Detection and Prediction of Communities in Friendster

Instructor: Prof. Zhu Yifei, UMJI

Group 3

Lu Hengyi	ID: 517370910054
Gracia Stefani	ID: 517370990022
Wang Xiheng	ID: 517021910095

December 11 2020

University of Michigan - Shanghai Jiao Tong University Joint Institute (UM-SJTU JI)

Contents

Abstract

The idea of community detection is to divide nodes in complex networks into several groups by which the nodes belonging to the same group are tightly connected whereas those belonging to different groups are sparsely connected [6]. Community detection has received a great interest as it can be used to reveal useful information regarding a network. For instance, in social networks, it can reveal group of individuals with mutual interests [2]. In purchase networks, it can reveal group of customers with the same purchase habit, thus enabling the development of a more efficient recommendation system [3]. In criminal networks, it can also be used to track down criminal leaders from their subordinates [4]. There are numerous algorithms that have been proposed to perform community detection within networks with varying performance. However, the performance of these algorithms deteriorate as the structure of networks become more complex and the fraction of the neighbor of nodes belonging to different communities than that of the benchmarking node increases. Thus, choosing the proper algorithm to give a reliable community detection result is a tricky matter. The central node based link prediction (CLPE), was developed especially to detect communities in complex network [1]. What we want to show in this paper is that CLPE is a suitable algorithm for detecting communities in social network. In this project, used the CLPE strategy proposed in [1] to detect communities in real-world networks using the data of a set of 100,199 Friendster users with 14, 067, 887 friendship pairs that we have obtained from [5]. We then bagged the Friendster dataset into five groups and calculated the average conductance of detected communities (COND) of the CLPE strategy and compare it with the COND of the Girvan-Newman Algorithm for Community Detection.

1 Problem introduction

With the advancements of technology and the progress of Internet and social media, people are now more connected than ever. According to We Are Social's 2020 report, more than 4.5 billion people are on the internet with over 3.8 billion active social media users, which is around 49% of the world population [7]. Detecting communities within a social network is no longer a simple task, especially with the increasing number of nodes that may serve as a connection between that node's particular community to other communities. This phenomena can be explained with what is now called mutual friends in social media. Within a real-life network, it is very likely that someone within a tightly knit community knows another person from another community. Thus even if node A and B are not friends in the social network, they may have a mutual friend C. This effect is very significant in real-life networks. As the ratio of these cross-community connected nodes increases, it becomes increasingly difficult to separate different communities and the performance of most of the widely used community detection algorithm deteriorate with the complexity of the networks. Thus it is important to choose the right algorithm to detect communities properly within a social network with a high mixing parameter for a reliable community detection result. For this, we propose to use the CLPE method described in [1] to detect communities within social network.

2 Related articles

2.1 CLPE Algorithm[1]

The central node based link prediction (CLPE) method was developed to tackle the deteriorating performance of other community detection algorithms in the case of complex network with ambiguous community structure. Ambiguous community structure refers to networks having the mixing parameter greater than 0.5 [1]. The mixing parameter is the ratio of the neighboring nodes that belong to other communities in the network [1].

The CLPE algorithm is divided into three main steps:

1. Identifying central nodes in complex network
2. Predicting edges related with the central nodes
3. Community expansion and merging the communities according to their modularity

The implementation of the CLPE algorithm will be provided in the latter section. In the next sections, we will summarize some key ideas regarding the algorithm.

2.1.1 Central node based link prediction

The central node based link prediction refers to the first two steps of the CLPE algorithm. It is used to enhance the community structure of the complex network. Link prediction has been applied in community detection to find undiscovered edges within a network or to predict edges that might emerge in the future. The main idea behind link prediction is to use a prediction index to score the probability that an edge will emerge between each two nodes and adding edges with a high score into the network [1]. Node pairs with a high score are more likely to belong to the same community. However, according to the authors in [1], the current link prediction strategy's performance is unsatisfactory in dealing with networks with an ambiguous community structure.

In the case of complex networks, node pairs belonging to different communities will also get a high score. Thus, the author adapted the current link prediction strategy to the central node based link prediction.

A central node is a node located at the center of community, with edges connecting to the majority of the nodes in that community. By relying on central nodes to predict links, the edges within the community can be increased while the edges connecting to different communities can be reduced [1]. Thus, the central node based link prediction strategy can perform better than the commonly used link prediction strategy in dealing with complex networks.

To find the central node, the authors use a local density based central node identification strategy. For each node i, we define the node density ρ_i as

$$\rho_i = \sum_j \psi(d_{i,j} - d_c).$$

We define $\varphi(x) = 1$ if $x \leq 0$ and 0 otherwise. $d_{i,j}$ is the distance between nodes i and j and d_c is the cutoff distance, which is set to 1 for the CLPE algorithm. In this case, the node density is the degree of the node.

