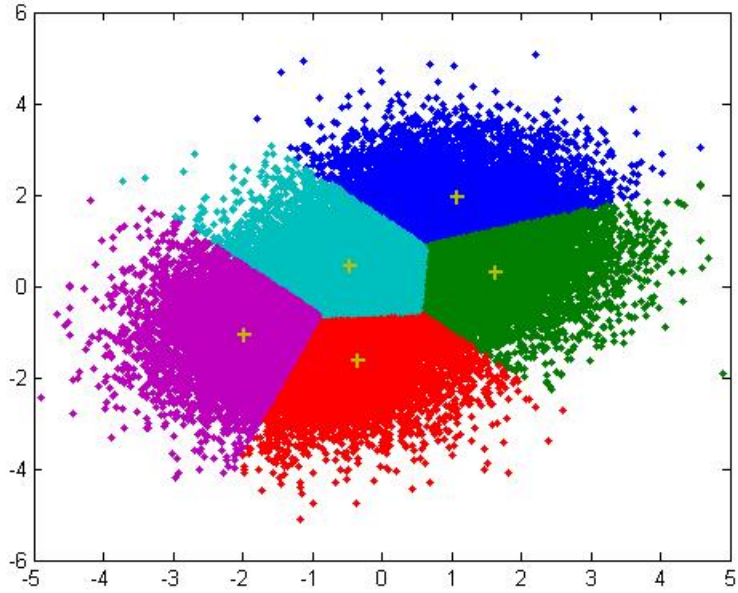

Fig.9 Example of k-means clustering. Taken from http://www.mathworks.com/matlabcentral/fileexchange/19344-efficient-k-means-clustering-using-jit

In Fig.9, there is one cross in every area, which is the center of every part. K-means keeps re-computing the center of them when a new member added into, and stops the calculation until there is no centroid move in the last iteration.

The limitation of this kind of algorithms is that the number of the clusters must be specified at the very beginning, rather than derive it in the computing process. What is more, placing nodes in a specific metric space may be natural for some networks, but all networks.

3) Spectral clustering

For this kind of algorithms, one thing we have to know first is the similarity matrix, which indicates the similarity between two nodes by a matrix of scores. It has a strong relationship with distance matrices and substitution matrices.

For a specific network N, the similarity matrix S is the one where S_{ij} indicates the similarity between nodes $i, j \in N$. The spectrum of the similarity matrix of the network will be used by spectral clustering, which performs a function called dimensionality reduction, so it will cluster in fewer dimension.

One of the most frequent used algorithms is the *Normalized Cuts Algorithm* by *Shi-Malik*, widely used in image segmentation. Most of the time, a network is partitioned into two sub-networks (N_1, N_2) based on the eigenvector v relate to the second-smallest eigenvalue of the *Laplacian matrix* [40].

$$L = I - D^{-1/2} N D^{-1/2}$$

of N, where D is the diagonal matrix

$D_{ii} = \sum_j N_{ij}$.

This partition can be done in different ways, for example, choosing the median *m* of the component in *v*, and then group the nodes whose component in *v* is larger than m into subset N_1, the rest to N_2.

It may sound strange to cluster nodes in network through the eigenvector based on the similarity matrix, especially when directly clustering the initial network is possible. Nevertheless, the reason for this is that the eigenvector indicates the change of the representation, and this can help to make the cluster attributes of the initial network much more manifest. Spectral clustering sometimes can divide nodes in the network, which may not be solved by k-means clustering, as it uses k-means to cluster the coordinates, which have already been transformed.

To some extent, spectral clustering can be said to be network partitioning. The measure of minimization of the ratio cut and normalized cut can be expressed in matrix form, acquiring similar cut size, with index vector grouping the network through the value of their entries. Then the problem is how to partition the network with the vectors, which can be done by k-means clustering.

Spectral clustering also has a great influence on random walks. The minimization of the number of the edges between forces the random walkers to spend more time within clusters rather than between clusters. Especially, unnormalized spectral clustering, which has the Laplacian $\mathbf{L}_{rw}$, links with random walks naturally, because $\mathbf{L}_{rw} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, where $\mathbf{D}^{-1}\mathbf{A}$ can be considered as the transfer matrix $\mathbf{T}$. This has interested some researchers to study. Meila and Shi have been able to demonstrate that the normalized cut for a bipartition is equivalent to the total possibility of movement of the random walker between clusters [66]. In this point of view, minimizing the normalized cut is equal to finding a partition, which minimizes the possibility of transition from one cluster to another.

B. Divisive algorithms

Finding the edges lying between different communities and then removing them is a way to find communities in a network. Therefore, clusters disconnect form each other. How to find an attribute of the inter-community edges, which allow to be identified, is important to this kind of algorithms. But there is no guarantee that the edge removed connects nodes from different clusters with low similarity. In some cases, the algorithm may remove several nodes or whole sub-networks, not a single edge.

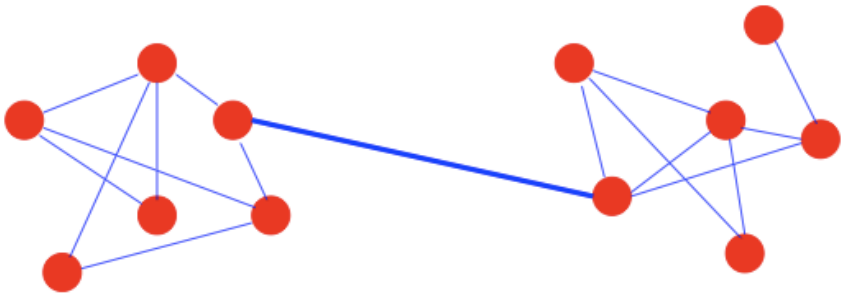

Fig.10 Edge betweenness is the most common measure for the calculation in divisive algorithm. In the figure, the edge between clusters has the highest betweenness of all the edges in the network, as all the shortest paths from one cluster to the other must run through it [6]. This picture is copied from Ref. [15].

Girvan and Newman have proposed an algorithm that is being used commonly, which is always known as GN algorithm. This algorithm is so historically important that is opened a new era in the area of community detection. The edges are chosen according to their value of edge betweenness. The steps for this algorithm is:

1. Calculate the betweenness of all the edges in the network.
2. The edge that has the highest betweenness is removed from the network.
3. Recalculate the betweenness of all edges, which are still in the network and are affected by the removal.
4. Repeat the step 2 and step 3 until all the edges are removed.

Edge betweenness is the number of shortest paths between all nodes pairs of the network that run through the edge. It also expresses how important the edge is in the processes like information spreading, as information tends to flow along the shortest paths. Anthonisse has proposed a theory that the inter-community edges have a high edge betweenness, as many shortest paths connecting nodes from different communities will run through them [41].

Although there are some other centrality measure like current-flow and random walk, edges betweenness is the fastest of them, and in practice, Girvan-Newman algorithm adopting edge betweenness gets better results than with others [11]. The step 3 of the Girvan-Newman algorithm is proved to be the most important to detect communities in a lot of studies. If network has a strong community structure, breaking into communities is quick, and only the connected component, which has the last removed edge, needs to be recalculated, as the other betweenness remain the same. In this case, it helps to save time for the computing, but it cannot estimate the gain as it depends on the network studied. The original Girvan-Newman algorithm has no criterion to define the best partition [9], but in the recent refinement [11], the partition with the largest value of modularity is chosen as the best one, which is frequently used from that time on. This method can also easily deal with weighted network, by calculating the edge betweenness appropriately. The betweenness of a weighted edge is the same as the one of an edge in the unweighted network, just divided by the weight of edge [42].

Tyler et al. has modified the Girvan-Newman algorithm to save the time of calculation [43, 44]. The Girvan-Newman algorithm will compute edge betweenness from all vertices and calculate the contribution to betweenness from all paths, which start at the node of the network, and repeat it for all the nodes in the network [11, 45, 46]. But Tyler et al.'s modified algorithm just computes the contribution of the limited randomly chosen centers. The stopping criterion of this modified algorithm is quite different; it relies on the definition of community, rather than the calculation of the modularity on the results [15]. However, the final partition of the network is depended on the initial set of center nodes, but it could be solved by repeating the calculation for several times to get a good result [47].

However, the weakness of the Girvan-Newman algorithm is that it cannot find overlapping communities, because it always assigns every node to only one cluster. Several studies have proposed to solve this problem. Steve Gregory [49] has developed an approach, named CONGA, in which a maximum value of betweenness of the edge is set, and the network is split if the nodes' site betweenness exceeds the maximum. Gregory introduces split betweenness in his modified algorithm; this new measure will calculate the number of shortest paths that would pass that two part of the nodes if they were split. Some edges of the split nodes will be assigned to one of the node's duplicates, and the rest to the other [15]. The code can be found at http://www.cs.bris.ac.uk/~steve/networks/index.html.

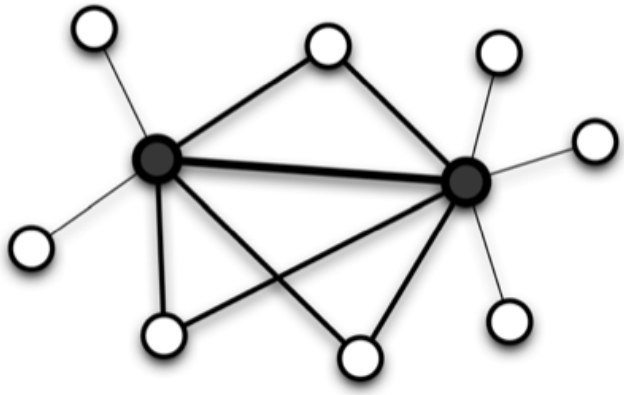

Fig.11 The right part and the left part are similar to each other. But the black nodes seem that they can be assigned to either community [48]. This picture is copied from Ref. [15].

C. Modularity-based methods

Modularity is used to define when to stop the Girvan-Newman algorithm, but it has been a crucial part of many clustering algorithms. It helps to understand the clustering problem, and how good the partition is.

Modularity optimization

In the definition, higher value of the modularity is considered to be a better partition. Especially when the partition's modularity value is closed to the maximum,

it should be a very good partition at least. However, the maximum of modularity is impossible to reach, as it is proved to be an NP-complete problem [50]. But there are still several algorithms that can find a good modularity in reasonable time.

1) Greedy techniques

The first modularity-clustering algorithm using greedy techniques was proposed by Newman [51]. It is an agglomerative clustering method, where nodes are merging and as a result, modularity increases. For a network containing n nodes, it has n clusters. At first, there is no edge in it. Edges are added one by one. Once a new edge is added, the network has a new partition. The edges added is always the one that will increase the modularity most based on the previous configuration. The procedure will continue until all the edges are added to the network. If the partition does not change when an edge is inserted, modularity stays the same. The largest modularity in the subset of partitions is the approximate largest modularity, and the algorithm will always choose the merger that can lead to largest increase of modularity. Merging communities, between which there are no edge, cannot ever lead to any change of modularity.

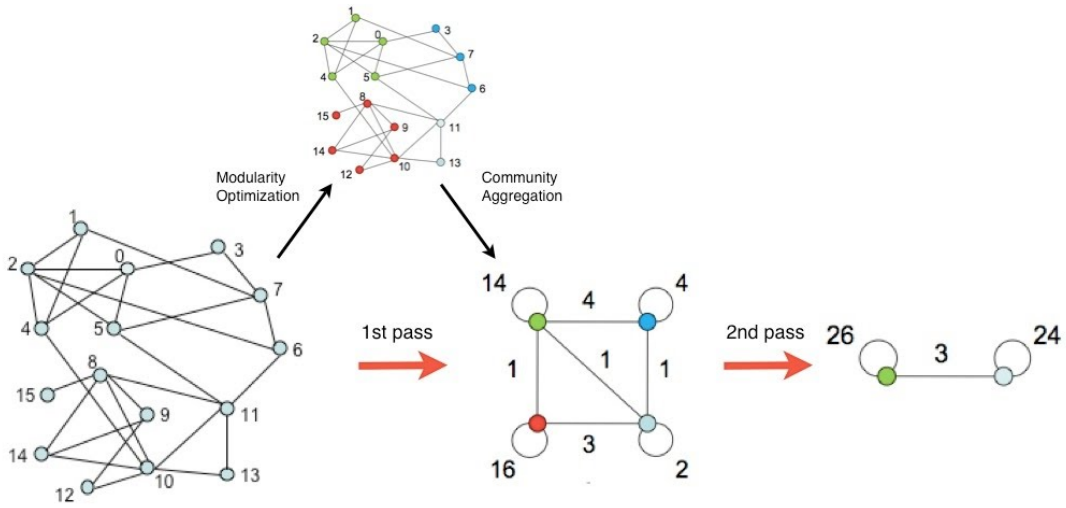

Fig.12 This is an example of modularity optimization. The method starts from the graph on the left. Nodes are assigned to the local cluster that will generate the largest modularity. Taken from http://sites.google.com/site/findcommunities/.

Blondel et al. have introduced a different approach. At first, all nodes of the network are belonging to different communities. The algorithm will compute the gain of the modularity once a given node is put in the community of its neighbor, and then choose the one that will result in largest increase of modularity. This step will end with a first level partition. In the following step, the clusters will be replaced by supervertices, and these supervertices are connected if there is one edge between the nodes of the communities they represent. The weight of the corresponding edges is the sum of the one between the original communities. The procedure of this algorithm can be viewed from the above figure (Fig.12).

2) Simulated annealing

Guimera et al. [52] first employed simulated annealing in their modularity optimization. It contains two kinds of moves: taken one node randomly and put it to from one cluster to another, which is called local moves, and global moves, including merging and splitting communities.

Simulated annealing is a probabilistic procedure, which explores the space of possible states and looks for the maximum [15]. The increase of the maximum will result in states transitions with probability 1, while the decrease of it with a probability $\exp(\beta \Delta M)$, where ΔM is the decrease and β is an index, which increases after each iteration [53].

3) Extremal optimization

Extremal optimization is a heuristic search procedure, which is based on the optimization of local variables. The local modularity of a node is the value of the sum on this node. Dividing the modularity by the degree of nodes will obtain the fitness measure. At each iteration, the node with the lowest fitness is put to the other community as random initial partition with two groups [15]. This move results in the change of the partition, so the local fitness needs to be recomputed. The procedure will continue until the global modularity, which is the sum of the local modularity, remains the same after the previous move [15].

2.5 Layout algorithms

While clustering is often used for community detection, most of the time, layout algorithm is used to place the vertices of a network in an aesthetically pleasing way. The purpose of layout algorithm is to place the nodes of a network into a two-dimensional or three-dimensional space according to the coordinate that has been get by some community detection algorithms. A combination of constrains, rules and parameters is used to arrange the nodes of a network.

It assigns forces to set of vertices and edges. The whole network is then simulated as a physical system. The forces are applied to nodes, pulling them closer together or pushing them apart from each other. And this repeats until all the vertices are in an equilibrium state. At this time, the coordinates of every node can be fixed. And the edges tend to be of nearly the same length and not to cross over the other one. Force-directed algorithm is typical case in layout algorithm, which calculates layouts of simple undirected networks. It is also known as spring-embedders, which use the information of the network structure to generate the layout, but domain-specific knowledge. Generally, force-directed algorithms define an objective function, which gives the nodes' coordinates so that they can map into an R^d metric spaces representing the energy of the layout. This function is designed to give the layout lowest energies by put adjacent nodes closer to each other, while put those non-adjacent ones in a relative distant away.

2.5.1 Different kinds of Layout Algorithms

1) The Barycentric method

This algorithm is the first "force-directed" algorithm, used by Tutte [23] in 1963. It obtains a straight-line, crossing free drawing for a given 3-connected planar graph [25]. The idea for this algorithm is that if a planar graph can be assigned to a plane, then finding suitable positions for the remaining vertices can be found by solving a system of linear equations [23].

A typical example for this algorithm is summarized by Di Battista et al. [24]:

Barycenter-Draw
Input: $G = (V, E)$; a partition $V = V_0 \cap V_1$ of V into a set V_0 of at least three *fixed* vertices and a set V_1 of *free* vertices; a strictly convex polygon P with $|V_0|$ vertices
Output: a position p_v for each vertex of V, such that the fixed vertices form a convex polygon P.
1. Place each fixed vertex $u \in V_0$ at a vertex of P, and each free vertex at the origin.
2. **repeat**
   foreach free vertex $v \in V_1$ **do**
   $x_v = \frac{1}{deg(v)} \sum_{(u,v)\in E} x_u$
   $y_v = \frac{1}{deg(v)} \sum_{(u,v)\in E} y_u$
until x_v and y_v converge for all free vertices v.

