Complex Networks
A3: Community detection

Anne Schreiber Brito
Ruben Vera Garcia

Contents

Part 1

Introduction

This third assignment in Complex Networks will be focused on community detection. Three different algorithms will be used to detect communities in a variety of undirected networks, plus at least one of the algorithms must be based on the optimization of modularity. Moreover, the chosen algorithms can not be from the same program. For program, it is understood that utilizing different libraries is also considered as distinct programs.

When the partitions are made, some networks have provided partitions as .clu files. For these networks, there is the necessity of implementing the comparison between partitions.

Part 2

Process Description

In this part, it will be explained which tools were used for the process of the task.

For running the script, it is necessary to run the command "python main.py". This will start the process of community detection.

To compare the partitions, we used Radatools, which will give all the information about the two compared partitions.

2.1 Scripts descriptions

For the task in hand, we created five different python files, that would have distinct tasks. The first one is the main script, which reads the different files, organizing them, calls for the comparison between partitions and calls for the calculation of the partition's modularity. It also calls for the implementation of plotting the different generated partitions and saving it in a folder as a .clu file.

Then we have the plotting script that, as its name implies, it creates the generated partitions plots.

Furthermore, the algorithms script which has the three different algorithms to community detection. There is also a save partition script, which saves the partitions as .clu files.

Finally, the modularity_graph script, which calculates the modularity of the created partitions as well as the existing ones given within the task.

To see the complete work and how the scripts work, you can access to our GitHub repository.[5]

2.2 Models descriptions

In this section, we will explain each community detection algorithm functionality.

2.2.1 Louvain comparison - Networkx

The first algorithm chosen is the Louvain method for community detection in the Networkx Python library. This method is preferable for large networks, and it was created by Blondel et al. As expected from the assignment statement, Louvain focus on the optimization of modularity.

This method from the NetworkX library works in 2 steps, first it assigns every node of the graph to be its own community and as a second step, the algorithm tries to find the maximum positive modularity gain for each node, by moving each node to all of its neighbor communities, and if this is not possible, then the node stays in the community it was already in.[3]

Last thing to remark is the method being a greedy optimization with a complexity of $O(n \cdot \log n)$.

2.2.2 Greedy comparison - Networkx

The second algorithm is also a greedy modularity maximization to find the community partitions with the largest modularity. The algorithm uses Clauset-Newman-Moore version that can be found in the Networkx Python library.

Greedy modularity maximization begins with each node in its own community and repeatedly joins the pair of communities that lead to the largest modularity until no further increase in modularity is possible.[4]

2.2.3 Walktrap - Igraph

Walktrap is an algorithm developed by Pascal Pons that detects communities with random walks. It will be used the already implemented Walktrap in the Python library Igraph. The method is preferable for large networks.

The functionality is as follows, the random walks are used to calculate the distances between the individuals (nodes). These nodes are assigned to a unique community in order to achieve the smaller intra-community (between nodes of the same community) distances and larger inter-community (between full communities) distances with bottom-up hierarchical clustering.[2] [1]

Part 3

Results

3.1 Generated Partitions Plots

In this section, different plots of the existing networks will be displayed, where each color that the node has, represents the community they belong to.

3.1.1 Model

3.1.1.1 256_4_4_2_15_18_p

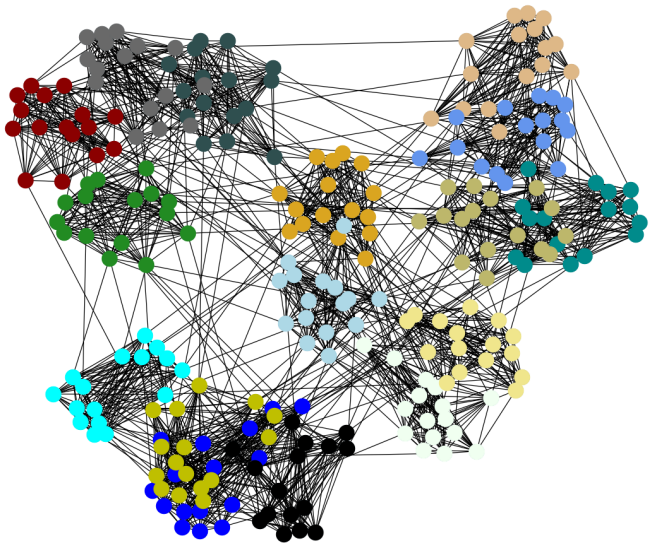

Figure 1: 256_4_4_2_15_18_p Louvain community detection results.

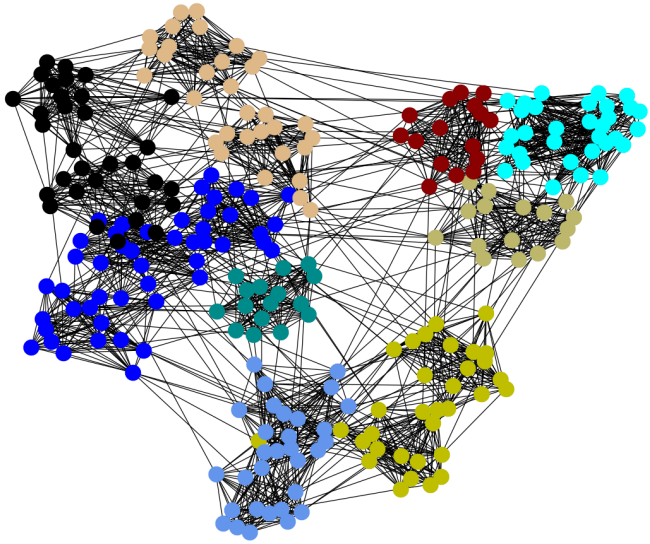

Figure 2: 256_4_4_2_15_18_p Greedy community detection results.

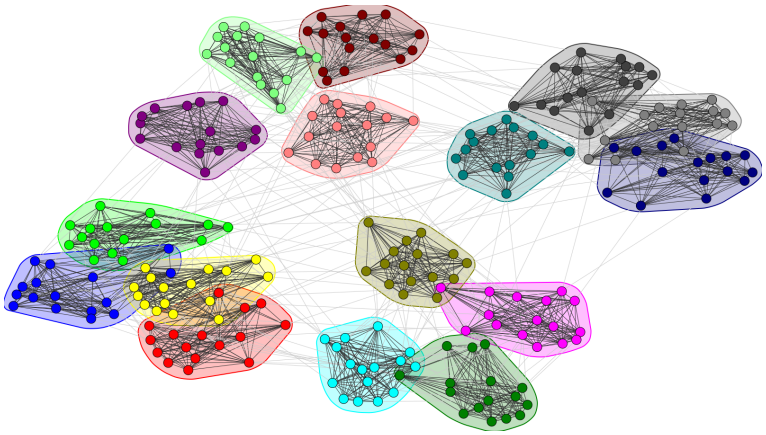

Figure 3: 256_4_4_2_15_18_p Walktrap community detection results.

