CN - Activity 3

Community detection

Carlos García

May 2017

Contents

1 Project description

Apply at least three different community detection algorithms for the attached networks. It is not necessary to implement them, you may use any freely available software. At least one of the algorithms must be based on the optimization of modularity (but not all of them), and you must use at least two different programs (i.e. do not use the same application all the time).

Some of the provided networks have a partition of reference, obtained from external information. In these cases, you have to compare your partitions with them, using at least the following standard measures: Jaccard Index, Normalized Mutual Information (arithmetic normalization), and Normalized Variation of Information. It is not necessary to implement the calculation of these indices, you may use any program (e.g. Radatools).

The objective is to compare the partitions obtained with the different algorithms, to try to conclude which is the best method you have found.

2 Program and libraries

As for prior projects, I decided working with Python, mainly because it is a robust language with a lot of useful libraries and lots of documentation. According with the project description we should use at least two different programs, then, I will use two different well-known libraries, they are igraph [1] and NetworkX [2], the first one will be working with the first two community detection algorithms (Edge betweenness and greedy optimization of modularity) and the second one will work with the third community detection algorithm (Louvain method), we will focus on this algorithms in the next chapter.

3 Algorithms used

There are several methods to detect communities in networks, this ones I am using were selected simply because they are popular and people usually talk about them in several posts.

3.1 Edge betweenness

Community structure detection based on the betweenness of the edges in the network. The algorithm was invented by M. Girvan and M. Newman, see: M. Girvan and M. E. J. Newman: Community structure in social and biological networks, Proc. Nat. Acad. Sci. USA 99, 7821-7826 (2002).

The idea is that the betweenness of the edges connecting two communities is typically high, as many of the shortest paths between nodes in separate communities go through them. So we gradually remove the edge with highest betweenness from the network, and recalculate edge betweenness after every removal. This way sooner or later the network falls off to two components, then after a while one of these components falls off to two smaller components, etc.

until all edges are removed. This is a divisive hierarchical approach, the result is a dendrogram [3].

3.2 Greedy optimization of modularity

This function implements the fast greedy modularity optimization algorithm for finding community structure, see A Clauset, MEJ Newman, C Moore: Finding community structure in very large networks [4].

Quoting from the paper: "Here we present a hierarchical agglomeration algorithm for detecting community structure which is faster than many competing algorithms: its running time on a network with n vertices and m edges is O(m d log n) where d is the depth of the dendrogram describing the community structure. Many real-world networks are sparse and hierarchical, with m ~ n and d ~ log n, in which case our algorithm runs in essentially linear time, O(n $\log^2 n$)."

Some improvements proposed in K Wakita, T Tsurumi: Finding community structure in mega-scale social networks, have also been implemented.

3.3 Louvain method

This method consists of two phases. First, it looks for "small" communities by optimizing modularity in a local way. Second, it aggregates nodes of the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained [5].

The output of the program therefore gives several partitions. The partition found after the first step typically consists of many communities of small sizes. At subsequent steps, larger and larger communities are found due to the aggregation mechanism. This process naturally leads to hierarchical decomposition of the network.

This is obviously an approximate method and nothing ensures that the global maximum of modularity is attained, but several tests have confirmed that our algorithm has an excellent accuracy and often provides a decomposition in communities that has a modularity that is close to optimality.

Moreover, due to its hierarchical structure, which is reminiscent of renormalization methods, it allows to look at communities at different resolutions.

The method was first published in: Fast unfolding of communities in large networks, Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000.

4 Resulting plots

4.1 Toy networks

4.1.1 20x2+5x2

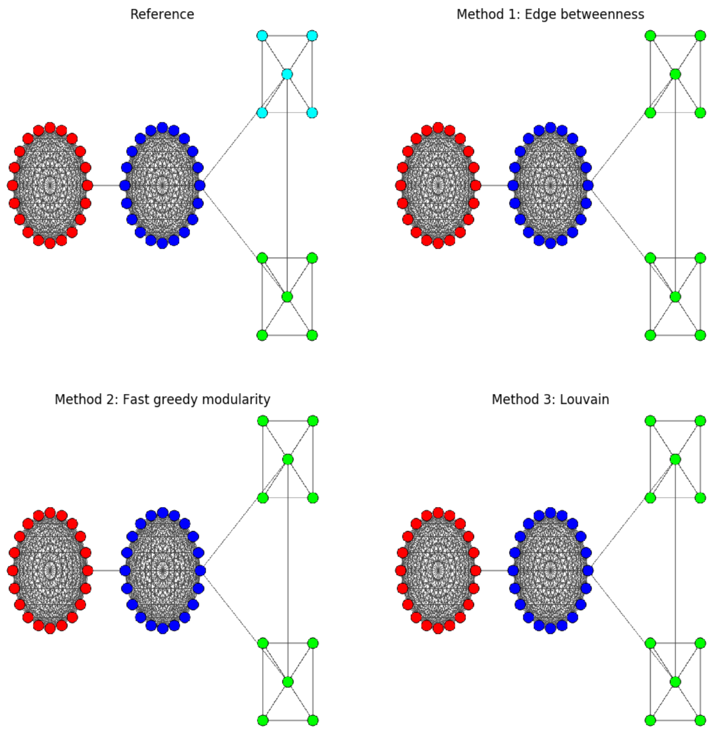
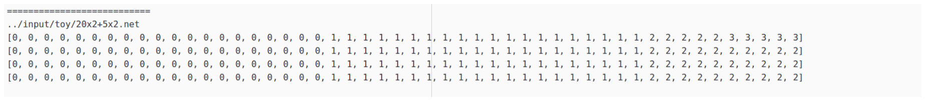