The force at a vertex V is described by

$$F(v) = \sum_{(u,v)\in E} (p_u - p_v)$$ [54],

where p_u and p_v are the positions of vertices u and v.

However, there is a drawback of this algorithm, which always results in poor vertex resolution [25].

2) Spring system and electrical force

The spring layout algorithm of Eades [26] was first used to assign up to 30 vertices and produced an "aesthetically pleasing" 2D layouts for plotters and CRT screens [25]. Here is the pseudo-code of it:

---

algorithm SPRING(G:graph);
place vertices of G in random locations;
repeat M times
    calculate the force on each vertex;
    move the vertex $c_4 * (force\ on\ vertex)$
draw graph on CRT or plotter.

In 1991, Fruchterman and Reingold add "even vertex distribution" to the earlier two criteria, and every vertex in their system exerts attractive and repulsive forces from each other. The attractive and repulsive forces are defined as

$$f_a(d) = d^2/k, \qquad f_r(d) = -k^2/d \text{ [27]},$$

so that the distance *d* between two nodes are defined as

$$k = C\sqrt{\frac{area}{number\ of\ vertices}} \text{ [27]}.$$

The pseudo-code for the Fruchterman and Reingold gives us insight into the workings of a spring-embedder [54]:

area:= $W * L$; {W and L are the width and length of the frame}
$G := (V, E)$; {the vertices are assigned random initial positions}
$k := \sqrt{area/|V|}$;
function $f_a(x)$:= **begin return** x^2/k **end**;
function $f_r(x)$:= **begin return** k^2/x **end**;
for $i := 1$ **to** *iterations* **do begin**
  {calculate repulsive forces}
  for v in V **do begin**
    {each vertex has two vectors: *.pos* and *.disp*
    $v.disp := 0$;
    for u in V **do**
      if $(u \neq v)$ **then begin**
        {δ is the difference vector between the positions of the two vertices }
        $\delta := v.pos - u.pos$;
        $v.disp := v.disp + (\delta/|\delta|) * f_r(|\delta|)$
      end
  end
  {calculate attractive forces}
  for e in E **do begin**
    {each edges is an ordered pair of vertices *.vand.u*}
    $\delta := e.v.pos - e.u.pos$;
    $e.v.disp := e.v.disp - (\delta/|\delta|) * f_a(|\delta|)$;
    $e.u.disp := e.u.disp + (\delta/|\delta|) * f_a(|\delta|)$
  end
  {limit max displacement to temperature t and prevent from displacement outside frame}
  for v in V **do begin**
    $v.pos := v.pos + (v.disp/|v.disp|) * \min(v.disp, t)$;
    $v.pos.x := \min(W/2, \max(-W/2, v.pos.x))$;
    $v.pos.y := \min(L/2, \max(-L/2, v.pos.y))$
  end
  {reduce the temperature as the layout approaches a better configuration}
  $t := cool(t)$
end

3) Graph theoretic distances approach

Kamada and Kawai [28] introduced a new way of generate better graph layouts in 1989. They take the graph theoretic approach to ensure the vertices are not too close to each other. The strength of the spring between vertices *i* and *j* is defined as

$$k_{i,j} = K/d_{i,j}^2$$ [54],

where K is a constant. This can lead to an overall energy function when treating an n-vertices graph:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} (|p_i - p_j| - l_{i,j})^2$$ [54].

If given with coordinates in the 2D Euclidean plane x_i and y_i, above function can be rewrite as:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} \left((x_i - x_j)^2 + (y_i - y_j)^2 + l_{i,j}^2 - 2 l_{i,j} \sqrt{(x_i - y_i)^2 + (y_i - y_j)^2} \right)$$ [54].

Kamada and Kawai's algorithm is to minimize the energy function E. All the partial derivatives are set to 0 at local minimum. Their algorithm keeps compute a stable position on particle p_m at a time. And each step, the particle p_m with the largest value of Δm is chosen, where

$$\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2}$$ [54].

This leads to the following algorithm [54]:

compute $d_{i,j}$ for $1 \le i \ne j \le n$;
compute $l_{i,j}$ for $1 \le i \ne j \le n$;
compute $k_{i,j}$ for $1 \le i \ne j \le n$;
initialize $p_1, p_2, \ldots, p_n$;
while $(max_i \Delta_i > \epsilon)\{$
    let p_m be the particle satisfying $\Delta_m = max_i \Delta_i$;
    while $(\Delta_m > \epsilon)\{$
        compute δx and δy by solving the following system of equations:
        $\frac{\partial^2 E}{\partial x_m^2}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial x_m}(x_m^{(t)}, y_m^{(t)})$;
        $\frac{\partial^2 E}{\partial y_m \partial x_m}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial y_m^2}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial y_m}(x_m^{(t)}, y_m^{(t)})$
        $x_m := x_m + \delta x$;
        $y_m := y_m + \delta y$;
    }
}

4) Large graphs

The algorithms described above are performing badly when they deal with large networks. In order to solve this problem, Handany and Harel introduced a multi-scale technique in 1999. They suggested put contractions on the edge so as to preserve certain properties of the graph: cluster size, vertex degrees and homotopy [29]. This algorithm can generate good layouts for much larger graph in reasonable time [29].

Harel and Koren introduced an algorithm similar to Hadany and Harel, but based on a k-centers approximation [30]. Their algorithm is also relies on Breadth-First search [25]. The Harel and Koren algorithm's pseudo-code is [54]:

Layout$(G(V,E))$
% Goal: Find L, a nice layout of G
% Constants:
% Rad[= 7] – determines radius of local neighborhoods
% Iterations[= 4] – determines number of iterations in local beautification
% Ratio[= 3] – ratio between number of vertices in two consecutive levels
% MinSize[= 10] – size of the coarsest graph
  Compute the all-pairs shortest path length: d_{VV}
  Set up a random layout L
  $k \leftarrow MinSize$
  while $k \leq |V|$ **do**
    $centers \leftarrow$**K-Centers**$(G(V,E),k)$
    $radius = \max_{v \in centers} \min_{u \in centers}\{d_{vu}\} * Rad$
    LocalLayout$(d_{centers \times centers}, L(centers), radius, Iterations)$
    for every $v \in V$ **do**
      $L(v) \in L(center(v)) + rand$
    $k \leftarrow kRatio$
  return L

K-Centers$(G(V,E),k)$
% Goal: Find a set $S \subseteq V$ of size k, such that $\max_{v \in V} \min_{s \in S}\{d_{sv}\}$ is minimized.
  $S \leftarrow \{v\}$ for some arbitrary $v \in V$
  for $i = 2$ to k **do**
    1. Find the vertex u farthest away from S
      (i.e., such that $\min_{s \in S}\{d_{us}\} \geq \min_{s \in S}\{d_{ws}\}, \forall w \in V$)
    2. $S \leftarrow S \cup \{u\}$
  return S

LocalLayout$(d_{V \times V}, L, k, Iterations)$
% Goal: Find a locally nice layout L by beautifying k-neighborhoods
% $d_{V \times V}$: all-pairs shortest path length
% L: initialized layout
% k: radius of neighborhoods
  for $i = 1$ to $Iterations * |V|$ **do**
    1. Choose the vertex v with the maximal Δ_v^k
    2. Compute δ_v^k as in Kamada-Kawai
    3. $L(v) \leftarrow L(v) + (\delta_v^k(x), \delta_v^k(y))$
  end

2.5.2 Advantages

1). Good quality of result: the result is obtained in an aesthetic state.

2). Flexibility: it is easy to adapt force-directed algorithms and add some other standards, which also make it popular for graph drawing.

3). Intuitive: because the basic of force-directed algorithms is physical analogies of common objects, and it is easy to understand and predict.

4). Simplicity: force-directed algorithms are simple and easy to implement.

5). Interactivity: users can know how the vertices are assigned to a cluster and watch the process from a mess to a good-looking layout.

2.5.3 Disadvantages

So many advantages they have, disadvantage is unavoidable.

1). High running time: the complexity of time of force-directed algorithms is $O(n^3)$, where n is the quantity of vertices in the network. Because each node needs to be computed, and it is estimated to be $O(n)$, and every node needs to be compared with the other nodes, it is $O(n\text{-}1)$. The repulsive forces of every vertex are computed between one nodes and all the others, that means the complexity is $O(n^3)$ now.

2). Poor local minima: it is easy to found a negative minimal energy of the whole network. In many cases, this kind of communities can be result in a low-quality drawing. It is sometimes influenced by the initial input. And this situation goes worse when the number of vertices grows, so it need some other application to solve this problem.

2.5.4 Energy models for layouts

As representations of the cluster structure, energy models are useful than common partitions of the network. Because they do not just assign the nodes to their closest cluster center, the distance between them also indicates the relation between them. A node is set closely to the cluster means they are strongly related.

Here is a typical energy model for layouts - (a, r)-energy model. In a d-dimensional layout p, each vertex v of a network maps with a position p_v in R^d, a distance between them is also assigned with an edge. It is important for the quality measure for layouts. Smaller energy means this layout is better than the other.

Two most popular models are stress functions of multi-dimensional scaling and force systems of pair-wise attraction and repulsion between vertices. But for this time, I will focus on the latter type.

In the force system, vertices tends to be pull together by the connected vertices, however, the repulsion push them apart at the same time. Formally, a layout p and two vertices u and v, $u \neq v$, the attractive force exerted on u by v is

$w_{\{u,v\}} \|p_u - p_v\|^a \overrightarrow{p_u p_v}$ [21],

and the repulsion is

$w_u w_v \|p_u - p_v\|^r \overrightarrow{p_v p_u}$ [21],

where $\|p_u - p_v\|$ is the distance between u and v, $\overrightarrow{p_u p_v}$ is the unit-length vector from u to v, while a and r are real constants with $a > r$, which make sure the attraction grows faster than the repulsion [21].

2.6 Previous work on layout for community detection

2.6.1 Modularity clustering is force-directed layout

In one of Andreas Noack's paper, he has demonstrated that modularity clustering is force-directed layout [21]. Existing theoretical results already show that layout of (a, r) – energy model can reflect the community structure of a network, so do clustering with modularity.

And he also found that modularity measure is mathematically similar to the density.

2.6.1.1 Transformation of modularity into (a, r)-energy

The modularity of a clustering p can be defined as

$$\sum_{c\in p(V)}\left(\frac{w_{\{c,c\}}}{w_{\{V,V\}}}-\frac{\frac{1}{2}w_c^2}{\frac{1}{2}w_V^2}\right),$$

the difference between the actual and expected fraction of intra-cluster edge weight [21].

And each edge is either inside a cluster or between clusters, so the fraction of intra and inter cluster edges weight can add up to 1:

$$\sum_{c\in p(V)}\frac{w_{\{c,c\}}}{w_{\{V,V\}}}+\sum_{\{c,d\}\subseteq p(V):\,c\neq d}\frac{w_{\{c,d\}}}{w_{\{V,V\}}}=1$$ [21];

similar to the corresponding expected fraction. Therefore, the modularity of clustering p can be written in this term:

$$\sum_{\{c,d\}\subseteq p(V):\,c\neq d}\left(-\frac{w_{\{c,d\}}}{w_{\{V,V\}}}+\frac{w_c w_d}{\frac{1}{2}w_V^2}\right)=-\sum_{\{u,v\}\subseteq V:\,p_u\neq p_v}\left(\frac{w_{\{u,v\}}}{w_{\{V,V\}}}-\frac{w_u w_v}{\frac{1}{2}w_V^2}\right)$$ [21].

k is the number of clusters in p. Without modifying p, the k clusters can be conveyed in a space R^{k-1}, and distance 1 is defined as the distance between each pair of different clusters [21].

Because the distances between each pair of nodes are 0 or 1, the modularity p can be written in this form:

$$-\sum_{\{u,v\}:\,u\neq v}\left(\frac{w_{\{u,v\}}}{w_{\{V,V\}}}\|p_u-p_v\|^{a+1}-\frac{w_u w_v}{\frac{1}{2}w_V^2}\|p_u-p_v\|^{r+1}\right)$$

for all a, r belong to R with $a > -1$ and $r > -1$ [21].

This transformation should have its conditions. Noack has pointed them out in his paper:

1). Most energy models for layout are to produce an easy-read layout for the user in box-and-line visualization, which show the network with community structure [21].

2). Energy models encourage the nodes to be put at different place, although they should in the same position according to the result of computation. So the nodes will not overlap on the other one [21].

3). Vertices are unassigned weight in both modularity and energy model. But according to some recent study, vertices assigned with weight may display a network better [21].

However, in most of the studies, analysts tend to use respective quality measures for clustering and layouts. Some clustering algorithms get the network community for layouts to visualize it, and this seems to be all the work layouts can do.

2.6.1.2 Energy layouts conform to modularity clustering

Layouts do inferior to clustering in high-dimensional structure, but it can provide information that may miss in clustering:

1. The relationship between clusters [21].
2. The internal structure of clusters [21].
3. The density between vertices and clusters, and this also indicates the relationship between them [21].

There is another condition for these interpretations to be shown in a layout. That is layouts and clustering group nodes according to the same standard [21].

However, in previous work, some authors never consider groups of vertices in force-directed layout as clusters, while the others think it is wrong to support such interpretations.

2.6.1.3 Evidence

With description in the previous subsections, (a, r)-energy model can be equal to modularity measure, if clusterings with k clusters is able to be considered as (k - 1)-dimensional layouts under the condition of $a > -1$ and $r > -1$.

On the one hand, $r > -1$ is necessary for several vertices to be assigned to the same cluster, but not for layouts to put several vertices at the same position. On the other hand, the values of a and r have little influence on clustering as the distance between two nodes is either 0 or 1, but it is important for layouts. The following are the condition when the layouts can reproduce modularity clustering:

1. $a > r$, $a \geq 0$ and $r \leq 0$ [21].
2. $a \approx 0$, in this case, distance do not mean the length of paths [21].
3. $a - r \approx 1$, or at least $a - r$ not too much larger than 1. In this case, densities can be shown by the distances [21].

2.6.2 LinLog energy model

In this section, introduction of the energy model, which used in Andreas Noack's *LinLogLayout*, and the cut ratio as the measure to indicate the coupling of the two sets of nodes that are disjoint will be given. The distance between each cluster and the remaining network is interpretable with the cut ratio.

2.6.2.1 Definition of the energy model of *LinLog*

The drawing p based on energy model of *LinLog* can be defined as

$$U_{LinLog}(p) = \sum_{\{u,v\}\in E} ||p_u - p_v|| - \sum_{\{u,v\}\in V^{(2)}} \ln ||p_u - p_v||$$ [55],

where $U_{LinLog}(p)$ is the LinLog energy of the network. $\sum_{\{U, v\}\in E}||p_u - p_v||$ is the attraction between two adjacent nodes, while $\sum_{\{u, v\}\in V(2)} \ln ||p_u - p_v||$ is the repulsion between any two different nodes.

There is another thing need to make sure is that the position of nodes should be different, to avoid infinite energies.

2.6.2.2 The cut ratio

The cluster will represent a set of nodes that has many internal edges and few edges to the nodes outside the cluster. That is high cohesion, and low coupling.

Formally, cluster will defined by a measure of coupling: the cut ratio. For a cut (V_1, V_2), the cut ratio is

$$\text{cutratio}(V_1, V_2) = \frac{|E[V_1, V_2]|}{|V_1| \cdot |V_2|}$$ [55].

The cut ratio is normalized to make its interpretation independent of the size of V_1 and V_2. Normally, the value of cut ratio is expected to be equal for all cuts of random network, while the edges between V_1 and V_2 is still influenced by the size of V_1 and V_2.

Chapter 3: Software Design and Implementation

3.1 Overview

Java is used as programming language, because Java is platform independent, it is easy to run on many platforms.