3.1.1.2 256_4_4_4_13_18_p

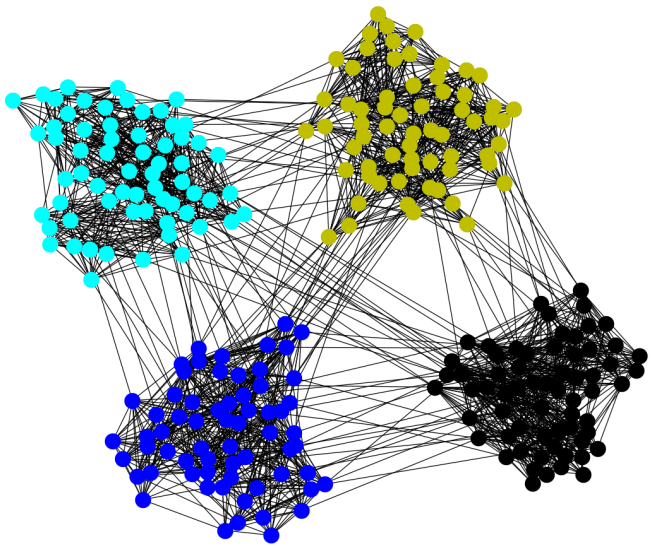

Figure 4: 256_4_4_4_13_18_p Louvain community detection results.

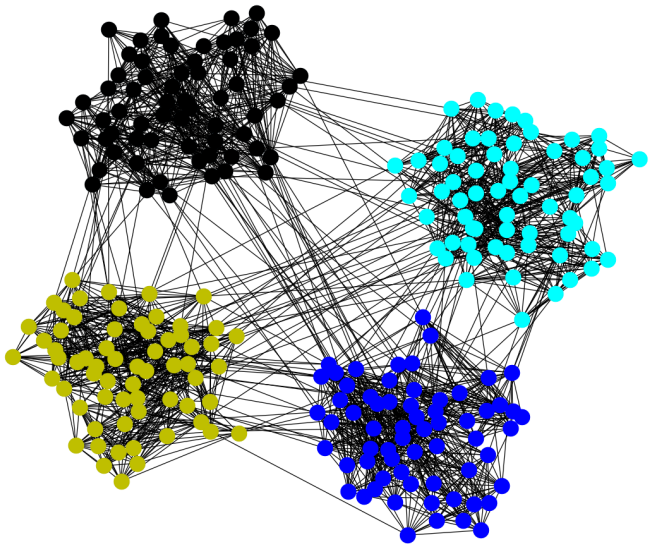

Figure 5: 256_4_4_4_13_18_p Greedy community detection results.

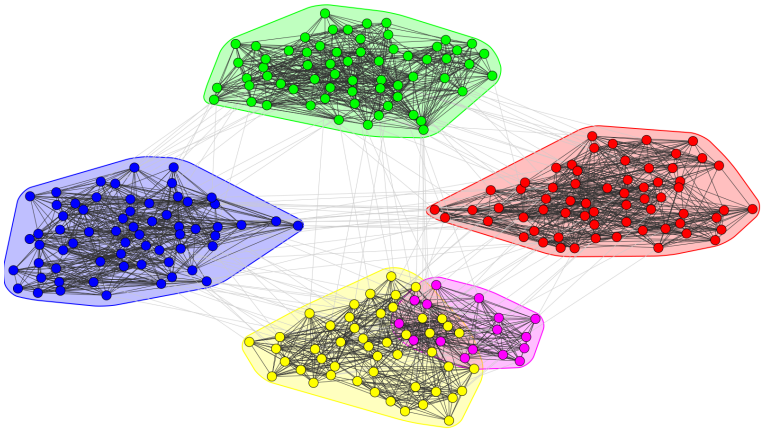

Figure 6: 256_4_4_4_13_18_p Walktrap community detection results.

3.1.1.3 rb125

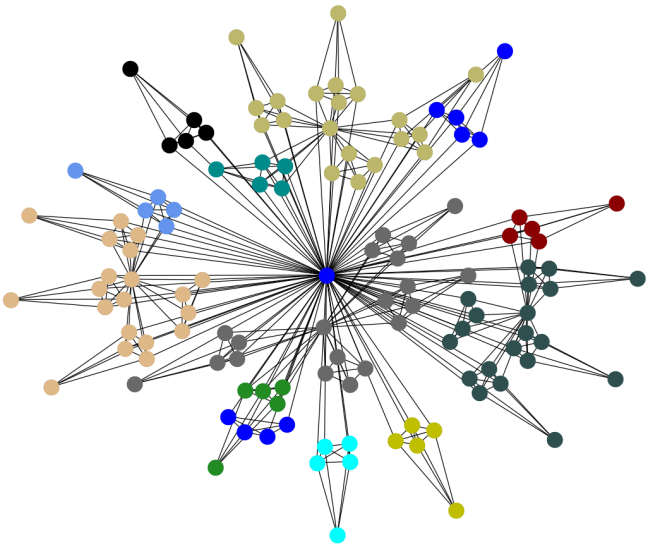

Figure 7: rb125 Louvain community detection results.

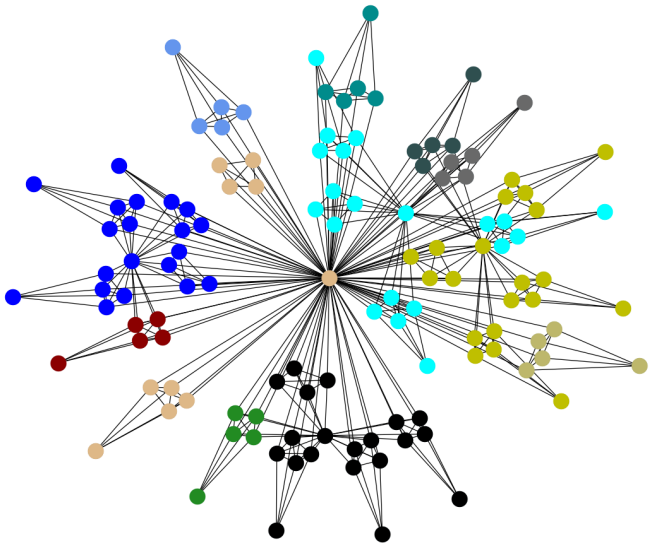

Figure 8: rb125 Greedy community detection results.

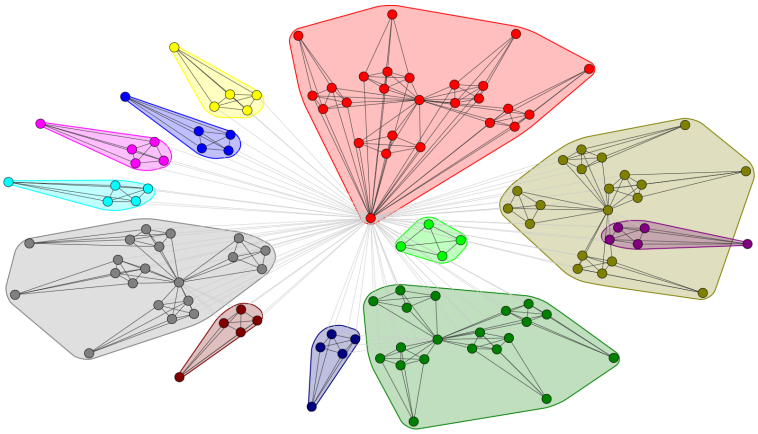

Figure 9: rb125 Walktrap community detection results.

