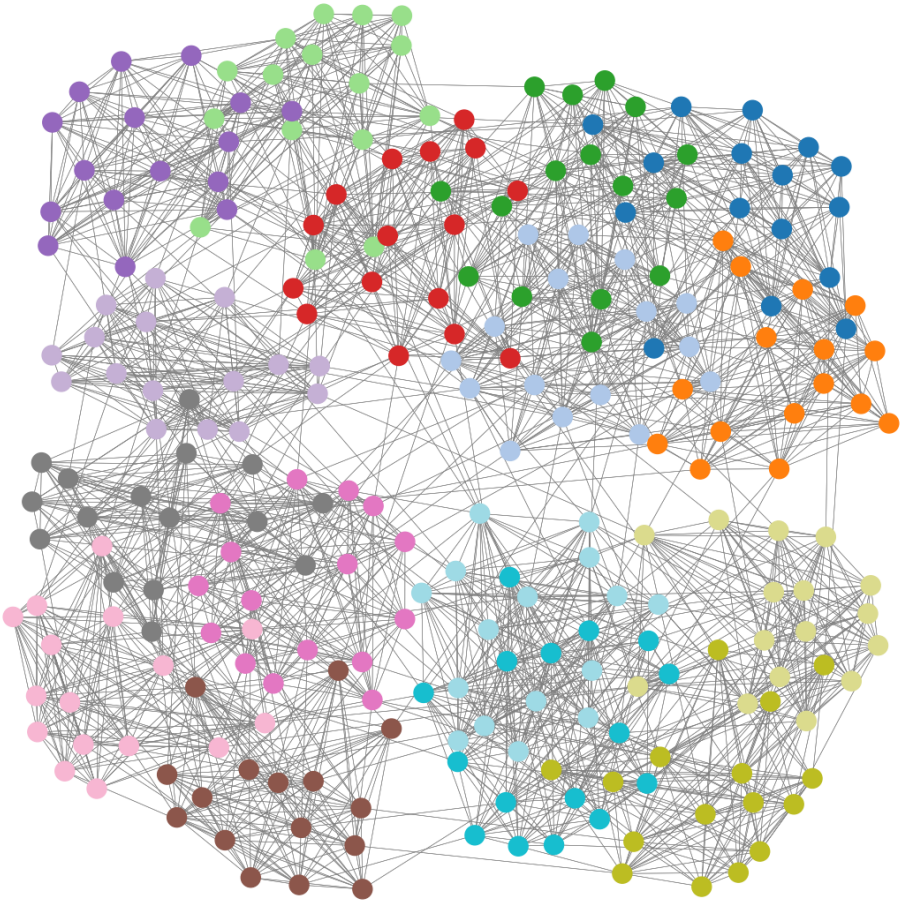
Complex Networks
Assignment 3: Community detection

Denaldo Lapi and Samy Chouiti

April 18, 2022



Community detection example

Contents

1 Programs and algorithms used

The goal of this assignment was to apply and compare different community detection algorithms. We used some already implemented algorithms available in the "networkx" module of Python [1] and others implemented in the "igraph" package [2]. In order to compare the obtained network partitions with the reference ones we used "Radatools" [3].

In particular, the algorithms from "networkx" we used are:

- *Clauset-Newman-Moore greedy modularity maximization* [4]. The algorithm finds communities by using a modularity maximization approach: it begins with each node in its own community and repeatedly joins the pair of communities that lead to the largest modularity until no futher increase in modularity is possible. We applied the algorithm with the default parameters of the "networkx" implementation.

- *Asynchronous label propagation* [5]. The algorithm proceeds as follows: after initializing each node with a unique label, the algorithm repeatedly sets the label of a node to be the label that appears most frequently among that nodes neighbors. The algorithm halts when each node has the label that appears most frequently among its neighbors. The algorithm is asynchronous because each node is updated without waiting for updates on the remaining nodes.

- *Louvain community detection* [6]. The algorithm is based on the optimization of the modularity measure and a hierarchical approach. Initially, each vertex is assigned to a community on its own. In every step, vertices are re-assigned to communities in a local, greedy way: in a random order, each vertex is moved to the community with which it achieves the highest contribution to modularity. When no vertices can be reassigned, each community is considered a vertex on its own, and the process starts again with the merged communities. The process stops when there is only a single vertex left or when the modularity cannot be increased any more in a step.

We also applied the following algorithms implemented in the "igraph" library:

- *Spinglass community detection by Reichardt and Bornholdt* [7]. This algorithm tries to find communities in graphs via a spin-glass model and simulated annealing. It is an approach from statistical physics, based on the so-called Potts model. In this model, each particle (i.e. vertex) can be in one of k spin states, and the interactions between the particles (i.e. the edges of the graph) specify which pairs of vertices would prefer to stay in the same spin state and which ones prefer to have different spin states. The model is then simulated for a given number of steps, and the spin states of the particles in the end define the communities.

- *Walktrap community detection by Pons and Latapy* [8]. The general idea of the algorithm is that if you perform random walks on the graph, then the walks are more likely to stay

[1] networkx documentation
[2] igraph documentation
[3] Radatools documentation
[4] Greedy modularity maximization
[5] Asynchronous label propagation
[6] Louvain community detection
[7] Spinglass community detection
[8] Walktrap community detection

within the same community because there are only a few edges that lead outside a given community. This method runs short random walks of 3-4-5 steps (depending on one of its parameters, in our case we use the default value set to 4) and uses the results of these random walks to merge separate communities in a bottom-up manner. You can use the modularity score to select where to cut the dendrogram: in our case we'll choose the partition with the highest modularity score.