The diagram shows the structure of LayoutClustering:

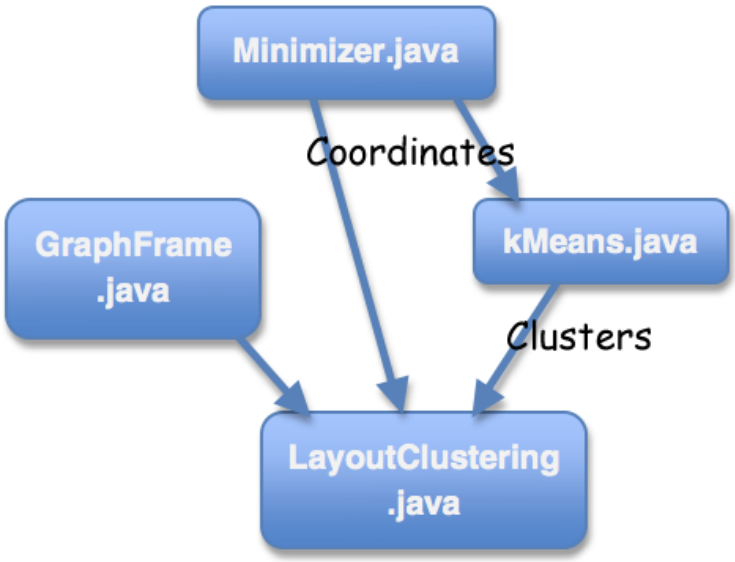

Fig.13 The input and output relationship between classes.

There are other two Java files called Edges.java and Node.java. They deal with the information of the relate object as their names indicate. They declare what properties an edge or a node should have. And they are basic classes, which will be used in all the other class to create objects of edges and nodes.

A network in the form of "list of edges" is the input of this application. A method in LayoutClustering.java will read it and store the information of nodes and edges in arrays. These arrays will be passed to Minimizer.java and used to compute the coordinates. After kMeans.java gets the coordinates and calculates the clusters, both the information of coordinates and clusters will go back to LayoutClustering.java and used to display in a dialog with the methods supplying in the GraphFrame.

Steps of my implementation:

1) Randomly distribute all the nodes in a metric space, 2D or 3D as specify at input.
2) Calculate the attraction, repulsion and gravity energy of every node, and move them to the final stable positions.
3) Randomly choose k centroids from nodes, use k-means to cluster the nodes using the coordinates generated in the last step, when new node is

added to a cluster, the central position of this cluster would be changed. This operation will repeat until no node is assigned to another cluster.

4) Output the coordinates of nodes and the clustering result to output files.
5) Draw a graph of the final layout of the network.

Input: network in form of "list of edges". It is easy to find network in this form. There is one edge in one line, starting with the start node of the edge, following an end node; a weight of the edge is in the last in weighted networks. All these information are separated by a tab.

Output: there are two files for the output. One is storing information of each node after clustering, one node one line, including the coordinates of the nodes and the cluster the node belong to; the other is the clustering result, one cluster one line. The later one can be used to calculate the modularity of the clustered networks.

3.2 Designs and Implementation

In my application, one class is responsible for one task, and at last all the output will pass to the main class to generate the visualization of the network.

1. Minimizer.java

   This class is so important that it generates the coordinates of nodes. It gets the random position of all nodes and moves them to the final stable positions. Because the initial positions of nodes are random, so the final layout of network should be different every time it runs.

Minimizer.java
• public Minimizer • public void minimizeEnergy • public void initEnergyFactors • public final double getDist • private double getRepulsionEnergy • private double getAttractionEnergy • private double getGravitationEnergy • private double getEnergy • private double addRepulsionDir • private double addAttractionDir • private double addGravitationDir • private void getDirection • private void computeBaryCenter • private void printStatistics

Table.1 List of methods in Minimizer.java.

2. kMeans.java

In this class, the program will cluster the network based with the coordinates computed by minimizing the energy. How many clusters in the network will be specified at input with the running command.

The means are initialized randomly at first. In the procedure of clustering, the new means would be those most closed to the centers of the clusters. All the coordinates of nodes are stable now.

My kMeans will choose a node randomly in the network and find the other centroids until k means is found. Every node will be assigned to the nearest centroid, and these nodes closed to the same centroid will be in the same cluster. And then the center of the cluster will be recalculated and set to the node closest to the new center. The above will repeat until no node is moved from one cluster to the other.

kMeans.java
• public Map<Node,Integer> execute • private double computeNewMeans • private double computeDisOfNodeToCenter • private Map<Node,Integer> initialMeans • private double computeDistant

Table.2 List of methods in kMeans.java.

3. GraphFrame.java

This class is used to display the layout of network and the clustering result. All nodes are displayed as a circle with different color, but the nodes in the same cluster will have the same color. All nodes are put in the positions according to the coordinates calculated by the Minimizer.class.

GraphCanvas
• public GraphCanvas • private void showPopup • private Set<Node> nodesAt • public void setLabelEnabled • public void paint

Table.3 List of methods in GraphCanvas.class.

GraphFrame
• public GraphFrame

Table.4 Method in GraphFrame.class.

GraphFrame: This is the constructor for the visualization dialog. It arranges the components in the layout panel. The attributes of the viewer are set here, and the other methods needed are defined in the GraphCanvas.class.

4. LayoutClustering.java

This class reads network from a file, invokes method in the other classes to compute the layout and clustering, at last, writes the result of layout and clustering to two files and displays the network in a dialog. It can display the structure of the clusters in the dialog with nodes in the same cluster putting closer. As I mentioned in the first method of Minimizer.java, the parameters in that method will affect the final layout of the network. The input in the running command will tell the program to compute the network in a two- or three-dimensional space. This class calls the method in the GraphFrame.class to display the network in the dialog.

LayoutClustering.java
• private static Map<String,Map<String,Double>> readGraph • private static Map<String,Map<String,Double>> makeSymmetricGraph • private static Map<String,Node> makeNodes • private static List<Edge> makeEdges • private static Map<Node,double[]> makeInitialPositions • private static void writeFiles • public static void main

Table.5 List of methods in LayoutClustering.java.

Chapter 4: Experiments

4.1 Overview

Experiment is a way to supply information about the quality of the software. It also provides an objective view of the software to allow users to understand potential risk of the software, and helps the users to understand how the software works to some extent.

It also tells users how well the software achieves the aims of the development, making sure the software matches the specification. Finding bugs is another crucial aim of testing.

As the exponents mentioned above, changing of these parameters will be tested to find what results it will get. The changing of the iteration also affects the computation time and the final layout of the network.

Therefore, tests of the parameters will be in the following subsection, and then different network will be used to test my application, and the modularity of the clustering result of the network will be calculated and compared with the results get by the other kinds of clustering algorithms, such as GN (Girvan-Newman) algorithm, CNM (Clauset-Newman-Moore) algorithm. As my application can also calculate the network in a 3D metric space, tests in 3D space are also given. The modularity is calculated by the classes provided by my supervisor Dr. Steve Gregory.

4.2 Experiments over parameters

For testing the parameters, I will set the parameters except the one I want to discuss to be stable. I consider the default value of parameters are repuExponent=0.0, attrExponent=1.0, gravfactor=0.05, iteration=100, and I use this to calculate the standard layout.

(1) repuExponent

repuExponent is the exponent that is used to calculate the repulsion energy and direction of repulsion of a node. Normally, it is set to 0.0, so the software can give logarithmic repulsion. It affects the uniformity of the node distance. The smaller it is, the more uniform node distance in the final layout.

The network of karate is used in the test of repuExponent. The size of the network is 34, and it is a weighted network, the size of the node in the layout indicates the importance of the node in the network. The number of the clusters is 4.

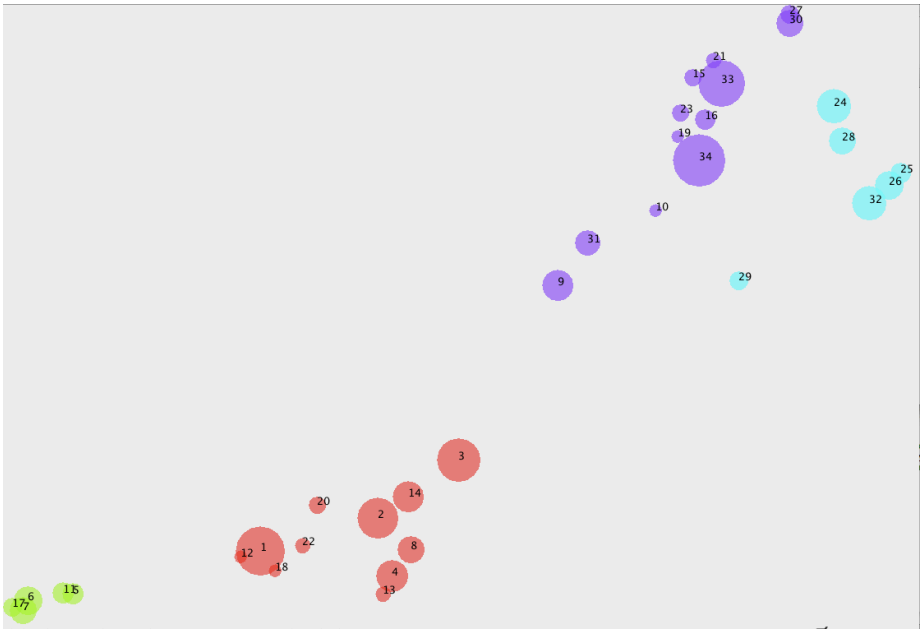

Fig.14 repuExponent=0.0, attrExponent=1.0, gravFactor=0.05, iteration=100, Modularity=0.41978

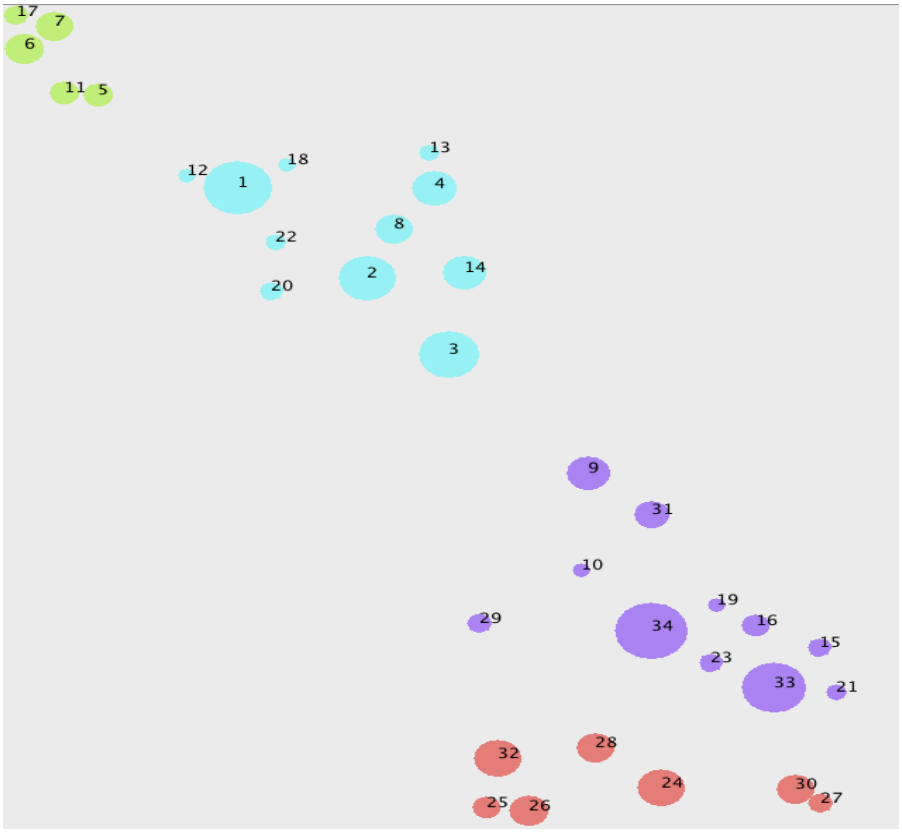

Fig.15 repuExponent=-0.5, attrExponent=1.0, gravFactor=0.05, iteration=100, Modularity=0.40129

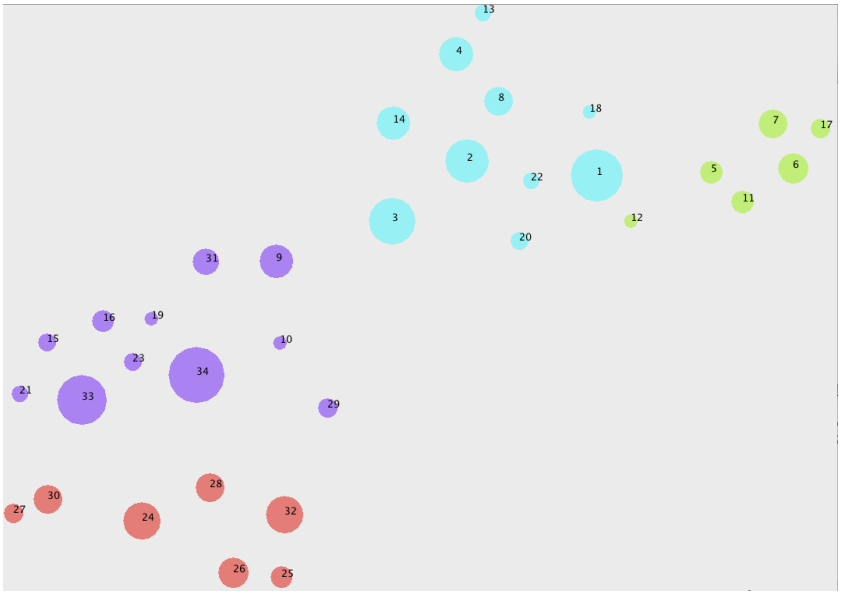

Fig.16 repuExponent=-1.0, attrExponent=1.0, gravFactor=0.05, iteration=100, Modularity=0.39201

As the results shown above, the decrease of the repuExponent will make the layout sparse, the distance between nodes are more and more similar. In my test, when the repuExponent is decreasing, the clustering results of the network are growing stable, because the uniform distances between nodes make the centroids calculated by my application more stable. However, the decrease of the repuExponent will not be able to reveal the dense clusters of the networks. Fig.14 is the best clustering result for the karate network. Therefore, for getting better and reasonable layout, default values are the best.

(2) attrExponent

Similar to repuExponent, this parameter affects the attraction energy, and then the final layout. The attrExponent affects the uniformity of the edge lengths. The greater it is, the more uniform the edge lengths are.

In this part, I use the network of world import of the year of 1999. The size of it is 66. It is a weighted network, and the number of clusters is set to 3, according to the realistic situation. However, as there are many inter-clusters within clusters, the modularity of the clustering is less than 0, that means the clustering is rather bad. Actually, the layout of the network still follows the economic relationship between countries.