3.1.2 Real

3.1.2.1 airports_UW

For this network we had difficulties plotting the graphs for the communities detected by the Louvain and Greedy algorithm, therfore this graphs are omitted. The graph displayed is from the Walktrap community detection algorithm.

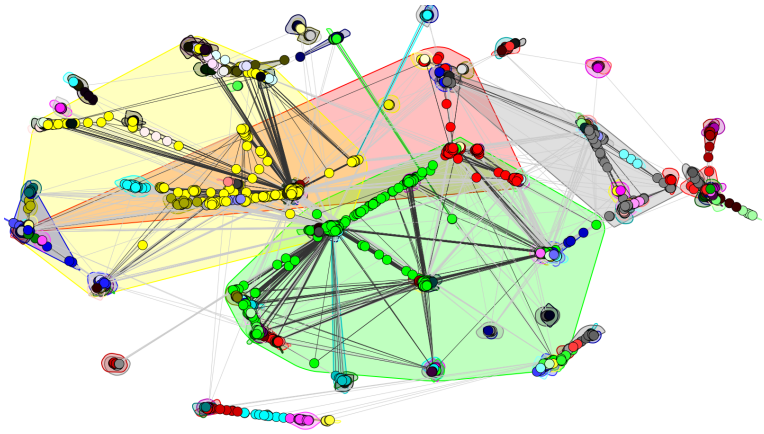

Figure 10: airports_UW Walktrap community detection results.

3.1.2.2 cat_cortex_sim

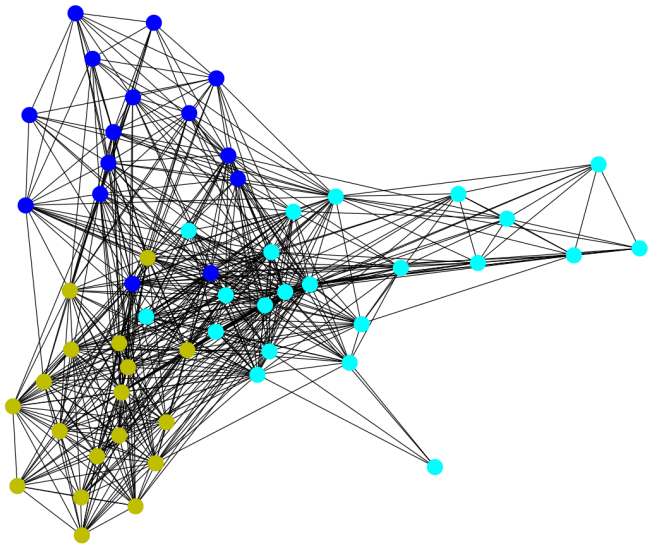

Figure 11: cat_cortex_sim Louvain community detection results.

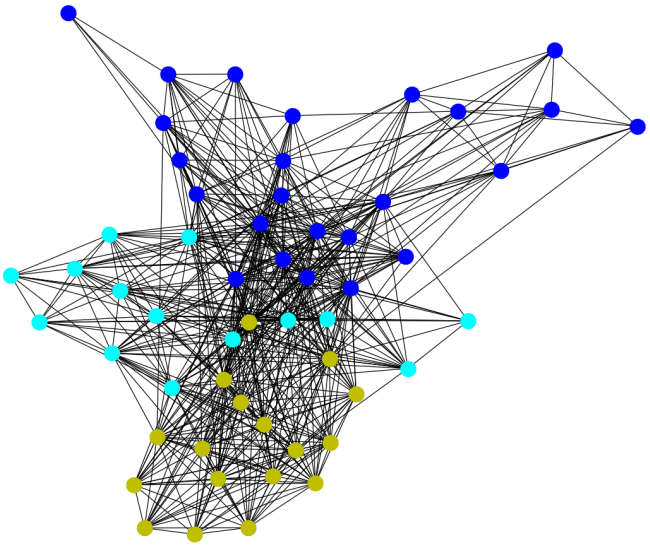

Figure 12: cat_cortex_sim Greedy community detection results.

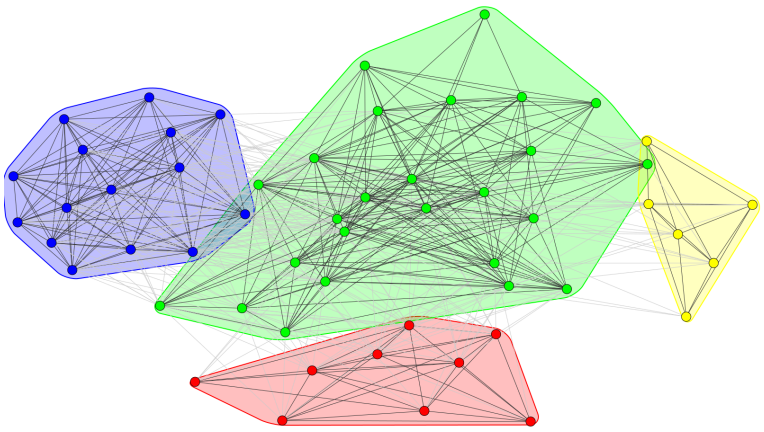

Figure 13: cat_cortex_sim Walktrap community detection results.

3.1.2.3 dolphins

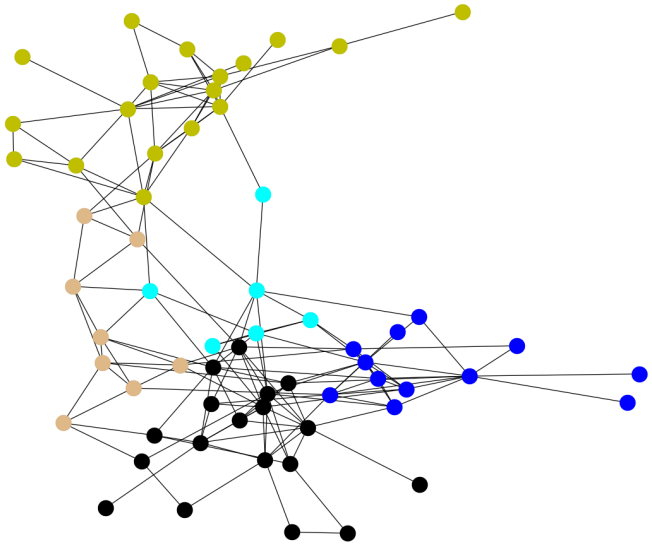

Figure 14: dolphins Louvain community detection results.

