Finding community structure

Complex and Social Networks (2017-2018)
Lab 5 Submission

ANKIT TEWARI & STEFAN WACHMANN

November 29, 2017

Introduction

In this laboratory session, we've tried to build an understanding of different quality metrics for comparing the performance of a given community detection algorithm. We've used 9 different community detection algorithms on three different kind of datasets.

The datasets that we had used for experimentation are the famous "**karate**" dataset about **Zachary**'s karate club network dataset about the karate network. It has about 78 edges and 34 nodes. The second dataset we have used is the "**UKfaculty**" dataset which is the personal friendship network of a faculty of a UK university, consisting of 81 vertices (individuals) and 817 directed and weighted connections. The school affiliation of each individual is stored as a vertex attribute. This dataset can serve as a testbed for community detection algorithms. Finally, the third dataset that we've used is the "**macaque**" which is graph model of the visuotactile brain areas and connections of the macaque monkey. The model consists of 45 areas and 463 directed connections.

Outline: The remainder of the article is summarized as follows. We've tried to describe the results of our 4 different quality metrics for our 9 different community algorithms applied on three different kinds of datasets in the **Section 2.** We've explained the methods that we'd adopted for computing the four quality metrics in the **Section 3**. Finally, we'll go through a discussion of the results we had obtained in the **Section 4**.

Results

In the tables below, we've attempted to describe the results that we had obtained for different community detection algorithms. The results include values of the quality metrics such as Triangle Partition Ratio (TPT), Expansion, Conductance and Modularity for each of the community detection algorithm.

These values can be understood as explained in the following description-

a) Triangular Partition Ration- This is a criteria based on internal connectivity and it can be interpreted as the higher it's value, the better is the community structure resulting in the better performance of the community detection algorithm.

b) Expansion-This criteria is based on external connectivity. Therefore, the lower its value, the better is the community structure and the better performance of the community detection algorithm.
c) Conductance- This criteria is based on a combination of internal and external connectivity. Therefore, the lower its value, the better is the community structure and the better performance of the community detection algorithm.
d) Modularity- This criteria is based on the network model. Therefore, the higher its value, the better is the community structure and the better performance of the community detection algorithm.

The first table described below explains the results that we have obtained on applying the 9 different community detection algorithms on the dataset “Karate”.

	TPT	Expansion	Conductance	Modularity
edge.betweenness.community	0.2058824	1.7647059	0.20355677	0.4012985
fastgreedy.community	0.2352941	1.5882353	0.18214574	0.3806706
label.propagation.community	0.4411765	0.5882353	0.09903282	0.3800953
leading.eigenvector.community	0.2058824	1.7352941	0.22348070	0.3934089
multilevel.community	0.2352941	1.2352941	0.16055305	0.4188034
optimal.community	0.2352941	1.5000000	0.18549805	0.4197896
spinglass.community	0.2352941	1.2352941	0.18777533	0.4197896
walktrap.community	0.1470588	1.8823529	0.26344732	0.3532216
infomap.community	0.2941176	1.3235294	0.14779381	0.4020381

Table I (Quality Metrics for “Karate” Dataset)

The second table below describes the results that we have obtained on application of different community detection algorithms on the dataset “UKfaculty”.

	TPT	Expansion	Conductance	Modularity
edge.betweenness.community	0.3703704	1.185185	0.1427621	0.4284729
fastgreedy.community	0.2592593	2.938272	0.3377596	0.5622017
label.propagation.community	0.2592593	2.493827	0.4552868	0.5175679
leading.eigenvector.community	0.2962963	2.123457	0.3072461	0.5572471
multilevel.community	0.2592593	2.395062	0.3377596	0.5622017
optimal.community	0.2716049	2.148148	0.3005767	0.5632116
spinglass.community	0.1728395	2.765432	0.4456680	0.1029871
walktrap.community	0.2469136	2.617284	0.3868720	0.5578907
infomap.community	0.2098765	2.666667	0.4323708	0.5563369

Table II (Quality Metrics for “UKfaculty” Dataset)

This third table describes the results of the four quality metrics for the dataset “macaque”. We’ve observed that the value of TPT remains the same for each of the community detection algorithm. The result is interesting and we’ve attempted to explain it.

	TPT	Expansion	Conductance	Modularity
edge.betweenness.community	0.1111111	4.577778	1.0000000	0.2711803
fastgreedy.community	0.1111111	3.777778	0.8400000	0.3504729
label.propagation.community	0.1111111	0.000000	0.0000000	0.0000000
leading.eigenvector.community	0.1111111	4.311111	0.9801537	0.3563245
multilevel.community	0.1111111	3.066667	0.8666667	0.3750404
optimal.community	0.1111111	3.066667	0.8666667	0.3750404
spinglass.community	0.1111111	3.888889	0.9685185	0.3741638
walktrap.community	0.1111111	3.066667	0.9685185	0.3213072
infomap.community	0.1111111	2.200000	0.9787749	0.2755863

Table III (Quality Metrics for "Macaque" Dataset)