The relative distance δ_i is the minimum distance between node i and the other nodes with density higher than i,

$$\delta_i = \max_{\rho_j > \rho_i} (d_{i,j})$$

For the node with the largest density, we use $\delta_i = \max_j(d_{i,j})$. The central index of a node i is defined as

$$SC_i = \rho_i * \delta_i$$

Since the central node of a community usually have a higher density than its neighbors and a relatively large distance to nodes with a higher density, nodes with higher SC_i values are likely to be the central node of a community. To summarize, to find the central node, we perform the following:

1. Calculate the SC_i of each node i in the network
2. Sort the nodes in the descending order of their SC_i values and add the top half of the nodes to a candidate set CC
3. The first node CN_k of the CC is taken as a central node. If the distance between CN_k and another node in CC is smaller than the cutoff distance d_c, delete those candidates from CC.
4. Repeat (3) with the next node CN_{k+1}, omitting CN_k at each step until CC is empty.

After identifying the central nodes, we can then proceed with the central node based link prediction strategy. The prediction index related to the central nodes is defined as

$$F(c,x) = \sum_{z \in \Gamma(c) \cap \Gamma(x)} \left(\frac{S(c,z)}{deg(c)} + \frac{S(x,z)}{deg(x)} \right) \sum_{k \in \Gamma(x) | (\Gamma(c) \cap \Gamma(x))} \frac{S(x,k)}{deg(x)},$$

$$S(c,x) = \frac{|\Gamma(c) \cap \Gamma(x)|}{|\Gamma(c) \cup \Gamma(x)|}.$$

where c is the central node of a network, $\Gamma(c)$ is the set of all neighbors of c, $deg(c)$ is the degree of c, $S(c,x)$ is the *Jaccard* similarity between node c and x, and $\Gamma(x)|(\Gamma(c) \cap \Gamma(x))$ is the set of nodes in $\Gamma(x)$ that does not belong to $\Gamma(c) \cap \Gamma(x)$. $F(c,x) > 0$ indicates dependency of node x to the central node c, which suggests that node x belongs to the same community as node c, otherwise they are more likely to belong to different communities.

Let NI be the set of neighbors of a central node c and SI is a set of nodes that have common neighbors but does have an edge connecting to c. For the central node based link prediction, we perform the following:

1. For each node $v \in NI$, if $F(c,v) < 0$, remove the edge between nodes c and v.
2. For each node $v \in SI$, if $F(c,v) > 0$, add an edge between nodes c and v

2.1.2 Community expansion

The central node based link prediction clears the community boundary to a certain extent. To expand the community, we do the following. If a node v has not been assigned to a community, the algorithm first find all the neighboring nodes NI of v and checks whether the nodes in NI need to be added to node v's community. Next, the algorithm traverses nodes in NI in the descending oder of their SC values. A node v' in NI will be added in the community of node v if:

1. The neighboring nodes of v' have SC values greater than $SC_{v'}$.
2. Among these nodes, the node with the largest *Jaccard* index with node v' have been added to the community of v.
3. The largest similarity is not smaller than the average similarity between node v' and its neighbors.

Once the community associated with v is found, the algorithm proceed to find the communities associated with other nodes which have not been assigned to a community.

2.1.3 The proposed CLPE

The CLPE algorithm is summarized as follows

1. Use the previously described central node based link prediction strategy
2. Use the previously described community expansion algorithm
3. Merge communities if the community modularity Q is greater than the threshold α, where $0.3 \leq \alpha \leq 0.4$ for better performance. The modularity is counted as

$$Q = \sum_i e_{ii} - \sum_{ijk} e_{ij} e_{ki} = \text{Tr } \mathbf{e} - ||\mathbf{e}^2||,$$

where $||\mathbf{x}||$ indicates the sum of all elements of x, $\mathbf{e}$ is a $g \times g$ matrix whose component e_{ij} is the fraction of edges in the original network that connect vertices in group i to those in group j.

To better illustrate the algorithm, we use a toy example. Figure 1 shows the original network containing 9 nodes. After identifying the central nodes using the algorithm described previously, we obtained nodes 1 and 5 as the central node. This is shown in figure 2. We then predict the edges related to the central node. The resulting edges is shown in figure 3. Lastly, we performed community expansion and merged the communities according to their modularities. The final graph is shown in figure 4, showing two different communities {1,2,3,9} and {4,5,7,6,8}. The nodes belonging to each community are colored accordingly.

