Community detection algorithms

for real and artificial networks

Project Report - Group 3

Fábio Cruz

Nr. 96957

fabio.rocha.cruz

@tecnico.ulisboa.pt

Luís Fernandes

Nr. 97018

luis.martins.fernandes

@tecnico.ulisboa.pt

**ABSTRACT**

In this paper, we analyse different community detection algorithms along with real and artificial benchmark networks. We also compare and evaluate those algorithms by analysing their internal and external measures, ranking them using some of those criteria and checking the accuracy of the methods by comparing the resulting partitions with the expected ones for the different networks. TBD

# INTRODUCTION

Complex networks are an important area of research. With the exponential growth of science and technology in the past few years, a lot of work and attention has been pointed at this area from the scientific community. In the context of network theory, complex networks are essentially graphs that are composed by connections between different entities, and can be characterized by different topological measures, such as degree distribution, clustering coefficient, community structure and others.

Community structure is defined on how the nodes of a network can be grouped into clusters/groups in such a way that a set of nodes is densely connected internally. These sets of nodes are called communities and can be found in several different types of networks such as biological, technological, and social networks. If a pair of nodes belongs to the same community the probability of them being connected is higher than if they don’t. Usually, nodes are exclusive to only one community, but in specific cases a network can have overlapping communities where a node can belong to more than one, and some networks may not even have community structure such as random graphs and the Barabási-Albert model.

Finding communities within a network can be computationally expensive since initially we usually don’t know how many communities exist and how big they are. There are several community detection algorithms that can be divided essentially in two types: Agglomerative and Divisive methods. The idea behind Agglomerative methods is to start with disconnected nodes and add edges one by one in order to group up nodes with a high similarity into the same community, while in the Divisive methods we already start with a complete network with connected nodes and cut low similarity edges that tend to connect different communities.

Methods based on modularity optimization are also one of the most important type of community detection methods. Modularity is a measure of the structure of networks that calculates the strength of the division of the network into different sets of nodes. Networks with high modularity have nodes more densely connected with other nodes from the same community but have a weak connection with other communities. These types of methods aim to maximize modularity at each iteration by, for example, changing links between different pairs of nodes and calculating which configuration of the network gives a higher modularity value. However, modularity is known to have some problems such as the resolution limit for large enough communities, which makes the algorithm sometimes incapable of detecting smaller communities.

In this paper we aim to analyse and evaluate different community detection algorithms and compare their accuracy when applied to real and artificial networks. In the next two sections we present what algorithms we will test and briefly explain how they work, and we will introduce the networks where we will apply those methods.

# COMMUNITY DETECTION ALGORITHMS

We chose to analyse and compare some popular community detection methods that are known to give good results and be efficient for large complex networks, but also that have different types of implementation and ideologies.

## Girvan-Newman algorithm

The first algorithm we decided to use was the Girvan-Newman algorithm, named after the popular complex network researchers Michelle Girvan and Mark Newman. The main idea of this algorithm is to sequentially remove edges/links that connect nodes that hopefully belong to different communities (Divisive method).

Divisive methods use centrality measures that are high for nodes that belong to different communities and low for the ones that belong to the same one. Girvan-Newman uses link betweenness as its centrality measure. The higher the link betweenness of an edge, the higher the probability of that edge connecting different communities.

The steps to run this algorithm can be simplified to this:

1. Compute centrality/betweenness for each link
2. Remove the link with the highest centrality value. In case of a tie, choose randomly
3. Recalculate centrality values after removing the edge
4. Repeat step 2 and 3 until all edges are cut

At the end of the algorithm we get a dendrogram. To obtain the final result with the community structure of the network we need to “cut” the dendrogram at a certain level. One way to choose at what level we should cut is to calculate the modularity for a set of different levels and choose the best one. The computational complexity of this algorithm is approximately O(*m*2*n*) for the worst case, where n is the number of nodes, and m the number of edges of the network.