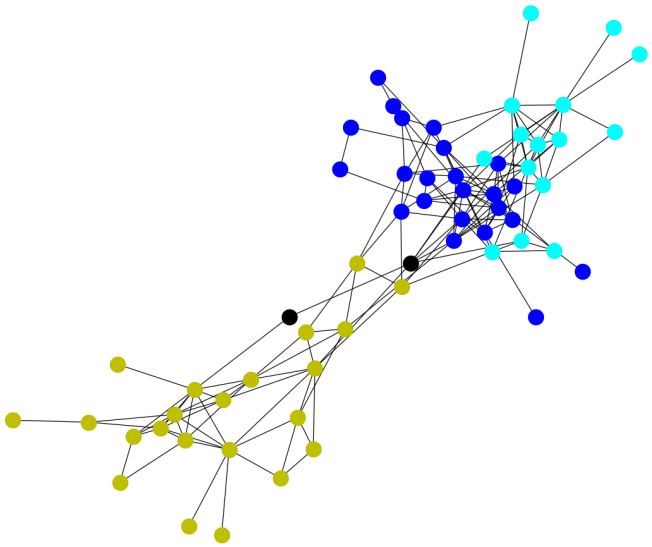

Figure 15: dolphins Greedy community detection results.

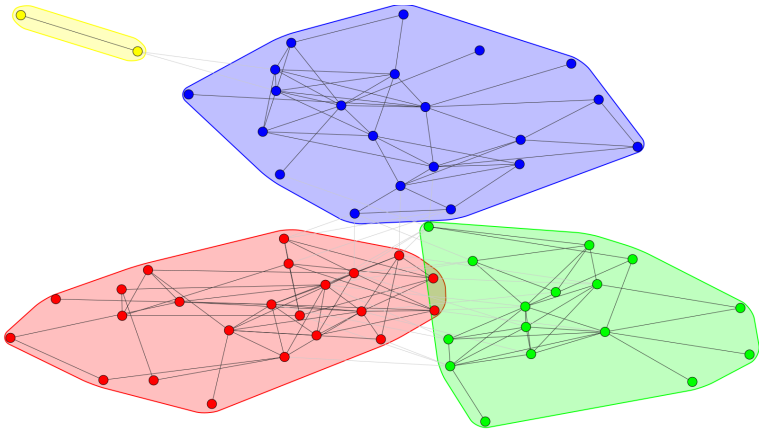

Figure 16: dolphins Walktrap community detection results.

3.1.2.4 football

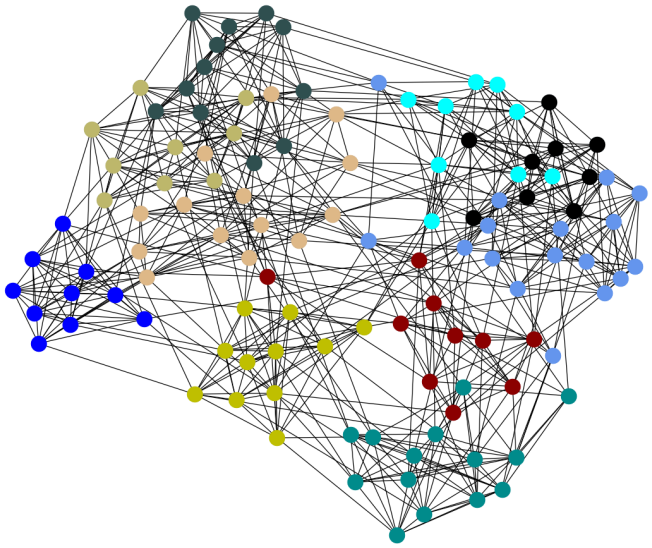

Figure 17: football Louvain community detection results.

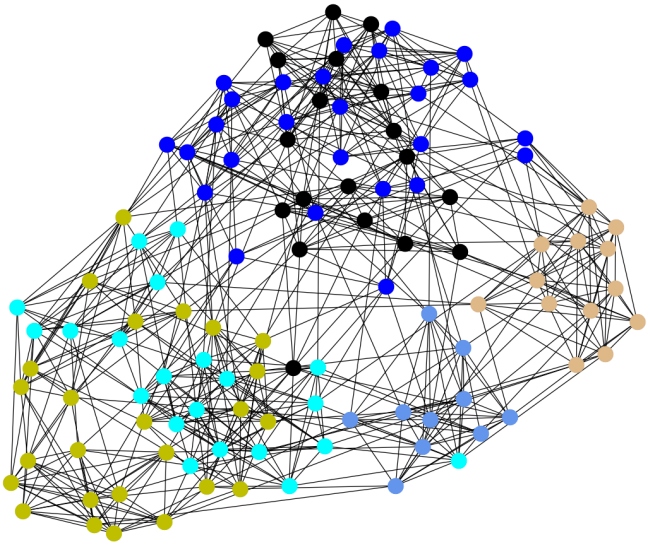

Figure 18: football Greedy community detection results.

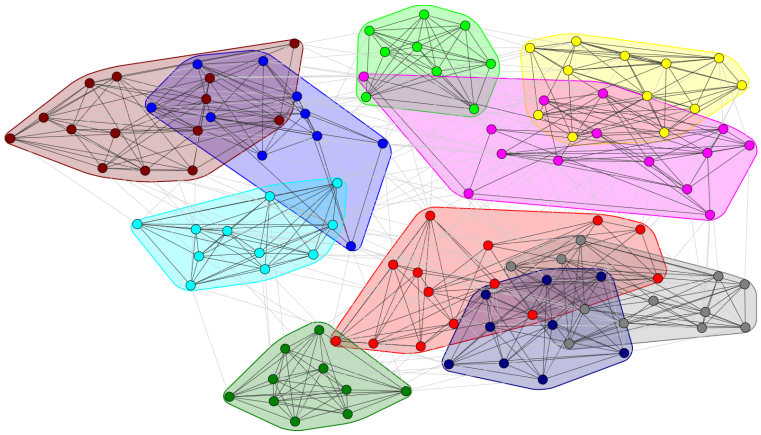

Figure 19: football Walktrap community detection results.

3.1.2.5 zachary_unwh

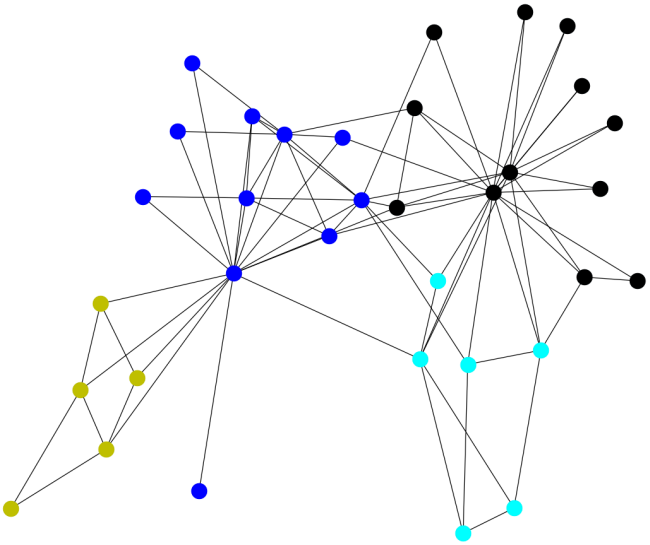

Figure 20: zachary_unwh Louvain community detection results.