Fig.17 repuExponent=0.0, attrExponent=1.0, gravFactor=0.05, iteration=100

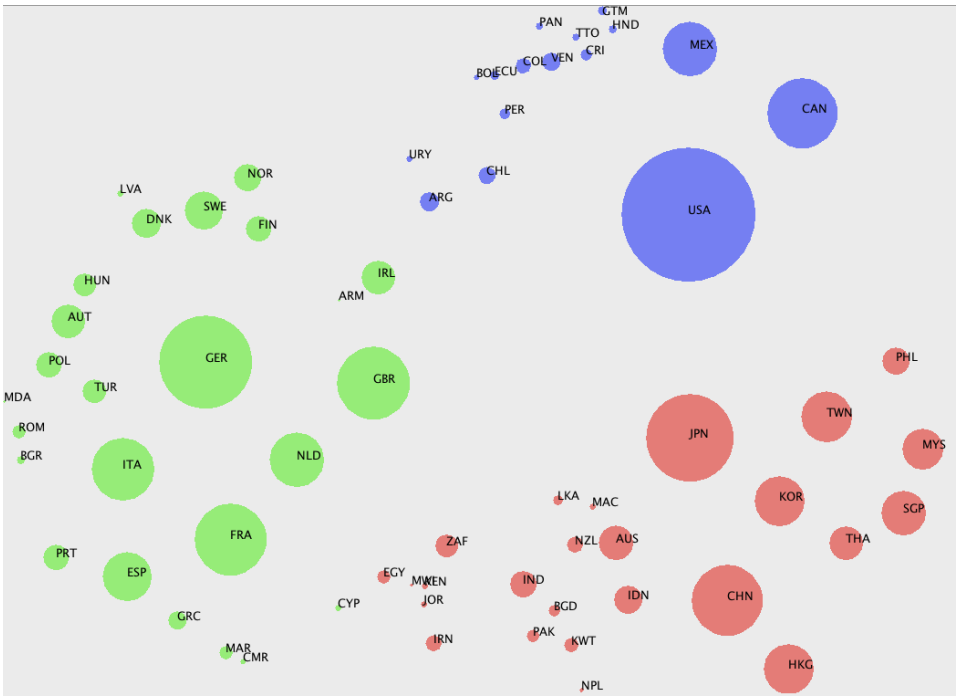

Fig.18 repuExponent=0.0, attrExponent=1.5, gravFactor=0.05, iteration=100

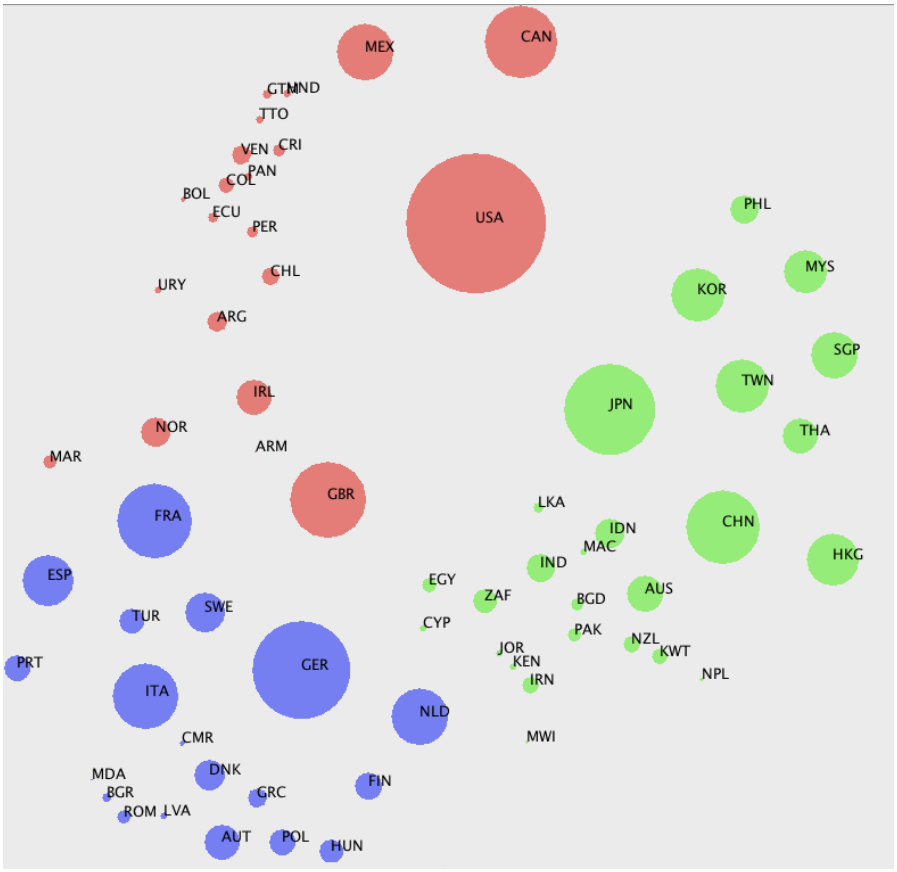

Fig.19 repuExponent=0.0, attrExponent=2.0, gravFactor=0.05, iteration=100

Opposite to the repuExponent, the increase of the attrExponent results in the sparse network layout. The distances between the nodes are increasingly similar. In my tests, when the attrExponent is increased to 1.5, the program can still compute reasonable layout and get good clustering results, but when it is 2.0, the probability of getting good clustering result is rather small. Because my kMeans initializes the centroids randomly, the increasing distances between the nodes will cause the nodes be assigned to the wrong cluster. Fig.19 is an example, GBR (Great Britain), IRL (Ireland) and NOR (Norway) are European countries, they should be clustered to the blue cluster, no the red one, although they all have connection with the American countries, they are certainly not as much as the connection with the other European countries as the regional economic groups exist in the world nowadays.

(3) gravFactor

This parameter also has influence on the layout. It decides how closed the nodes is attracted to the barycenter. The greater it is, the closer the nodes to the barycenter. A value of 1.0 will mean the attraction to the barycenter is about the same as the one by the other nodes. Normally, for connected networks, 0.0 is sufficient, but when we study networks with many nodes and edges, we cannot make sure whether the network is connected or not, so the value is always set slightly larger than 0.0, such as 0.05 or 0.1. Because when the value is 0.0 and the network is unconnected, the

components of the network would fly away to infinity. But as the area to show the network is finite, the layout of unconnected network would be hard to be recognized.

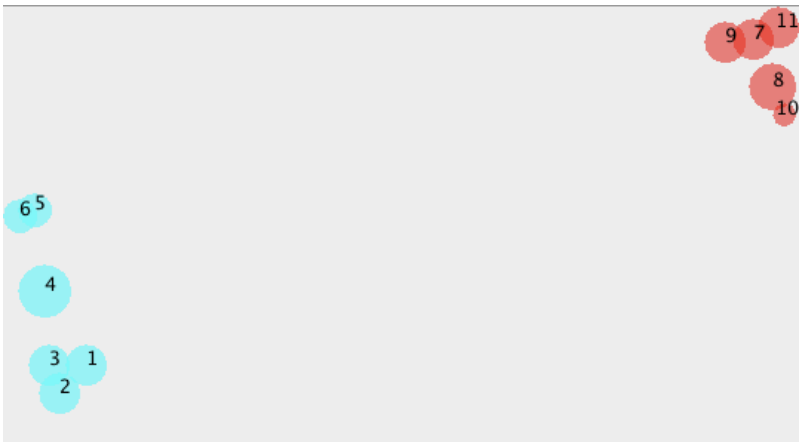

Fig.20 repuExponent=0.0, attrExponent=1.0, gravFactor=0.1, iteration=100

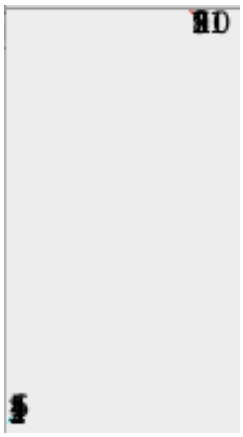

Fig.21 repuExponent=0.0, attrExponent=1.0, gravFactor=0.0, iteration=100

As the figures shown above, this is the unconnected network I create myself with 11 nodes. It is easy to see that when the gravFactor is 0.1, the unconnected network can be restricted in a limited area. But when the gravFactor is 0.0, two clusters would fly away from each other and we can only see two small colorful nodes. There are two black objects in the Fig.21, that is because the names of nodes is shown. However, if the number of clusters in the network can be known, it can surely calculate the correct clustering result, because the clusters are fly away from each other, and my kMeans always choose the farthest node as the new centroid. Take Fig.20 for an example, if node 4 is chosen as the first centroid, the next centroid should be chosen in the red cluster (the farthest to node 4 is node 11), but one in the blue cluster. Therefore, when studying unconnected networks, the gravFactor can be set much greater than 0.0 but no larger than 1.0 in a range of 0.0 to 1.0.

(4) attrExponent – repuExponent

As the parameters discussed above, attrExponent should be greater than 1.0, while repuExponent is smaller than 0.0. Therefore, the difference attrExponent-

repuExponent is greater than 1.0. If the difference is less than 1.0, there will be many overlapping nodes in the layout.

In my tests, the default configuration of parameters reveals best cluster structure in the network, as it shows group of the densely connected nodes and the separation of the sparsely connected nodes. When the difference is larger than 1.0, the distances between nodes tend to be uniform. There is one well-known model – Fruchterman-Reingold energy model. This model produces readable layouts, but gives less clearly community structures. The values of the parameters of this model are attrExponent=3.0 and repuExponent=0.0 [56].

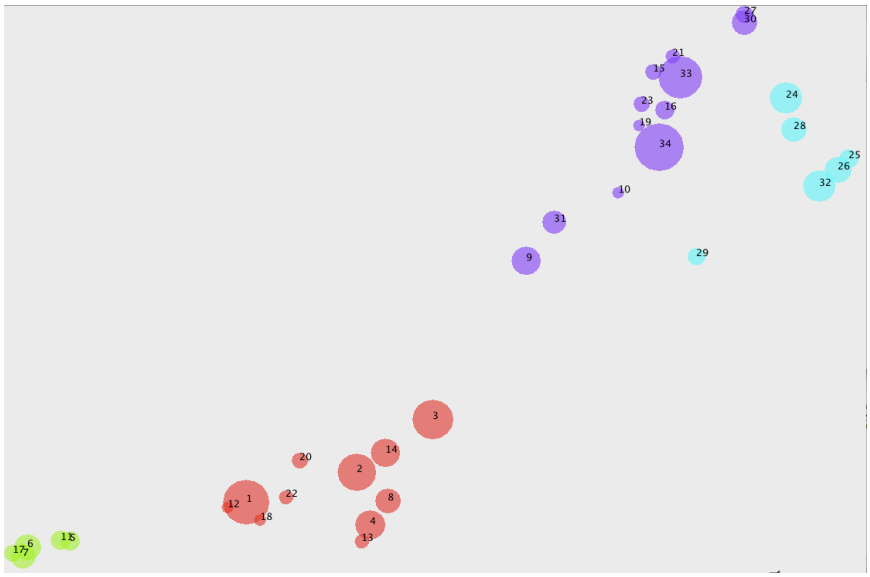

Fig.22 repuExponent=0.0, attrExponent=1.0, gravFactor=0.05, iteration=100, Modularity=0.41978

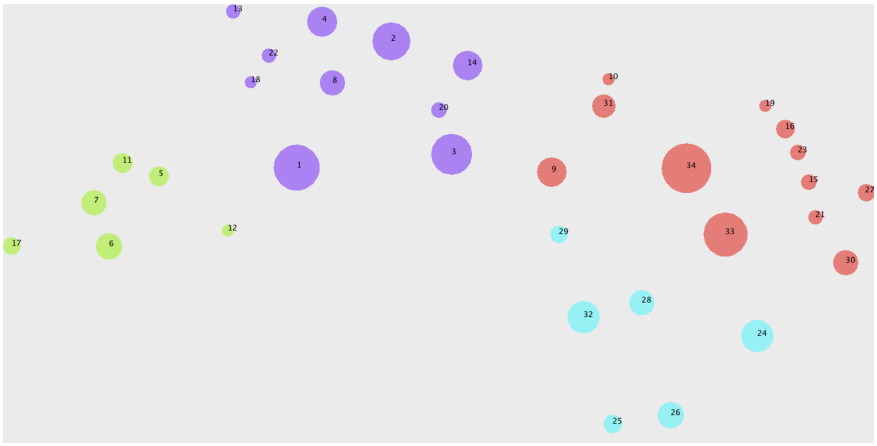

Fig.23 repuExponent=0.0, attrExponent=3.0, gravFactor=0.05, iteration=100, Modularity=0.41050

Fig.22 and Fig.23 give a clear view that the increase of the difference attrExponent-repuExponent will result in a sparse layout. The probability of getting a best layout is rather low, because I need to run the program more times to get this

result (Fig.23). Regardless to the colors of the clusters, Fig.22 shows clear cluster structure of the network, while Fig.23 not. Fig.23 distributes the nodes evenly in the available area; nodes will not be overlap as the distances between each other is longer, but it is difficult to tell which node is belong to which cluster.

(5) iteration

This parameter can be treated as the number of loops to calculate the energy of the layout with moving the nodes. Generally, the more iteration, the better final layout is. For most of networks with hundreds nodes, 100 iterations would be enough, but when computing larger or complicated networks, more iterations may be needed.

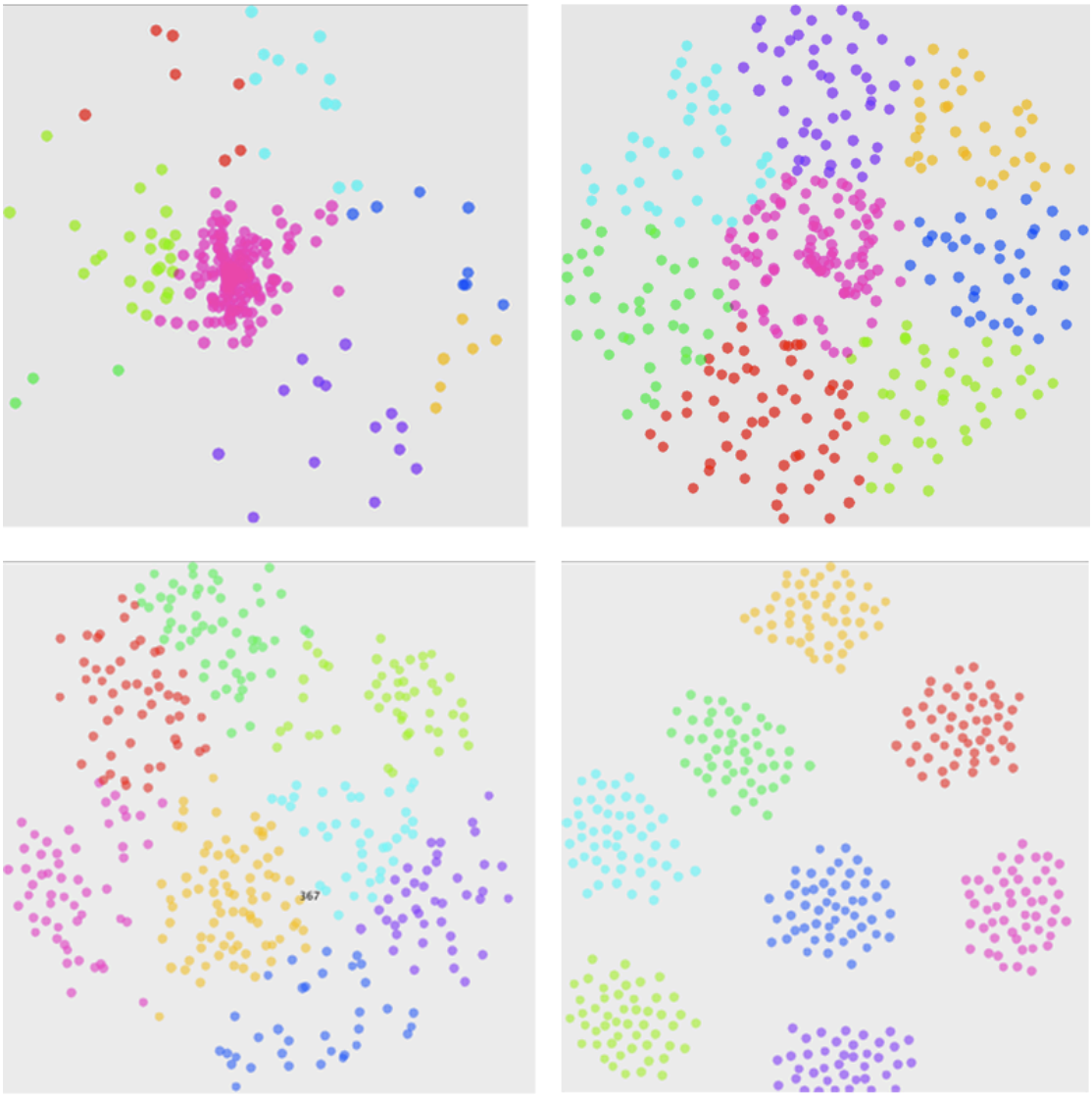

Fig.24 repuExponent=0.0, attrExponent=1.0, gravFactor=0.05. These four drawings are drawn after 6, 12, 24, 48 iterations.

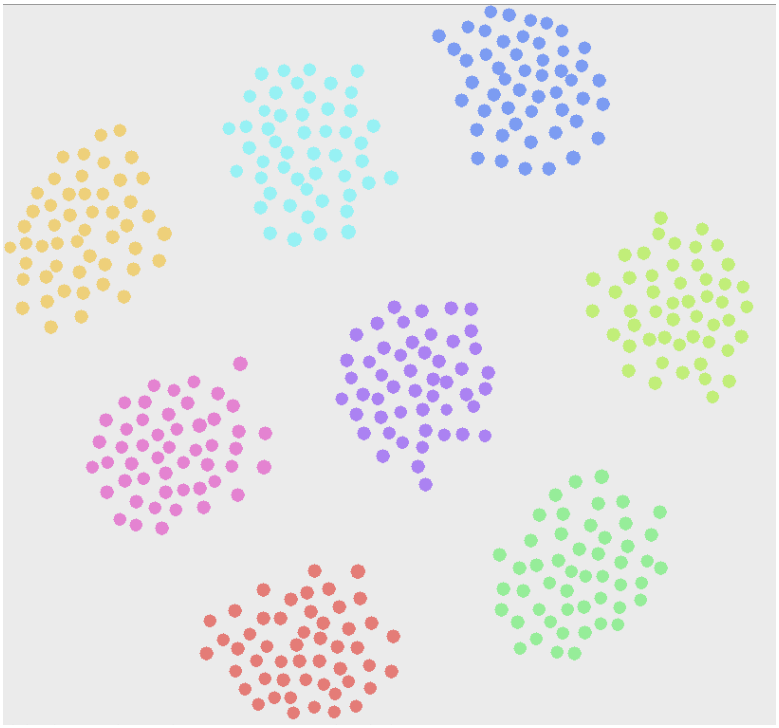

Fig.25 repuExponent=0.0, attrExponent=1.0, gravFactor=0.05, iteration=100.

