Activity A3 : Community Detection

Hajar Lachheb

1 Introduction

Through this paper, I am presenting the individual work I did. It was about the concept of Community Detection. In fact, finding clusters of nodes with strong connectedness and similarity inside a network or graph is a method known as community detection. In a number of disciplines, including social network analysis, biological network analysis, and recommendation systems, community detection is essential. For instance, using community discovery in social network analysis, it is possible to locate groups of individuals who share a common affiliation, set of values, or set of interests. This data can be used to create targeted advertising or to locate influential people inside a network. Overall, this work was quite challenging as well, but it helped me learn more about the importance of community detection as well as it importance to analyse and understand complex networks.

2 Work Description

2.1 Description of the algorithms used

In this task, I was very ambitious and I thought it will be the opportunity to learn more about the different algorithms that would help me to do community detection as best as possible. Some of the algorithms are used in igraph and will be as well helping me to plot the different partitions. An addition was to use algorithms using networkx to have a comparison table just as we did use igraph algorithms. The overall objective behind what I did is just to compare the overall performance of the algorithms and get to the conclusion "What's the best algorithm?"

- Fast Greedy : The communityfastgreedy algorithm bases its modularity optimization on a greedy strategy. In order to generate clusters that maximize the modularity score, nodes or groups of nodes are gradually merged. Although this method is quick, it might not always result in the modularity function's global maximum.

- Optimal Modularity : The communityoptimalmodularity algorithm is another that has been tested. To identify the partition that has the maximum modularity score, this algorithm does a thorough search. However, this computation may take a while for big networks.

- Label Propagation : The label propagation algorithm created by Raghavan, Albert, and Kumara serves as the foundation for the communitylabelpropagation algorithm. By using a majority vote, this technique first assigns a node an initial label before propagating that label to any nearby nodes. Up until there are no more label modifications, this process is repeated. Depending on the initial labeling, the algorithm's nondeterministic output may differ.

- Leading Eigenvector : Using the eigenvectors of the modularity matrix, the community-leading eigenvector method maximizes modularity. The pairwise connections between nodes in a network are encoded in the modularity matrix, which is a matrix. This matrix's eigenvectors can be used to determine which nodes are most crucial for maximizing modularity. Although this approach might take longer than the communityfastgreedy algorithm, it might yield superior outcomes.

- Louvain : A community recognition approach called the Louvain algorithm maximizes modularity, a metric for the effectiveness of dividing a network into communities. It accomplishes this by repeatedly merging communities that raise the modularity rating until no more advancements are possible. The technique is a preferred option for extensive network research because of its speed and scalability.

For the added Networkx algorithms, we can explain them as follows :

- Greedy Modularity : The communityfastgreedy algorithm in igraph is comparable to the greedymodularitycommunities algorithm.

- Label Propagation : The communitylabelpropagation algorithm in igraph is comparable to the labelpropagationcommunities algorithm.

- Louvain : The louvain algorithm in igraph is comparable to the louvainnx algorithm.

- Kclique : The kcliquecommunities approach was also tested, and we had to change k in order to get good comparison results. In fact, the algorithm searches for densely linked subgraphs of a network known as k-cliques, where k is a parameter that establishes the minimum size of the subgraph. All k-cliques in the network are first identified by the algorithm, which then combines them to create larger communities. The resulting communities' sizes are not specified, and they may overlap. The network's properties and the required level of granularity in the community structure must be taken into consideration while selecting the k value.

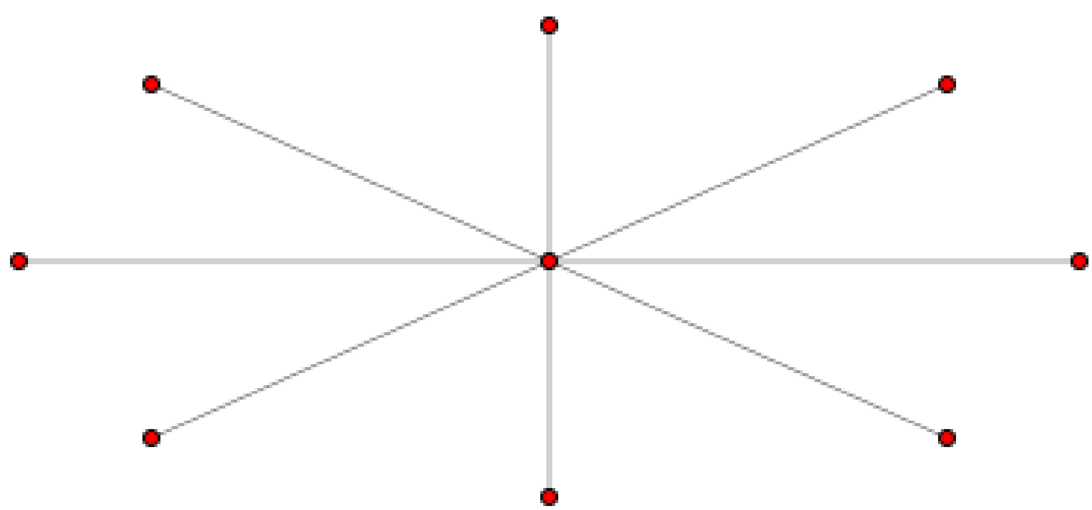
2.2 Representation of the plots with color-coded communities

By giving each community a distinct hue, color-coded communities can help in the visualization of the outcomes of community detection algorithms. This makes it simpler to recognize the many communities and comprehend how they interact inside a network. The purpose of color-coded communities is to give us an intuitive visual representation of the clustering results. It might be challenging to distinguish between different cultures and comprehend their makeup without the use of color-coding. Finally, color-coded communities can also be used to compare and assess the efficacy of various community detection techniques. It is feasible to compare the color-coded communities and see which algorithms are generating results that are similar or dissimilar, as well as any patterns or anomalies in the data.

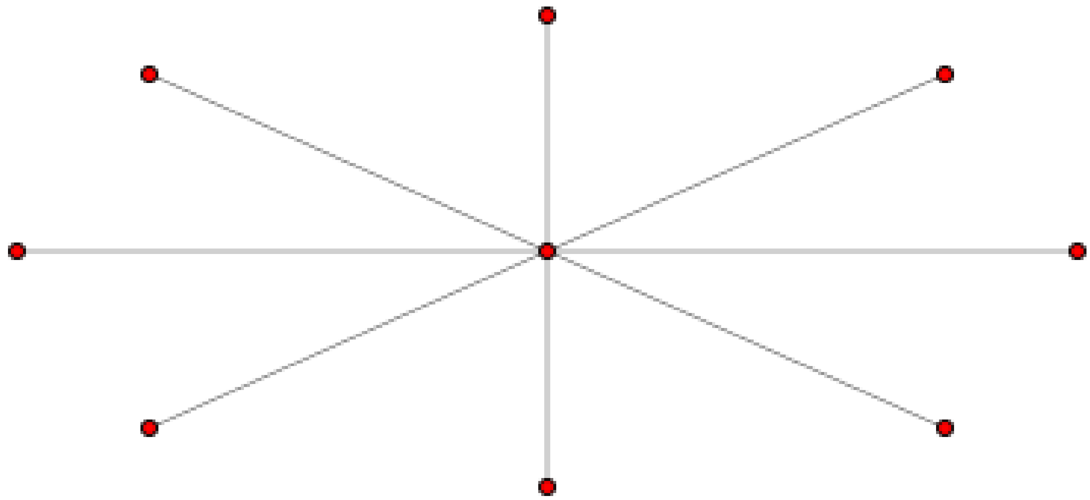
The representation in this part will be as follows: We represent each network separately and for each network, we plot the three different community detection representations following the respective algorithm.

Plots of the network: Star

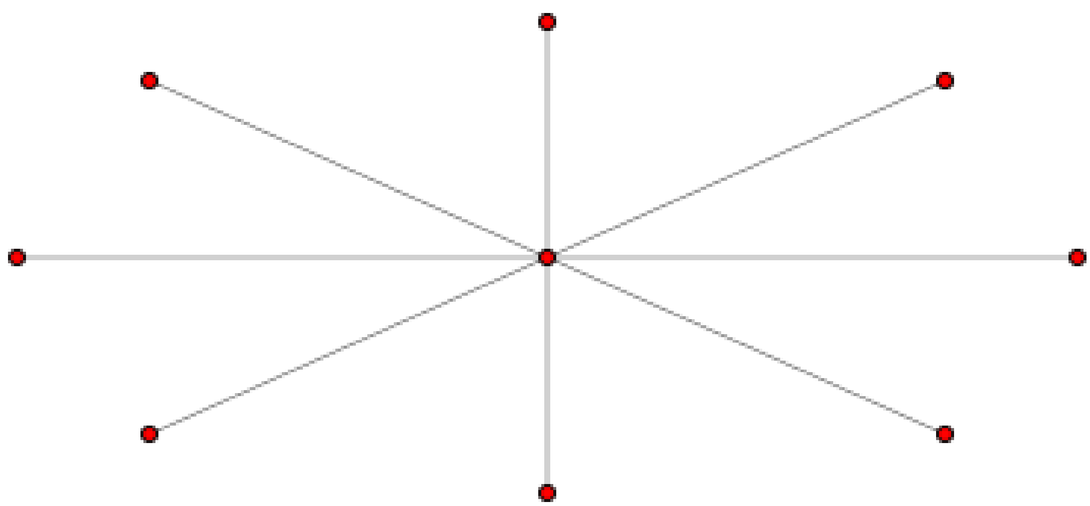
Star network plot using Label Propagation Algorithm



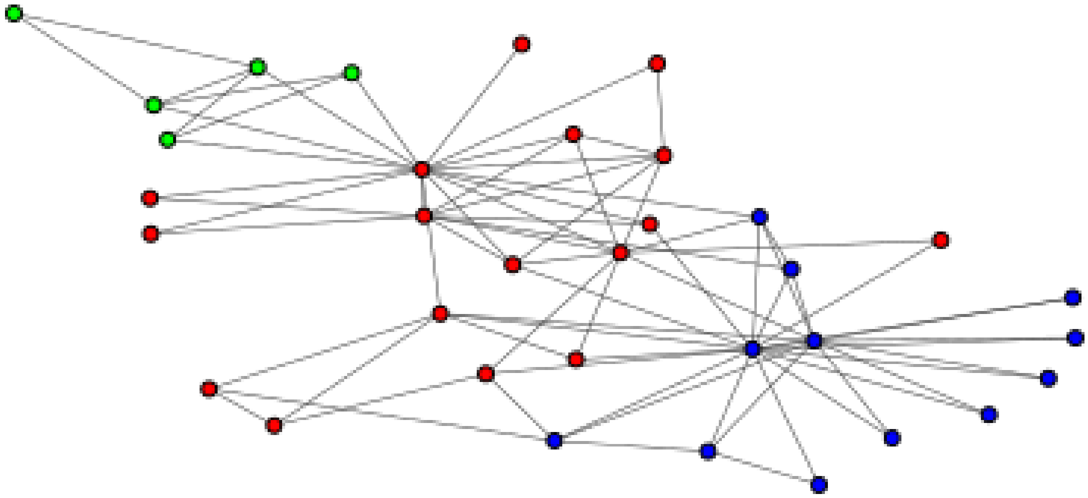
Star network plot using Louvain Algorithm