- *Girvan-Newman (GN)* [9]. The GN algorithm is one of the most famous community detection algorithm and also one of the most intuitive. The specificity of this algorithm is the use of *betweeness centrality* of an edge: the number of shortest path between nodes in which this edge is part of. 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 (again we choose the partition corresponding to the max modularity value).

All the details of the performed operations and the links to the algorithms' implementations in "networkx" and in "igraph" are showed in detail in the provided python notebook *cd_algorithms.ipynb*.

1.1 Solutions adopted

We would like to remark in this section some of the most significant solutions we adopted (which can be seen on the provided notebook). In particular, we decided to use "networkx" and the "draw" function to plot the networks. In order to obtain significant plots for all the considered networks and the partitions of each network, we decided to fix the position of the nodes of each network for each different partition of the network: this will allow us to easily compare the plots of the various partitions. We used the layout algorithm proposed by Kamada-Kawai [10] which allows to render the network with minimal edge overlapping.
For networks that were already provided with nodes coordinates, we used those coordinates for performing the plots, without resorting to the Kamada-Kawai layout algorithm. In order to do that we simply built an auxiliary python function that reads the network pajek file and populates a dictionary containing the coordinates of each node, indexed by node. This dictionary is then used as a parameter for the "networkx" "draw" function.

Another point we want to focus on is the computation of the modularities of the already provided network partitions: we use another auxiliary python function that reads the "pajek" partition file (by simply providing the path), parses it in order to build the community list and then we use the "networkx" function to compute the modularity.

Again, the detailed implementation of this functions can be visualized in the attached python notebook.

Finally, we would like to point out how we used "Radatools" to perform the comparison of the network's partitions with the reference ones, when such exists. Please note that the plots

[9] Girvan-Newman algorithm
[10] Kamada-Kawai layout

have been generated separately from the comparaisons indexes, even though the overall trend for each network and community detection technique is relatively similar.

1.2 Structure of the assignment folder

The assignment folder is organized as follows:

- *partitions folder*: Contains the partitions generated for each provided network by using the algorithms described above. The folder contains 3 subfolders:
  - *model*: it contains a folder for each network of type "model". Inside each network's folder there are the *..clu* files of the obtained partitions.
  - *real*: it contains a folder for each network of type "real". Inside each network's folder there are the *..clu* files of the obtained partitions.
  - *toy*: it contains a folder for each network of type "toy". Inside each network's folder there are the *..clu* files of the obtained partitions.
- *cd_algorithms.ipynb*: python notebook with all the implemented community detection algorithms and the auxiliary functions we used. The notebook is run for a single sample network. We strongly suggest to visualize this notebook to get a better understaning of the report.
- *cd_algorithms.py*: automated version of the notebook to generate partitions and modularities for each network.
- *auto_compare.py*: python script that executes the "Compare_Partitions.exe" executable of "Radatools" for calculating the similarity measures with the reference partitions.
- *auto_notebook.txt*: text file with all the partition results obtained after running the algorithms on all the networks (number of found communities and modularity value of each partition)
- *Report* file

2 Obtained partitions

Let's now see, one by one, the networks we analyzed and the partitions we obtained by applying the algorithms described above. We were provided with 3 folders containing various types of networks:

- "toy"
- "model"
- "real"

We'll plot together the partitions of each network according to the following order, from left to right and from top to bottom:

- Girvan-Newman
- Clauset-Newman-Moore greedy approach
- Louvain
- Asynchronous label propagation
- Spinglass
- Walktrap

In addition, we'll show for each network a table with the modularity values of the obtained partitions, and a table containing the comparison measures between our partitions and the reference ones. In particular, the considered measures are, in order:

- Normalized Mutual Information Index (arithmetic), indicated as "Mutual info"
- Normalized Variation of Information, indicated as "Variance"
- Jaccard Index

2.1 Networks in the "toy" folder

2.1.1 "20x2+5x2" network

In the table 1 we report the modularity values of the network partitions, while in the table 2 we report the comparison measures between our partitions and the reference one(s).
In figure 1 the 6 partitions obtained after applying the algorithms to the "20x2+5x2" network.

2.1.2 "graph3+1+3" network

In the table 3 we report the modularity values of the network partitions, while in the table 4 we report the comparison measures between our partitions and the reference one(s).
In figure 2 the 6 partitions obtained after applying the algorithms to the "graph3+1+3" network.

2.1.3 "graph4+4" network

In the table 5 we report the modularity values of the network partitions, while in the table 6 we report the comparison measures between our partitions and the reference one(s).
In figure 3 the 6 partitions obtained after applying the algorithms to the "graph4+4" network.