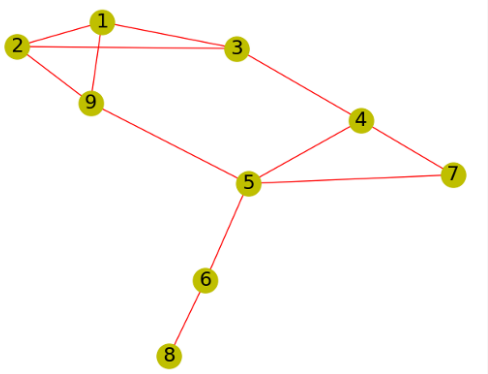

Figure 1: Original example network

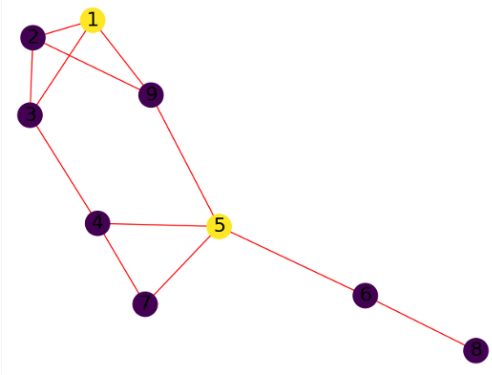

Figure 2: Network after finding the central nodes shown in yellow

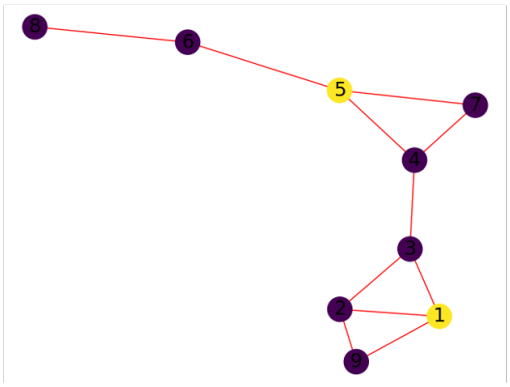

Figure 3: Network after link prediction

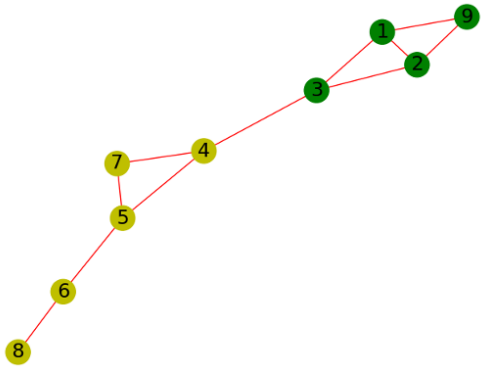

Figure 4: Founded communities (green and yellow).

2.1.4 Comparison between the CLPE algorithm and other community detection algorithms

The author has performed community detection on nine real-world networks using 7 different community detection algorithms, including the CLPE algorithm and compared their $COND$ values. The $COND$ value is calculated as

$$COND = \frac{1}{k}\sum_{s=1}^{k}\frac{cut(s)}{\min\{vol(s), vol(\overline{s})\}},$$

where k is the number of communities found in the network, $cut(s)$ is the number of edges with only one node in the community s and the other node in another community, $vol(s)$ is the sum of degree of all nodes in s and $\overline{s}$ is the complement of node s containing all nodes not in s. Smaller $COND$ values indicate better performance of the algorithm.

The result of the experiment is summarized in the table below

Network	EB	Louvain	SSCF	SCAN	CNM	CDFR	CLPE
karate	0.267	0.288	0.207	0.487	0.281	**0.132**	0.292
dolphin	0.243	0.292	**0.064**	0.533	0.316	0.153	0.219
football	0.318	0.285	0.340	0.482	0.288	0.309	**0.285**
Polbooks	0.278	0.276	0.189	0.502	0.255	0.185	**0.182**
netscience	0.005	0.002	0.201	0.039	0.003	0.858	**0.002**
PPI	0.401	0.199	0.294	0.496	**0.194**	0.727	0.198
Blogs	0.168	0.107	0.141	0.430	0.099	0.626	**0.097**
ca-HepTh1	0.284	**0.017**	0.254	0.326	0.029	0.705	0.257
Erdos	0.374	0.233	0.576	0.568	0.256	0.769	**0.233**

Table 1: Conductance of nine-real world networks from [1]

As we can see, the CLPE algorithm performs better on five real-world complex networks while still obtaining satisfactory values on the rest of the network. Due to this, we believe that using the CLPE algorithm is the right choice to analyze data from a complex network, which in our case is the social media Friendster.

2.2 Girvan-Newman Algorithm[6]

The Girvan-Newman algorithm was proposed to detect communities within a network basing on the property of community structure. Many system take the form of networks. Most network share the following properties:

1. Small world effect: the average distance between vertices in a network is usually short, scaling logarithmically with the total number of vertices.

2. Right-skewed degree distributions: in a network, there are typically many vertices with low degree and a few with high degree, the precise distribution often following a power-law or exponential form

3. Clustering (network transitivity): two vertices within network which are both neighbors of the same third vertex have a heightened probability of also being neighbors of one another

Girvan and Newman proposed another property, which is the property of community structure: network nodes are linked closely into tightly knit groups, between which there are looser connections [6]. The ability to detect community structure in a network can have many applications. For instance, communities in a social network which might represent real social groupings, communities in a citation network might represent related papers on a single topic, and many more. Thus, understanding communities within a network can be very useful in analyzing a network.