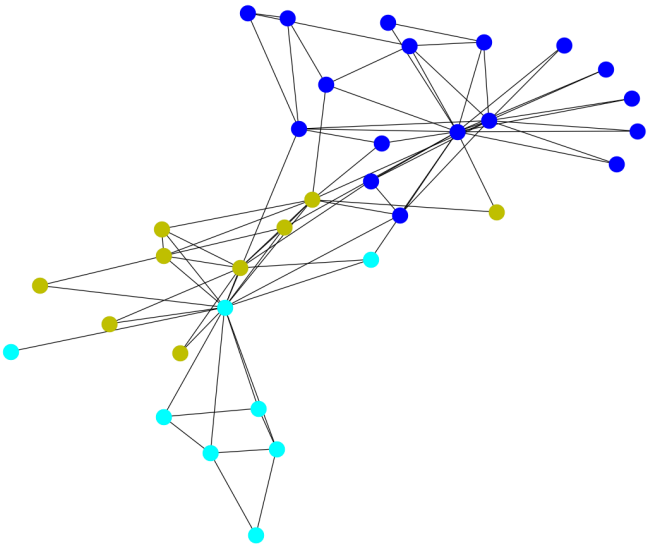

Figure 21: zachary_unwh Greedy community detection results.

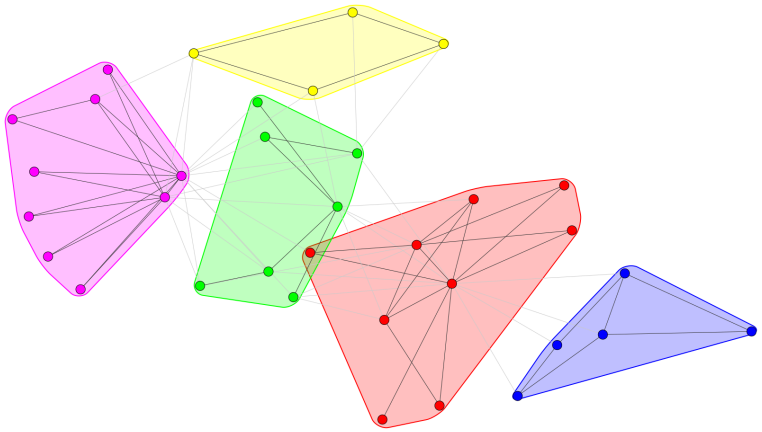

Figure 22: zachary_unwh Walktrap community detection results.

3.1.3 Toy

3.1.3.1 20x2+5x2

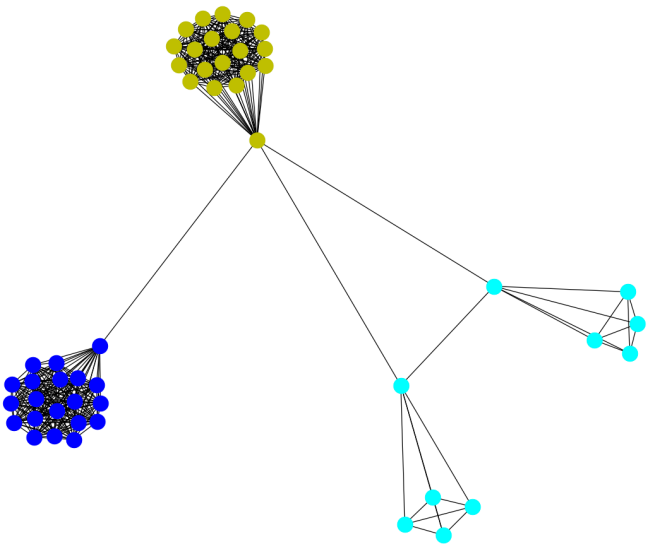

Figure 23: 20x2+5x2 Louvain community detection results.

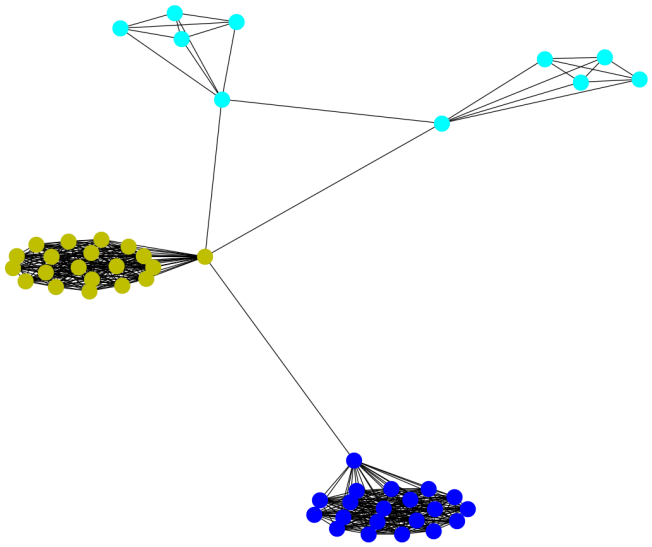

Figure 24: 20x2+5x2 Greedy community detection results.

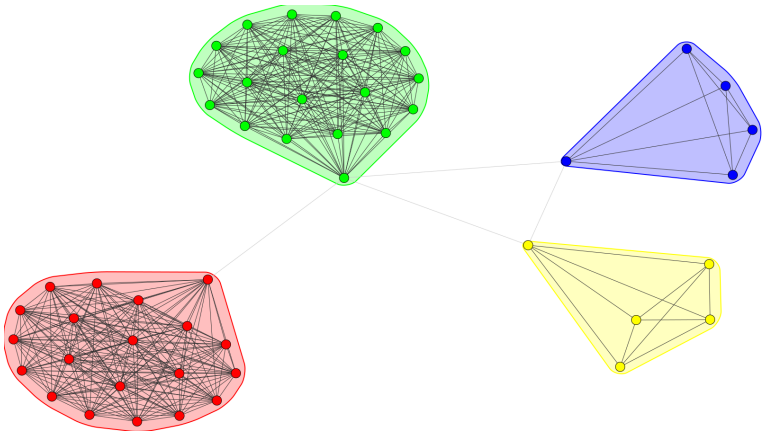

Figure 25: 20x2+5x2 Walktrap community detection results.

3.1.3.2 graph3+1+3

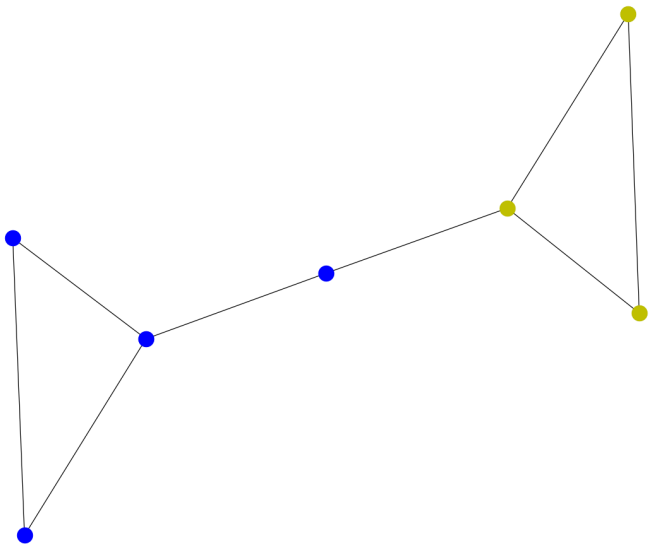

Figure 26: graph3+1+3 Louvain community detection results.