Figure 1: 20x2+5x2, plot from all methods



```
============================
../input/toy/20x2+5x2.net
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

Figure 2: 20x2+5x2, community detection sample values

4.1.2 graph3+1+3

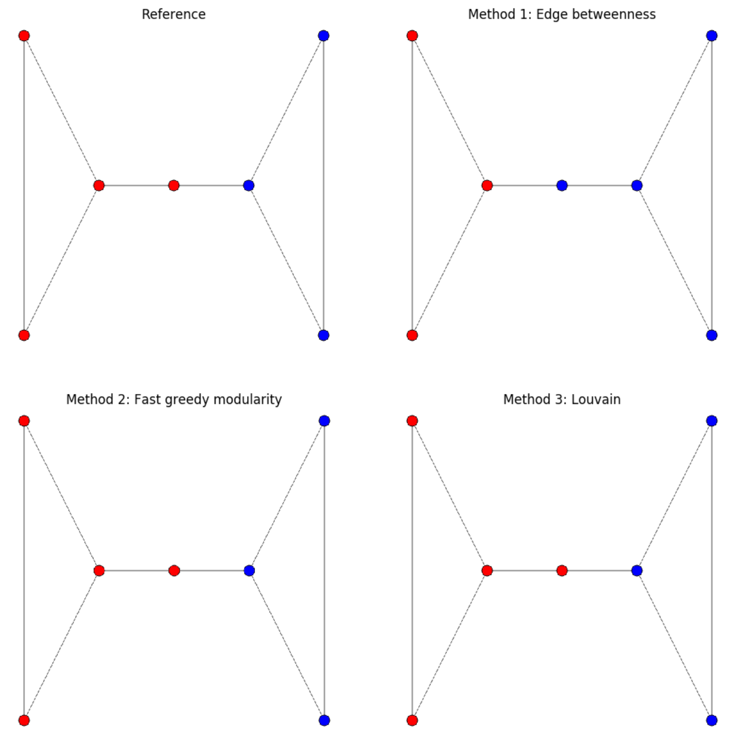
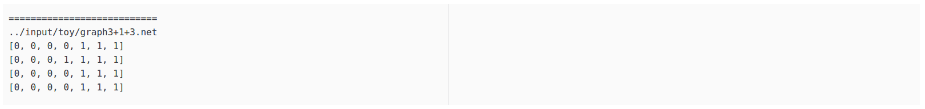


Figure 3: graph3+1+3, plot from all methods



```
===========================
../input/toy/graph3+1+3.net
[0, 0, 0, 0, 1, 1, 1]
[0, 0, 0, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 1, 1]
[0, 0, 0, 0, 1, 1, 1]
```

Figure 4: graph3+1+3, community detection sample values

4.1.3 graph4+4

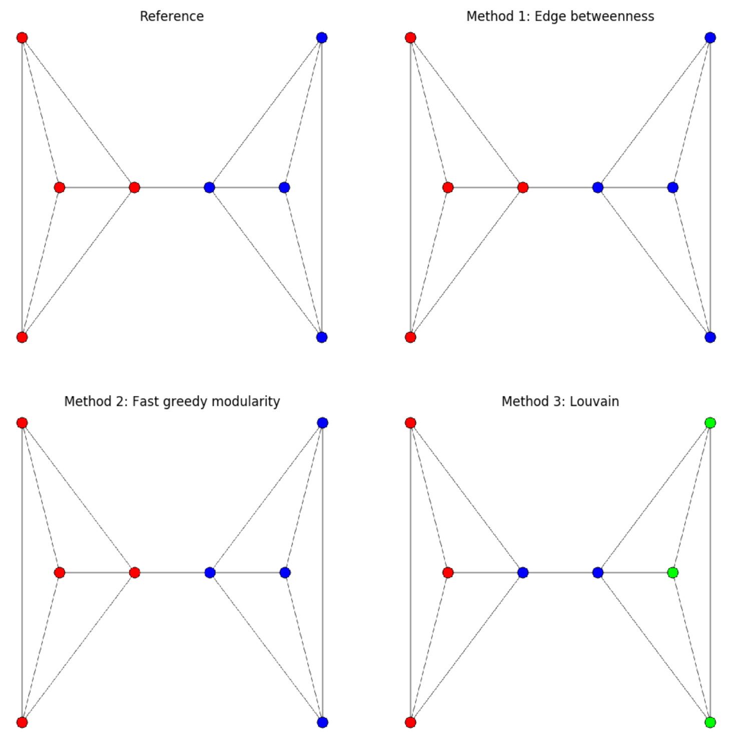
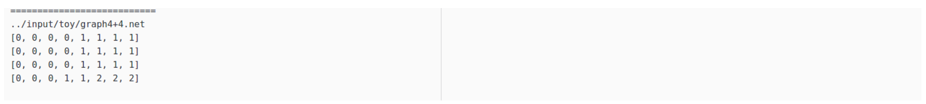


Figure 5: graph4+4, plot from all methods