The traditional method for detecting community structure is hierarchical clustering. The weight $W_{i,j}$ is calculated for every pair i, j of vertices in the network to give a sense of how closely connected the vertices are. There are many different ways of calculating the weight. Then, n vertices in the network with no edges between them are taken and edges are added in the decreasing order of their weights. This results in a nested set of increasingly large components which are taken to be the communities. There are however many shortcomings to this method. The commonly use definition of weight have a tendency to separate single peripheral vertices from the communities

to which they should rightly belong. To tackle this shortcoming, the Girvin-Newman algorithm is proposed. Instead of constructing communities by adding the strongest edges to an initially empty vertex set, the proposed method removes edges that are most "between communities" to construct communities. The edge betweenness of an edge is calculated as the number of shortest paths between pairs of vertices that run along it. If there are $n_s > 1$ shortest paths between a pair of vertices, each path is given the weight $1/n_s$ such that the sum of their weights is unity. The shortest paths between different communities that are loosely connected by a few intergroup edges must go along these few edges. Thus, edges between different communities will have a high edge betweenness. As a practical matter, the fast algorithm of Newman was used to calculate the betweenness. The algorithm proposed by Girvin and Newman is as follows:

1. Calculate the betweenness for all edges in the network
2. Remove the edge with the highest betweenness
3. Recalculate betweenness for all edges affected by the removal
4. Repeat from step 2 until no edges remain

The complexity of the fast algorithm of Newman for a graph of m edges and n vertices in $O(mn)$. Since after the removal of each edge we only need to calculate the betweenness of those nodes affected, the entire algorithm runs in worst-case time $O(m^2n)$.

2.2.1 Tests of the method

The algorithm was tested on both computer-generated graphs and real-world networks for which the community structure is already know.

A graph of 128 vertices divided into four communities of 32 vertices each was constructed with edges placed between vertex pairs at random. The average degree z of a vertex was kept at 16. The graph have a known community structure but was random in other aspects. The result of the application of the algorithm shows that the algorithm could classify 90% or more of the vertices correctly when $z_{out} < 6$. However, when $z_{out} \geq 6$ the fraction correctly classified start to fall of substantially. They concluded that the algorithm performs very well almost to the point at which each vertex has as many inter-community as intra-community connections.

The algorithm was also tested on two real-world networks with known structures: Zachary's karate club study and college football. The result shows that the algorithm was able to detect the communities with high success and perform better than the hierarchical clustering method. However, the algorithm fails in cases where the intra-community connections are nearly as much as the inter-community connections. In other words, their algorithm doesn't perform well in cases where the network structure does not correspond to the actual community structure.

Lastly, the Girvan-Newman algorithm was tested on two real-world networks for which the structure was not known: collaboration network of scientist and food web of marine organisms in the Chesapeake Bay. They found that the algorithm was able to extract clear communities that appear to correspond to plausible and informative divisions of the network nodes.

2.3 Detecting community structure in networks[8]

In this paper, Newman reviewed some algorithms that has been proposed for community detection in networks. He reviewed some traditional approaches that are not ideal for real-world

networks and described a number of more suitable algorithms which was new at the time the paper was published. In his paper, he assumed that the network structure of interest is such that each vertex belongs to one of the communities and of the simplest kind possible, with a single type of undirected, unweighted edge connecting unweighted vertices of a single type [8].

2.3.1 Traditional approaches

1. Spectral bisection method[10],[11]

   Using the spectral bisection method, the network is divided into two communities by looking at the eigenvector corresponding to the second lowest eigenvalue and separating the vertices by whether the corresponding element in this eigenvector is greater than or less than zero. This is because for the case of two weakly coupled communities, there will be one eigenvector with eigenvalue slightly greater than zero and elements all positive for one community and all negative for the other. The performance of this method, however, is limited to graph that split nicely into two communities. Another disadvantage of this method is that it only bisects graphs and division into a larger number of communities i usually achieved by repeated bisection, which does not guarantee a satisfactory result. Moreover, it would require us to know how many communities we want to divide the graph into.

2. Kernighan-Lin algorithm[12] The Kernighan-Lin algorithm is a greedy optimization method. The benefit function Q is the number of edges that lie within the two groups associated with the possible division minus the number that lie between them. The algorithm requires the user to specify the size of the two groups. The two stages of the algorithm are:

   (a) Consider all possible vertices pairs in which each vertex is chosen from each of the groups and calculate ΔQ that would result from swapping them. Swap the pair that maximizes this change in the benefit function. The process is repeated until all vertices in one of the groups have been swapped once, note than each vertex can only be swapped once.

   (b) Iterating over the sequence of swaps that were made, take the point at which Q was highest to be the bisection of the graph.

   The main disadvantage of the Kernighan-Lin algorithm is that the sizes of the two communities must be specified ahead of the division. Since in most real-world data the sizes of groups is unknown, this method is unsuitable. Furthermore, since it only divides the network into two groups, dividing into several groups require multiple bisections, which does not guarantee best result.

   The two methods mentioned below uses the principal of *hierarchical clustering*. For this, one needs to develop a measure of the similarity $x_{i,j}$ between pairs (i, j) of vertices, of which many different measures are possible. Starting with an empty network of n vertices and no edges, the edges are added one by one in order of decreasing similarity. The original network is used only for the calculation of similarity measure.