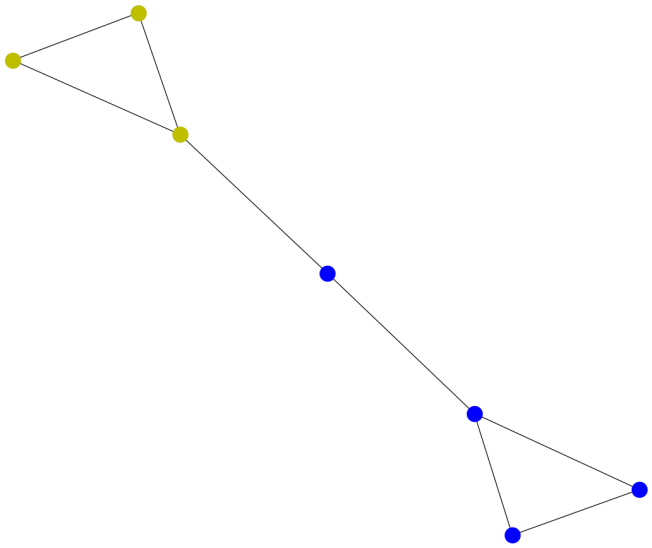

Figure 27: graph3+1+3 Greedy community detection results.

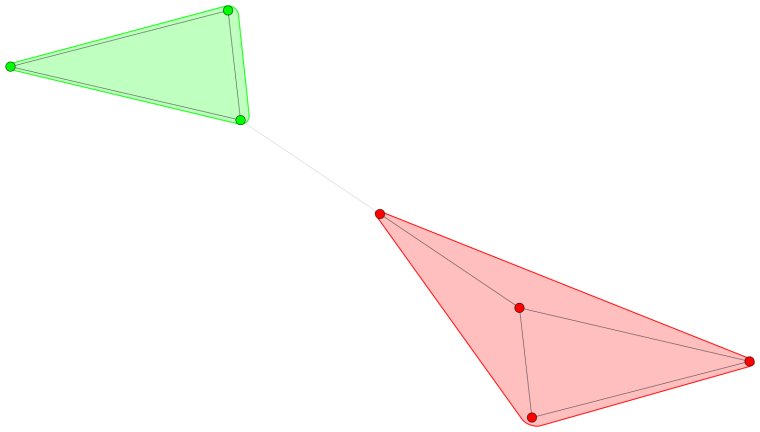

Figure 28: graph3+1+3 Walktrap community detection results.

3.1.3.3 graph4+4

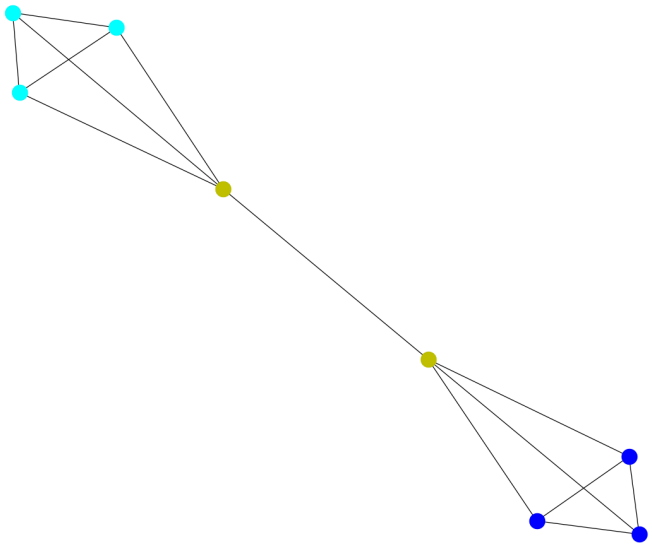

Figure 29: graph4+4 Louvain community detection results.

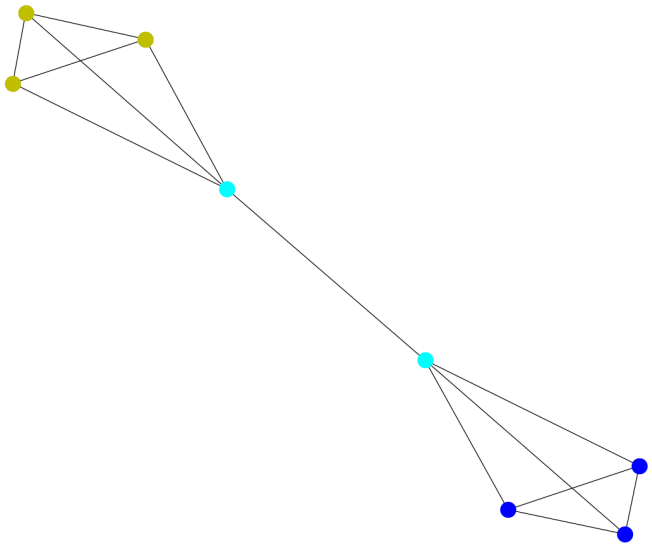

Figure 30: graph4+4 Greedy community detection results.

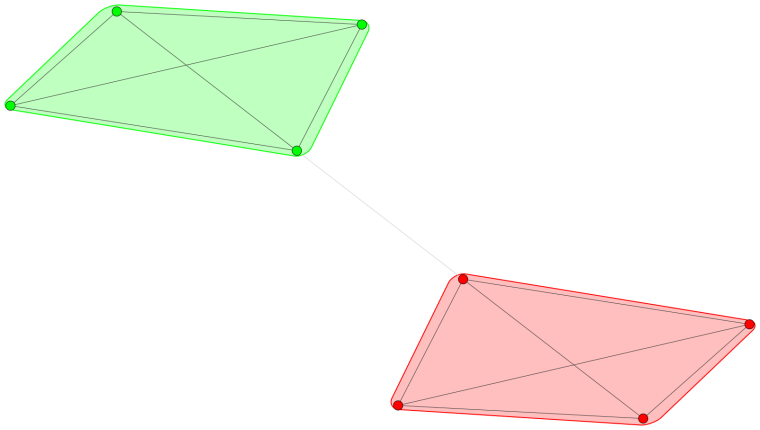

Figure 31: graph4+4 Walktrap community detection results.