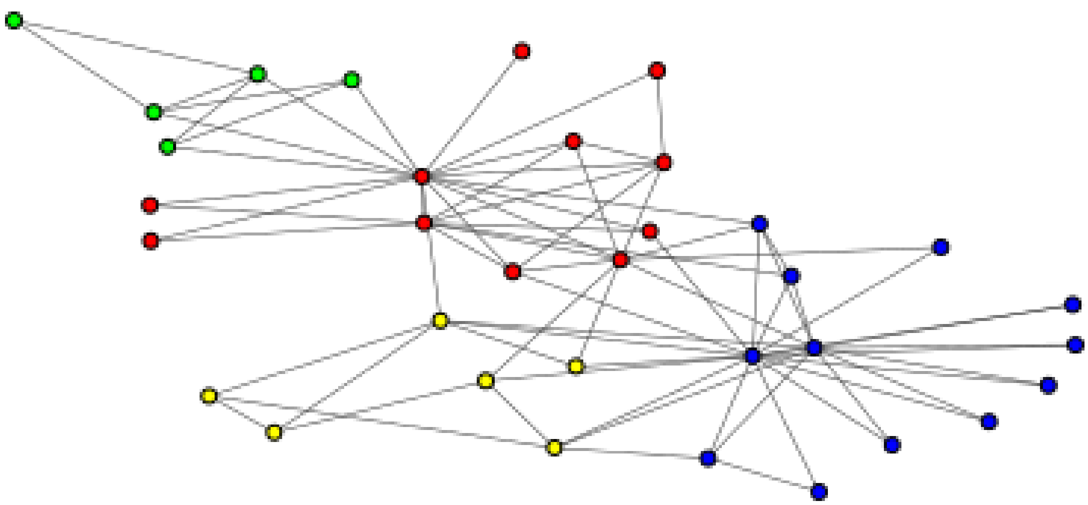
Star network plot using Fast Greedy Algorithm



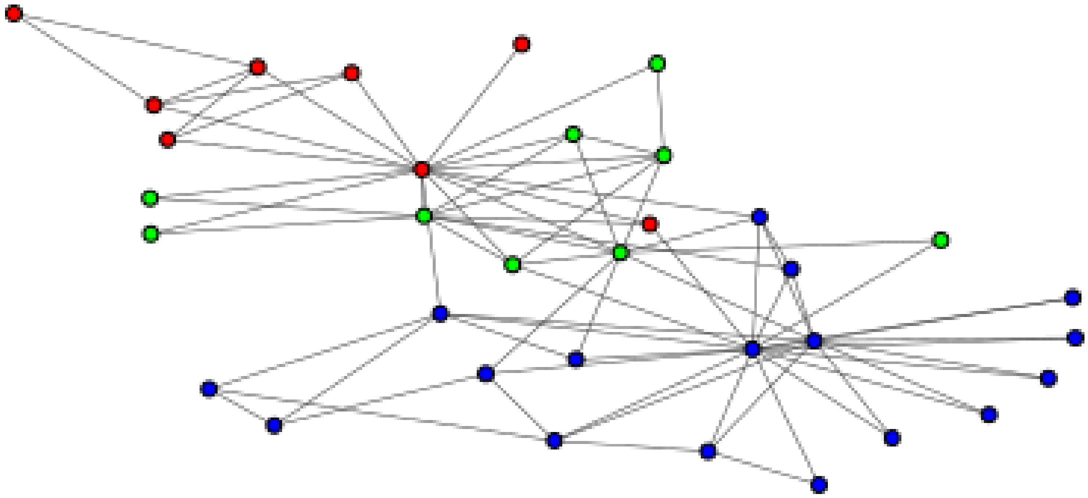
Plots of the network: Zacharyunwh
Zacharyunwh network plot using Label Propagation Algorithm



Zacharyunwh network plot using Louvain Algorithm

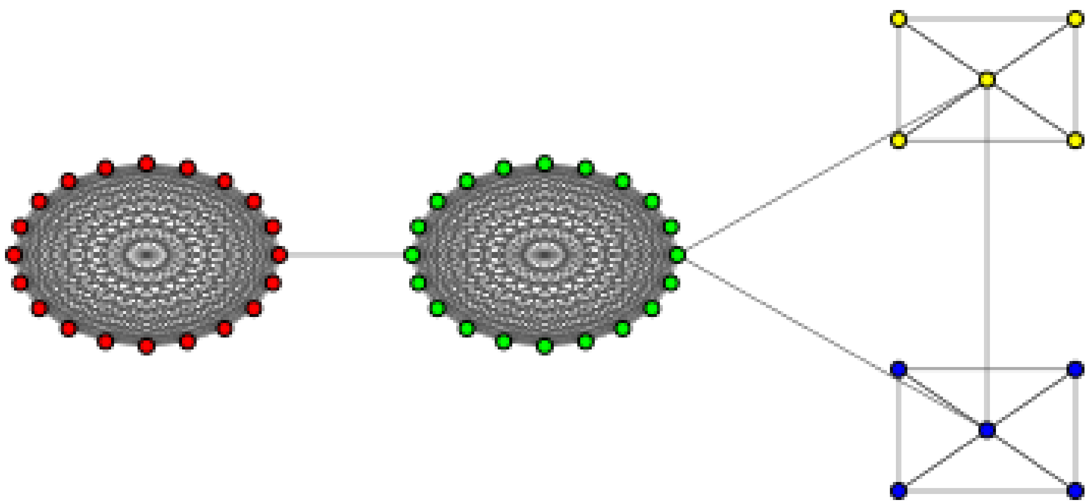


Zacharyunwh network plot using Fast Greedy Algorithm

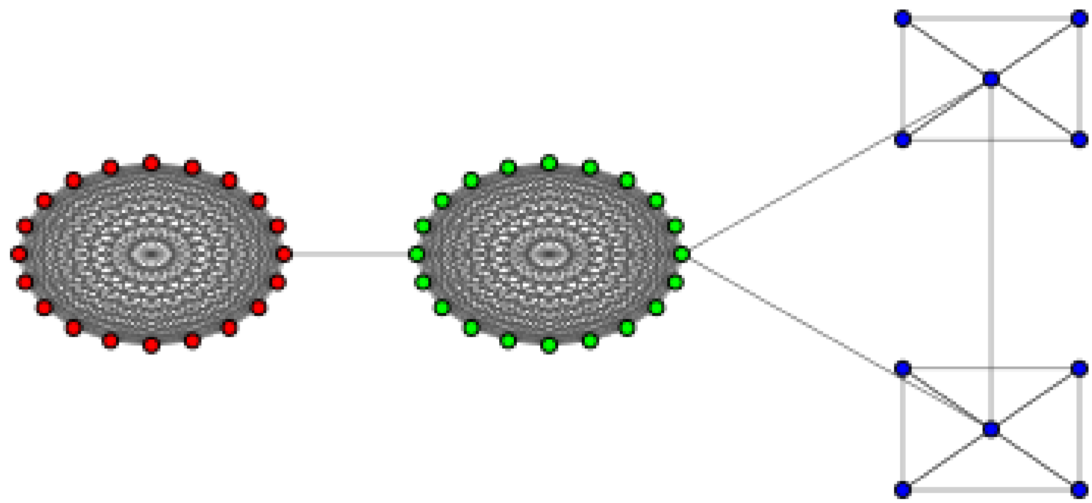


Plots of the network: 20*2+5*2

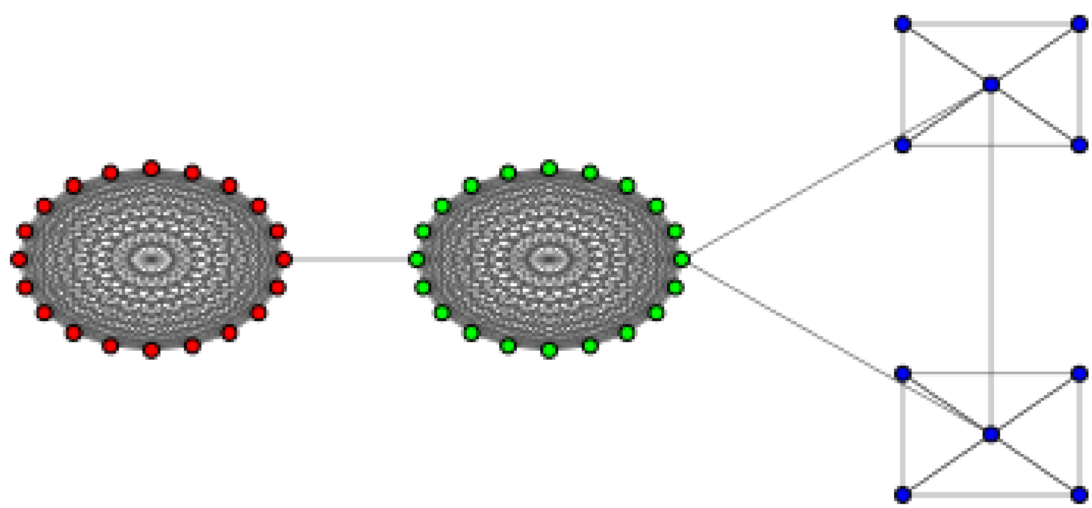
20*2+5*2 network plot using Label Propagation Algorithm