```
===========================
../input/toy/graph4+4.net
[0, 0, 0, 0, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 1, 1, 1]
[0, 0, 0, 1, 1, 2, 2, 2]
```

Figure 6: graph4+4, community detection sample values

4.1.4 star

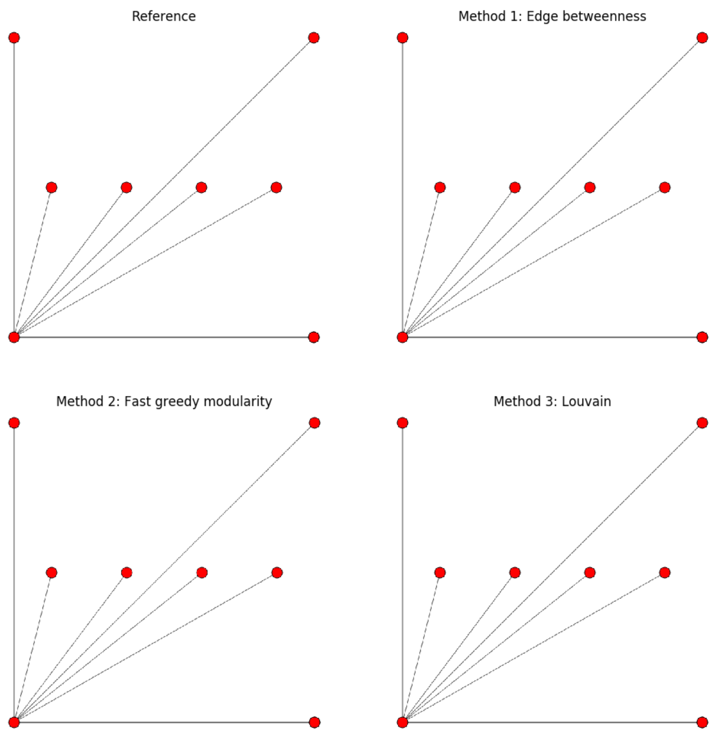
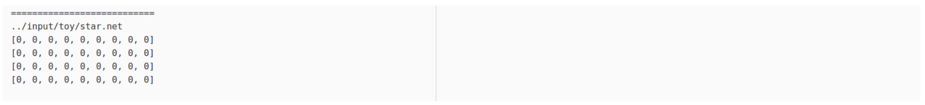


Figure 7: star, plot from all methods

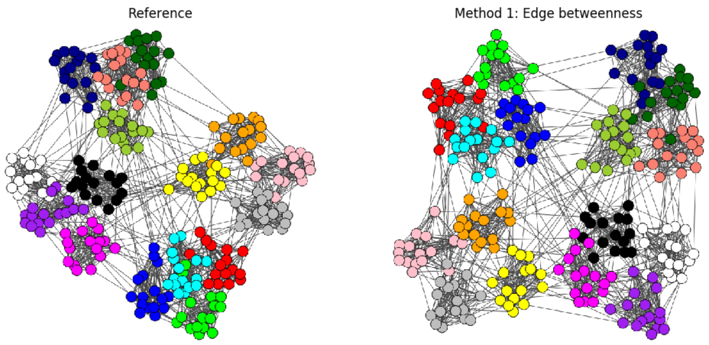


```
===========================
../input/toy/star.net
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 8: star, community detection sample values

4.2 Model networks

4.2.1 256_4_4_2_15_18_p



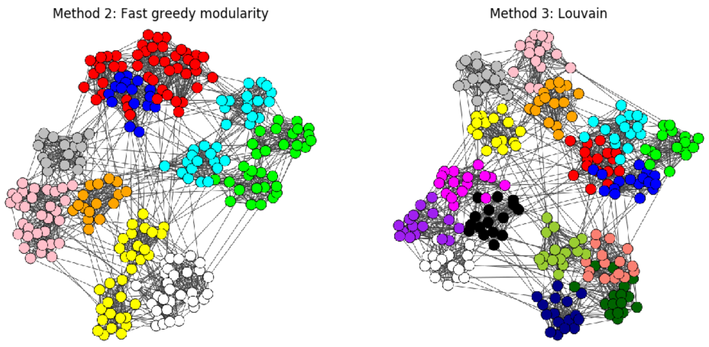
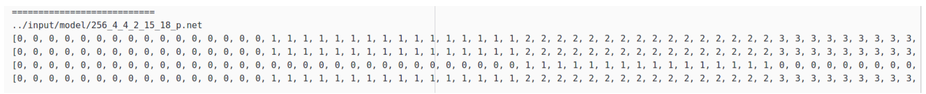


Figure 9: 256_4_4_2_15_18_p, plot from all methods



```
===========================
../input/model/256_4_4_2_15_18_p.net
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
```