3.1.3.4 grid-p-6x6

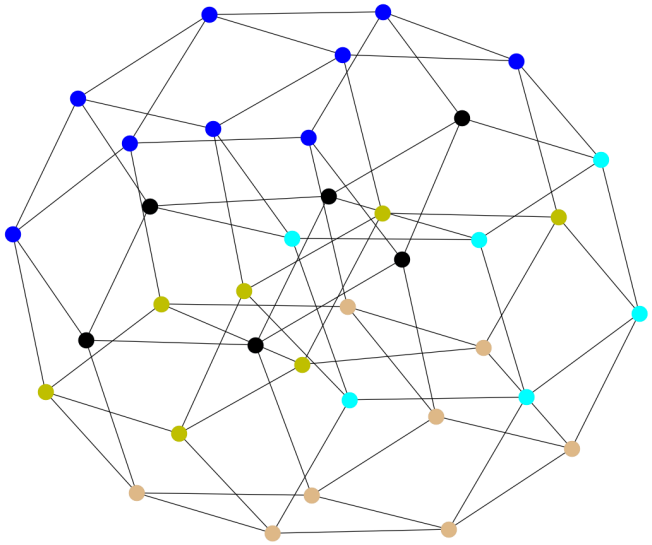

Figure 32: grid-p-6x6 Louvain community detection results.

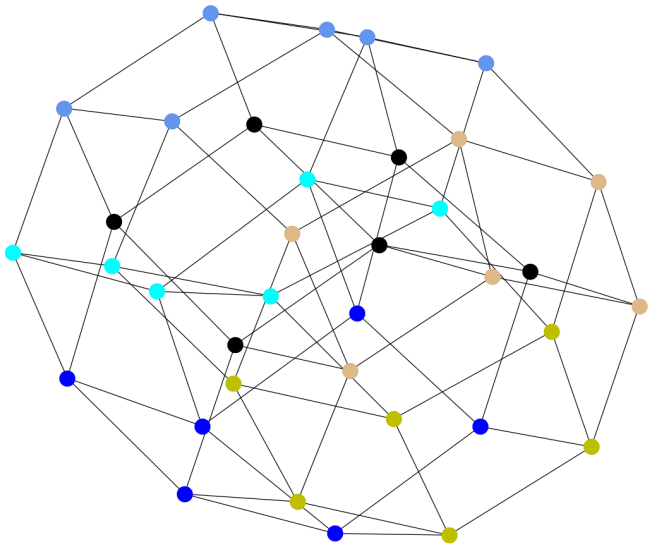

Figure 33: grid-p-6x6 Greedy community detection results.

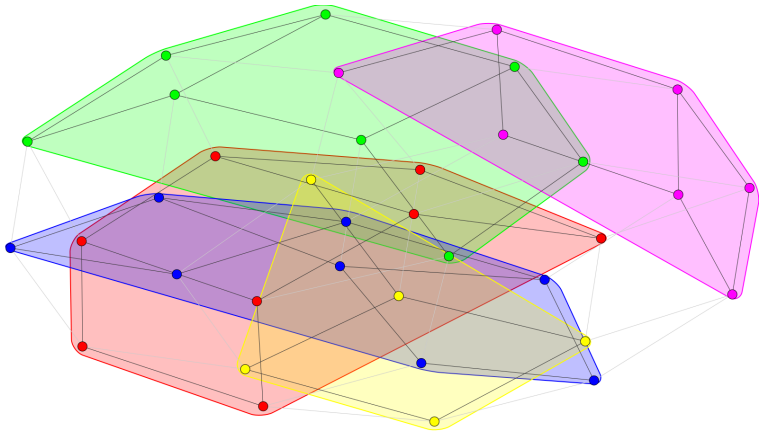

Figure 34: grid-p-6x6 Walktrap community detection results.

3.1.3.5 star

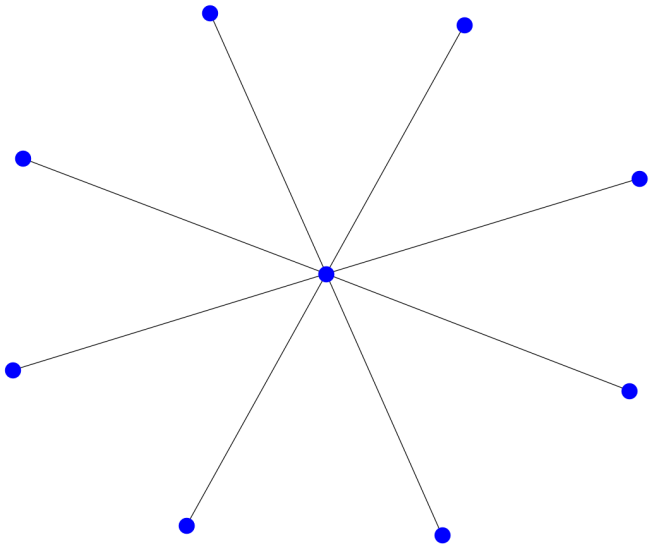

Figure 35: star Louvain community detection results.

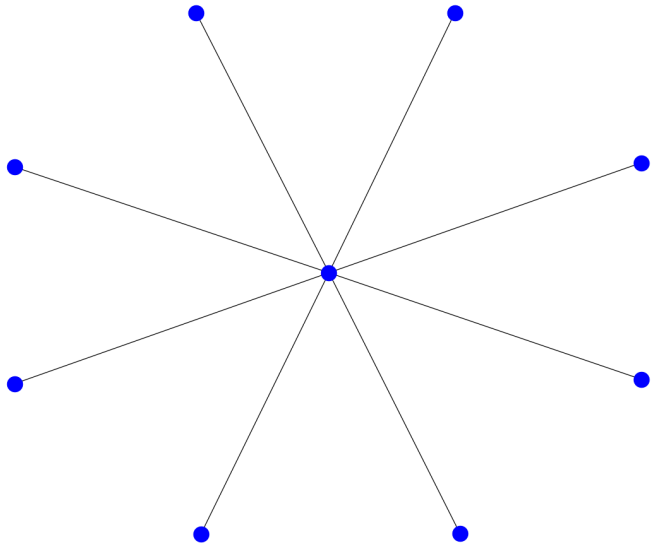

Figure 36: star Greedy community detection results.

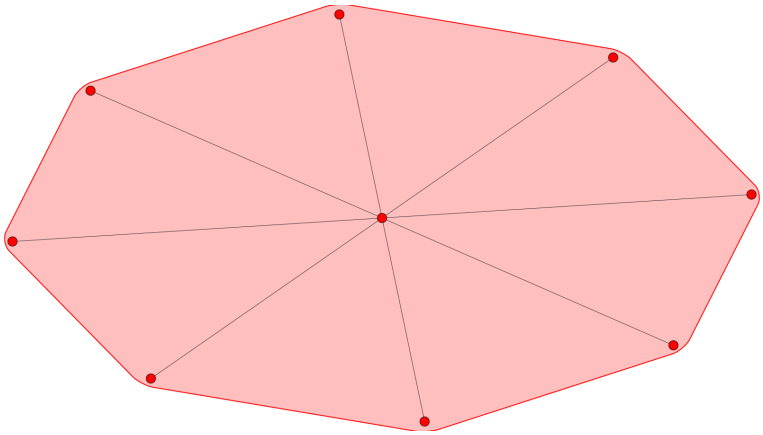

Figure 37: star Walktrap community detection results.

3.2 Tables

3.2.1 Comparison measures

The next table1 has the resulted comparisons metrics resulted from comparing the partitions with Radatools. The table shows first the two partitions being compared, then the Jaccard INdex (JI), the Normalized Mutual Information (arithmetic normalization) (NMI (arith.)) and finally the Normalized Variation of Information (NVI).