20*2+5*2 network plot using Louvain Algorithm

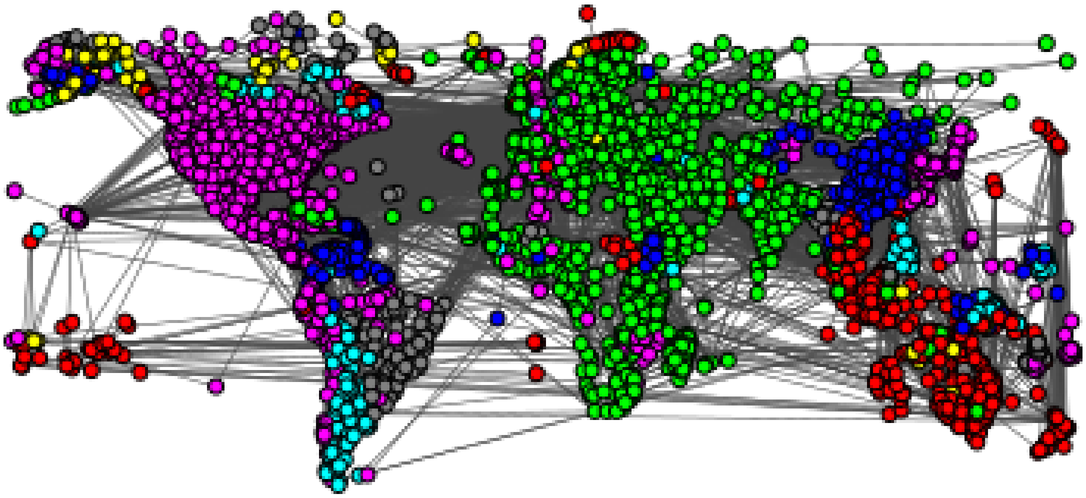


20*2+5*2 network plot using Fast Greedy Algorithm

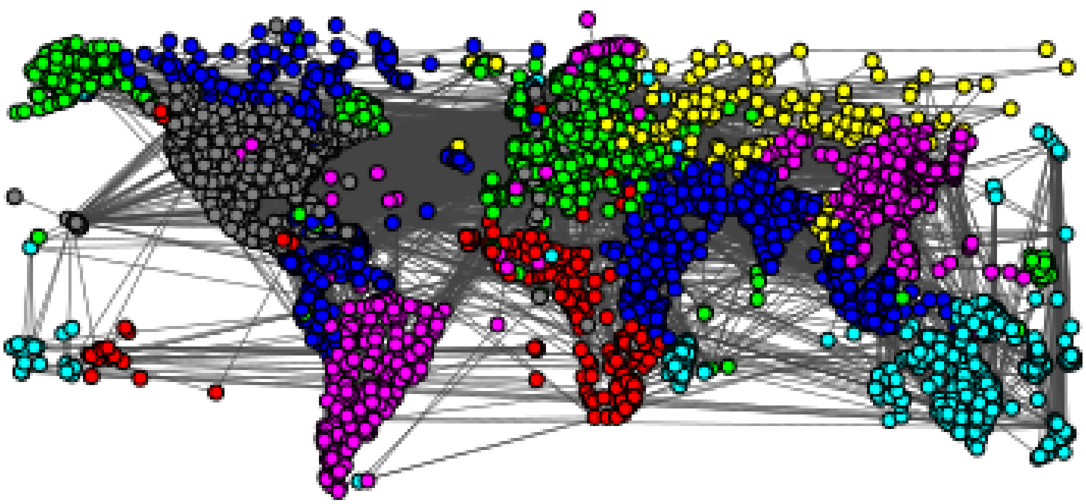


Plots of the network: AirportsUW

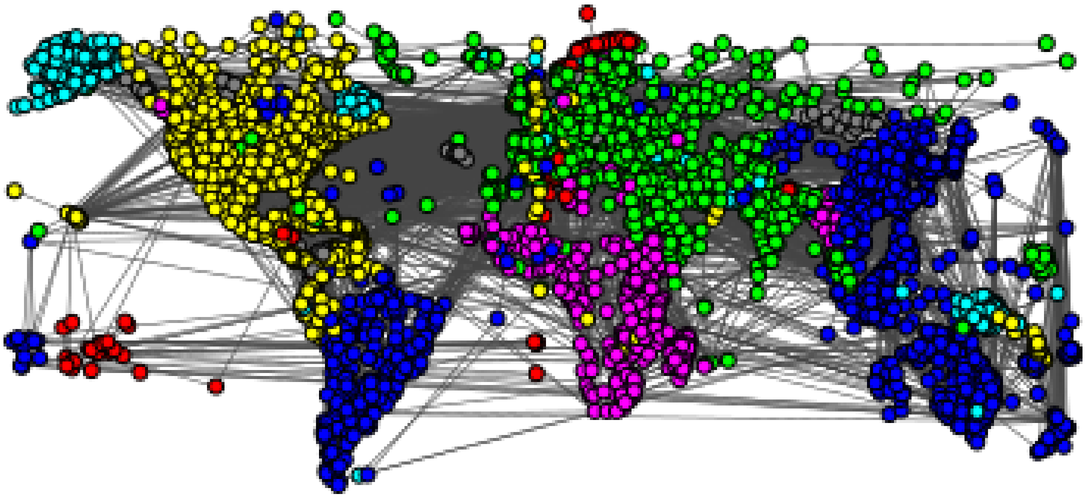
AirportsUW network plot using Label Propagation Algorithm



AirportsUW network plot using Louvain Algorithm

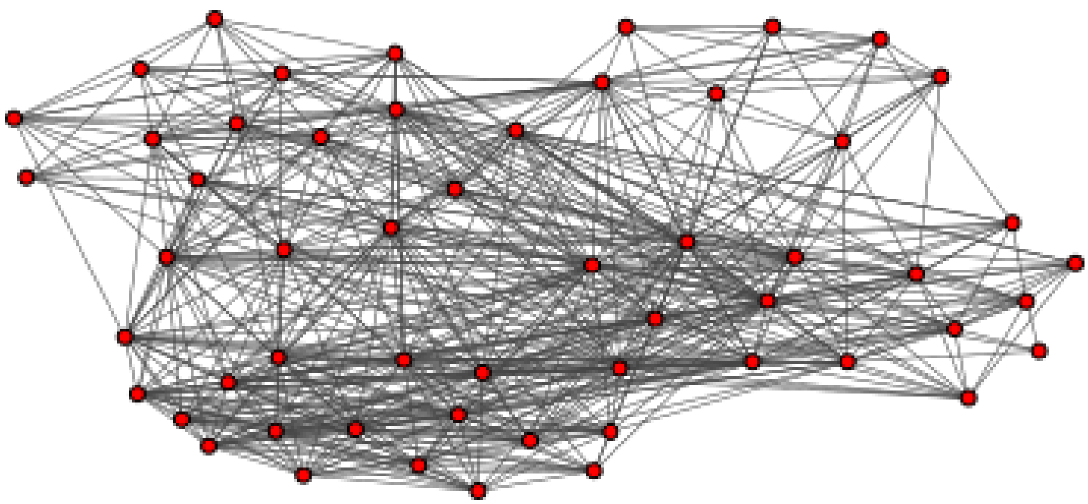


AirportsUW network plot using Fast Greedy Algorithm

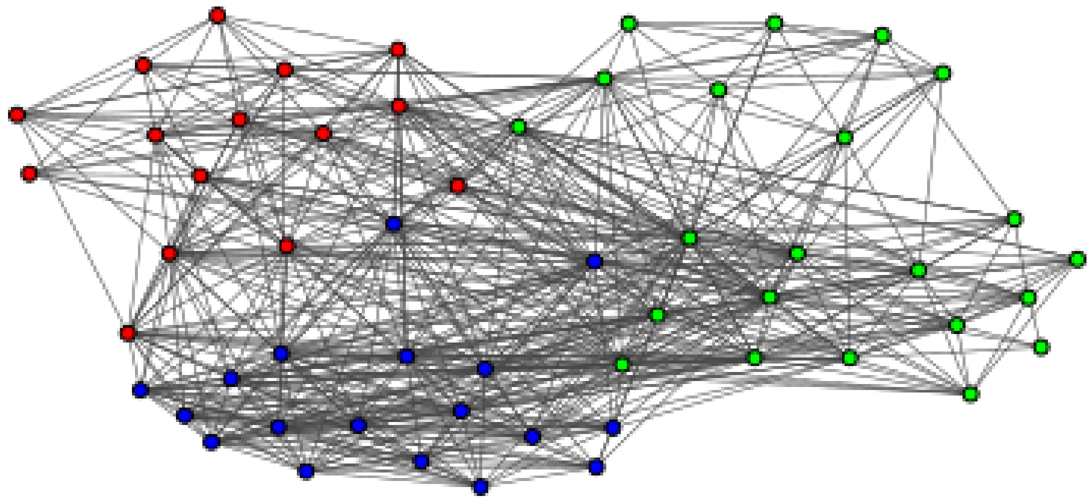


Plots of the network: CatCortexSim

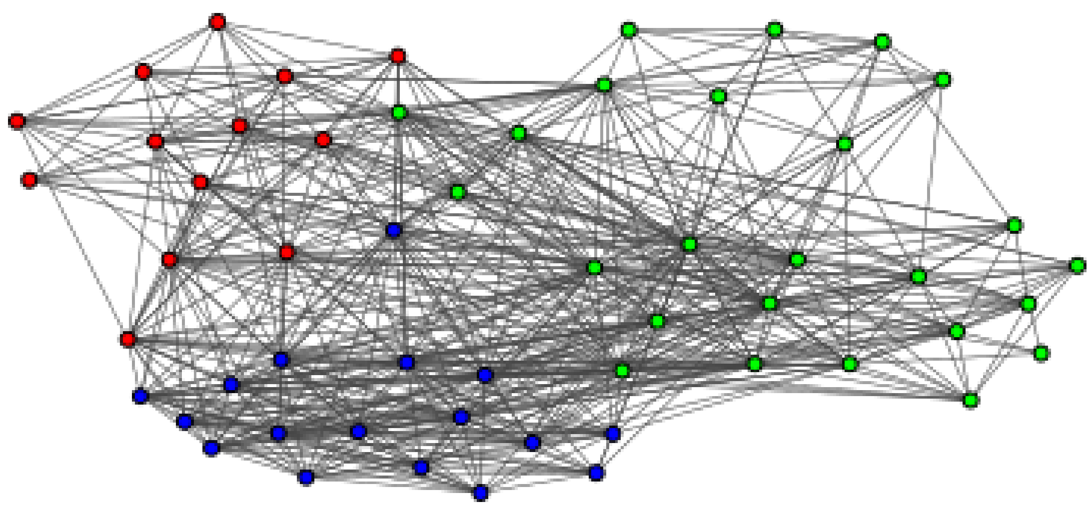
CatCortexSim network plot using Label Propagation Algorithm