Figure 10: 256_4_4_2_15_18_p, community detection sample values

4.2.2 256_4_4_4_13_18_p

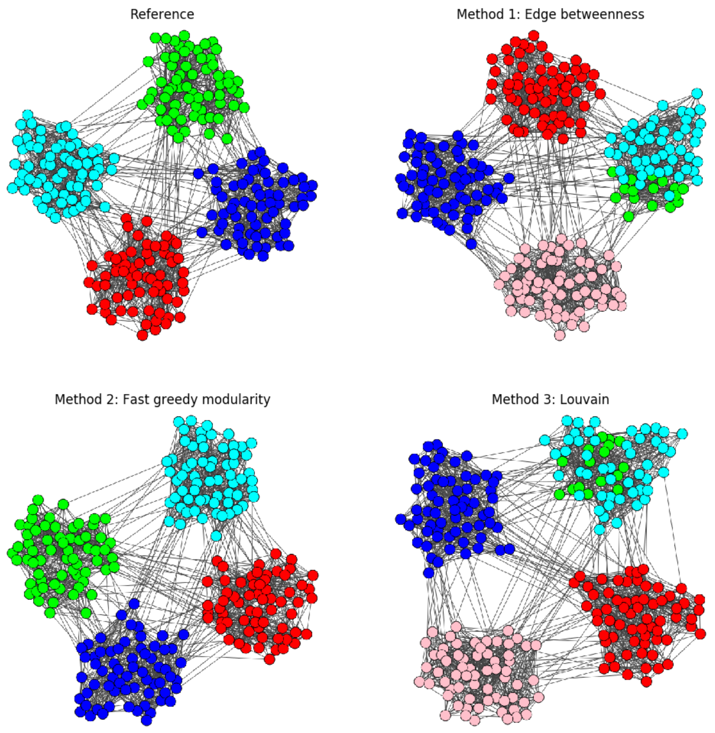
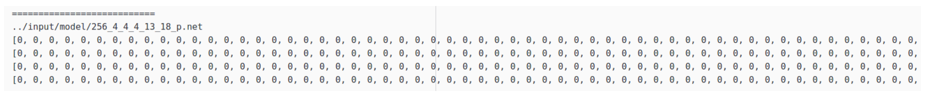


Figure 11: 256_4_4_4_13_18_p, plot from all methods



```
===========================
../input/model/256_4_4_4_13_18_p.net
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 12: 256_4_4_4_13_18_p, community detection sample values

4.2.3 rb125

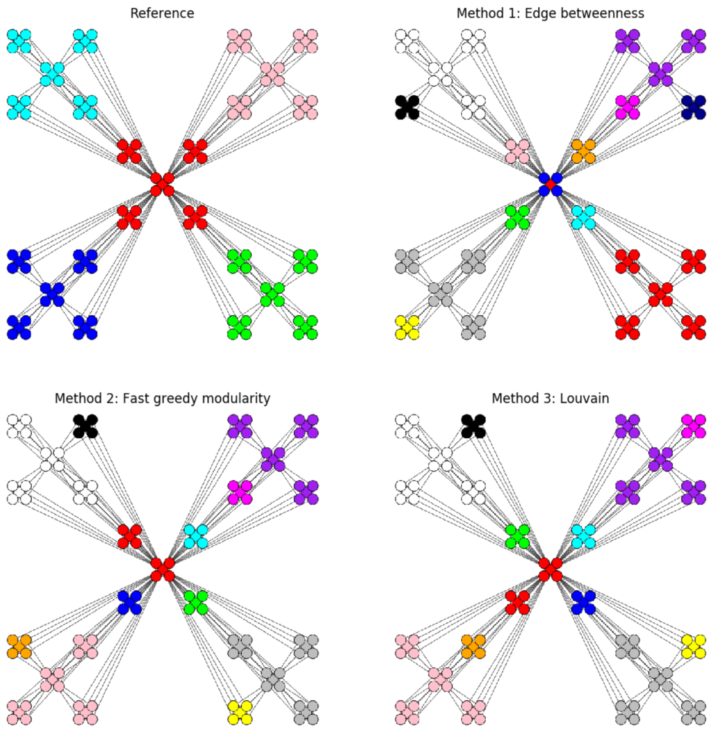
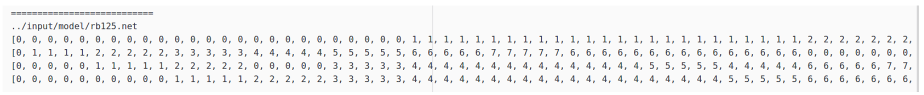


Figure 13: rb125, plot from all methods