Methods

a) **Computing the Triangular Partition Ratio**:
While computing the Triangular Partition Ratio for each of the object returned by a given community detection algorithm, the following algorithm was put to work-

Step 0: Initialize a vector **Sum** with as many elements as there are clusters in the object returned by applying a specific of the community detection algorithm on the graph data under consideration. We initialize all elements of Sum with zero value.
Step 1: Repeat for each of the communities C[i] in the object:
Step 2: Find a subgraph SG[i] of the community C[i]
Step 3: Count the number of distinct vertices that are forming a triangle in SG[i] (the **triangle** function of igraph was utilized for this task along with unique function)
Step 4: Check if the distinct vertices belong to the cluster under consideration
Step 5: If TRUE then count the vertex in SUM[i] else IGNORE and CONTINUE from **Step 1**

A major difficulty was encountered while applying the unique function for finding the unique vertices. The object returned in this case was unable to be utilized for comparison with vertices of the cluster under consideration in the present iteration. So, we had tried to copy the elements of the object returned by unique function to a new list and fortunately this list was able to be compared with the vertices of the cluster under consideration. Although, we had to create an additional loop but for the present scenario, It is working perfectly fine. We'll be thinking to optimize it in the nearby future.

b) **Computing the Conductance**:
The formula for computing the value of conductance requires the number of edges internal to a cluster (m_c). We have developed a pseudocode for computing the value m_c. At present, since this function attempts to use a very fundamental approach of $O(n^3)$ for achieving its objective and hence is slow. The pseudocode we had used is described below-

```
Algorithm for computing Number of edges internal to a cluster-
# C means the set of all clusters {C1, C2, ....., CN}
# C_i refers to the i-th cluster under consideration

STEP 1: for each 'temp' in C:          i.e. iterate over each of the clusters
STEP 2:    for each i in C[temp]   i.e. iterate over each of the nodes in a particular cluster C[temp]
STEP 3:       for each j in 1:number_of_vertices_of_graph_G
STEP 4:          check if node i and node j have a node between them
STEP 5:             if (node i and node j have a link between them)    then
STEP 6:                check if (node j belongs to cluster C[temp]
STEP 6:                 If TRUE then count this edge    else ignore and continue further in loop;
```

NOTE: The indentation in STEPs is intentionally provided for explaining the loop inside a previous loop. In future, we'll be working on it further to either use more inbuilt functions of igraph for computing the desired value else we'll be coding it completely in C++ for faster response.

c) **Computing the Expansion**:
The formula for computing the value of expansion requires the number of edges at the frontier of a cluster (f_c) i.e. the number of edges pointing outside a given cluster. We have developed a pseudocode for computing the value f_c. At present, since this function attempts to use a very fundamental approach of $O(n^3)$ for achieving its objective and hence is slow. The pseudocode we had used is described below-

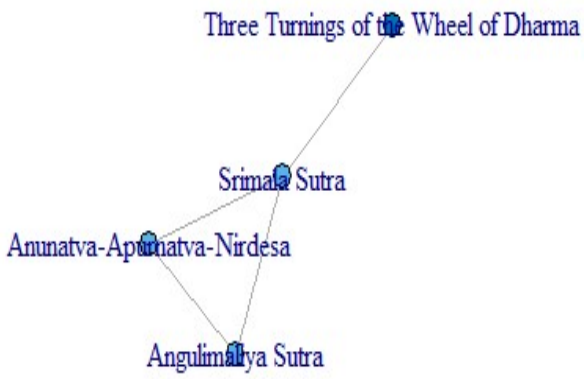
```
Algorithm for computing Number of edges internal to a cluster-
# C means the set of all clusters {C1, C2, ....., CN}
# C_i refers to the i-th cluster under consideration

STEP 1: for each 'temp' in C:          i.e. iterate over each of the clusters
STEP 2:    for each i in C[temp]   i.e. iterate over each of the nodes in a particular cluster C[temp]
STEP 3:       for each j in 1:number_of_vertices_of_graph_G
STEP 4:          check if node i and node j have a node between them
STEP 5:             if (node i and node j have a link between them)    then
STEP 6:                check if (node j does not belongs to cluster C[temp]
STEP 6:                 If TRUE then count this edge    else ignore and continue further in loop;
```

d) **Computing the Modularity**:
The assumptions of function **modularity** which is used for computing the value of modularity requires that the graph data should be strictly undirected. However, except the first dataset, the rest were directed graphs. In such a case, we had made an improvisation by using the **as.undirected** function from igraph for converting the directed graphs to undirected graphs before computing the four quality critera.