CatCortexSim network plot using Louvain Algorithm

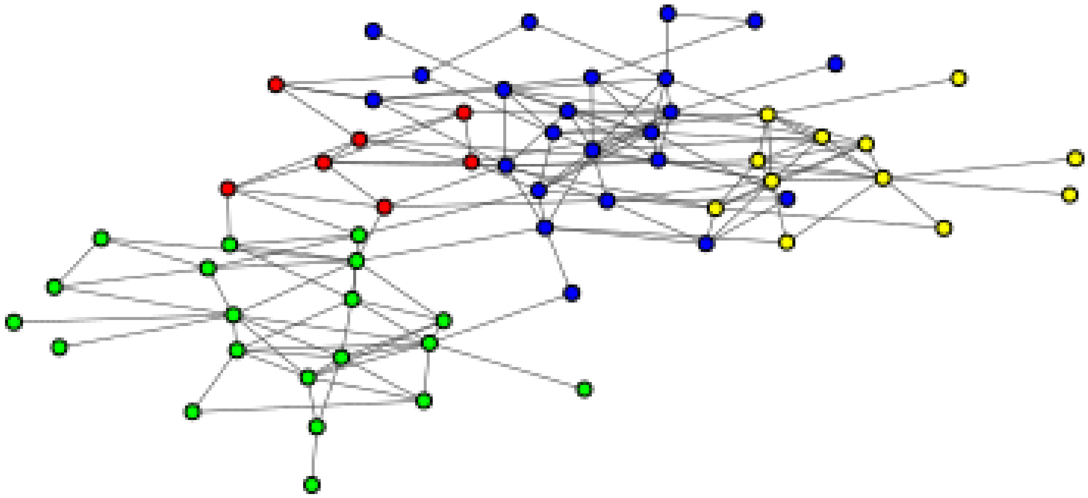


CatCortexSim network plot using Fast Greedy Algorithm

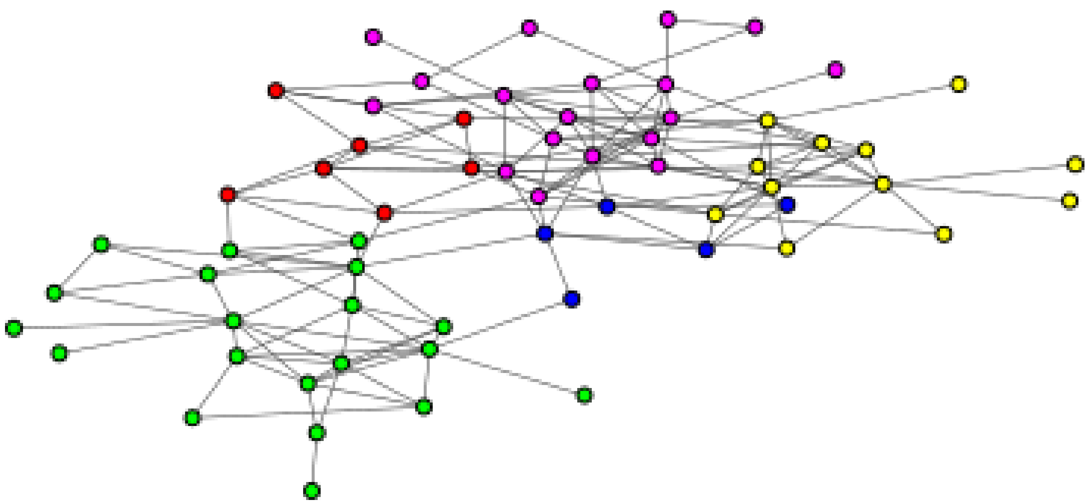


Plots of the network: Dolphins

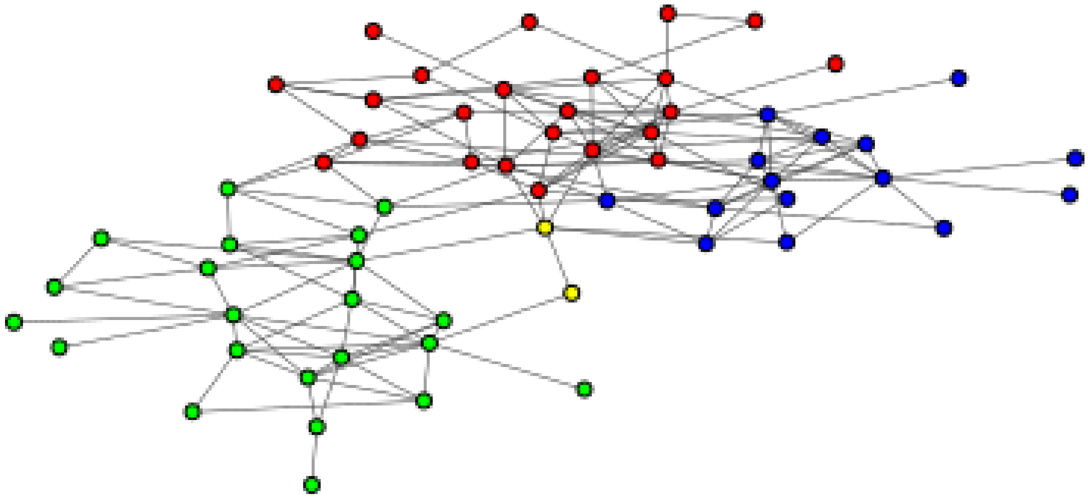
Dolphins network plot using Label Propagation Algorithm



Dolphins network plot using Louvain Algorithm

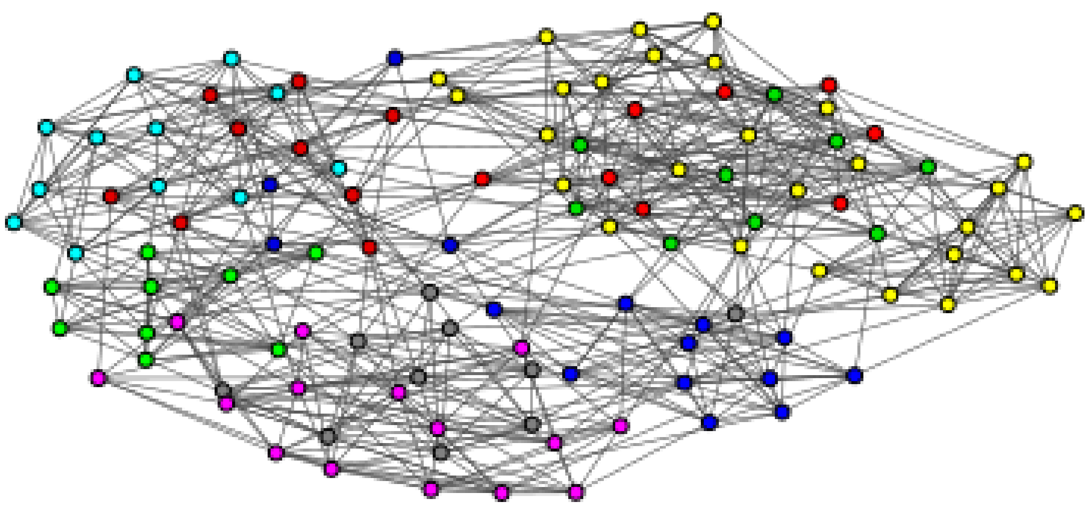


Dolphins network plot using Fast Greedy Algorithm

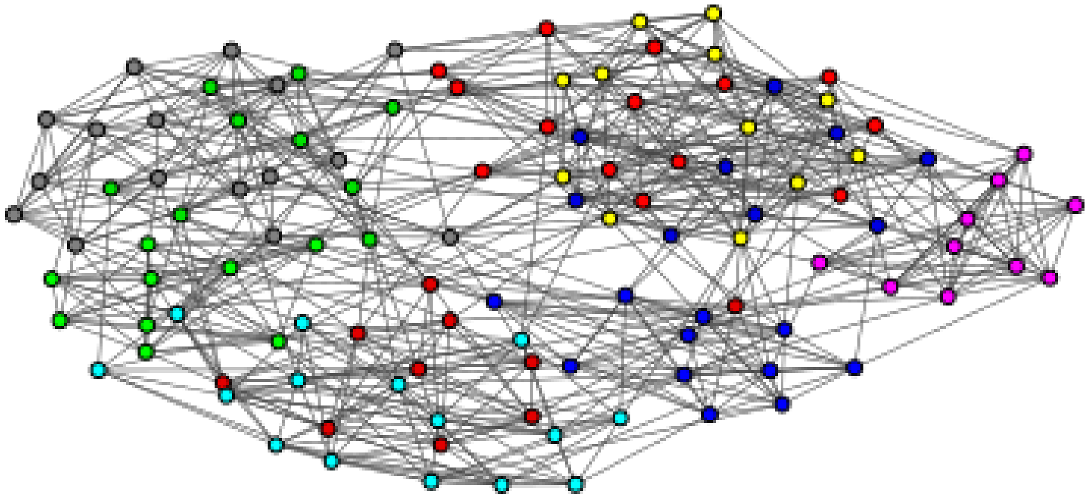


Plots of the network: Football

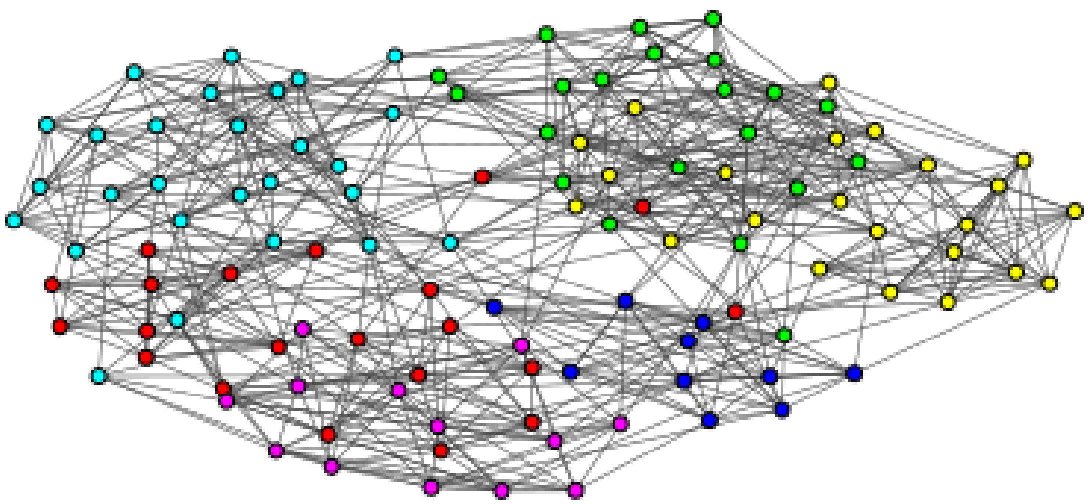
Football network plot using Label Propagation Algorithm