```
=============================
../input/model/rb125.net
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
[0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 7, 7,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6,
```

Figure 14: rb125, community detection sample values

4.3 Real networks

4.3.1 airports_UW

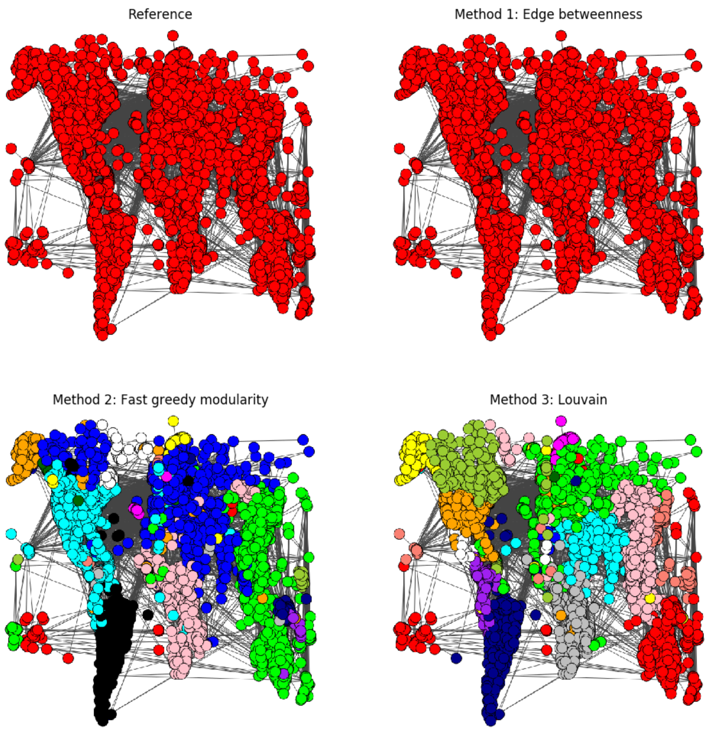
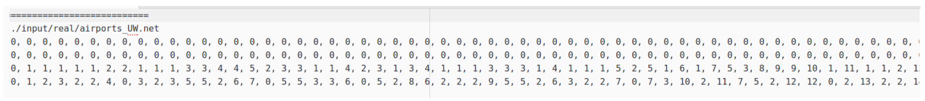


Figure 15: airports_UW, plot from all methods



```
=========================
./input/real/airports_UW.net
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 3, 3, 4, 4, 5, 2, 3, 3, 1, 1, 4, 2, 3, 1, 3, 4, 1, 1, 1, 3, 3, 3, 1, 4, 1, 1, 1, 5, 2, 5, 1, 6, 1, 7, 5, 3, 8, 9, 9, 10, 1, 11, 1, 1, 2, 1
0, 1, 2, 3, 2, 2, 4, 0, 3, 2, 3, 5, 5, 2, 6, 7, 0, 5, 5, 3, 3, 6, 0, 5, 2, 8, 6, 2, 2, 2, 9, 5, 5, 2, 6, 3, 2, 2, 7, 0, 7, 3, 10, 2, 11, 7, 5, 2, 12, 12, 0, 2, 13, 2, 2, 1
```

Figure 16: airports_UW, community detection sample values

4.3.2 cat_cortex_sim

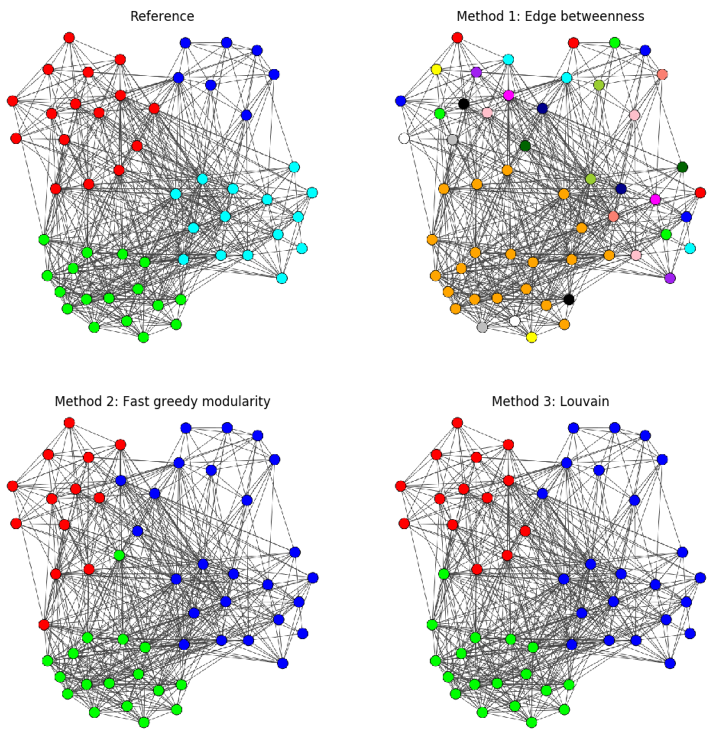
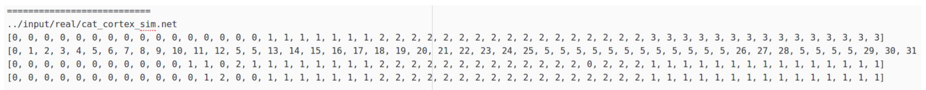


Figure 17: cat_cortex_sim, plot from all methods



```
===========================
../input/real/cat_cortex_sim.net
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 5, 5, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 26, 27, 28, 5, 5, 5, 5, 29, 30, 31
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Figure 18: cat_cortex_sim, community detection sample values