3. Single linkage method[8]

   The single linkage method takes the components formed by the hierarchical clustering method as communities. At each addition of k^{th} the edge, the definitive property of the community is that any two vertices in the same community have a similarity of at least $x^{(k)}$. However,

the method does not place any general conditions on the similarities of vertices within the same community. $x_{i,j} \geq x$ is a sufficient but not a necessary condition for vertices i and j to be in the same community. As x is decreased, the communities join together. At the start, there are n components consisting of a single vertex each, and at the end there is just one component containing all vertices. The disadvantage of this method is that it requires the investigator to choose how many communities the network should be split into.

4. Complete linkage method[8]

   This method also make use of the hierarchical clustering, adding edges to an initially empty graph in order of decreasing similarity. The communities are defined as being the maximal cliques int he network. A maximal cliques is the largest set of vertices each of which is connected directly to every other vertices in the set. The definitive property is that every two vertices in the same community have similarity of at least the similarity of the latest edge that was added. Similar to the previous method, this method does not place any general conditions on the similarities of vertices within the same community. This method has a more desirable property than the single linkage method but is rarely used because finding cliques in a graph is a hard problem and cliques are usually not unique.

2.3.2 Recent approaches

1. The algorithm of Girvan and Newman[6] The algorithm has been summarized in subsection 2.2. The result of the entire algorithm can be represented as a dendrogram, progressing from the root to the leaves. The communities can be taken as any horizontal cross-sections of the dendrogram, depending on the position of the cut. The algorithm is more useful than the spectral bisection as it allows us to split the network into any other number of communities instead of just two. The main disadvantages of the Girvan-Newman algorithm is that it provides no guide to how many communities a network should be split to [8]. However, the modularity can be used to give an insight to how good the division was. Second, the worst-case running time of the algorithm is $O(m^2n)$ or $O(n^3)$ on a sparse graph with m edges to be removed and n vertices, which is slow.

2. The algorithm of Tyler et al.[13]

   The algorithm introduced by Tyler et al. is a variation of the Girvan-Newman algorithm that improves the speed of calculation at the reduction of accuracy. The algorithm suggest that only a subset of vertices i need to be summed over in calculating the (partial) betweenness scores for all edges. The number of vertices sampled is chosen to make the betweenness of at least one edge in the network greater than a certain threshold, ensuring the error falls below some satisfactory level chosen by the investigator. Vertices whose community assignment is ambiguous will sometimes be put in one community and sometimes in another, one can get a sense of which is more reliable by running the calculation multiple times.

3. The algorithm of Radicchi et al.[14]

   The Raddichi et al. algorithm is based on iterative removal of edges. The removal is based on counting short loops of edges in the networks–triangles are used in the simplest case. Edges that run between communities are less likely to belong to many short loops since it would require another edge that run between the same two communities to create a loop. Its

complexity is $O(m^4/n^2)$ or $O(n^2)$ on a sparse graph. The main disadvantage of this method is that it relies on the presence of triangle in the network. If the network has insufficient triangles, the algorithm will fail. Even though triangle counts are unusually high in most social networks, it is not the case in nonsocial network. Thus, the algorithm may only yield an acceptable result when run on social networks.

2.4 Discussion on research articles

From the articles, we can see that although there are many community detection algorithm, the performance is limited to some conditions and become less reliable when the network becomes more complex. The traditional methods are not ideal for real-world networks while the more approaches discussed in 2.3.2 tend to perform less reliably on networks on which the structure are not known or on network that have insufficient triangles. We have then concluded to use the CLPE algorithm to detect communities on large real-world social network using Friendster dataset since the algorithm may yield the most promising result for complex network.

3 Description of dataset

The dataset we will use is obtained from Friendster's friendship network. Friendster was a big US social networking site which once had more than 100 million users. The data is a public dataset from Social Computing Data Repository, which is published by Arizona State University in 2009. In this project, we will transform the data into an undirected graph using nodes to represent users and edges to represent the friendship between two users. The basic information regarding the data is shown on the table below.

Num of nodes	Num of edges	Average degree	Average clustering coefficient
100199	14067887	280.8	0.0807

Table 2: Basic information of the network

The degree distribution is shown in figure 5. Here, k refers to the degree of the nodes and P(k) refers to the probability that a randomly chosen node has degree k.

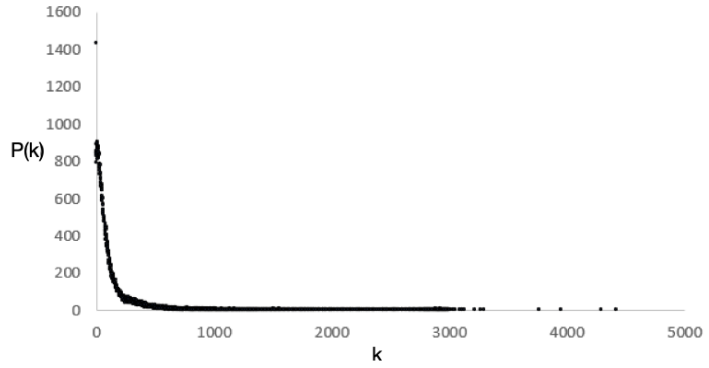

Figure 5: The distribution of degrees P(k)

4 Experimental results and analysis

4.1 Evaluation metrics and dataset

As CLPE is a combination of algorithm, to verify its correctness, it would be preferred to check the accuracy of both the subordinated and the total algorithms. While for the reference algorithm, Girvan-Newman, as it is deterministic, we only verify the total performance.