Football network plot using Louvain Algorithm

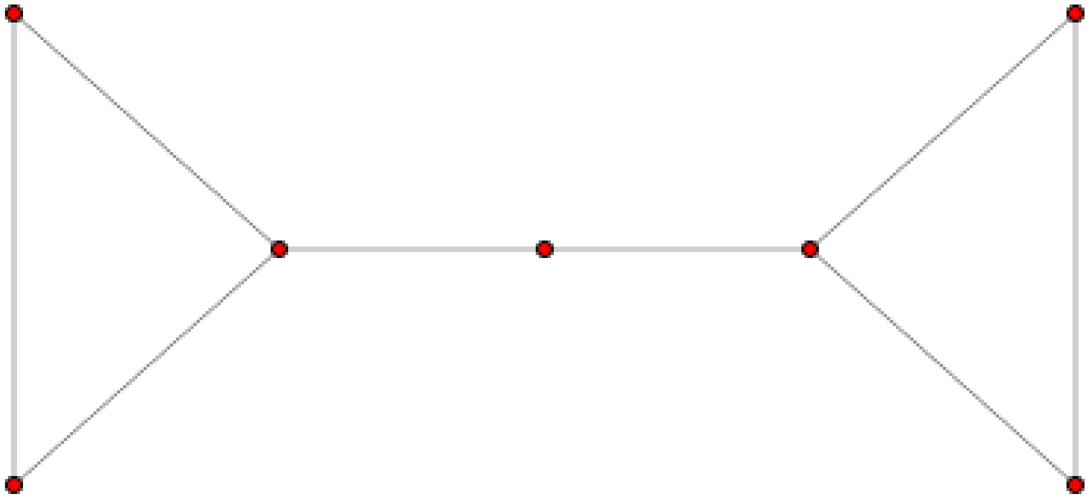


Football network plot using Fast Greedy Algorithm

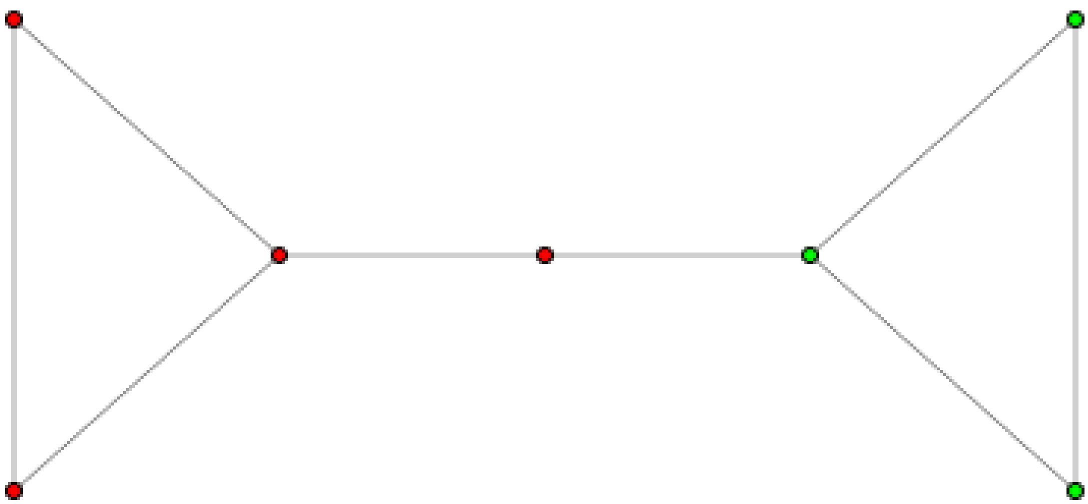


Plots of the network: Graph3+1+3

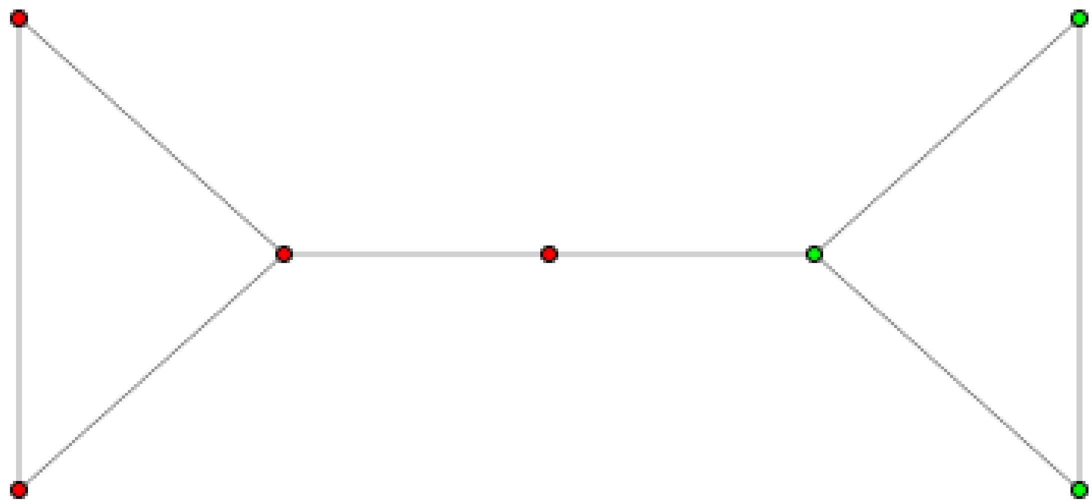
Graph3+1+3 network plot using Label Propagation Algorithm



Graph3+1+3 network plot using Louvain Algorithm

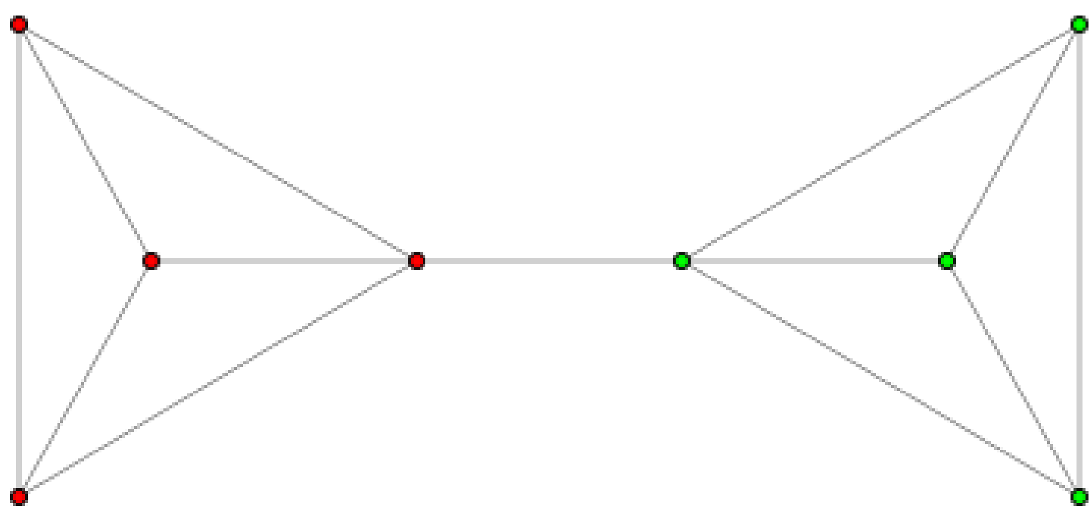


Graph3+1+3 network plot using Fast Greedy Algorithm

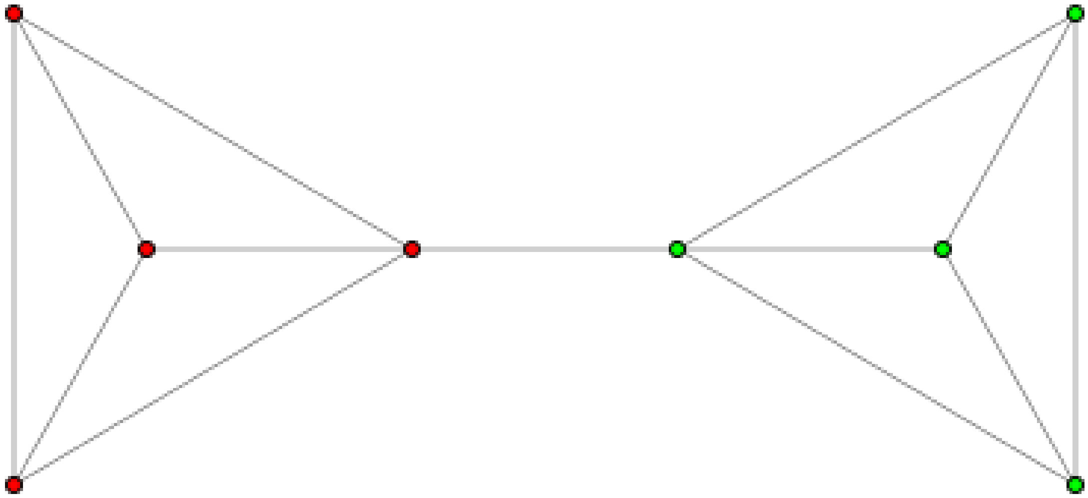


Plots of the network: Graph4+4

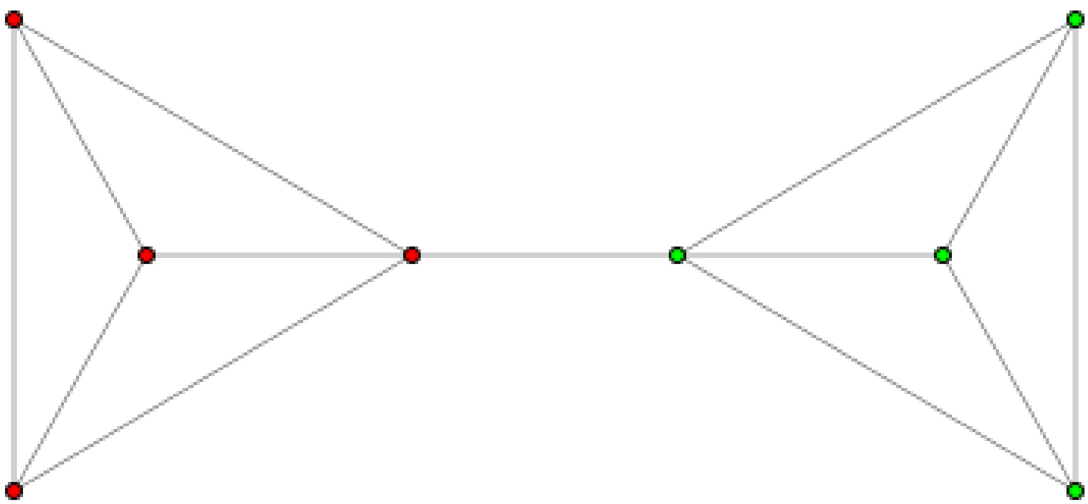
Graph4+4 network plot using Label Propagation Algorithm