4.3.3 dolphins

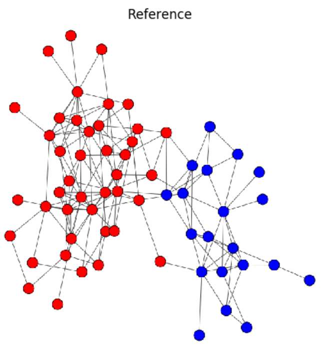
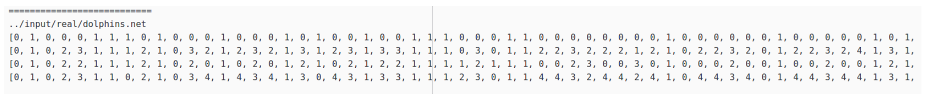


Figure 19: dolphins, plot from all methods



```
============================
../input/real/dolphins.net
[0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
[0, 1, 0, 2, 3, 1, 1, 1, 2, 1, 0, 3, 2, 1, 2, 3, 2, 1, 3, 1, 2, 3, 1, 3, 3, 1, 1, 1, 0, 3, 0, 1, 1, 2, 2, 3, 2, 2, 2, 1, 2, 1, 0, 2, 2, 3, 2, 0, 1, 2, 2, 3, 2, 4, 1, 3, 1,
[0, 1, 0, 2, 2, 1, 1, 1, 2, 1, 0, 2, 0, 1, 0, 2, 0, 1, 2, 1, 0, 2, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 0, 0, 2, 3, 0, 0, 3, 0, 1, 0, 0, 0, 2, 0, 0, 1, 0, 0, 2, 0, 0, 1, 2, 1,
[0, 1, 0, 2, 3, 1, 1, 0, 2, 1, 0, 3, 4, 1, 4, 3, 4, 1, 3, 0, 4, 3, 1, 3, 3, 1, 1, 1, 2, 3, 0, 1, 1, 4, 4, 3, 2, 4, 4, 2, 4, 1, 0, 4, 4, 3, 4, 0, 1, 4, 4, 3, 4, 4, 1, 3, 1,
```

Figure 20: dolphins, community detection sample values

4.3.4 football

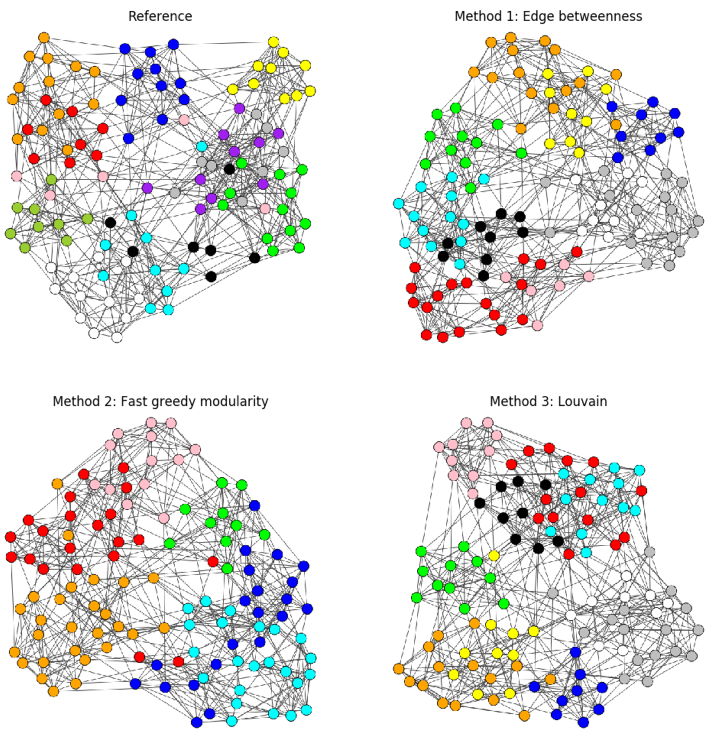
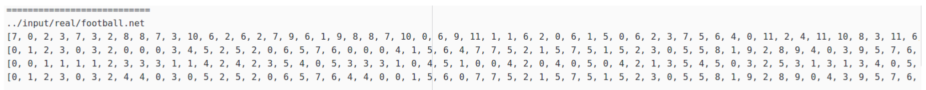