For the first target, we verify the accuracy of the method we choose on LinkPrediction, as it is the major source of error in community identification. We applied it to an independent benchmark graph given by `https://journals.aps.org/pre/abstract/10.1103/PhysRevE.78.046110`, which gives us the advantage of accounting for the heterogeneity in the distributions of node degrees and of community sizes. We count the correctness of prediction for different μ value, which is the ratio of each node connecting with the nodes in other communities, representing ambiguity of the community structure's border.

For the second target, we choose average conductance of detected communities (COND), which is defined by

$$COND = \frac{1}{k}\sum_{s=1}^{k} \mathrm{cut}(s) \,/\, \min\{\mathrm{vol}(s), \mathrm{vol}(\bar{s})\}$$

As it doesn't rely on some ground truth of the network, it is preferable for verification of real-world networks.

Because we are limited to comparing two algorithms on one data group, we tried to bag the data group to provide multiple verification. This is done by randomly deleting one thirds of the nodes in the Friendster data set and feeding it to the community detection algorithms.

Finally, we analyzed the performance by recording the time required by the two algorithms to run on both the bagged Friendster and the complete Friendster data set.

4.2 Results and anlysis

The results is shown as follows, the bar graph shows the correctly and wrongly added or deleced edges during LinkPrediction respectively. The line chart shows the error ratio. We can conclude that when the ratio μ of each node connecting with the nodes in other communities, which represents ambiguous community structure, is low, the prediction algorithm gives rather good performance, however, when μ increases, the performance degrades significantly, It now remains to be seen whether this will pose a problem to the detection of our dataset.

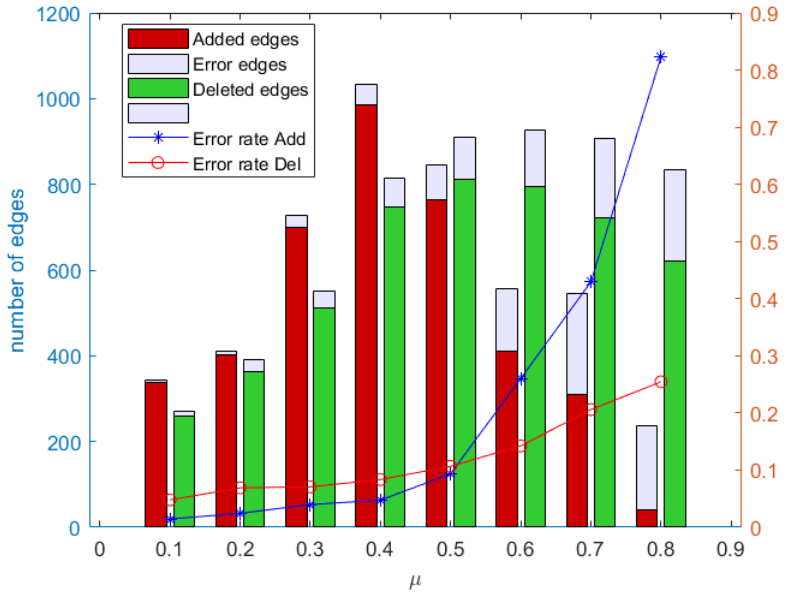

Figure 6: Correctness of link prediction

The COND values on the five bagged group of Friendster dataset is shown as follows.

Algorithm	GN	CLPE
Group 1	0.323	**0.219**
Group 2	0.302	**0.235**
Group 3	0.385	**0.297**
Group 4	0.376	**0.248**
Group 5	0.334	**0.227**

We can clearly see that despite the previously mentioned problem of link prediction used in CLPE, it shows clearly better result for every cases

The time complexity is also an important factor, according to papers, Girvan-Newman algorithm should have $O(NE^2)$ at worst case, but significantly better under most circumstances, especially when the community structure is clear. CLPE is said to have a (ON), which should lead to better performance for large E and N like Friendster Network.

The result is as follows:

	GN	CLPE
avg of bagged Friendster	217.8815s	417.0248s
complete Friendster	107.5323s	249.9237s

We can see that

- While CLPE is better than GN, the difference is not as significant as indicated by the theoretical time complexity, which suggests Friendster's structure is rather clear.
- GN performs relatively better in bagged sets, this is excepted as the network size is smaller.