Graph4+4 network plot using Louvain Algorithm

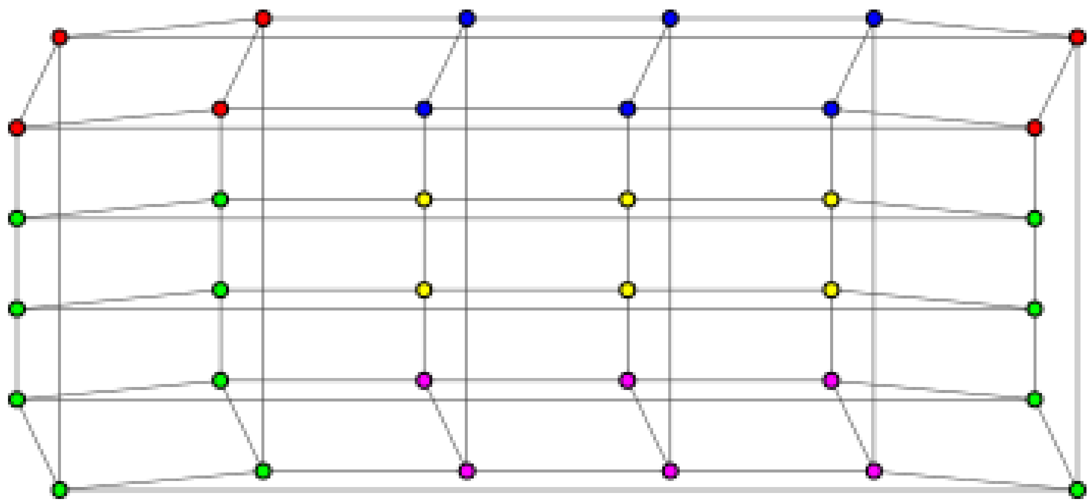


Graph4+4 network plot using Fast Greedy Algorithm

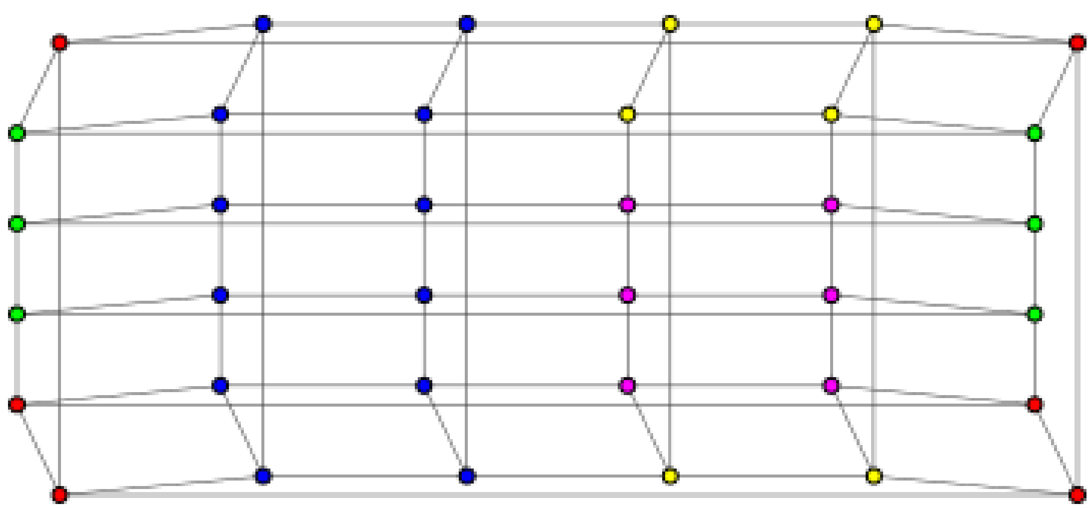


Plots of the network: Gridp 6*6

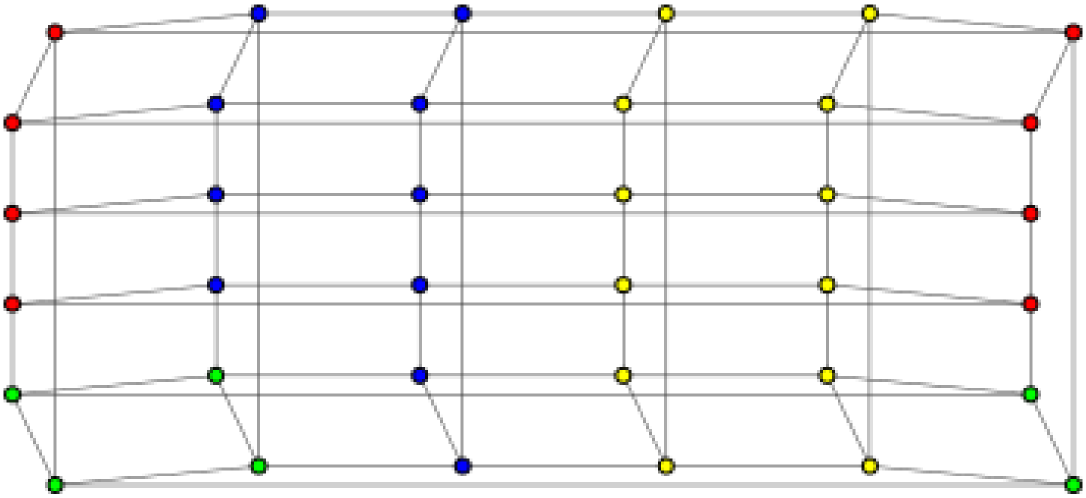
Gridp 6*6 network plot using Label Propagation Algorithm



Gridp 6*6 network plot using Louvain Algorithm

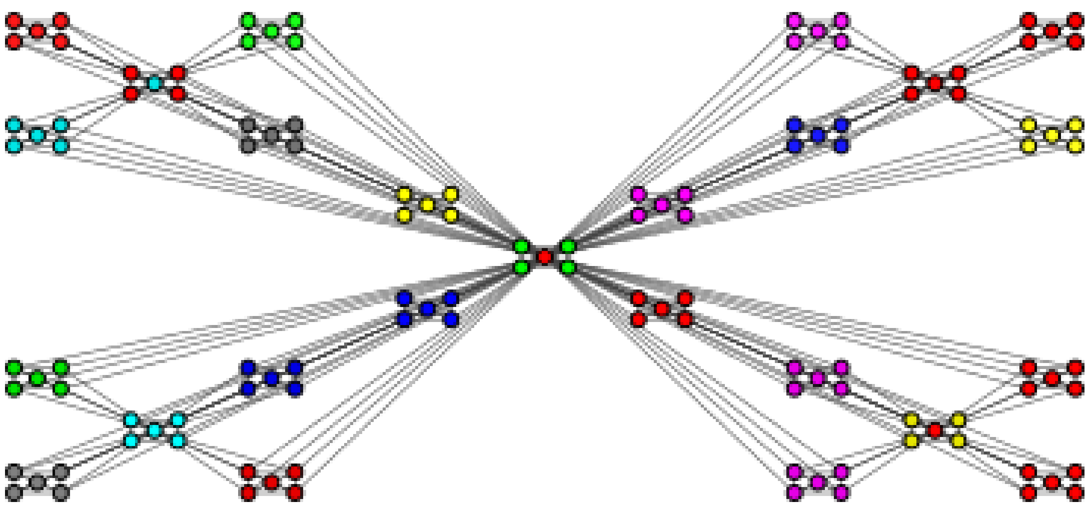


Gridp 6*6 network plot using Fast Greedy Algorithm

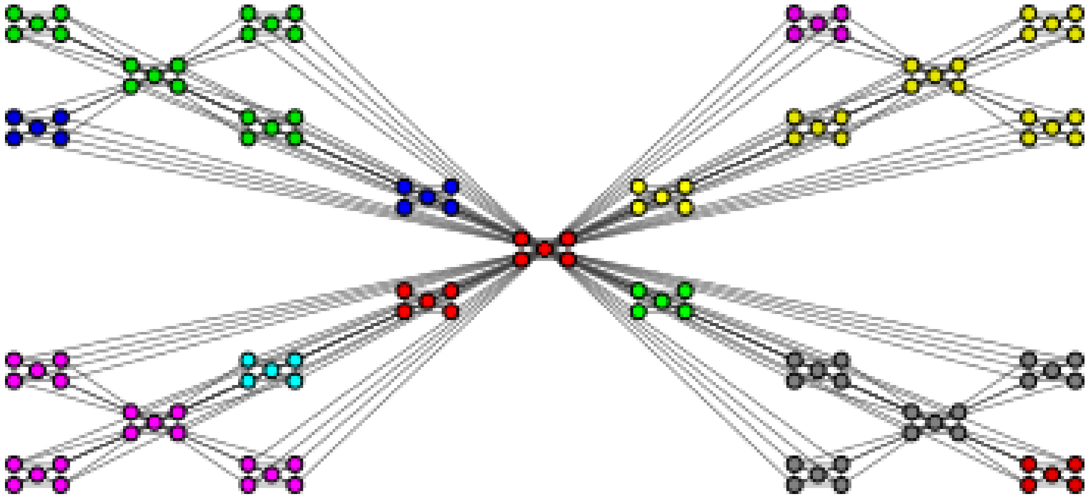


Plots of the network: Rb125

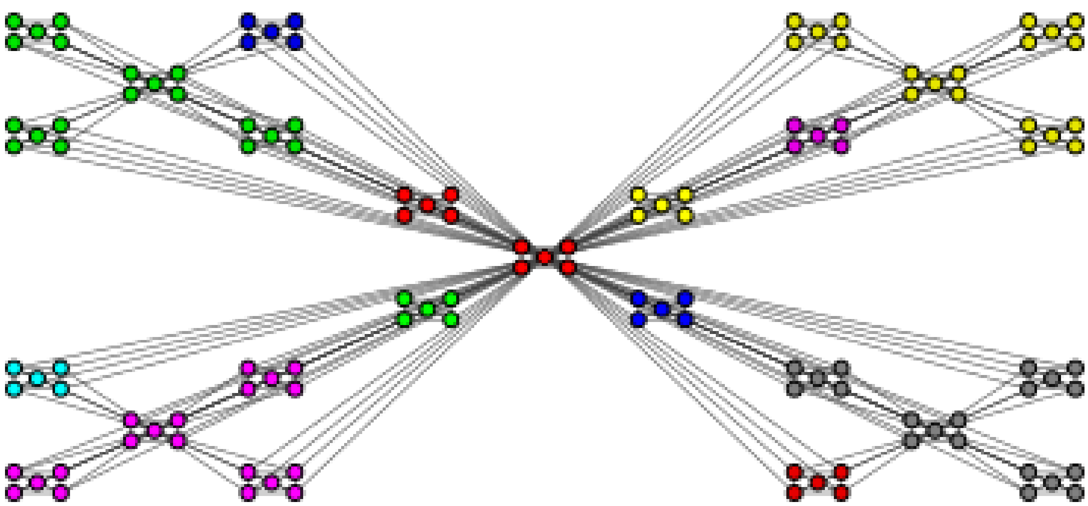
Rb125 network plot using Label Propagation Algorithm