Partition x	Partition y	JI	NMI (arith.)	NVI
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_louvain.clu	1.0000	1.0000	0.0000
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_greedy.clu	0.4782	0.8637	0.1206
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_walktrap.clu	0.1577	0.4718	0.5282
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_louvain.clu	1.0000	1.0000	0.0000
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_greedy.clu	1.0000	1.0000	0.0000
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_walktrap.clu	0.2875	0.3169	0.3589
rb125-1.clu	rb125_louvain.clu	0.5833	0.8284	0.1381
rb125-1.clu	rb125_greedy.clu	0.5833	0.8284	0.1381
rb125-1.clu	rb125_walktrap.clu	0.2077	0.4242	0.4564
rb125-2.clu	rb125_louvain.clu	0.2857	0.8284	0.1952
rb125-2.clu	rb125_greedy.clu	0.2857	0.8284	0.1952
rb125-2.clu	rb125_walktrap.clu	0.1552	0.6591	0.3838
rb125-3.clu	rb125_louvain.clu	0.2811	0.8254	0.1994
rb125-3.clu	rb125_greedy.clu	0.2811	0.8254	0.1994
rb125-3.clu	rb125_walktrap.clu	0.1557	0.6640	0.3797
cat_cortex_sim.clu	cat_cortex_sim_louvain.clu	0.6372	0.7297	0.1636
cat_cortex_sim.clu	cat_cortex_sim_greedy.clu	0.5804	0.6873	0.1882
cat_cortex_sim.clu	cat_cortex_sim_walktrap.clu	0.2452	0.3320	0.4316
dolphins-real.clu	dolphins_louvain.clu	0.3791	0.5162	0.2526
dolphins-real.clu	dolphins_greedy.clu	0.5041	0.5727	0.1883
dolphins-real.clu	dolphins_walktrap.clu	0.2377	0.0159	0.4348
football-conferences.clu	football_louvain.clu	0.6959	0.8850	0.1148
football-conferences.clu	football_greedy.clu	0.3709	0.7141	0.2528
football-conferences.clu	football_walktrap.clu	0.0466	0.2333	0.7667
zachary_unwh-real.clu	zachary_unwh_louvain.clu	0.5348	0.6873	0.1784
zachary_unwh-real.clu	zachary_unwh_greedy.clu	0.6833	0.6925	0.1509
zachary_unwh-real.clu	zachary_unwh_walktrap.clu	0.2089	0.1359	0.5524
20x2+5x2.clu	20x2+5x2_louvain.clu	0.9412	0.9383	0.0354
20x2+5x2.clu	20x2+5x2_greedy.clu	0.9412	0.9383	0.0354
20x2+5x2.clu	20x2+5x2_walktrap.clu	0.9608	0.9161	0.0512
graph3+1+3.clu	graph3+1+3_louvain.clu	0.6667	0.8095	0.1651
graph3+1+3.clu	graph3+1+3_greedy.clu	0.6667	0.8095	0.1651
graph3+1+3.clu	graph3+1+3_walktrap.clu	0.6667	0.8095	0.1651
graph4+4.clu	graph4+4_louvain.clu	0.4615	0.5856	0.3538
graph4+4.clu	graph4+4_greedy.clu	0.4615	0.5856	0.3538
graph4+4.clu	graph4+4_walktrap.clu	1.0000	1.0000	0.0000
star.clu	star_louvain.clu	1.0000	1.0000	0.0000
star.clu	star_greedy.clu	1.0000	1.0000	0.0000
star.clu	star_walktrap.clu	1.0000	1.0000	0.0000

Table 1: Table with the comparison measures between the partitions made and the existing ones, grouped by network.

3.2.2 Modularity values

Network	Louvain	Greedy	Walktrap	Existing file 1	Ex. file 2	Ex. file 3
256_4_4_2_15_18_p	0.7818	0.7603	0.7818	0.7818	-	-
256_4_4_2_13_18_p	0.6968	0.6968	0.6974	0.6968	-	-
rb125	0.6087	0.6087	0.6053	0.6006	0.5581	0.5531
airports_UW	0.6458	0.6393	0.6214	-	-	-
cat_cortex_sim	0.3655	0.3662	0.2042	0.3624	-	-
dolphins	0.5188	0.4955	0.4888	0.3735	-	-
football	0.6043	0.5564	0.6029	-	-	-
zachary_unwh	0.4198	0.3807	0.3532	0.3715	-	-
20x2+5x2	0.5426	0.5426	0.5416	0.5416	-	-
graph3+1+3	0.3672	0.3672	0.3672	0.3516	-	-
graph4+4	0.4408	0.4408	0.4231	0.4429	-	-
grid-p-6x6	0.4059	0.4167	0.3920	-	-	-
star	0	0	0	0	-	-

Table 2: Table with the modularity values of the partitions made and the existing ones, grouped by network.

Part 4

Conclusions

From the given results we can see that using algorithms based on the optimization of modularity, such as the Greedy algorithm or the Louvain algorithm, are more efficient in finding the different groups each node belongs to and both algorithms have very similar outcomes when it comes to do so. Just for a few networks, both algorithms gave different results, however the results weren't so far from each other.

When it comes to the Walktrap algorithm that we used from igraph, we can see that the community partition results were different from the previous two algorithms and in some occasion the quantity of partitions were different. However, the Walktrap algorithm realized an efficient job in partitioning simple networks, where there was a so to say predefined structure, where the nodes were not bulked together, a distinctive figure or structure could be seen. For other complex networks, such as the airport or the football networks, the algorithm did a fairly good job but not as good as the algorithms based on the optimization of modularity.

Furthermore, when comparing the created partitions from the three different algorithms with the already existing ones, the first two algorithms (Louvain & Greedy) resulted to have better similarities than the Walktrap algorithm. Most of the time, the metrics show better results for the algorithms based on modularity optimization than the Walktrap algorithm, just in few occasions the Walktrap algorithm showed best suited for the network or equally suited, when comparing to the given partition.

With this we can say that Louvain and Greedy algorithms are best suited for large and complex networks as for the Walktrap algorithm would be best suited for networks that are medially complex or small and medium networks.

References

[1] iGraph. (n.d.). *python-igraph API reference: List of all classes, functions and methods in python-igraph*.
https://igraph.org/python/doc/api/igraph.Graph.html#community_edge_betweenness

[2] Jayawickrama, T.D. (2021). *Community Detection Algorithms*. Towards Data Science.
https://towardsdatascience.com/community-detection-algorithms-9bd8951e7dae

[3] NetworkX. (2022). *louvain_communities*.
https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.louvain.louvain_communities.html#networkx.algorithms.community.louvain.louvain_communities

[4] NetworkX. (2022). *greedy_modularity_communities*.
https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.modularity_max.greedy_modularity_communities.html#networkx.algorithms.community.modularity_max.greedy_modularity_communities

[5] Schreiber, A. & Vera, R. (2022). *A3-ComplexNet*. Github
https://github.com/ACSBSC/A3-ComplexNet