Figure 21: football, plot from all methods



```
========================
../input/real/football.net
[7, 0, 2, 3, 7, 3, 2, 8, 8, 7, 3, 10, 6, 2, 6, 2, 7, 9, 6, 1, 9, 8, 8, 7, 10, 0, 6, 9, 11, 1, 1, 6, 2, 0, 6, 1, 5, 0, 6, 2, 3, 7, 5, 6, 4, 0, 11, 2, 4, 11, 10, 8, 3, 11, 6
[0, 1, 2, 3, 0, 3, 2, 0, 0, 0, 3, 4, 5, 2, 5, 2, 0, 6, 5, 7, 6, 0, 0, 0, 4, 1, 5, 6, 4, 7, 7, 5, 2, 1, 5, 7, 5, 1, 5, 2, 3, 0, 5, 5, 8, 1, 9, 2, 8, 9, 4, 0, 3, 9, 5, 7, 6,
[0, 0, 1, 1, 1, 1, 2, 3, 3, 3, 1, 1, 4, 2, 4, 2, 3, 5, 4, 0, 5, 3, 3, 3, 1, 0, 4, 5, 1, 0, 0, 4, 2, 0, 4, 0, 5, 0, 4, 2, 1, 3, 5, 4, 5, 0, 3, 2, 5, 3, 1, 3, 1, 3, 4, 0, 5,
[0, 1, 2, 3, 0, 3, 2, 4, 4, 0, 3, 0, 5, 2, 5, 2, 0, 6, 5, 7, 6, 4, 4, 0, 0, 1, 5, 6, 0, 7, 7, 5, 2, 1, 5, 7, 5, 1, 5, 2, 3, 0, 5, 5, 8, 1, 9, 2, 8, 9, 0, 4, 3, 9, 5, 7, 6,
```

Figure 22: football, community detection sample values

4.3.5 zachary_unwh

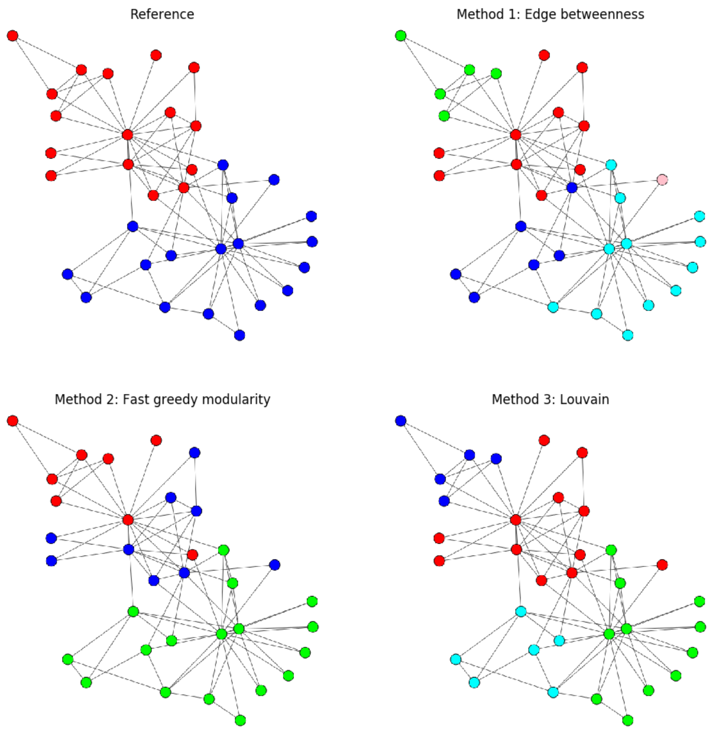
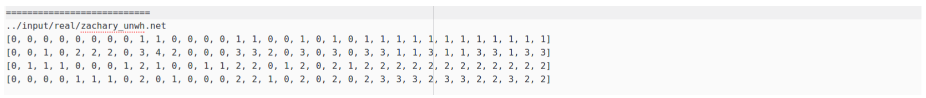


Figure 23: zachary_unwh, plot from all methods



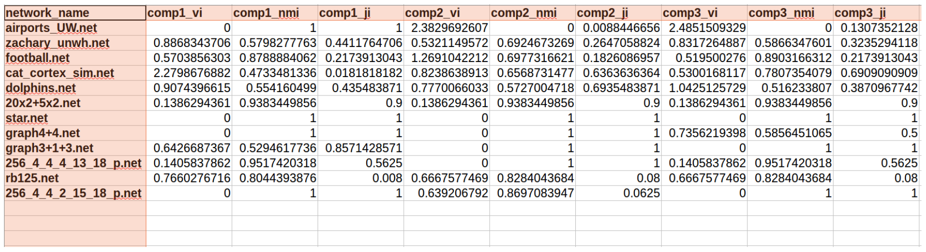
```
===========================
../input/real/zachary_unwh.net
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 1, 0, 2, 2, 2, 0, 3, 4, 2, 0, 0, 0, 3, 3, 2, 0, 3, 0, 3, 0, 3, 3, 1, 1, 3, 1, 1, 3, 3, 1, 3, 3]
[0, 1, 1, 1, 0, 0, 0, 1, 2, 1, 0, 0, 1, 1, 2, 2, 0, 1, 2, 0, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 0, 0, 0, 1, 1, 1, 0, 2, 0, 1, 0, 0, 0, 2, 2, 1, 0, 2, 0, 2, 0, 2, 3, 3, 3, 2, 3, 3, 2, 2, 3, 2, 2]
```

Figure 24: zachary_unwh, community detection sample values

5 Comparison tables

5.1 Measures comparison

In these part we can compare our partitions with the references, using the following standard measures: Jaccard Index (ji), Normalized Mutual Information (nmi), and Variation of Information (vi).