Rb125 network plot using Louvain Algorithm

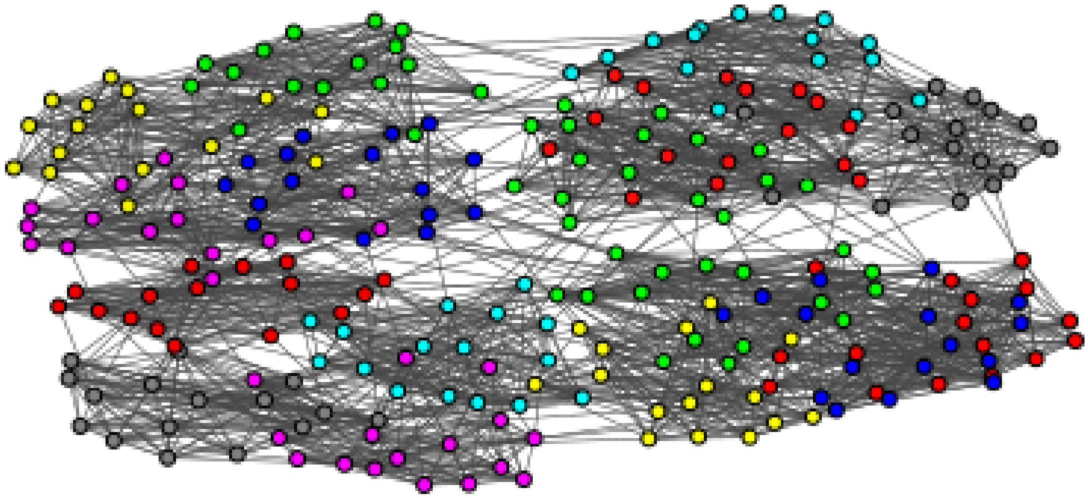


Rb125 network plot using Fast Greedy Algorithm

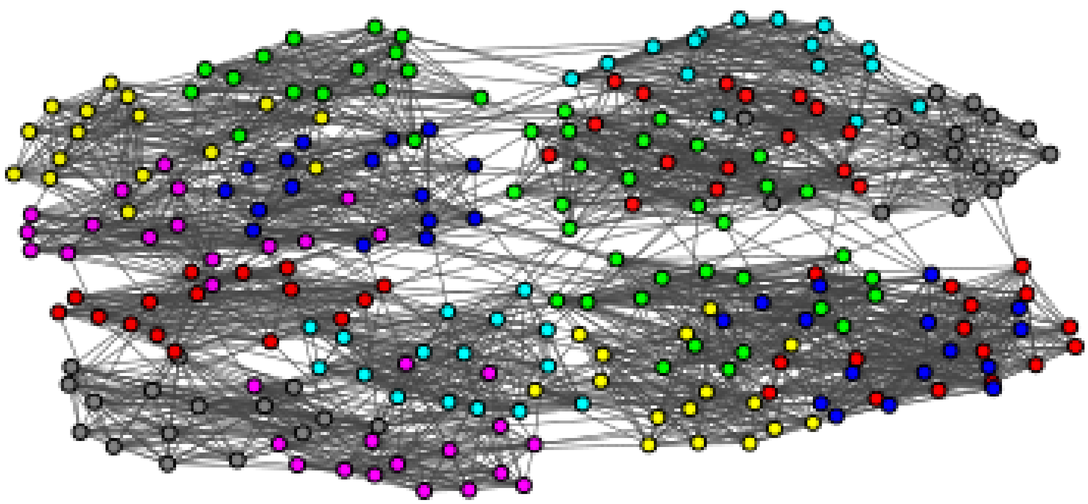


Plots of the network: 256 4421518 p

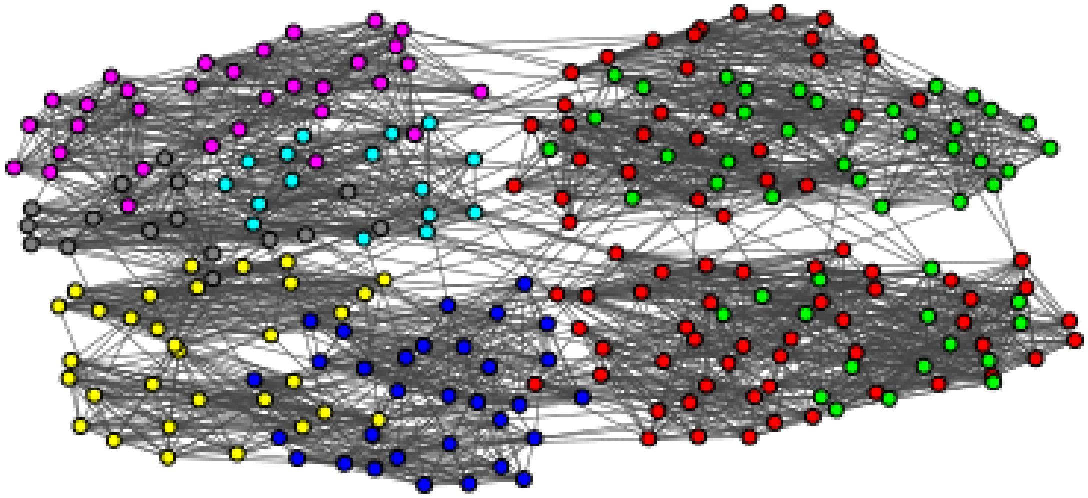
256 4421518 p network plot using Label Propagation Algorithm



256 4421518 p network plot using Louvain Algorithm

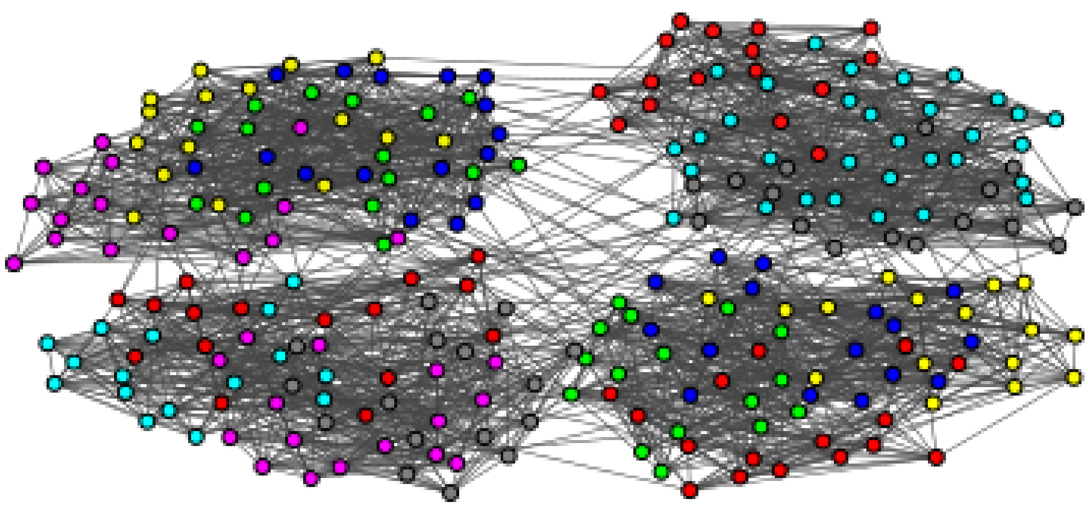


256 4421518 p network plot using Fast Greedy Algorithm

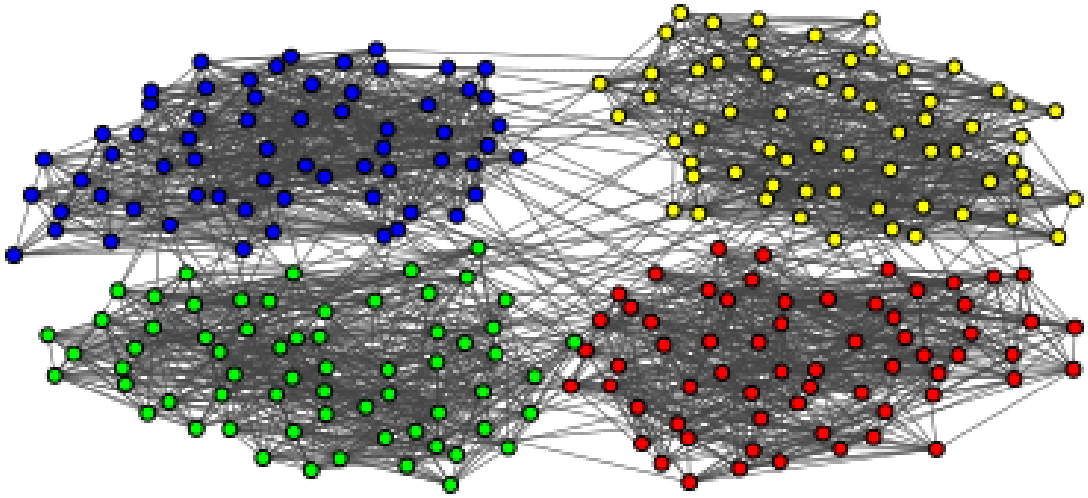


Plots of the network: 256 4441318 p

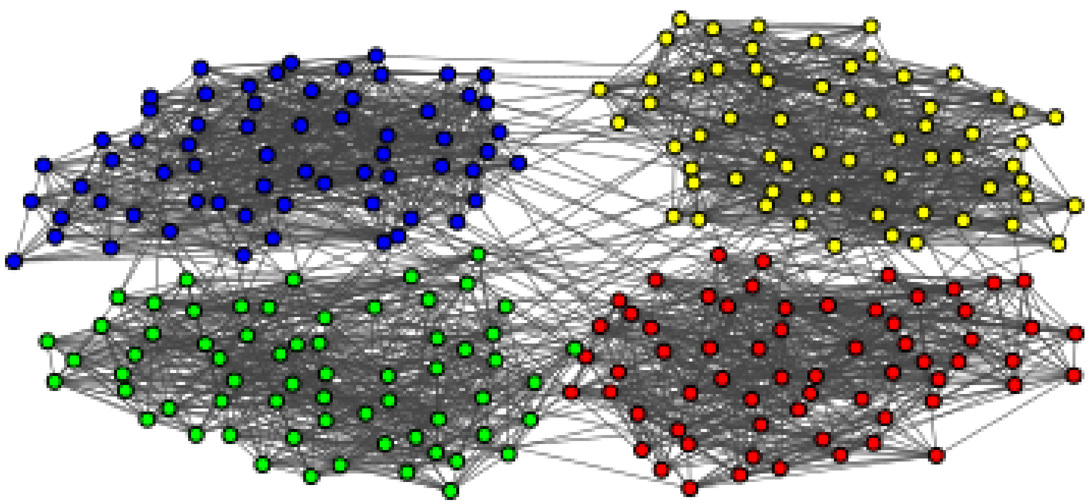
256 4441318 p network plot using Label Propagation Algorithm



256 4441318 p network plot using Louvain Algorithm



256 4441318 p network plot using Fast Greedy Algorithm



2.3 Comparison measures

This section's goal is to compare the partitions produced by various community detection techniques to reference partitions, which are thought to represent the network's nodes' actual clustering in reality. In the field of community detection, comparison measures like the Jaccard index, NMI, and NVI are frequently employed to assess the quality of a partition.
We can see how each algorithm performs on each network by grouping the comparison measurements by network. This enables us to examine how well various algorithms perform across various networks

and determine which algorithms function well uniformly across all networks and which ones could be better suited for particular types of networks.
Following we will share the tables representing the comparison using Jaccard, NVI and NMI. We will also share the networkx results as well.