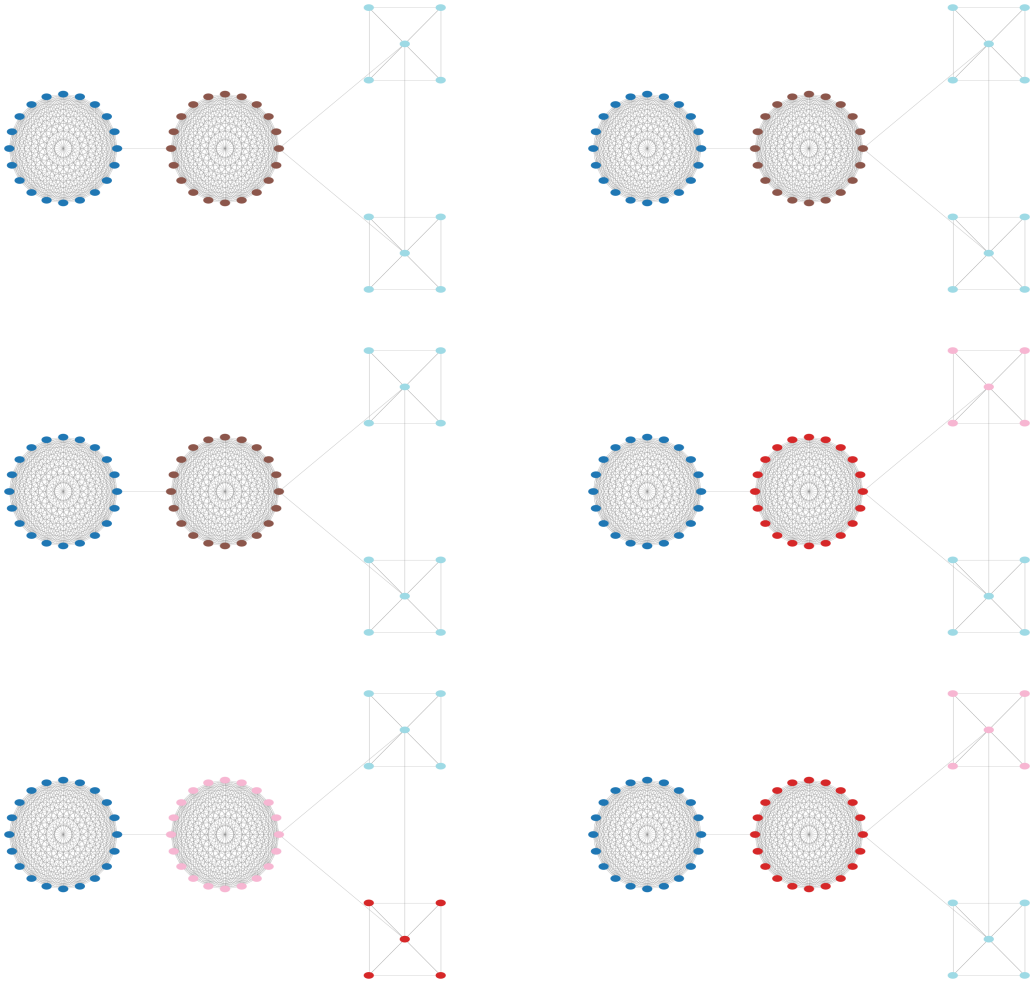


Figure 1: “20x2+5x2” network partitions

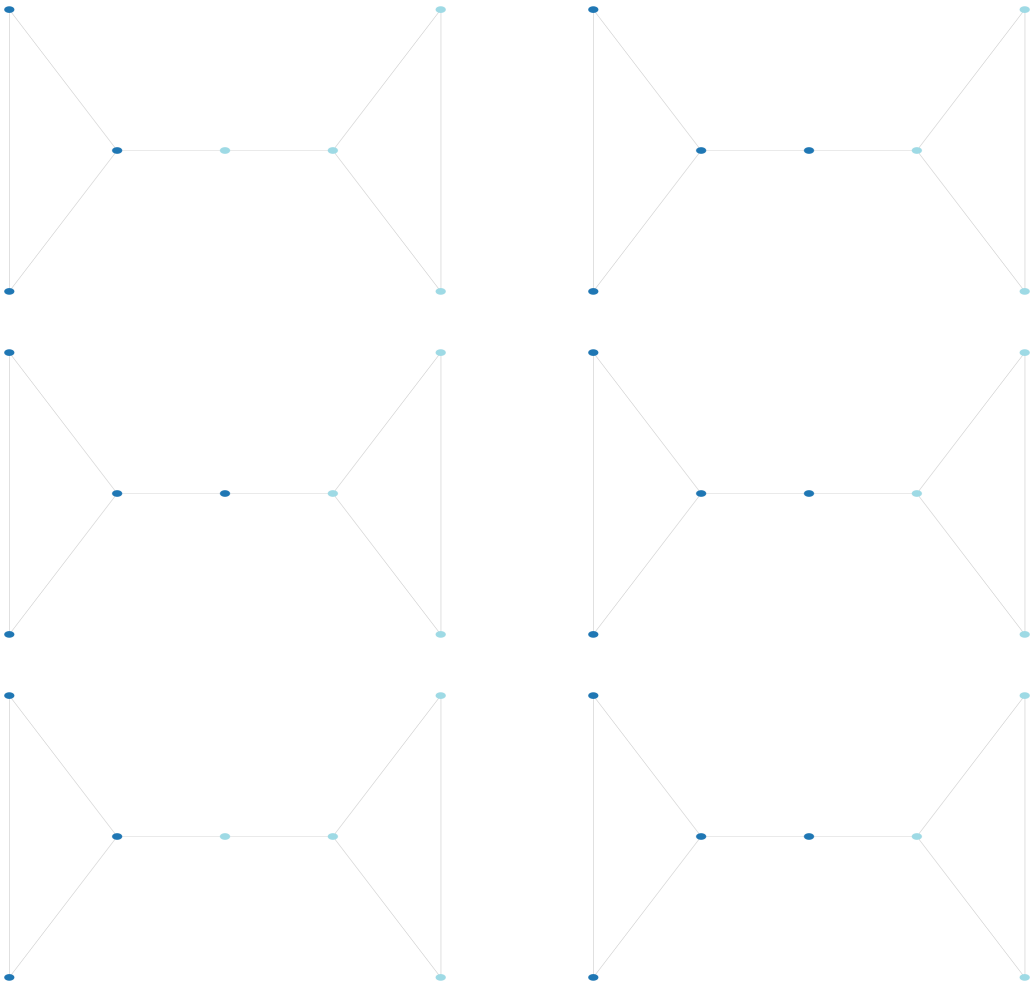


Figure 2: “graph3+1+3” network partitions

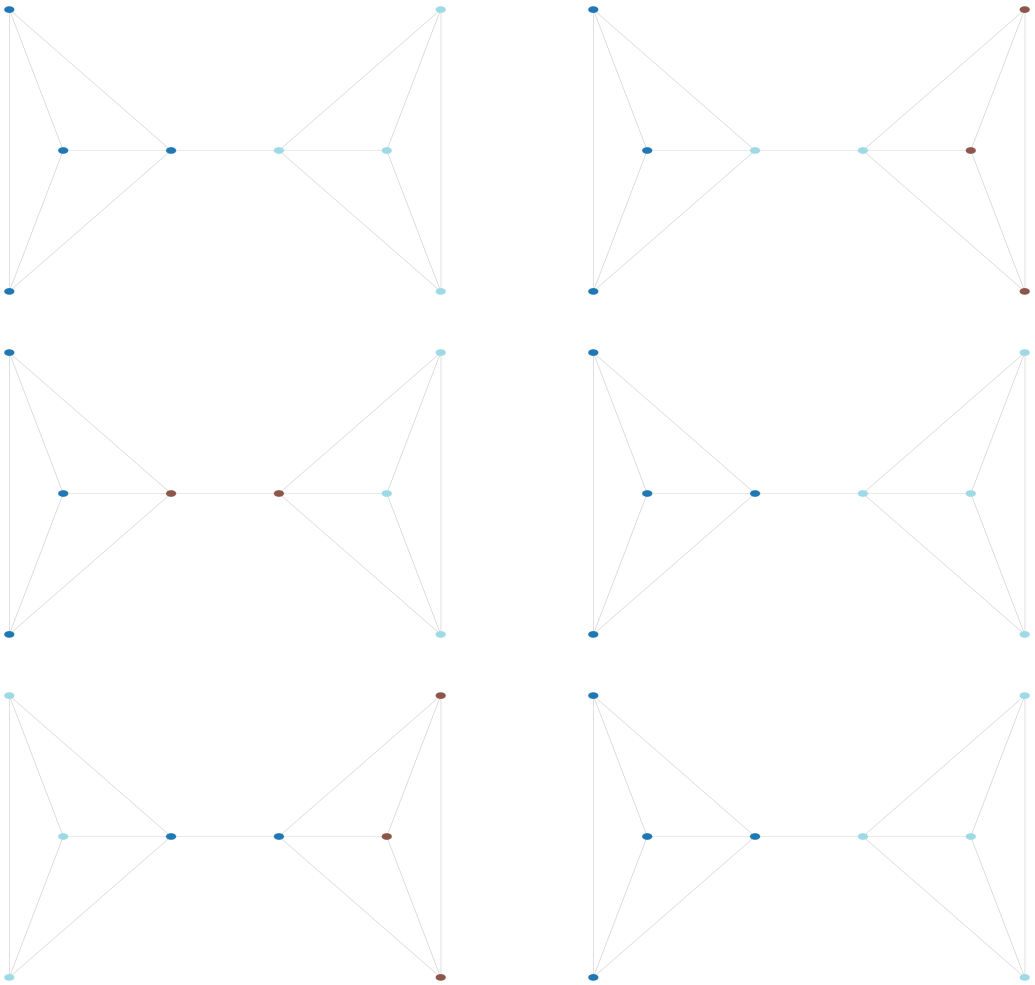


Figure 3: “graph4+4” network partitions

Network name	Method	Modularity
nets\toy\20x2+5x2	Reference	0.5415859964709342
nets\toy\20x2+5x2	Greedy	0.5425785462209587
nets\toy\20x2+5x2	Asyn	0.5415859964709342
nets\toy\20x2+5x2	Louvain	0.5425785462209587
nets\toy\20x2+5x2	Spinglass	0.5415859964709343
nets\toy\20x2+5x2	Walktrap	0.5415859964709343
nets\toy\20x2+5x2	Girvan-Newman	0.5425785462209588

Table 1: "20x2+5x2" network modularities

Reference Name	Compared Name	Mutual info [11]	Variance [12]	Jaccard Index
20x2+5x2.clu	20x2+5x2_asyn.clu	0.0000	1.0000	1.0000
20x2+5x2.clu	20x2+5x2_girvan-newman.clu	0.0354	0.9383	0.9412
20x2+5x2.clu	20x2+5x2_greedy.clu	0.0354	0.9383	0.9412
20x2+5x2.clu	20x2+5x2_louvain.clu	0.0354	0.9383	0.9412
20x2+5x2.clu	20x2+5x2_spinglass.clu	0.0000	1.0000	1.0000
20x2+5x2.clu	20x2+5x2_walktrap.clu	0.0000	1.0000	1.0000

Table 2: "20x2+5x2" network measures

Network name	Method	Modularity
nets\toy\graph3+1+3	Reference	0.3515625
nets\toy\graph3+1+3	Greedy	0.3671875
nets\toy\graph3+1+3	Asyn	0.3671875
nets\toy\graph3+1+3	Louvain	0.3671875
nets\toy\graph3+1+3	Spinglass	0.3671875
nets\toy\graph3+1+3	Walktrap	0.3671875
nets\toy\graph3+1+3	Girvan-Newman	0.3671875