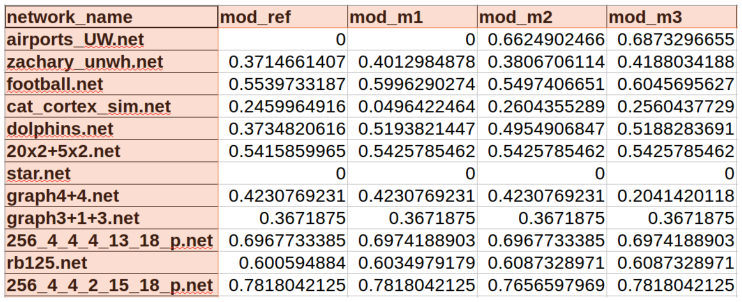


network_name	comp1_vi	comp1_nmi	comp1_ji	comp2_vi	comp2_nmi	comp2_ji	comp3_vi	comp3_nmi	comp3_ji
airports_UW.net	0	1	1	2.3829692607	0	0.0088446656	2.4851509329	0	0.1307352128
zachary_unwh.net	0.8868343706	0.5798277763	0.4411764706	0.5321149572	0.6924673269	0.2647058824	0.8317264887	0.5866347601	0.3235294118
football.net	0.5703856303	0.8788884062	0.2173913043	1.2691042212	0.6977316621	0.1826086957	0.519500276	0.8903166312	0.2173913043
cat_cortex_sim.net	2.2798676882	0.4733481336	0.0181818182	0.8238638913	0.6568731477	0.6363636364	0.5300168117	0.7807354079	0.6909090909
dolphins.net	0.9074396615	0.554160499	0.435483871	0.7770066033	0.5727004718	0.6935483871	1.0425125729	0.516233807	0.3870967742
20x2+5x2.net	0.1386294361	0.9383449856	0.9	0.1386294361	0.9383449856	0.9	0.1386294361	0.9383449856	0.9
star.net	0	1	1	0	1	1	0	1	1
graph4+4.net	0	1	1	0	1	1	0.7356219398	0.5856451065	0.5
graph3+1+3.net	0.6426687367	0.5294617736	0.8571428571	0	1	1	0	1	1
256_4_4_4_13_18_p.net	0.1405837862	0.9517420318	0.5625	0	1	1	0.1405837862	0.9517420318	0.5625
rb125.net	0.7660276716	0.8044393876	0.008	0.6667577469	0.8284043684	0.08	0.6667577469	0.8284043684	0.08
256_4_4_2_15_18_p.net	0	1	1	0.639206792	0.8697083947	0.0625	0	1	1

Figure 25: Method 1 is Edge betweenness, method 2 is Greedy optimization of modularity and method 3 is Louvain

5.2 Modularity comparison

Here we show a table with the modularity values of all the partitions (including the reference ones), grouped by network.



network_name	mod_ref	mod_m1	mod_m2	mod_m3
airports_UW.net	0	0	0.6624902466	0.6873296655
zachary_unwh.net	0.3714661407	0.4012984878	0.3806706114	0.4188034188
football.net	0.5539733187	0.5996290274	0.5497406651	0.6045695627
cat_cortex_sim.net	0.2459964916	0.0496422464	0.2604355289	0.2560437729
dolphins.net	0.3734820616	0.5193821447	0.4954906847	0.5188283691
20x2+5x2.net	0.5415859965	0.5425785462	0.5425785462	0.5425785462
star.net	0	0	0	0
graph4+4.net	0.4230769231	0.4230769231	0.4230769231	0.2041420118
graph3+1+3.net	0.3671875	0.3671875	0.3671875	0.3671875
256_4_4_4_13_18_p.net	0.6967733385	0.6974188903	0.6967733385	0.6974188903
rb125.net	0.600594884	0.6034979179	0.6087328971	0.6087328971
256_4_4_2_15_18_p.net	0.7818042125	0.7818042125	0.7656597969	0.7818042125

Figure 26: Method 1 is Edge betweenness, method 2 is Greedy optimization of modularity and method 3 is Louvain

6 Conclusion

After trying the different community detection methods, maybe the most consistent in good results for almost every network is the Louvain method, and besides, it worked with very good performance in all networks, taking only a few seconds even for the real ones.

The greedy optimization of modularity also got good results, almost as good as the ones obtained with the Louvain method.

On the other hand, the Edge betweenness seemed quite inconsistent with results, and even more, it usually spend more time than other methods, and it even got stuck for the network airports_UW where we could not get the final membership detection.

References

[1] igraph – the network analysis package. `http://igraph.org/redirect.html`. [Online; accessed 25-May-2017].

[2] Networkx, high-productivity software for complex networks. `https://networkx.github.io/`. [Online; accessed 25-May-2017].

[3] Community finding based on edge betweenness. `http://igraph.org/c/doc/igraph-Community.html#idm470942730032`. [Online; accessed 25-May-2017].

[4] Finding community structure by greedy optimization of modularity. `http://igraph.org/c/doc/igraph-Community.html#idm470942725872`. [Online; accessed 25-May-2017].

[5] Louvain method: Finding communities in large networks. `https://sites.google.com/site/findcommunities/`. [Online; accessed 25-May-2017].