Table representing the comparison using igraph

graph3+1+3

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.666667	0.80954	0.238237
1	leading_eigenvector	0.666667	0.80954	0.238237
2	label_propagation	0.666667	0.80954	0.238237
3	louvain	0.666667	0.80954	0.238237

20x2+5x2

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.941176	0.938345	0.051124
1	leading_eigenvector	0.941176	0.938345	0.051124
2	label_propagation	1.000000	1.000000	0.000000
3	louvain	0.941176	0.938345	0.051124

graph4+4

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	1.0	1.0	0.0
1	leading_eigenvector	1.0	1.0	0.0
2	label_propagation	1.0	1.0	0.0
3	louvain	1.0	1.0	0.0

star

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	1.0	1.0	0.0
1	leading_eigenvector	1.0	1.0	0.0
2	label_propagation	1.0	1.0	0.0
3	louvain	1.0	1.0	0.0

cat_cortex_sim

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.542169	0.656873	0.296602
1	leading_eigenvector	0.547872	0.618651	0.332598
2	label_propagation	0.257239	0.000000	0.481994
3	louvain	0.589416	0.672641	0.285823

zachary_unwh

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.683274	0.692467	0.217697
1	leading_eigenvector	0.505495	0.677092	0.269804
2	label_propagation	0.758741	0.723881	0.182728
3	louvain	0.450355	0.581858	0.350338

dolphins

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.504125	0.572700	0.271613
1	leading_eigenvector	0.329314	0.448914	0.423804
2	label_propagation	1.000000	1.000000	0.000000
3	louvain	0.304798	0.474253	0.431107

football

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.362153	0.697732	0.385871
1	leading_eigenvector	0.350324	0.698670	0.407185
2	label_propagation	0.763200	0.910680	0.131033
3	louvain	0.695906	0.884962	0.165626

256_4_4_4_13_18_p

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	1.000000	1.000000	0.000000
1	leading_eigenvector	1.000000	1.000000	0.000000
2	label_propagation	0.269841	0.680851	0.338132
3	louvain	0.904762	0.951742	0.036576

rb125

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.281143	0.825443	0.287653
1	leading_eigenvector	0.228837	0.781209	0.347461
2	label_propagation	0.429358	0.913109	0.157606
3	louvain	0.281143	0.825443	0.287653

256_4_4_2_15_18_p

	algorithm	Jaccard	NMI	NVI
0	fastgreedy	0.483871	0.869708	0.166303
1	leading_eigenvector	0.542431	0.924071	0.102676
2	label_propagation	1.000000	1.000000	0.000000
3	louvain	1.000000	1.000000	0.000000

Table representing the comparison using networkx

```
graph3+1+3
             algorithm   Jaccard      NMI       NVI
0  greedy_modularity  0.666667  0.80954  0.238237
1            louvain  0.666667  0.80954  0.238237
2           k-clique  0.285714  0.00000  0.744544
3  label_propagation  0.666667  0.80954  0.238237

20x2+5x2
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.941176  0.938345  0.051124
1            louvain  0.941176  0.938345  0.051124
2           k-clique  0.326531  0.000000  0.440163
3  label_propagation  1.000000  1.000000  0.000000

graph4+4
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  1.000000  1.000000  0.000000
1            louvain  0.461538  0.585645  0.510367
2           k-clique  0.428571  0.000000  0.480898
3  label_propagation  1.000000  1.000000  0.000000

star
             algorithm  Jaccard  NMI  NVI
0  greedy_modularity      1.0  1.0  0.0
1            louvain      1.0  1.0  0.0
2           k-clique      1.0  1.0  0.0
3  label_propagation      1.0  1.0  0.0

cat_cortex_sim
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.571930  0.702341  0.257299
1            louvain  0.637218  0.729666  0.236034
2           k-clique  0.257239  0.000000  0.481994
3  label_propagation  0.257239  0.000000  0.481994

zachary_unwh
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.379009  0.289122  0.503219
1            louvain  0.465950  0.592773  0.341193
2           k-clique  0.486631  0.000000  0.282870
3  label_propagation  0.403361  0.254134  0.494754
```

```
dolphins
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.245068  0.010911  0.628715
1            louvain  0.379116  0.516234  0.364424
2           k-clique  0.555791  0.000000  0.219805
3  label_propagation  0.203363  0.041141  0.717362

football
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.058675  0.147975  1.086858
1            louvain  0.713004  0.892296  0.155381
2           k-clique  0.079786  0.000000  0.746947
3  label_propagation  0.044610  0.238707  1.113253

256_4_4_4_13_18_p
             algorithm   Jaccard     NMI       NVI
0  greedy_modularity  1.000000  1.0000  0.000000
1            louvain  1.000000  1.0000  0.000000
2           k-clique  0.247059  0.0000  0.360674
3  label_propagation  0.413333  0.6993  0.263123

rb125
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.281143  0.825443  0.287653
1            louvain  0.281143  0.825443  0.287653
2           k-clique  0.031742  0.000000  0.967777
3  label_propagation  0.159132  0.754033  0.420127

256_4_4_2_15_18_p
             algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.461653  0.845852  0.196748
1            louvain  1.000000  1.000000  0.000000
2           k-clique  0.058824  0.000000  0.721348
3  label_propagation  0.245902  0.773341  0.266577
```

2.4 Modularity measures

This section's goal is to be able to provide a comparison of the modularity values between all the partitions obtained from the different algorithms, including the reference partitions. Modularity is a measure of the quality of a partition, which represents the degree to which a network can be divided into non-overlapping and densely connected groups.

By grouping the modularity values by network, we can compare the performance of each algorithm in identifying meaningful communities within each network. The table allows for a quick and easy comparison between the modularity values of the different partitions, allowing researchers to identify which algorithm performed better in identifying community structures in a particular network.

Table representing the modularity measures using igraph algorithms

	fastgreedy	louvain	label_prop
rb125	0.608733	0.608733	0.525830
256_4_4_4_13_18_p	0.696773	0.696773	0.663378
256_4_4_2_15_18_p	0.765660	0.781804	0.781804
zachary_unwh	0.380671	0.398176	0.132807
football	0.549741	0.604407	0.589858
dolphins	0.495491	0.523338	0.426526
cat_cortex_sim	0.260436	0.266097	0.000000
airports_UW	0.662490	0.699136	0.627621
grid-p-6x6	0.401235	0.416667	0.333333
graph4+4	0.423077	0.423077	0.423077
star	0.000000	0.000000	0.000000
graph3+1+3	0.367188	0.367188	0.367188
20x2+5x2	0.542579	0.542579	0.541586

	modularity
graph3+1+3	0.351562
20x2+5x2	0.541586
graph4+4	0.423077
star	0.000000
cat_cortex_sim	0.245996
zachary_unwh-real	0.371466
dolphins-real	0.373482
football-conferences	0.553973
256_4_4_4_13_18_p	0.696773
rb125-1	0.600595
rb125-2	0.558144
rb125-3	0.553147
256_4_4_2_15_18_p	0.781804

2.5 Conclusions about the algorithms

First, in the comparison part, we can observe that all algorithms successfully get a perfect score for every metric for several datasets, including graph4+4, star, and dolphins. This shows that clustering these datasets might be simple. Other datasets i te other hand, like catcortexsim, football, and zacharyunwh, have scores that differ between algorithms, indicating that clustering may be more difficult for these datasets.

When it comes to the algorithms presented in the igraph algorithms comparison table, we can see that the "labelpropagation" algorithm performs well when the data contains distinct clusters or communities, as shown by the high Jaccard and NMI scores it receives on numerous datasets. The "fastgreedy" and "leadingeigenvector" algorithms, on the other hand, typically perform well across many graph types, scoring highly in the Jaccard and NMI metrics on the majority of the datasets we used. Last but not least, the "louvain" algorithm performs occasionally well but less regularly than the others.

If we compare the results we got after using networkx algorithms, we can conclude that in terms

of Jaccard, NMI, and NVI scores, the algorithms "greedymodularity" and "louvain" generally outperform "kclique" and "labelpropagation." On many datasets, these two algorithms frequently gave us excellent results, occasionally even perfect results and scores. However, 'k-clique' and 'labelpropagation' frequently received lower scores, particularly in terms of NMI and NVI.
What can we conclude from these tables is that the algorithm "label propagation" is the one that most often gives more similar results to the reference partitions. Therefore it is not always the case since the algorithm is not deterministic, so running it again with the same graph may give us different outcomes.

In the modularity scores part, we can see that both Louvain and Label Propagation methods have greater modularity scores than Fast Greedy for the majority of network graphs. For instance, the Louvain and Label Propagation algorithms have modularity scores of 0.604407 and 0.589858, respectively, for the "football" network graph, while Fast Greedy got a score of 0.549741.

However, there are several exceptions to this observed pattern or conclusion. For instance, the Louvain and Label Propagation methods have higher modularity scores than Fast Greedy in the case of the "dolphins" network graph and the difference is not as pronounced as in other examples.

To conclude, we can say that selecting a community detection algorithm might affect the modularity score of a network graph and that in some cases we are required to test out various algorithms in order to get the best outcomes.