The test results easily show that more iteration will give better layout. Therefore, the nodes belong to one cluster are separated from the ones from the other cluster. It would be easier to cluster the nodes and give a reasonable clustering result. As shown above, it is hard to get a reasonable clustering result if only a few iterations were running; the nodes are randomly put in the drawing, some nodes are even overlap, the boundary of the cluster is not clear. The size of this network is 400. And when there are more than 48 iterations, the program starts to give a reasonable layout with clear clusters. Fig.25 gives nice layout and correct clustering result.

4.3 Experiments over different networks

After testing the parameters to show the changing of them will result in what layout, default configuration will be applied to the program in this part to run with different networks. As mentioned above, the default configuration is repuExponent=0.0, attrExponent=1.0, gravFactor=0.05, iteration=100.

In this part, real networks will first be tested, following with artificial networks generated by the benchmark. For every network, the layout with clustering result will be given, and then the clustering result will be used to calculate the modularity, comparing with the one calculated by GN and CNM algorithms.

For the GN and CNM algorithms, Dr. Steve Gregory's CONGA (Cluster Overlap Newman-Girvan Algorithm) [49] will be used. To run CNM algorithm, one more step is needed, which is to run the FastCommunity [57] implemented by Aaron Clauset.

All the results computed by my application are the best clustering that could be found with running the program over and over again on the same network. The better result has higher modularity.

(1) Real networks

A summary of the results of the following networks will be given at the end of this subsection.

a) Karate [7]

This is a social network of friendships between the members of a karate club. The size of it is 34. The best result my program found is shown in Fig.26, with 4 clusters. The time to compute this result is 341ms.

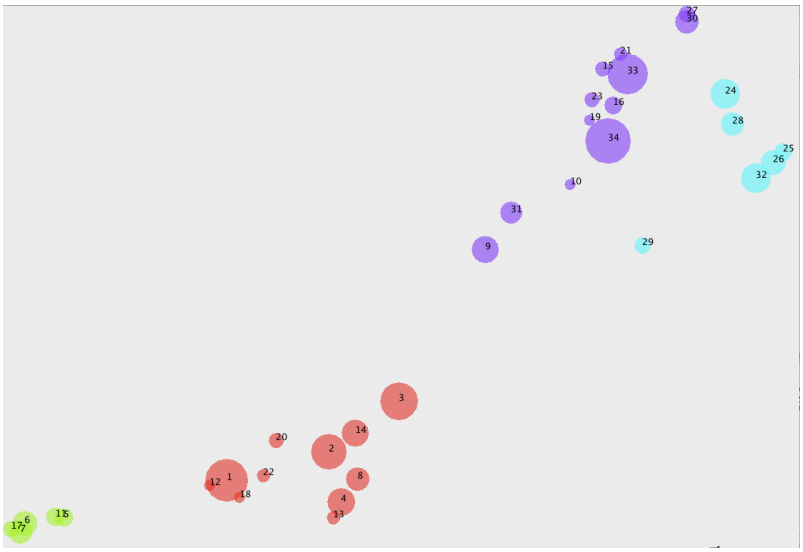

Fig.26 Zachary's karate club in 2D.

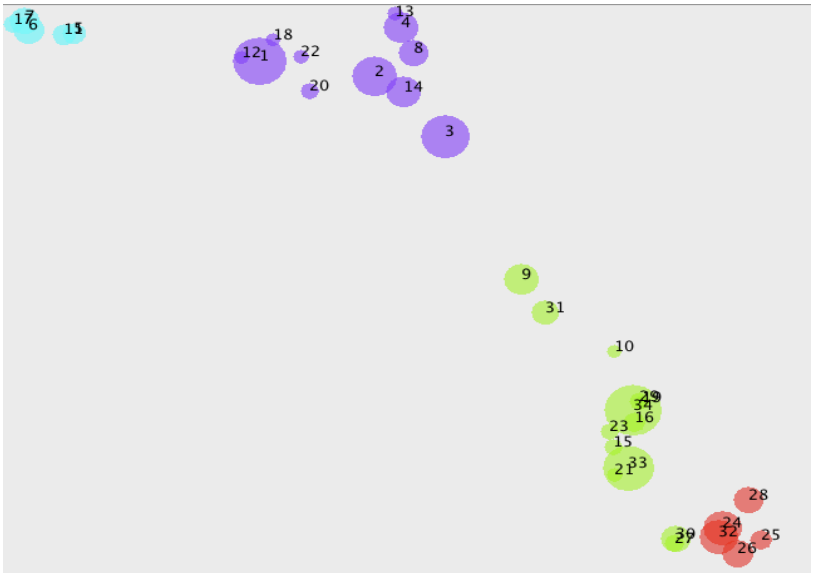

Fig.27 Zachary's karate club in 3D.

There is another result calculated by optimizing of the modularity of Newman and Girvan, shown in Fig.2, the clustering result is the same as the one shown in Fig.26. The NMI (Normalized Mutual Information) between these two results is 1. That means my program can get good result for this network. The modularity of the clustering result in 2D is 0.419. When the network is computed in 3D space, the NMI is 0.8994, while the modularity is 0.411.

However, the modularity of the best clustering GN and CNM can get is 0.401 and 0.370 respectively, with 5 clusters.

b) Dolphins [58]

This is a social network of frequent associations between dolphins in a community living off Doubtful Sound, New Zealand. The size of this network is 62. The best result is shown in Fig.27, with 4 clusters. The time to get the result is 681ms.

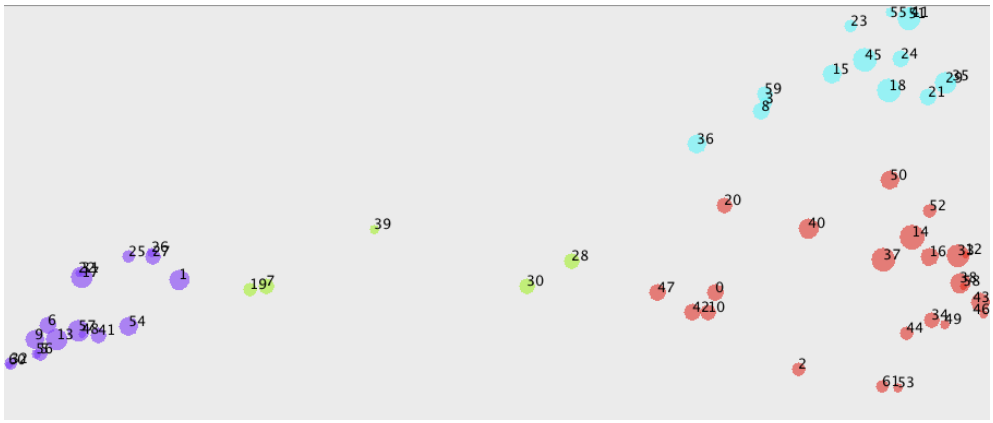

Fig.28 Dolphins in 2D.

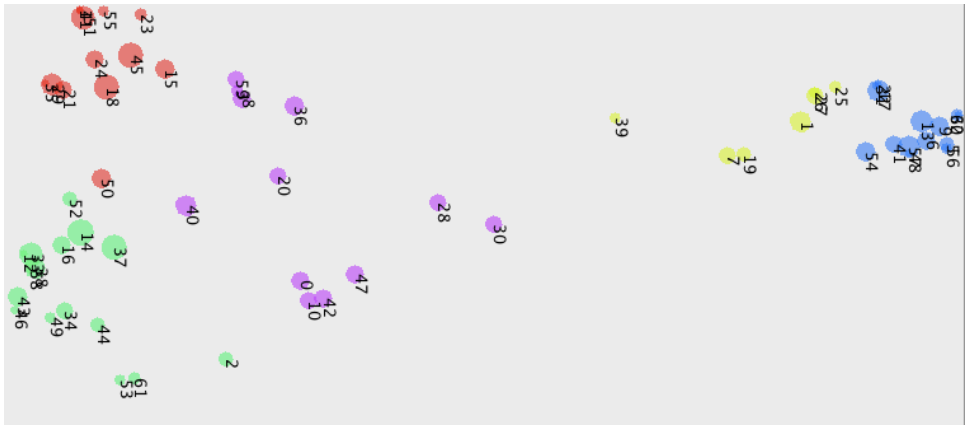

Fig.29 Dolphins in 3D. For saving space, the figure is rotated.

The modularity of the 2D result is 0.471. In many papers, the best cluster structure of this network is 2-clusters. However, the clustering result should be better than that because it reveals better separation of clusters. And the modularity of the results computed by GN and CNM is 0.383 with 4

clusters and 0.351 with 6 clusters respectively but 2 clusters. The modularity in 3D space is 0.465 with 5 clusters (Fig.28).

c) Wirz [59]

The size of this network is 84. The best result is 7 clusters (Fig.30), with 1185ms to compute it.

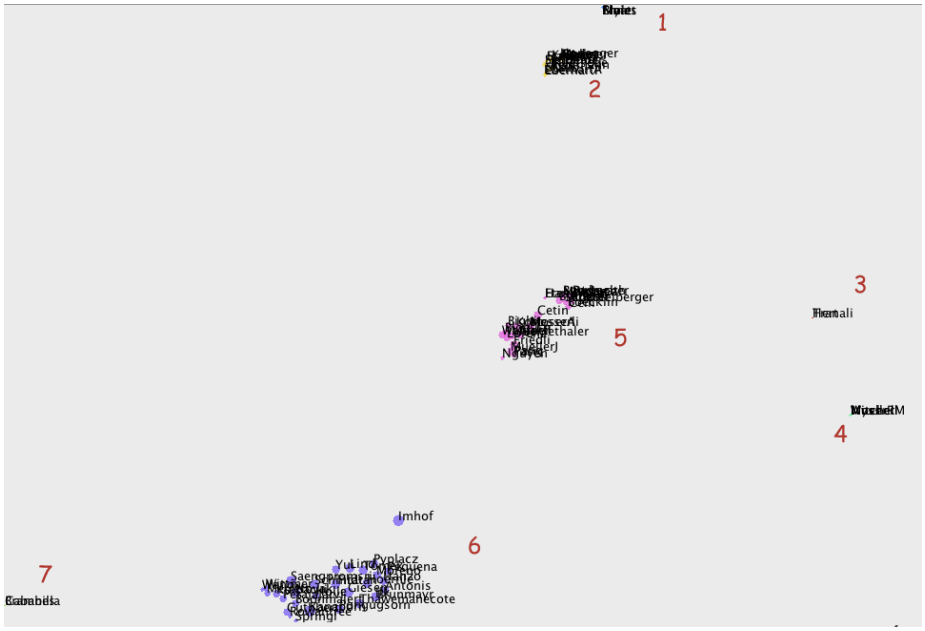

Fig.30 Wirz in 2D. The layout is so huge that makes nodes in the available area shown small. But as shown, the nodes closer to each other are in the same cluster.

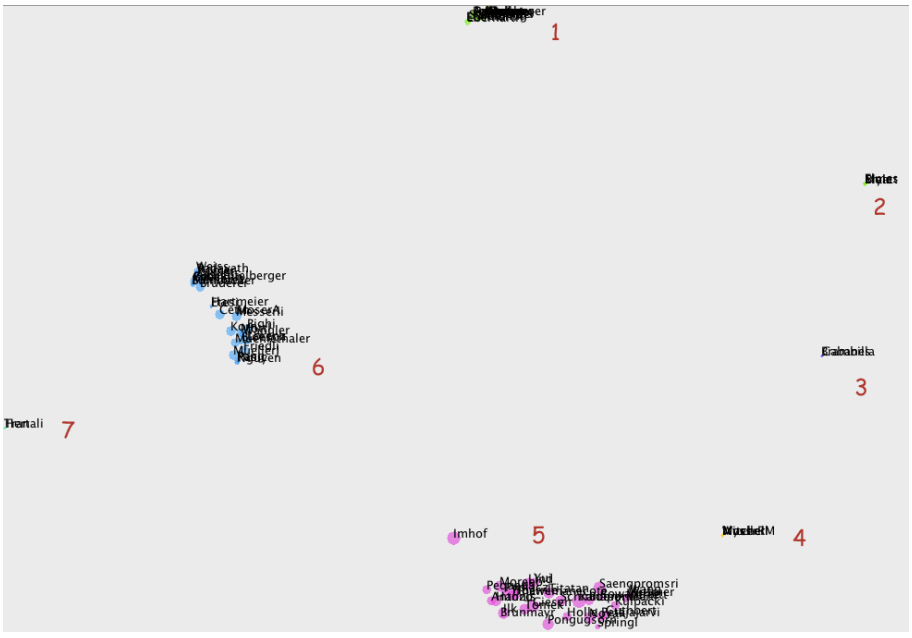

Fig.31 Wirz in 3D.

The modularity of the clusters in 2D is 0.5226, while the one in 3D is 0.518 (Fig.31). However, the clustering calculated by GN and CNM are 0.5220 with 7 clusters and 0.464 with 6 clusters respectively.

d) Football [60]

This is the network of American football games between Division IA colleges during the regular season Fall 2000. The size of it is 115. The best clustering result in 2D is shown in Fig.32, with 9 clusters. Modularity of Fig.32 is 0.596. The running time is 2035ms.

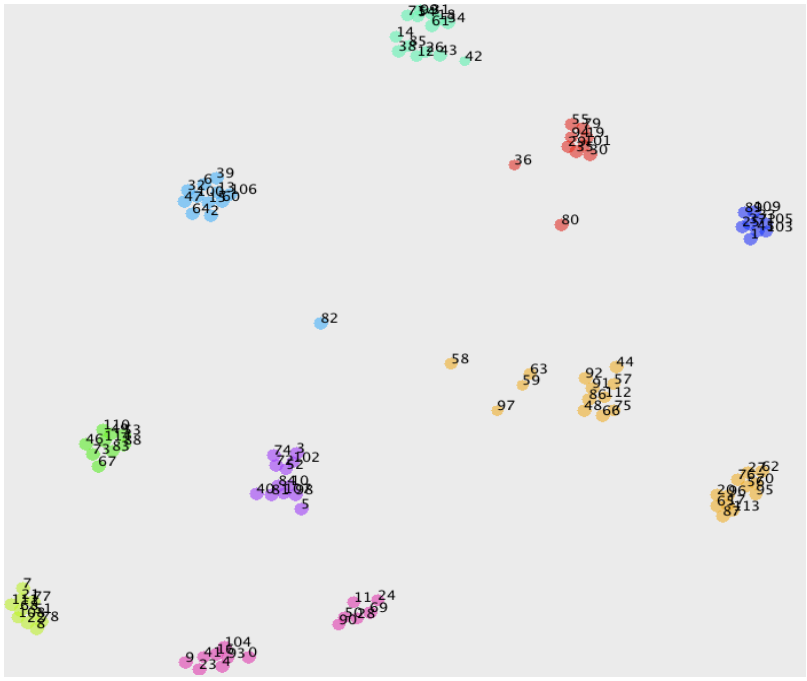

Fig.32 Football network in 2D.