Table 3: "graph3+1+3" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
graph3+1+3.clu	graph3+1+3_asyn.clu	0.1651	0.8095	0.6667
graph3+1+3.clu	graph3+1+3_girvan-newman.clu	0.1651	0.8095	0.6667
graph3+1+3.clu	graph3+1+3_greedy.clu	0.1651	0.8095	0.6667
graph3+1+3.clu	graph3+1+3_louvain.clu	0.1651	0.8095	0.6667
graph3+1+3.clu	graph3+1+3_spinglass.clu	0.1651	0.8095	0.6667
graph3+1+3.clu	graph3+1+3_walktrap.clu	0.1651	0.8095	0.6667

Table 4: "graph3+1+3" network metrics

2.1.4 "grid-p-6x6" network

In the table 7 we report the modularity values of the network partitions
In figure 4 the 6 partitions obtained after applying the algorithms to the "grid-p-6x6" network.

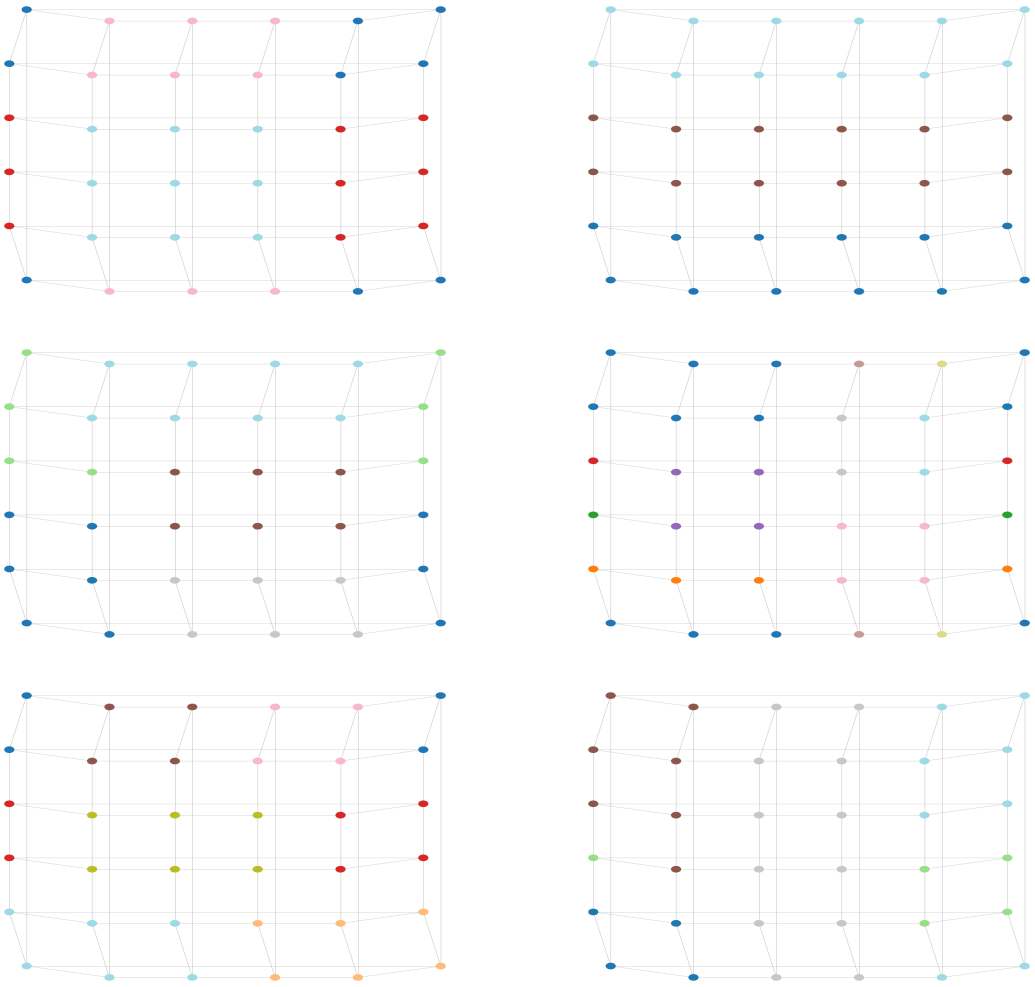


Figure 4: “grid-p-6x6” network partitions

Network name	Method	Modularity
nets\toy\graph4+4	Reference	0.44285714285714284
nets\toy\graph4+4	Greedy	0.44081632653061226
nets\toy\graph4+4	Asyn	0.44285714285714284
nets\toy\graph4+4	Louvain	0.44081632653061226
nets\toy\graph4+4	Spinglass	0.4408163265306122
nets\toy\graph4+4	Walktrap	0.44285714285714284
nets\toy\graph4+4	Girvan-Newman	0.44285714285714284

Table 5: "graph4+4" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
graph4+4.clu	graph4+4_asyn.clu	0.0000	1.0000	1.0000
graph4+4.clu	graph4+4_girvan-newman.clu	0.0000	1.0000	1.0000
graph4+4.clu	graph4+4_greedy.clu	0.3538	0.5856	0.4615
graph4+4.clu	graph4+4_louvain.clu	0.3538	0.5856	0.4615
graph4+4.clu	graph4+4_spinglass.clu	0.3538	0.5856	0.4615
graph4+4.clu	graph4+4_walktrap.clu	0.0000	1.0000	1.0000