5 Conclusion

Community detection has received a great interest since it can reveal useful information regarding a network. However, as the number cross-community connected nodes increase, the performance of most of the proposed community detection algorithm deteriorate. Since cross-community connected nodes are very common in social networks, it is crucial to choose the correct algorithm to detect communities in social networks as to give a reliable result. After conducting research and comparing several algorithms, such as the widely known Girvan-Newman algorithm, we decided to use the central node based link prediction algorithm proposed in [1] to detect communities in Friendster network. Our data consists of 100,199 nodes with 14,067,887 edges. We calculated the average degree to be 280.8 and the average clustering coefficient to be 0.0807.

There are three main steps in the CLPE algorithm: identifying central nodes in complex network, predicting edges related with the central nodes, and community expansion and merging the communities according to their modularity. We divided the Friendster data to five groups and ran both the CLPE and Girvan-Newman algorithm on each group. We then calculated the average conductance of detected communities of each group. Even though the correctness of the link prediction degrades when the ration of node connected with nodes in other communities increases, the result shows that the CLPE algorithm still has a lower COND value for each group compared to the Girvan-Newman algorithm, which indicates a better community detection performance.

References

[1] J. Hao, Z. Liu, C. Liu, Y. Su, and X. Zhang, "Community detection in complex networks with an ambiguous structure using central node based link prediction". *Elsevier*, Feb. 2020.

[2] J. Moody, and D. R. White, "Structural cohesion and embeddedness: A hierarchical concept of social groups". *American Sociological Review*, vol. 68, no. 1, pp. 103–127, Feb. 2003.

[3] P. K. Reddy, M. Kitsuregawa, P. Sreekanth, and S. S. Rao, "A graph based approach to extract a neighborhood customer community for collaborative filtering". *Databases in Networked Information System*, vol. 2544, pp. 188-200, Dec. 2002.

[4] T. Sangkaran, A. Abdullah, and N. Zaman, "Criminal network community detection using graphical analytic methods: a surver". *EAI Endorsed Transactions on Energy Web*, Jan. 2020.

[5] "Friendster dataset", 17zixueba, May. 2018. [Online]. Available: `https://www.17zixueba.com/thread-3520-1-1.html`

[6] M.Girvan, and M. E. J. Newman, "Community structure in social and biological networks". *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821-7826, Jun. 2002.

[7] "Digital 2020", We Are Social. [Online]. Available: `https://wearesocial.com/digital-2020`.

[8] M. E. J. Newman, "Detecting community structure in networks". *The European Physical Journal B*, vol. 38, pp. 321-339, Mar. 2004.

[9] Santo Fortunato, Darko Hric, "Community detection in networks: A user guide", 2016. `https://doi.org/10.1016/j.physrep.2016.09.002`.

[10] M. Fiedler, "Algebraic connectivity of graphs", *Czech. Math. J.*, no. 23, pp. 298–305, 1973.

[11] A. Pothen, H. Simon, and K.P. Liou, "Partitioning sparse matrices with eigenvectors of graphs", *SIAM J. Matrix Anal. Appl.*, no. 11, pp. 430–452, 1990.

[12] B. W. Kernighan, and S. Lin, "An efficient heuristic procedure for partitioning graphs", *Bell System Technical Journal*, no. 49, pp. 291–307, 1970.

[13] J. R. Tyler, D. M. Wilkinson, and B. A. Huberman, "Email as spectroscopy: Automated discovery of community structure within organizations", *Proceedings of the First International Conference on Communities and Technologies*, 2003.

[14] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks", Preprint cond-mat/0309488, 2003.

Appendix

The appendix includes the codes to the main function that has been implemented.

```
1
2  import csv
3  import Graph
4  nodes = csv.reader(open('nodes.csv','r'))
5  edges = csv.reader(open('edges.csv','r'))
6
7  network = Graph.Graph()
8
9  for e in edges:
10     network.AddEdge(e[0],e\cite{Ref1})
11
12
13 def r(graph,vertex):
14     length = 0
15     visited = set()
16     queue = []
17     queue.append(vertex)
18     visited.add(vertex)
19     while len(queue) != 0:
20         length += 1
21         v = queue.pop(0)
22         neighbors = graph.GetNeighbors(v)
23         for n in neighbors:
24             if n not in visited:
25                 if len(graph.GetNeighbors(n))>len(neighbors):
26                     return length
27                 queue.append(n)
28                 visited.add(n)
29     return length
30
31
32 def SC(graph,vertex):
33     return len(graph.GetNeighbors(vertex))*r(graph,vertex)
34
35 dic = {}
36 for node in nodes:
37     dic[node[0]] = SC(network,node[0])
38
39 V = sorted(dic.items(),key=lambda x:x\cite{Ref1},reverse=True)
40
41
42 CC = []
43 for i in range(int(len(V)/2)):
44     CC.append(V[i][0])
45
46 CN = []
47 while len(CC)!=0:
48     Ck = CC.pop(0)
49     for each in CC:
50         if network.IsAdjacent(Ck,each):
51             CC.remove(each)
52     CN.append(Ck)
53
```

```
f = open('CN.csv','w',encoding='utf-8', newline='')
csv_writer = csv.writer(f)
for each in CN:
    csv_writer.writerow([each])
```