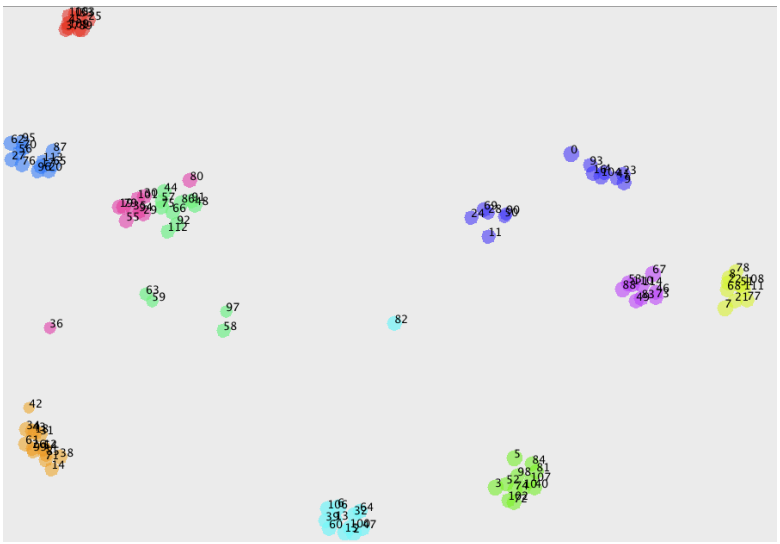

Fig.33 Football network in 3D.

However, the modularity of Fig.33 that is computed in 3D space is even better, 0.597 with 10 clusters, and the NMI reaches 0.731, while the one in the 2D space is just 0.691. The NMI result is compared with the conference partitions.

The clustering results get by GN and CNM is 0.599 and 0.294 respectively, both 10 clusters. The result of GN is best.

e) Jazz [61]

Jazz network has 198 nodes. The best clustering in 2D is shown in Fig.34, with 13 clusters. The modularity of this result is 0.330, running in 5929ms.

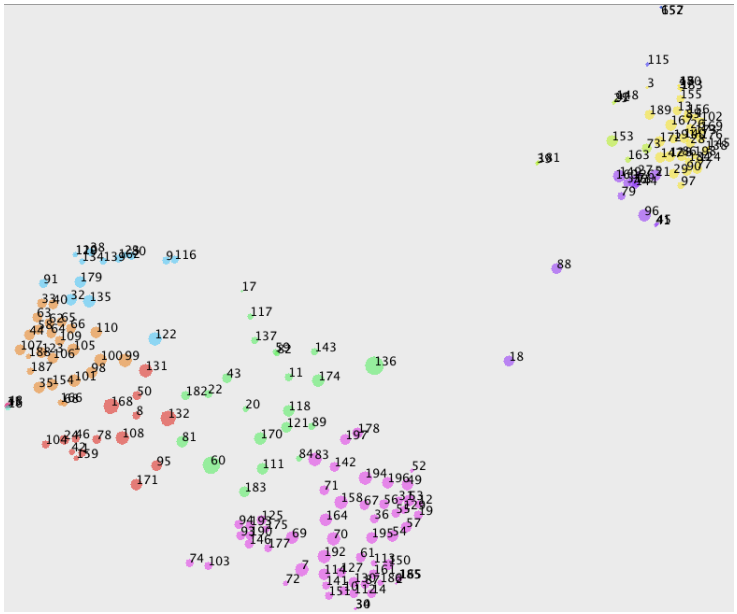

Fig.34 Jazz network in 2D.

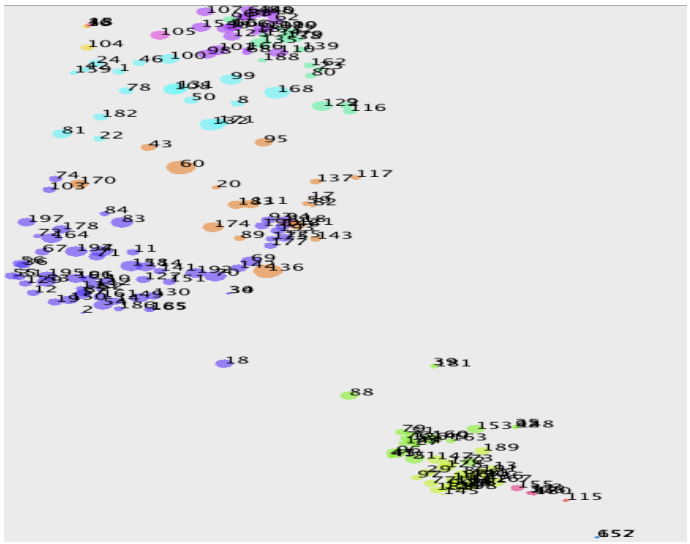

Fig.35 Jazz network in 3D.

For this network, when computed in 3D space, it also gets a better result, with 14 clusters, and the modularity is 0.332.

The GN and CNM get the results with the modularity of 0.286 with 14 clusters and 0.289 with 13 clusters respectively.

f) Science [62]

This network is the coauthorship of scientists working on the network theory and experiment. The original network is compiled by M. Newman in May 2006 [64], but the version of mine will be tested is the largest component of it get by my supervisor Dr. Steve [63]. The size of it is 379.

Fig.36 Network science in 2D.

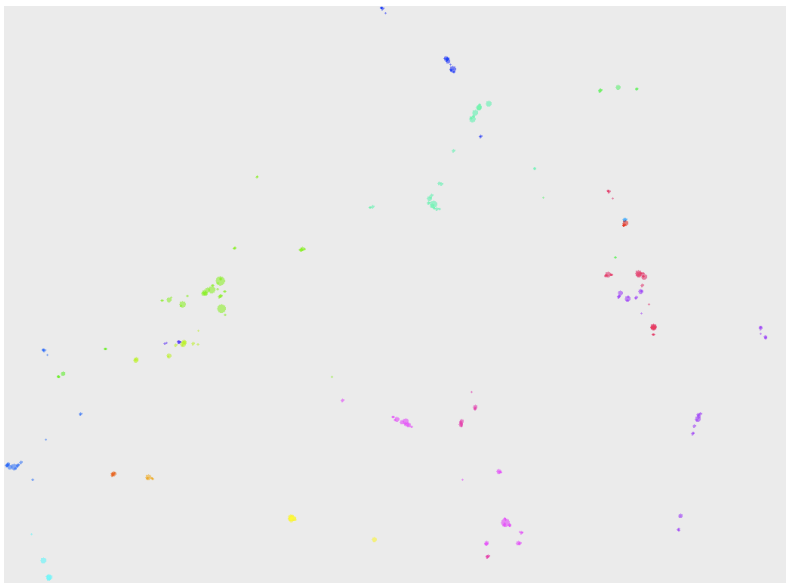

Fig.37 Network science in 3D.

Because the nodes are scattered in the viewer, the nodes is too small to see, but there are still some colorful spots in the figures. The modularity of the best result in 2D is 0.789 with 18 clusters, while the one of 3D is 0.814 with 18 clusters. The running time is about 19142ms.

GN and CNM can get the best result with 0.842 (17 clusters) and 0.760 (11 clusters) respectively. The GN algorithm gets the best result for this network.

Summary of the results above

Networks	2D-Layoutclustering	3D-LayoutClustering	GN	CNM
Karate(34)	0.419(4)	0.411(4)	0.401(5)	0.370(5)
Dolphins(62)	0.471(4)	0.465(5)	0.383(4)	0.351(6)
Wirz(84)	0.5226(7)	0.518(7)	0.5220(7)	0.464(6)
Football(115)	0.596(9)	0.597(10)	0.599(10)	0.294(10)
Jazz(198)	0.330(13)	0.332(14)	0.286(14)	0.289(13)
Scientists(379)	0.789(18)	0.814(18)	0.842(17)	0.760(11)

Table.6 Summary of the results of real networks. (Based on Modularity).

The highest modularity is highlighted for each network. The number after the name of the network is the size of it, while the one after modularity is the number of clusters when the program get this modularity.

As the table shown above, LayoutClustering performs well both in 2D and 3D metric spaces. The results of different dimensional space on the same network are quite close. However, there is a detail needed to be paid attention to. When the size of the network becomes larger, the results in 3D tend to be slightly better than the ones in 2D. The reason for this is that though the display area is scalable, the nodes in the network may be easy to overlap on other nodes at initialization. 3D space gives the nodes one more dimension so that nodes can have the same coordinates on x-dimension and y-dimension. Therefore, the program can compute better energy on every node, which will help to get a better result.

(2) Artificial network

In this part, networks generated by benchmark [65] will be tested. Because there is no standard result to be compared with, modularity of the network will be calculated to show how good the clustering result is.

a) Samples

8 clusters with 100 nodes: modularity is 0.606. The running time is 1527ms.

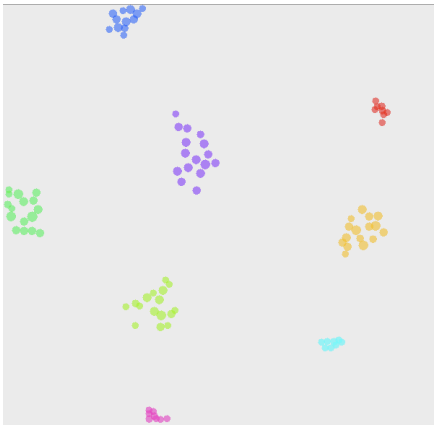

Fig.38 8 clusters with 100 nodes.

4 clusters with 200 nodes: modularity is 0.350. The running time is 6526ms.

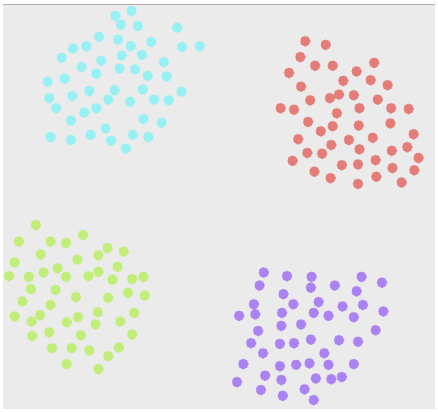

Fig.39 4 clusters with 200 nodes.

5 clusters with 300 nodes: modularity is 0.381. The running time is 13607ms.

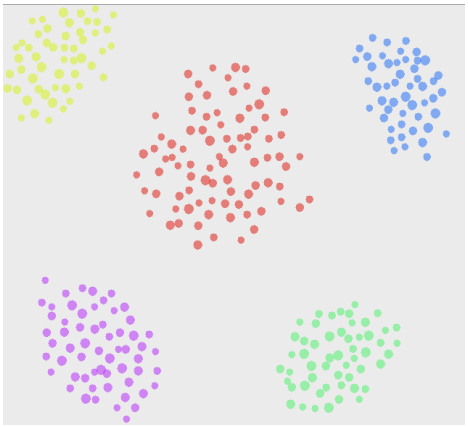

Fig.40 5 clusters with 300 nodes.

5 clusters with 400 nodes: modularity is 0.600. The running time is 24757ms.

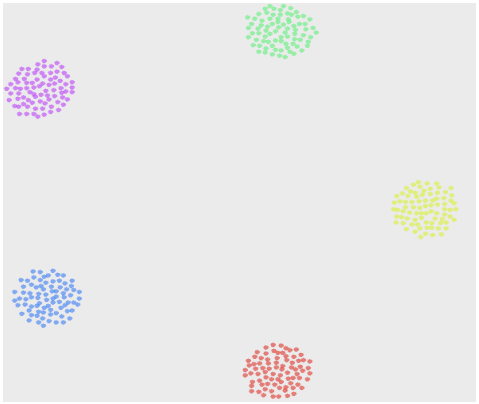

Fig.41 5 clusters with 400 nodes.

b) Conclusion for the results of the artificial networks

As the figures shown above, LayoutClustering can detect clusters in the network impressively.

The running time is increasing proportional to the size of nodes in the network. If the size of the network grows by n times, the time to detect the clusters will increase by nearly n^2 times. This conclusion can be found easily by checking the running for the above results (Chart.1). The number of edges also has influence on the running time. The reason for this consuming time is that every node has repulsion on all the other nodes, while they only have attraction to the nodes that they have edges connect to. Therefore, the time is not exactly the same as the multiple of n.

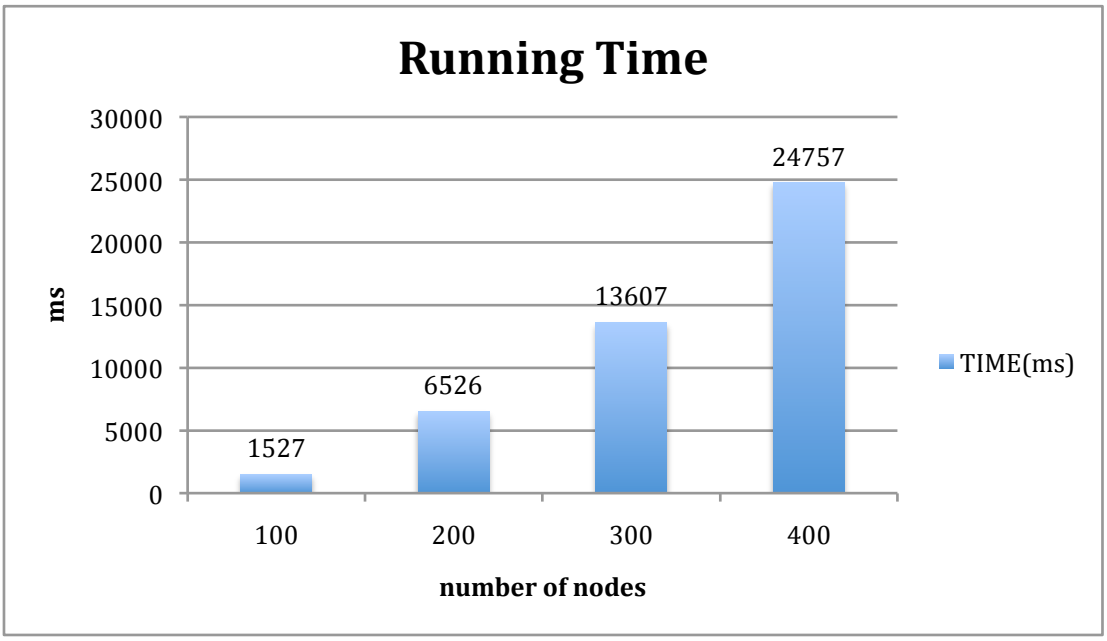

Chart.1 Running Time

However, even the final layout looks reasonable, one may notice that the modularity of the Fig.39 and Fig.40 is not as high as the one of the Fig.38 and Fig.41. It is because their ratio of the external degree to the total degree for each node is higher. It is to say that there is more edges of each node connect to the nodes in other clusters. Therefore, the community structure of Fig.39 and Fig.40 may not be as obvious as the one of Fig.38 and Fig.41 even the nodes in the same cluster are close to each other.

Chapter 5: Evaluation

In Chapter 5, LayoutClustering has been tested to cluster some real networks and artificial networks. It shows that the program works well and gets good clustering results. In this part, evaluation for the program is given as follow:

1) Performance

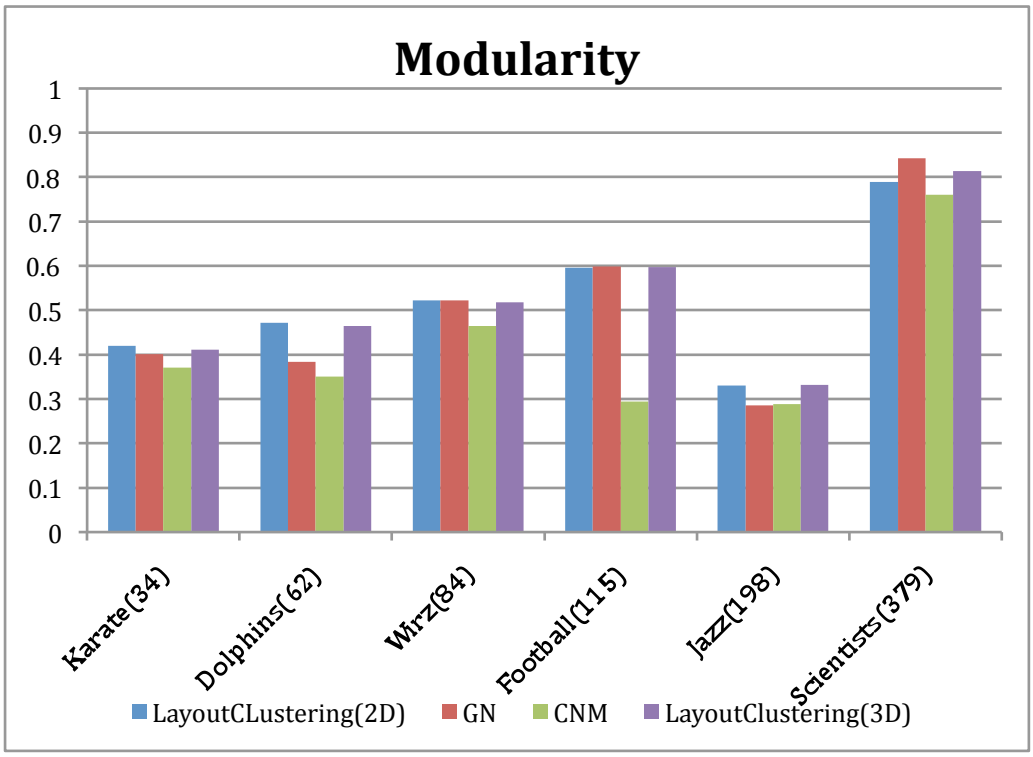

Chart.2 Modularity. The results of LayoutClustering are based on the best situation.