Table 6: "graph4+4" network metrics

Network name	Method	Modularity
nets\toy\grid-p-6x6	Greedy	0.4166666666666667
nets\toy\grid-p-6x6	Asyn	0.3055555555555555
nets\toy\grid-p-6x6	Louvain	0.40586419753086417
nets\toy\grid-p-6x6	Spinglass	0.40740740740740733
nets\toy\grid-p-6x6	Walktrap	0.39506172839506176
nets\toy\grid-p-6x6	Girvan-Newman	0.41666666666666663

Table 7: "grid-p-6x6" network modularities

2.1.5 "star" network

In the table 8 we report the modularity values of the network partitions, while in the table 9 we report the comparison measures between our partitions and the reference one(s).
In figure 5 the 6 partitions obtained after applying the algorithms to the "star" network.

Network name	Method	Modularity
nets\toy\star	Reference	0.0
nets\toy\star	Asyn	0.0
nets\toy\star	Louvain	0.0
nets\toy\star	Spinglass	0.0
nets\toy\star	Walktrap	0.0
nets\toy\star	Girvan-Newman	0.0

Table 8: "star" network modularities

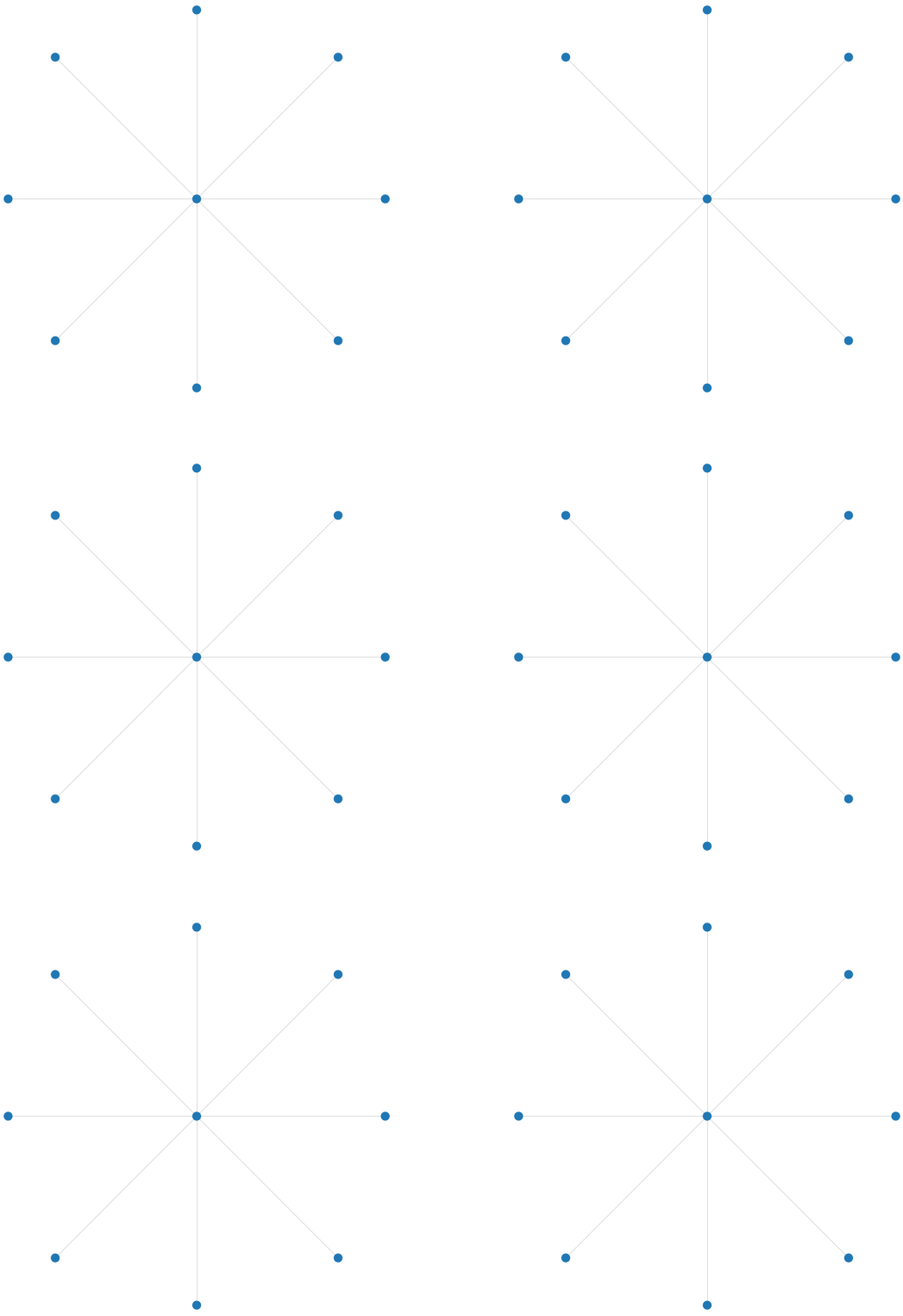


Figure 5: “star” network partitions

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
star.clu	star_asyn.clu	0.0000	1.0000	1.0000
star.clu	star_girvan-newman.clu	0.0000	1.0000	1.0000
star.clu	star_greedy.clu	0.0000	1.0000	1.0000
star.clu	star_louvain.clu	0.0000	1.0000	1.0000
star.clu	star_spinglass.clu	0.0000	1.0000	1.0000
star.clu	star_walktrap.clu	0.0000	1.0000	1.0000

Table 9: "star" network metrics

2.2 Networks in the "model" folder

2.2.1 "256_4_4_2_15_18_p" network

In the table 10 we report the modularity values of the network partitions, while in the table 11 we report the comparison measures between our partitions and the reference one(s).
In figure 6 the 6 partitions obtained after applying the algorithms to the "256_4_4_2_15_18_p" network.