Listing 1: FindCentralNode implementation

```

import csv
import Graph
nodes = csv.reader(open('nodes.csv','r'))
edges = csv.reader(open('edges.csv','r'))

network = Graph.Graph()

for e in edges:
    network.AddEdge(e[0],e\cite{Ref1})


def r(graph,vertex):
    length = 0
    visited = set()
    queue = []
    queue.append(vertex)
    visited.add(vertex)
    while len(queue) != 0:
        length += 1
        v = queue.pop(0)
        neighbors = graph.GetNeighbors(v)
        for n in neighbors:
            if n not in visited:
                if len(graph.GetNeighbors(n))>len(neighbors):
                    return length
                queue.append(n)
                visited.add(n)
    return length


def SC(graph,vertex):
    return len(graph.GetNeighbors(vertex))*r(graph,vertex)

dic = {}
for node in nodes:
    dic[node[0]] = SC(network,node[0])

V = sorted(dic.items(),key=lambda x:x\cite{Ref1},reverse=True)


CC = []
for i in range(int(len(V)/2)):
    CC.append(V[i][0])

```

```
CN = []
while len(CC)!=0:
    Ck = CC.pop(0)
    for each in CC:
        if network.IsAdjacent(Ck,each):
            CC.remove(each)
    CN.append(Ck)

f = open('CN.csv','w',encoding='utf-8', newline='')
csv_writer = csv.writer(f)
for each in CN:
    csv_writer.writerow([each])
```

Listing 2: LocalExpand implementation

```

import csv

import Graph

nodes = csv.reader(open('nodes.csv', 'r'))
edges = csv.reader(open('edges.csv', 'r'))

network = Graph.Graph()

for e in edges:
    network.AddEdge(e[0], e\cite{Ref1})


def r(graph, vertex):  # get the node density
    length = 0
    visited = set()
    queue = []
    queue.append(vertex)
    visited.add(vertex)
    while len(queue) != 0:
        length += 1
        v = queue.pop(0)
        neighbors = graph.GetNeighbors(v)
        for n in neighbors:
            if n not in visited:
                if len(graph.GetNeighbors(n)) > len(neighbors):
                    return length
                queue.append(n)
                visited.add(n)
    return length


def SC(graph, vertex):
    return len(graph.GetNeighbors(vertex)) * r(graph,
                                               vertex)  # return the central index
                                                   value, i.e.  relative distance x
```

```
                                                    node density
37
38
39  CN = []
40  dic = {}
41  Com = set()
42  for node in nodes:
43      dic[node[0]] = SC(network, node[0])
44  #  sort them in descending order according to SC
45  V = sorted(dic.items(), key=lambda x: x\cite{Ref1}, reverse=True)
46  visited = set()  # set all nodes in the network to be unvisited
47
48  for v in network.v_dict.values():
49      visited.add(v)
50      expand = set()
51      expand.add(v)
52      C = set()
53      C.add(v)
54      NI = v.get_neighbors() - visited
55      while len(expand):
56          expand.clear()
57          n_dic = {}
58          for node in NI:
59              n_dic[node[0]] = dic[node[0]]
60          #sort nodes in NI in descending order according to SC value
61          NI = sorted(n_dic.items(), key=lambda x: x\cite{Ref1}, reverse=True)
62          for ni in NI:
63              NN = set()
64              for n_ni in ni.get_neighbors():
65                  if dic[n_ni] > dic[ni] :# neighboring nodes of ni whose SC are
                        larger ni
66                      NN.add(n_ni)
67              if len(NN) > 0:
68                  max = 0
69                  max_i = list(NN)[0]
70                  oval = 0
71                  for i in NN:
72                      s_nn = (ni.get_neighbors()&i.get_neighbors())/(ni.get_neighbors
                            ()|i.get_neighbors())
73                      if s_nn > max:
74                          max = s_nn
75                          max_i = i
76                  oval = oval/len(NN) #  the average similarity between ni and its
                        neighbors
77                  S = (ni.get_neighbors()&max_i.get_neighbors())/(ni.get_neighbors()|
                        max_i.get_neighbors())
78                  if max_i in C & S > oval:
79                      expand.add(ni)
80                      C.add(ni)
81                      visited.add(ni)
82          NI = v.get_neighbors() - visited
83      Com.add(C)
84
85  # The function to calculate the COND value for test results
86      def Cond_test(Com,G):
87          Sol=[]
88          for (u,C) in enumerate(Com):
89              Sol.append(sum(n.degree for n in C))
```

```
90          Sol_sum=sum(Sol)
91          cond=0
92          for (u,C) in enumerate(Com):
93              cut = 0
94              for k in C:
95                  for i in G.neighours(k):
96                      if i not in C:
97                          cut+=1
98              cond+=cut/min(Sol[u], Sol_sum-Sol[u])
99          cond/= len(Com)
```

Listing 3: LocalExpand implementation