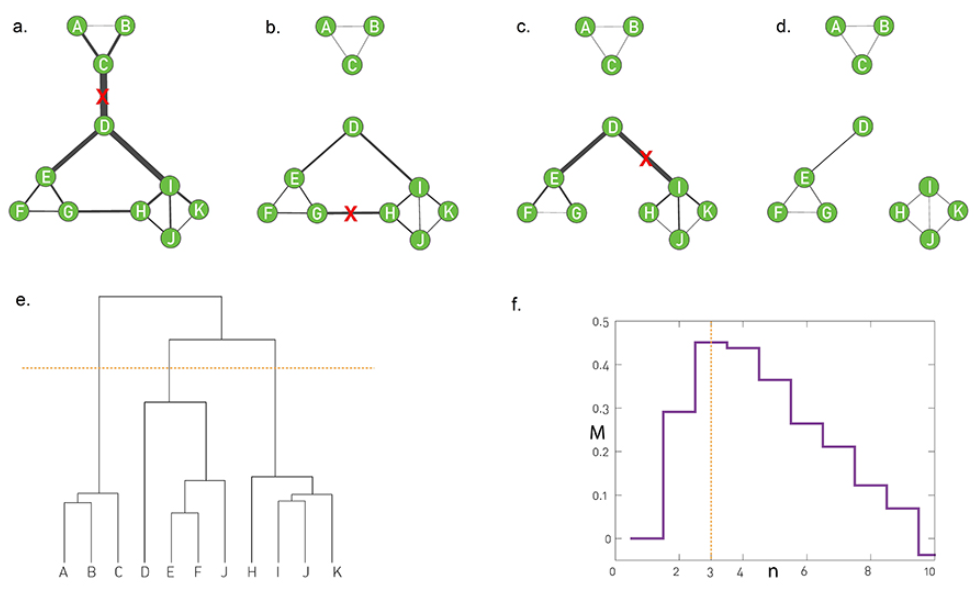


Figure 1 - http://networksciencebook.com/chapter/9#hierarchical

## Louvain method

The next algorithm we decided to test was the Louvain method, created by Blondel et al. from the University of Louvain. This is an Agglomerative method that uses modularity optimization to find communities and can be used on very large networks since it is very efficient from a computational point of view (complexity of O(*m*)).

Initially, the algorithm assigns a different community for each node of the network. Then for each node it calculates the modularity value if the node is changed from the current community to a neighbour community. The change will occur if modularity is maximized that way, otherwise nothing changes. This process is repeated until no more improvements to the modularity of the network can be done. After this, the algorithm transforms the current communities into nodes and reapplies the previous procedure. Again, this is repeated until no more improvements are accomplished in the network. Because of this the Louvain method is capable of finding communities inside communities, and it is easy to implement and very time efficient. However it has some problems…

## Leiden method

One of the Louvain method problems is that it sometimes can discover weekly connected communities. To solve this problem, T.A. Traag has created the Leiden method which is also faster than the previously mentioned method.

This algorithm is very similar to Louvain. In the first phase Leiden only visits the nodes whose neighbourhood has changed instead of all the nodes, and it adds a new phase after the ones explained before, where it essentially tries to refine the partitions made in the second phase where new communities can be created from the previous existing communities. This refinement means that a node may be merged with a randomly chosen community which increases some quality function.

## Infomap

Last but not least, we decided to add to our set of algorithms the Infomap method to compare it with the other more known and used algorithms such as the Girvan-Newman and the Louvain, and see if Infomap is also a viable option in terms of accuracy and time complexity.

This algorithm is a little complex to describe, but essentially the Infomap algorithm tries to minimize a cost function and is based on data compression for community detection. It is accomplished by encoding the best trajectory of a random walker in the network, using something called a map equation.

Every node is labelled with a code. If a random walker walked inside our network, we would want to describe its path with the least number of symbols possible using those codes. But we know that normally a random walker tends to stay “trapped” for longer when it is inside a community, so we can partition the network and give a code to every community as well so we can optimize and find the shortest message possible with those community codes. Essentially, we associate the paths that are more used by the random walker with edges between nodes that belong to the same community, and paths or edges that are not travelled a lot are probably connecting different clusters of nodes. If we partition the network into too many modules the message encoded becomes bigger than what it needs to be, so this way we can find an optimal partition that assigns nodes to modules in such a way that the information needed to compress the movement of our random walkers is minimized (the cost function).

The reason this method is also very good in terms of time complexity (O(*m*)) is because instead of brute forcing this algorithm, which would be bad, they use a variation of the Louvain method to help find good partitions.

# METHODS

To implement and apply all of these community detection methods we usedNetworkXand CDlib, a community detection library for python with lots of algorithms already implemented

# RESULTS

## Networks

## Best Girvan-Newman parameter

## Best Infomap parameter

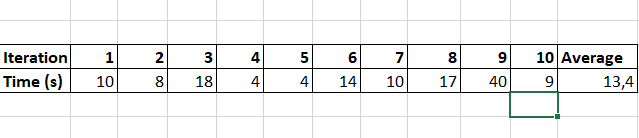
## Running times

## Community properties

## Accuracy

Regarding the results,

Reactive Pac-Man vs Original Ghosts:



Reactive Pac-Man vs Improved Ghosts:

# CONCLUSIONS

First, regarding the results observed above we can take some conclusions.

Testar clique percolation

Testar robustness modularity

.

# ACKNOWLEDGMENTS

Our thanks to all the professors of the course for the help and material provided, to Armin Kazemi for the base Pacman project provided in the Github platform and to all the creators of the information that we used to implement this project.

# REFERENCES

1. Course materials regarding agents and multi-agent systems
2. <https://www.computerhope.com/jargon/p/pacman.htm>
3. <https://en.wikipedia.org/wiki/Pac-Man>
4. <https://www.youtube.com/watch?v=qwhXIzNrb9w>
5. <https://github.com/arminkz/Pacman>