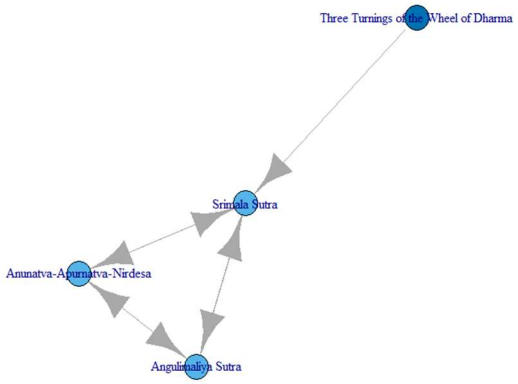
Discussion

a) While performing the Task 1, we had noticed that on applying the 9 different community detection algorithms on the dataset “macaque”, the value of Triangle Partition Ratio remains same for all of the algorithms. There is a significant reason behind this fact. On investigation, we noticed that for every community detection algorithm applied on the dataset “macaque”, the first cluster of the object returned by the community detection algorithm always had the same number of vertices forming a triangle or a triad internally in the cluster. This value was ‘5’. So, consider the object returned by applying the fastgreedy algorithm on the macaque dataset. The number of nodes in each community were 18, 14 and 13. So, the Weighed average value of Triangle Partition Ratio for this algorithm was (18/ (18+14+13)*(5/18) + 0 + 0) which turns out to be 5/45 or 0.111111.

b) We had applied the fastgreedy community detection algorithm on the wikipedia.gml dataset. The result is interesting. The algorithm had partitioned the dataset into 1037 communities. Of these 1037 communities, the first five communities had about 17000 nodes altogether. But, interestingly, the remaining communities had roughly few hundred nodes in each community. Finally, in most of the communities, there were less than 10 nodes each.
Although, plotting the whole graph doesn’t makes sense visually for obvious reasons. So, we decided to conduct a visual examination reveals that there were some communities that really make sense. For e.g. there exists a community of religious documents from Hindu religion’s books. It has been plotted below-

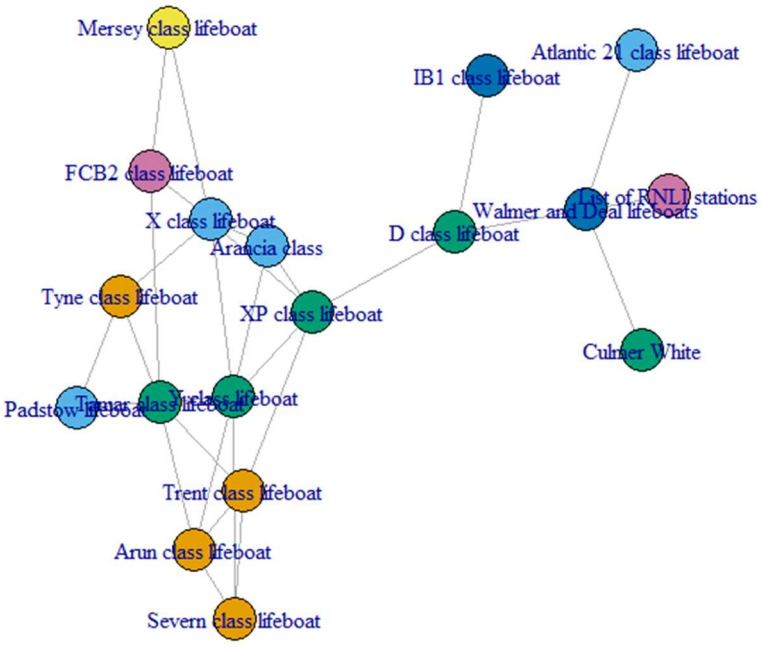


Undirected Graph of Hindu Religious Documents
(1)



Directed Graph of Hindu Religious Documents
(2)

Similarly, we have a graph of the following community. It was the 40th community chosen randomly for investigation. This community contains the documents associated with class lifeboat.



Overall, we can conclude that the communities with relatively fewer number of nodes did really make sense. However, for communities that had comparatively higher number of nodes, we need to explore them further. In such initiatives, there are some researches which includes approached of Natural Language Processing for properly detecting communities. One such fundamental approach has been described in the following paper- Improving Community Detection in Wikipedia Articles using Semantic Features. We plan to conduct some such experiments using semantic features in the nearby future to further develop a robust method for understanding community structure in graphs such as wikipedia.gml.

References

a) [Brandes et al., 2008] Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008). On Modularity Clustering. IEEE Transactions on Knowledge and Data Engineering, 20.

b) [Clauset et al., 2004] Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. Physical Review E.

c) [Newman, 2006] Newman, M. E. J. (2006). Finding community structure in networks using the eigenvectors of matrices. Physical review. E, Statistical, nonlinear, and soft matter physics, 74:036104.

d) [Newman and Girvan, 2004] Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. Physical review. E, Statistical, nonlinear, and soft matter physics, 69(2 Pt 2):026113.

e) [Pons and Latapy, 2005] Pons, P. and Latapy, M. (2005). Computing communities in large networks using random walks. Journal of Graph Algorithms and Applications, 10:191–218

f) [Raghavan et al., 2007] Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. Physical review. E, Statistical, nonlinear, and soft matter physics, 76:036106.
g) [Reichardt and Bornholdt, 2006] Reichardt, J. and Bornholdt, S. (2006). Statistical mechanics of community detection. Physical Review E, 74.
h) [Rosvall and Bergstrom, 2008] Rosvall, M. and Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. Proceedings of the National Academy of Sciences of the United States of America, 105:1118–1123.