Modularity is a measure to decide how good the partition of network is, and higher the modularity, the better it is. In the chart shown above, LayoutClustering works well on all the real networks experimented, both in 2D and 3D metric spaces. The results computed by LayoutClustering in 2D and 3D are quite closed to each other. LayoutClustering performs best on the networks of karate, dolphins, wirz and jazz, second to GN on the networks of football and scientists, but not too much, and all of the results computed by LayoutClustering are better than the ones computed by CNM. LayoutClustering performs obviously better than GN and CNM on the network of dolphins and jazz, the gap between them can be seen easily, while slightly better than GN on the network of karate and wirz.

For the LayoutClustering itself, it works better and more stably in 2D than in 3D metric space on the network tested. However, the difference between them is increasingly small when the size of the network becomes larger and larger. When dealing with small networks, 2D-LayoutClustering is significantly better, but on the network of football, 3D-LayoutClustering has already surmounted 2D-

LayoutClustering slightly. And it gets higher modularity of clustering result than 2D-LayoutClustering.

In a word, both 2D-LayoutClustering and 3D-LayoutClustering perform well on different real networks, while the 3D-LayoutClustering tends to get better results with the size of the network growing.

2) Stability

The comparison of the performance of different algorithms on different networks is given in the last part. In this part, stability of LayoutClustering will be tested.

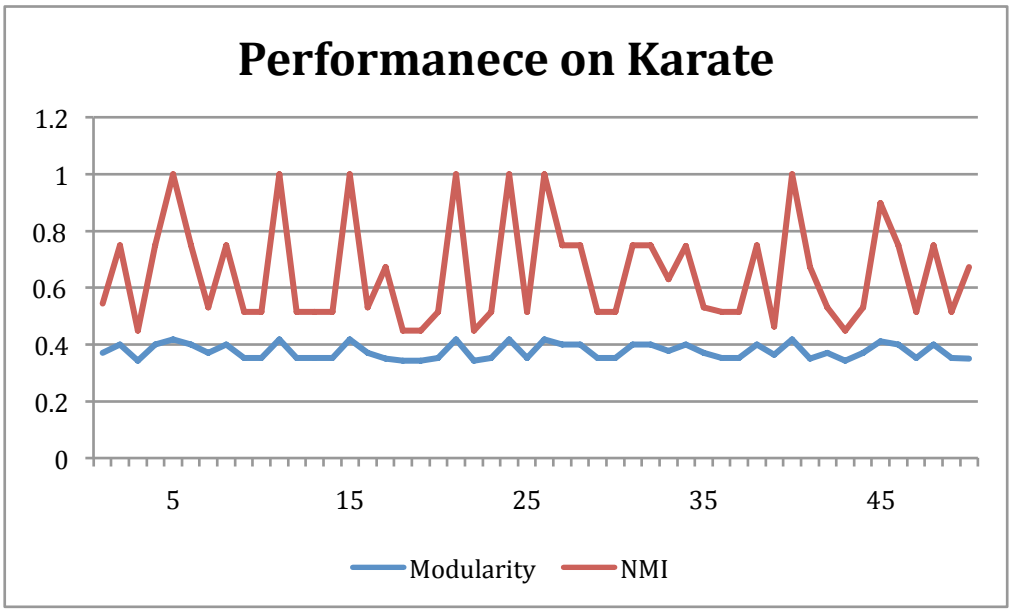

Chart.3 Performance on Karate.

Chart.3 shows the performance LayoutClustering on the network of karate. The program was run 50 times on the same network. Another measure is used here-NMI, which uses the clustering result compare with the known best partition of the network. The value of 1.0 means the result is the same as the known best partition. However, these two curves are put in the same chart because they have the same range of value, they are meaningless to compare with each other. The curves in Chart.3 vibrate because LayoutClutering may get different clustering results from time to time. It can compute the best partition for the network, but it is not guaranteed to get the best one all the time. If one did want to get best partition of the network, the program needs to be run more times. The program works well on karate club network, it can get best result in acceptable probability. However, it may be unlucky to get the other result as the chart shows between 26 to about 40. The best modularity of karate network is 0.419, even the program cannot get the best partition, and it vibrates near 0.38 and does not suddenly get the one with less than 0.20. Therefore, even the result is not the best one, it is acceptable and reasonable as LayoutClustering shows in the final layout of the network.

3) Time complexity

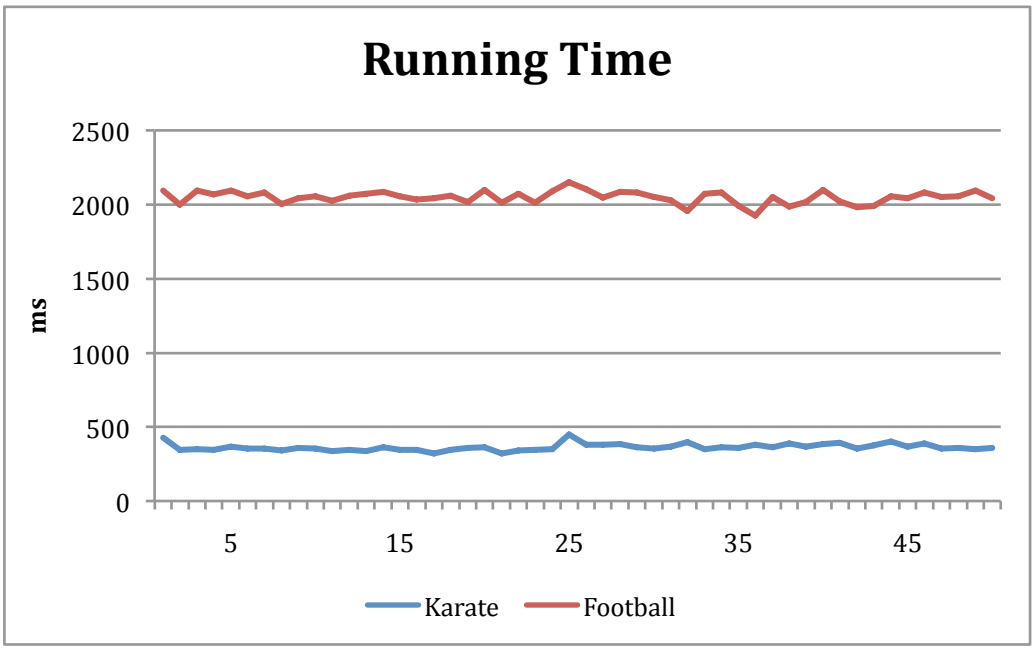

Chart.4 The time to compute the network of karate and football.

Similar to the Chart.1, we can see that the running time are increasing with the size of network grows. The size of karate is 34, while the one of football is 115, nearly four times as the size of karate. The number of edges does have influence on the running time, because the running time of football is not as four times as the one of karate, and the exact reason inside is that the program will calculate the attraction energy, which are affected by the number of edges. The running time of karate vibrates near 400ms, while the football is near 2020, almost 4 times longer.

Conclusion:

In short, LayoutClustering works well on all networks. It does not only generate pleasing layout, but also get reasonable clustering results. It places the nodes in the reasonable position in a metric space, shows densely connected clusters and separates the nodes in different clusters. The clustering method works well, too. Though the result is different from time to time, it sometimes discovers better partition that has never found, and the result it gets is acceptable.

However, the speed of this program is quite slow. When computing small networks, the result will come out within seconds, but as the network grows larger, it needs minutes and even hours to wait for the result. The time grows too fast as the size of network increases.

Chapter 6: Conclusion & Future Work

In this project, LayoutClustering aims to provide another way to cluster a network. Andreas Noack has proved that modularity clustering is a force-directed layout, and it is able to use layout algorithm to cluster networks. This theory is proved by LayoutClustering in my project. In LayoutClustering, not only can it visualize the network but also get the result how the network is partitioned.

LayoutClustering works well on the networks tested, including real networks and artificial networks. It gets good clustering results on these networks with higher modularity, comparing to the one by GN and CNM algorithms.

LayoutClustering is helpful in studying networks, because the researchers will not only get the clustering result how the network is partitioned, but also watch the relationship between the nodes of the network in sight. LayoutClustering puts the densely connected nodes in the near positions while the sparsely connected nodes in different clusters at distant positions. For classic community detection algorithms, they only output the result of the clusters, and the researchers will get no idea about the relationship of nodes in space.

It is interesting to find another feature that when testing the program, LayoutClustering can sometimes discover new and better partition of a network. Classic community detection algorithms get the clusters based on the edges- or nodes-betweenness of the network. When the network is given, the betweenness of edges or nodes are set and unchangeable. But LayoutClustering is different, it uses k-means algorithm to detect the clusters in the network, and the initial centroids that affect the final clusters are chosen randomly. It makes the result unstable, but in another aspect, it would be great help to find a new better partition of a network.

As the development of LayoutClustering keeps going on, there are some ideas appear. But because of time consuming, these ideas cannot be implemented during this period. They are considered as future work so that LayoutClustering can be perfected. These main ideas include:

1) More stable.

   Although the benefits of unstable results may give us a better partition of the network, the probability for this is quite small. The weight of nodes is taken into consideration when developing the program, and the program did perform more stable, but the result it gets is not as good as the one without the weight of nodes. Nodes with large weight act importantly in the cluster, but there may be more than one node that has the same influence in the same cluster, adding weight to compute the centroids will separate these two nodes away. Because the nodes are randomly distributed to a space and k-means randomly chooses *k* centroids, the solution should be focused on the k-means algorithm.
2) Work with overlapping networks.

LayoutClustering only works with un-overlapping networks. However, most networks in real world are considered to overlap at some members. For example, social networks, one may belong to a community who were study in the same high school, while belong to another one who study in the same university. Because LayoutClustering only considers the distance between nodes and the centroids and there is only one nearest cenroid, it cannot compute overlapping clustering result for a network. Therefore, new measure is needed to handle this overlapping situation.

3) Improve the speed.

   The speed to compute clustering result for small networks is acceptable. But when the network is larger than 1000 nodes, the program works rather slowly. Because every node has repulsion on the other nodes and attraction on the ones that they are connected to, the calculation needs nearly $O(n^2)$ time per iteration, where n is the size of the network. Barnes and Hut proposed a new method to store the nodes – Octree, which every node in the tree structure has exactly eight children. It can reduce the time to $O(n \log n)$ and less. The speed of the program may be also improved if LayoutClustering just calculates the repulsion force with the nodes within finite range. But separating the clusters needs the program to consider the repulsion forces globally.

4) Work with bipartite networks.

   In real world, there is another special kind of networks existing – bipartite networks. These networks are special because the attributes of nodes at each end of every edge are different, and they have to treat separately. There is a famous bipartite network – southern women. The edges of this network reveal what kind of activities the women attend. One side of an edge is a woman while the other is the activity. That is quite different from the normal networks, because the nodes at the end of edges are not belonging to the same kind of object.

Bibliography

[1]. S. H. Strogatz, Nature (London) **410**, 268 (2001).

[2]. R. Albert and A.-L. Barabási, Rev. Mod. Phys. **74**, 47 (2002).

[3]. D. J. de S. Price, Science **149**, 510 (1965).

[4]. S. Redner, Eur. Phys. J. B **4**, 131 (1998).

[5]. J. A. Dunne, R. J. Williams, and N. D. Martinez, Proc. Natl. Acad. Sci. U.S.A. **99**, 12917 (2002).

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

[7]. W.W. Zachary, An information flow model for conflict and fission in small groups, J. Anthropol. Res. 33 (1977) 452–473.

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

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

[10]. P.F.Jonsson, T.Cavanna, D.Zicha, P.A.Bates, Cluster analysis of networks generated through homology: Automatic identification of important protein communities involved in cancer metastasis, BMC Bioinf. 7 (2006) 2.

[11]. M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.

[12]. G. Gan, C. Ma, J. Wu, Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability), Society for Industrial and Applied Mathematics, Philadelphia, USA, 2007.

[13]. G.E. Andrews, The Theory of Partitions, Addison-Wesley, Boston, USA, 1976.

[14]. L. Lovász, Combinatorial Problems and Exercises, North-Holland, Amsterdam, The Netherlands, 1993.

[15]. S. Fortunato, Community detection in graphs, Physics Reports 486 (2010) 75–174

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

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

[18]. B.H. Good, Y. de Montjoye, A. Clauset, The performance of modularity maximization in practical contexts, eprint arXiv:0910.0165.

[19]. M.E.J. Newman, Finding community structure in networks using the eigenvectors of matrices, Phys. Rev. E 74 (3) (2006) 036104.

[20]. C.P. Massen, J.P.K. Doye, Thermodynamics of Community Structure, eprint arXiv:cond-mat/0610077.

[21]. A. Noack, Modularity clustering is force-directed layout, Phys. Rev. E 79 (2) (2009) 026102.

[22]. A. Arenas, J. Duch, A. Fernández, S. Gómez, Size reduction of complex networks preserving modularity, New J. Phys. 9 (2007) 176.

[23]. William T. Tutte. How to draw a graph. Proc. London Math. Society, 13(52):743–768, 1963.

[24]. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Englewood Cliffs, NJ, 1999.

[25]. S G, Kobourov, Force-directed Drawing Algorithm, CRC Press, LLC

[26]. Peter Eades. A heuristic for graph drawing. Congressus Numerantium, 42:149–160, 1984

[27]. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. Softw. – Pract. Exp., 21(11):1129–1164, 1991.

[28]. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. Inform. Process. Lett., 31:7–15, 1989.

[29]. R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. Discrete Applied Mathematics, 113(1):3–21, 2001.

[30]. David Harel and Yehuda Koren. A fast multi-scale method for drawing large graphs. Journal of Graph Algorithms and Applications, 6(3):179–2002, 2002.

[31]. H. A. Simon. The architecture of complexity. Proceedings of the American Philosophical Society, 106(6):467–482, 1962.

[32]. M. E. J. Newman. The structure and function of complex networks. SIAM Review, 45(2):167–256, 2003.

[33]. C.J.AlpertandA.B.Kahng.Recentdirectionsinnetlistpartitioning:Asurvey.Integration, the VLSI Journal, 19(1-2):1–81, 1995.

[34]. A.Pothen.Graphpartitioningalgorithmswithapplicationstoscientificcomputing.InD.E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, Parallel Numerical Algorithms, pages 323–368. Kluwer, 1997.

[35]. S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In Proc. 6th IEEE International Workshop on Program Understanding (IWPC 1998), pages 45–52, 1998.

[36]. R. Brockenauer and S. Cornelsen. Drawing clusters and hierarchies. In M. Kaufmann and D. Wagner, editors, Drawing Graphs: Methods and Models, LNCS 2025. Springer-Verlag, 2001.

[37]. A. Noack. An energy model for visual graph clustering. In G. Liotta, editor, Proceedings of the 11th International Symposium on Graph Drawing (GD 2003), LNCS 2912, pages 425–436, Berlin, 2004. Springer-Verlag.

[38]. F.-J.Brandenburg,M.Himsolt,andC.Rohrer.Anexperimentalcomparisonofforce-directed and randomized graph drawing algorithms. In F.-J. Brandenburg, editor, Proc. Symposium on Graph Drawing (GD 1995), LNCS 1027, pages 76–87, 1996. Springer-Verlag.

[39]. G. Pólya, G. Szegö, Problems and Theorems in Analysis I, Springer-Verlag, Berlin, Germany, 1998.

[40]. M. Meilă, J. Shi, A random walks view of spectral segmentation, in: AI and STATISTICS (AISTATS) 2001.