Network name	Method	Modularity
nets\model\256_4_4_2_15_18_p	Reference	0.7818042125081899
nets\model\256_4_4_2_15_18_p	Greedy	0.7603253172067089
nets\model\256_4_4_2_15_18_p	Asyn	0.7818042125081899
nets\model\256_4_4_2_15_18_p	Louvain	0.7818042125081899
nets\model\256_4_4_2_15_18_p	Spinglass	0.7818042125081898
nets\model\256_4_4_2_15_18_p	Walktrap	0.7818042125081897
nets\model\256_4_4_2_15_18_p	Girvan-Newman	0.7818042125081897

Table 10: "256_4_4_2_15_18_p" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_asyn.clu	0.0000	1.0000	1.0000
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_girvan-newman.clu	0.0000	1.0000	1.0000
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_greedy.clu	0.1206	0.8637	0.4782
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_louvain.clu	0.0000	1.0000	1.0000
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_spinglass.clu	0.0000	1.0000	1.0000
256_4_4_2_15_18_p.clu	256_4_4_2_15_18_p_walktrap.clu	0.0000	1.0000	1.0000

Table 11: "256_4_4_2_15_18_p" network metrics

2.2.2 "256_4_4_4_13_18_p" network

In the table 12 we report the modularity values of the network partitions, while in the table 13 we report the comparison measures between our partitions and the reference one(s).
In figure 7 the 6 partitions obtained after applying the algorithms to the "256_4_4_4_13_18_p" network.