[41]. J.M. Anthonisse, The rush in a directed graph, Tech. rep., Stichting Mathematisch Centrum, 2e Boerhaavestraat 49 Amsterdam, The Netherlands, 1971.

[42]. M.E.J. Newman, Analysis of weighted networks, Phys. Rev. E 70 (5) (2004) 056131.

[43]. J.R. Tyler, D.M. Wilkinson, B.A. Huberman, Email as spectroscopy: Automated discovery of community structure within organizations, in: Communities and Technologies, Kluwer, B.V., Deventer, The Netherlands, 2003, pp. 81–96.

[44]. D.M. Wilkinson, B.A. Huberman, A method for finding communities of related genes, Proc. Natl. Acad. Sci. U.S.A. 101 (2004) 5241–5248.

[45]. U. Brandes, A faster algorithm for betweenness centrality, J. Math. Sociol. 25 (2001) 163–177.

[46]. T. Zhou, J.-G. Liu, B.-H. Wang, Notes on the calculation of node betweenness, Chin. Phys. Lett. 23 (2006) 2327–2329.

[47]. D.M. Wilkinson, B.A. Huberman, A method for finding communities of related genes, Proc. Natl. Acad. Sci. U.S.A. 101 (2004) 5241–5248.

[48]. F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks, Proc. Natl. Acad. Sci. USA 101 (2004) 2658–2663.

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

[50]. U. Brandes, D.Delling, M.Gaertler, R.Görke, M.Hoefer, Z.Nikolski, D.Wagner, On modularity — np-completeness and beyond. URL http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/3255.

[51]. M.E.J. Newman, Fast algorithm for detecting community structure in networks, Phys. Rev. E 69 (6) (2004) 066133.

[52].R.Guimerà,M.Sales-Pardo, L.A.N. Amaral, Modularity from fluctuations in random graphs and complex networks, Phys. Rev. E70 (2) (2004) 025101 (R).

[53]. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[54]. S.G. Kobourov, Force-Directed Drawing Algorithms. Handbook of Graph Drawing and Visualization, R. Tamassia Ed, CRC press.

[55]. A. Noack. An energy model for visual graph clustering, 2003.

[56]. A.Noack.Energy models for drawing clustered small-world graphs. Technical Report 07/03, Institute of Computer Science, Brandenburg University of Technology at Cottbus, 2003.

[57]. A.Clauset. Structure and Strangeness. Retrieve on Sep 3rd 2010. URL: http://cs.unm.edu/~aaron/research/fastmodularity.htm.

[58]. D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, Behavioral Ecology and Sociobiology 54, 396-405 (2003).

[59]. Get from http://www.cs.bris.ac.uk/~steve/networks/congopaper/.

[60]. M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. USA 99, 7821-7826 (2002).

[61]. P.Gleiser and L. Danon , Adv. Complex Syst.6, 565 (2003).

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

[63]. Get from http://www.cs.bris.ac.uk/~steve/networks/peacockpaper/.

[64]. Network data by Mark Newman. Retrieve on Sep 3rd 2010. URL: http://www-personal.umich.edu/~mejn/netdata/.

[65]. A software package to generate the benchmark graphs can be downloaded from http:// santo.fortunato.googlepages.com/benchmark.tgz.

[66]. M. Meilă, J. Shi, A random walks view of spectral segmentation, in: AI and STATISTICS (AISTATS) 2001.

Appendices: Source Code

In this part, some main method in the following file will be list.

(1) Minimizer.java:

```
public void minimizeEnergy(final
Map<Node,double[]> positions, final int nrIterations)
{
        if (nodes.size() <= 1) return;
       this.positions = positions;

        initEnergyFactors();
        final double finalAttrExponent = attrExponent;
        final double finalRepuExponent =
repuExponent;

        // compute initial energy
        computeBaryCenter();
        printStatistics();
        double energySum = 0.0;
        for (Node node : nodes) energySum +=
getEnergy(node);
        System.out.println("initial energy " +
energySum);

        // minimize energy
        final double[] oldPos = new double[nrDims];
        final double[] bestDir = new double[nrDims];
        for (int step = 1; step <= nrIterations; step++) {
                        computeBaryCenter();

        if (nrIterations >= 50 && finalRepuExponent <
1.0) {
        attrExponent = finalAttrExponent;
        repuExponent = finalRepuExponent;
        if (step <= 0.6*nrIterations) {
        // use energy model with few local minima
        attrExponent += 1.1 * (1.0 -
finalRepuExponent);
        repuExponent += 0.9 * (1.0 -
finalRepuExponent);
        } else if (step <= 0.9*nrIterations) {
        // gradually move to final energy model
        attrExponent += 1.1 * (1.0 -
finalRepuExponent)* (0.9 -
((double)step)/nrIterations) / 0.3;
    repuExponent += 0.9 * (1.0 - finalRepuExponent)
        * (0.9 - ((double)step)/nrIterations) / 0.3;
                                        }
                        }
        // move each node
        energySum = 0.0;
        for (Node node : nodes) {
        final double oldEnergy = getEnergy(node);

        // compute direction of the move of the node
getDirection(node, bestDir);

        // line search: compute length of the move
             double[] pos = positions.get(node);
        for (int d=0; d<nrDims; d++) oldPos[d] =
pos[d];
        double bestEnergy = oldEnergy;
        int bestMultiple = 0;
             for (int d=0; d<nrDims; d++) bestDir[d] /=
32;
        for (int multiple = 32;multiple >= 1 &&
(bestMultiple==0 || bestMultiple/2==multiple);
        multiple /= 2) {
          for (int d=0; d<nrDims; d++) pos[d] =
oldPos[d] + bestDir[d] * multiple;
        double curEnergy = getEnergy(node);
        if (curEnergy < bestEnergy) {
        bestEnergy = curEnergy;
        bestMultiple = multiple;
        }
        }
        for (int multiple = 64; multiple <= 128 &&
bestMultiple == multiple/2; multiple *= 2) {
        for (int d=0; d<nrDims; d++) pos[d] =
oldPos[d] + bestDir[d] * multiple;
         double curEnergy = getEnergy(node);
         if (curEnergy < bestEnergy) {
                  bestEnergy = curEnergy;
                  bestMultiple = multiple;
                }
             }

          for (int d=0; d<nrDims; d++) pos[d] =
oldPos[d] + bestDir[d] * bestMultiple;
        energySum += bestEnergy;
                    }
        System.out.println("iteration " + step
                    + "   energy " + energySum
                    + "   repulsion " + repuExponent);
        }

        // print statistics and warnings
        printStatistics();
       double minDistance = Double.MAX_VALUE,
maxDistance = 0.0;
       for (Node node1 : nodes) {
        if (node1.weight == 0.0) continue;
        final double[] position1 = positions.get(node1);
          for (Node node2 : nodes) {
            if (node2.weight != 0.0 && node1 !=
node2) {
                double dist = getDist(position1,
positions.get(node2));
          minDistance = Math.min(minDistance, dist);
          maxDistance = Math.max(maxDistance, dist);
            }
          }
       }
       if (maxDistance / minDistance > 1e9) {
          System.err.println(
              "The node distances in the layout are
extremely nonuniform.\n"
            + " The graph likely has unconnected or
very sparsely connected components.\n"
            + " Set random layout to recover, and
increase gravitation factor.");
       }

}
```

(2) kMeans.java:

```
import java.util.*;

public class kMeans {
```

```
    public Map<Node,Integer>
execute(Map<Node,double[]> nodeToPosition, int k)
    {
        Map<Node,Integer> means =
initialMeans(nodeToPosition, k);//store the mapping
of means and clusters
        int count = 0;
        //print all initial means
        for (Node node : means.keySet()) {
        System.out.println("mean: " + count + "-" +
node);
        count++;
        }

        Map<Integer, double[]> meansPosition = new
HashMap<Integer, double[]>();//store the positions
of means
        Map<Node,Integer> nodeToCluster = new
HashMap<Node,Integer>();//store the mapping of
nodes and clusters
        int cluster = 0;
        for (Node node : means.keySet()) {
        meansPosition.put(cluster,
nodeToPosition.get(node));//means' position
        cluster++;
        }

        //means' position is shown
        count = 0;
        for (Integer c : meansPosition.keySet()) {
        double[] p = meansPosition.get(c);
        System.out.println("mean "+ count + ": " + p[0]
+ " " + p[1] + " " + p[2]);
        count++;
        }

        System.out.println(meansPosition.size());
        int change = 1;
        while (change > 0) {
        change = 0;
        //compute the cluster which the node belong to
        double shortDis, disNodeToCenter;
        for (Node node : nodeToPosition.keySet()) {
        shortDis = Double.MAX_VALUE;
        for (int c = 0; c < meansPosition.size(); c++) {
        disNodeToCenter =
computeDisOfNodeToCenter(nodeToPosition, node,
meansPosition, c);
        if (disNodeToCenter < shortDis) {
        shortDis = disNodeToCenter;
        nodeToCluster.put(node, c);
                                        }
                                }
                }

        //recompute the positions of means
        for (int c = 0 ; c < meansPosition.size(); c++) {
        double[] mPosition = meansPosition.get(c);
        double[] mNewPosition = {0.0, 0.0, 0.0};
        int countNode = 0;
        for (Node node : nodeToPosition.keySet()) {
        if (nodeToCluster.get(node) == c) {
        double[] nodePosition =
nodeToPosition.get(node);
        mNewPosition[0] = mNewPosition[0] +
nodePosition[0];
    mNewPosition[1] = mNewPosition[1] +
nodePosition[1];
        mNewPosition[2] = mNewPosition[2] +
nodePosition[2];
        countNode++;
                                        }
                                }
        mNewPosition[0] = (double)
mNewPosition[0]/countNode;
        mNewPosition[1] = (double)
mNewPosition[1]/countNode;
        mNewPosition[2] = (double)
mNewPosition[2]/countNode;
        double nearestNode = Double.MAX_VALUE;
        double nNode = 0.0;
        Node nnNode = new Node();
        for (Node node : nodeToPosition.keySet()) {
        if (nodeToCluster.get(node) == c) {
        nNode = computeNewMeans(nodeToPosition,
node, mNewPosition);
        if (nNode < nearestNode) {
        nnNode = node;
        nearestNode = nNode;
                                                }
                                        }
                                }
        double[] newCenter =
nodeToPosition.get(nnNode);
        mNewPosition[0] = newCenter[0];
        mNewPosition[1] = newCenter[1];
        mNewPosition[2] = newCenter[2];
        if (mNewPosition[0] != mPosition[0] ||
mNewPosition[1] != mPosition[1] ||
mNewPosition[2] != mPosition[2]) {
        change++;
        meansPosition.put(c, mNewPosition);
                                }
                }
        }
        return nodeToCluster;
    }

    //compute the distance between the node and the
cluster center it belong to
    private double
computeNewMeans(Map<Node,double[]>
nodeToPosition, Node node, double[]
mNewPosition)
    {
        double[] position1 = nodeToPosition.get(node);
        return Math.pow((position1[0]-
mNewPosition[0]),2)+Math.pow((position1[1]-
mNewPosition[1]),2)+Math.pow((position1[2]-
mNewPosition[2]),2);
    }

    //compute the distant between centroids and nodes
    private double
computeDisOfNodeToCenter(Map<Node,double[]>
nodeToPosition, Node node1, Map<Integer,
double[]> meansPosition, int cluster)
    {
        double[] position1 =
nodeToPosition.get(node1);
        double[] position2 =
meansPosition.get(cluster);
        return Math.pow((position1[0]-position2[0]),2)
+ Math.pow((position1[1]-position2[1]),2) +
Math.pow((position1[2]-position2[2]),2);
    }

    private Map<Node,Integer>
initialMeans(Map<Node,double[]> nodeToPosition,
int k)
    {
        int count = 0;

        Map<Node, Double> iniNodes = new
HashMap<Node, Double>();
        Node maxNode = new Node();
```

```
        double mNode = 0.0;
        for (Node node : nodeToPosition.keySet()) {
        double randomNumber = Math.random();
        double nodeWeight = randomNumber *
node.weight;
        if(nodeWeight >= mNode) {
        maxNode = node;
                    }
        }

        Map<Node, Integer> means = new
HashMap<Node, Integer>();              // store the k
means
        means.put(maxNode,count);
        count++;   //count == 1

        double MAXDistant;  //Temporarily store the
distant information of the farthest node
        //get k means
        while (count < k) {
        MAXDistant = 0.0;
        Node newMean = new Node();
        for (Node node : nodeToPosition.keySet()) {
        double dis = 0.0;
        if (!means.containsKey(node)) {
        //compute the distant from current node to all
means
        for (Node node1 : means.keySet()) {
        dis = dis + computeDistant(nodeToPosition,
node, node1);
                                        }
        if (dis >= MAXDistant) {
    //store the farthest node, if two nodes have the
same farthest distant, get the later one
        if (means.containsValue(count)) {
        means.remove(newMean);
        }
    MAXDistant = dis;
    newMean = node;
    means.put(newMean, count);
        }
        }
        }
        if (means.size() > count) {
        count++;
        }
        }
        return means;
    }

    private double
computeDistant(Map<Node,double[]>
nodeToPosition, Node node1, Node node2)
    {
        double[] position1 =
nodeToPosition.get(node1);
        double[] position2 =
nodeToPosition.get(node2);
        return Math.pow((position1[0]-
position2[0]),2)+Math.pow((position1[1]-
position2[1]),2)+Math.pow((position1[2]-
position2[2]),2);
    }

}
```

(3) LayoutClustering.java:

```
private static Map<String,Map<String,Double>>
readGraph(String filename) {
        Map<String,Map<String,Double>> result =
new HashMap<String,Map<String,Double>>();
        try {
        BufferedReader file = new BufferedReader(new
FileReader(filename));
        String line;
        while ((line = file.readLine()) != null) {
            StringTokenizer st = new
StringTokenizer(line);
            if(!st.hasMoreTokens())            continue;
            String source = st.nextToken();
            String target = st.nextToken();
            double weight = st.hasMoreTokens() ?
Double.parseDouble(st.nextToken()) : 1.0f;
            if (result.get(source) == null) result.put(source,
new HashMap<String,Double>());
        result.get(source).put(target, weight);
            }
            file.close();
            } catch (IOException e) {
                System.err.println("Exception while reading
the graph:");
                        System.err.println(e);
                        System.exit(1);
            }
            return result;

            }


private static Map<String,Node>
makeNodes(Map<String,Map<String,Double>> graph) {
        Map<String,Node> result = new
HashMap<String,Node>();
        for (String nodeName : graph.keySet()) {
        double nodeWeight = 0.0;
        for (double edgeWeight :
graph.get(nodeName).values()) {
            nodeWeight += edgeWeight;
        }
            result.put(nodeName, new Node(nodeName,
nodeWeight));
            }
            return result;

            }


private static List<Edge>
makeEdges(Map<String,Map<String,Double>> graph,
        Map<String,Node> nameToNode) {
    List<Edge> result = new ArrayList<Edge>();
    for (String sourceName : graph.keySet()) {
        for (String targetName :
graph.get(sourceName).keySet()) {
            Node sourceNode =
nameToNode.get(sourceName);
            Node targetNode =
nameToNode.get(targetName);
            double weight =
graph.get(sourceName).get(targetName);
            result.add( new Edge(sourceNode, targetNode,
weight) );
        }
    }
    return result;

}
```

```
private static void writeFiles(Map<Node,double[]>
nodeToPosition, Map<Node, Integer> nodeToCluster,
String filename, int k) {
```

```
        try {
        BufferedWriter file = new BufferedWriter(new
FileWriter(filename.concat("-node.txt")));
        for (Node node : nodeToPosition.keySet()) {
        double[] position = nodeToPosition.get(node);
         int cluster = nodeToCluster.get(node);
        file.write(node.name + " " + position[0] + " " +
position[1] + " " + position[2] + " " + cluster);
          file.write("\n");
            }

        //save the cluster in another file
        String clusterfile = new String(filename.concat("-
clustering.txt"));
        BufferedWriter cfile = new BufferedWriter(new
FileWriter(clusterfile));
        for (int i = 0 ; i < k; i++) {
        for (Node node : nodeToCluster.keySet()) {
        if (nodeToCluster.get(node) == i) {
        cfile.write(node.name + " ");
            }
            }
            cfile.write("\n");
        }
            file.close();
            cfile.close();
            } catch (IOException e) {
              System.err.println("Exception while writing
the graph:");
            System.err.println(e);
            System.exit(1);
            }
        }
```