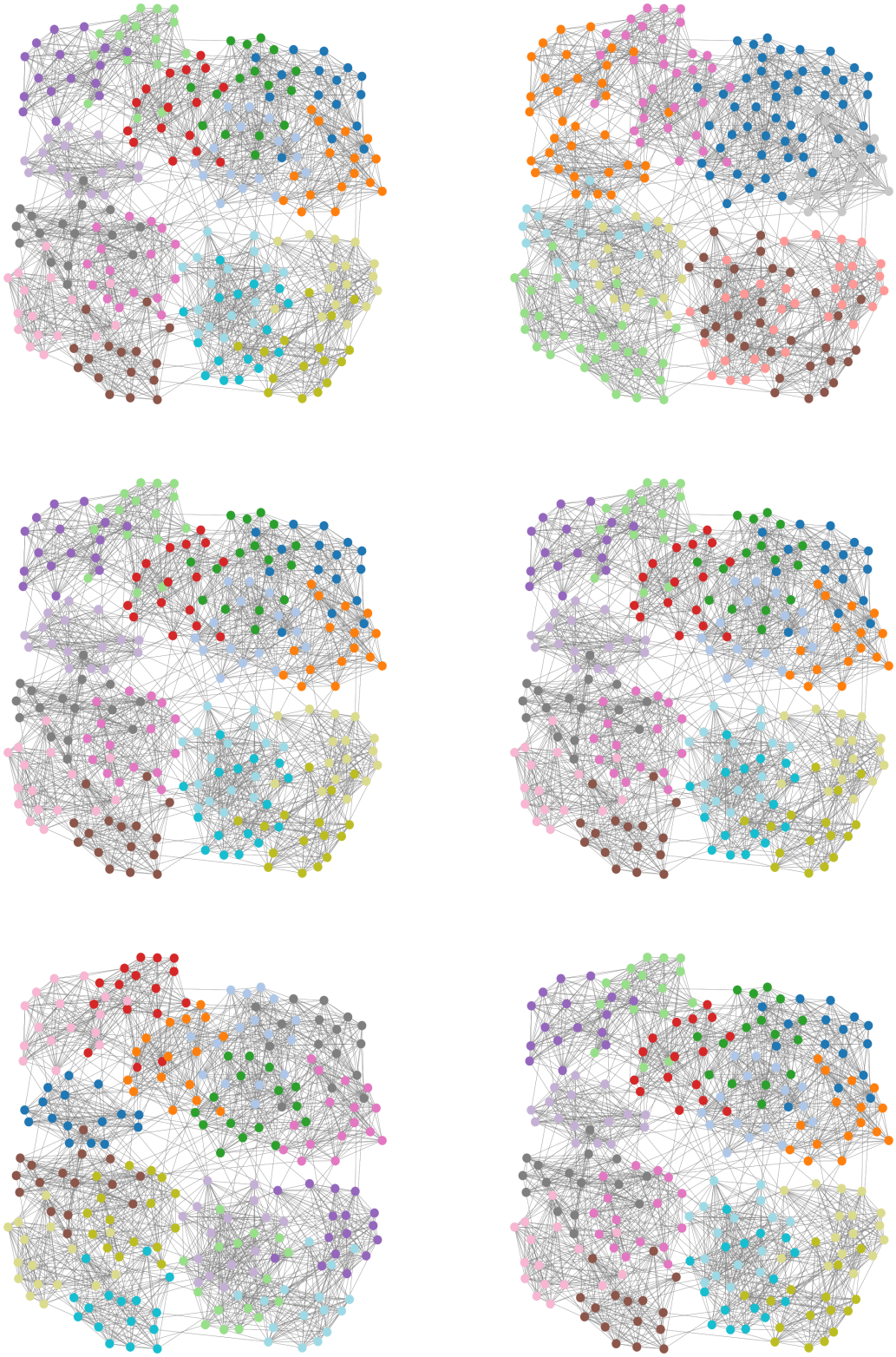


Figure 6: “256_4_4_2_15_18_p” network partitions

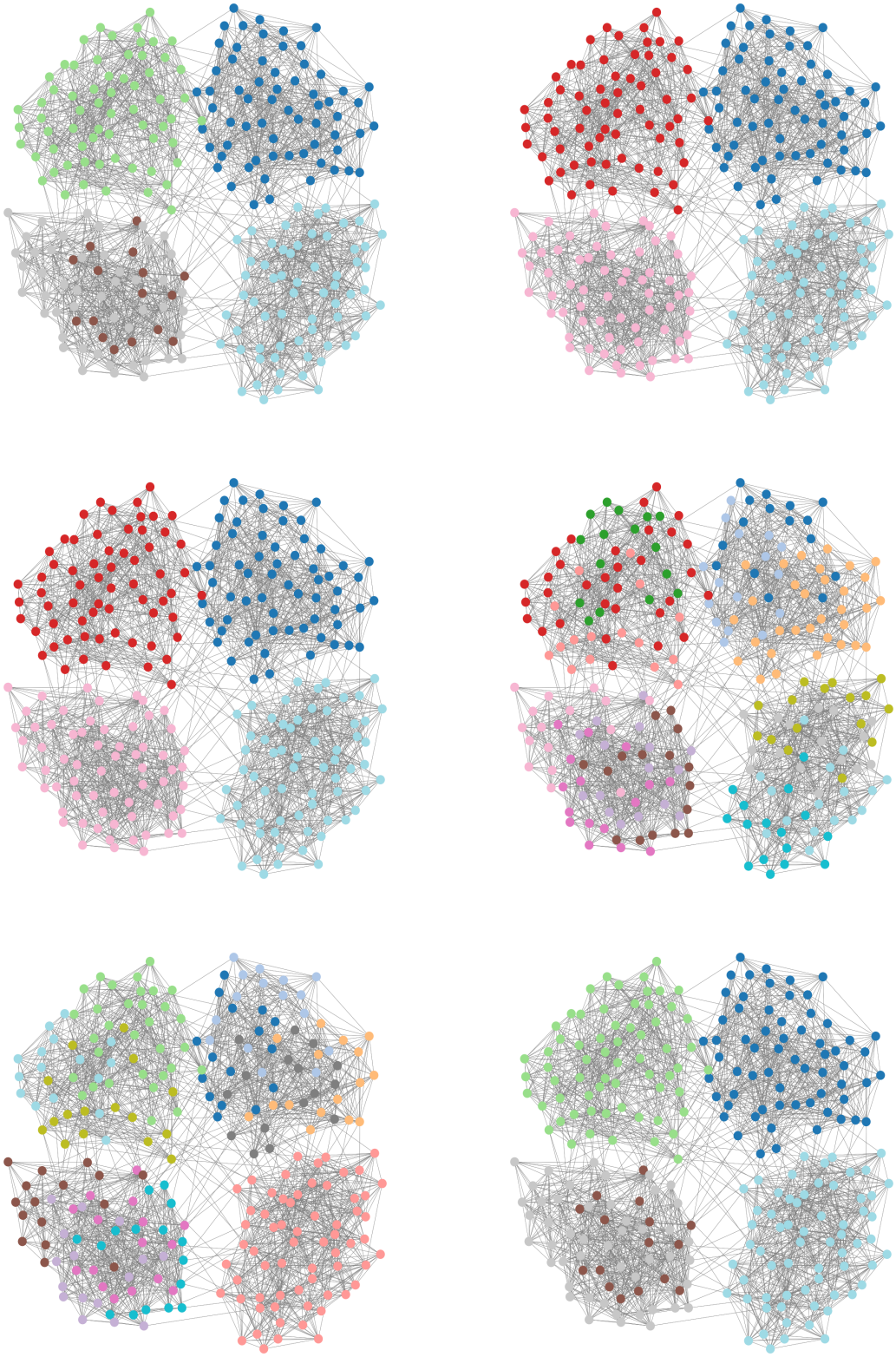


Figure 7: “256_4_4_4_13_18_p” network partitions

Network name	Method	Modularity
nets\model\256_4_4_4_13_18_p	Reference	0.6967733384846297
nets\model\256_4_4_4_13_18_p	Greedy	0.6967733384846297
nets\model\256_4_4_4_13_18_p	Asyn	0.6686714026050247
nets\model\256_4_4_4_13_18_p	Louvain	0.6967733384846297
nets\model\256_4_4_4_13_18_p	Spinglass	0.680847867550636
nets\model\256_4_4_4_13_18_p	Walktrap	0.6974188902601712
nets\model\256_4_4_4_13_18_p	Girvan-Newman	0.6974188902601712

Table 12: "256_4_4_4_13_18_p" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_asyn.clu	0.2188	0.6957	0.3016
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_girvan-newman.clu	0.0254	0.9517	0.9048
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_greedy.clu	0.0000	1.0000	1.0000
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_louvain.clu	0.0000	1.0000	1.0000
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_spinglass.clu	0.1719	0.7442	0.4603
256_4_4_4_13_18_p.clu	256_4_4_4_13_18_p_walktrap.clu	0.0254	0.9517	0.9048

Table 13: "256_4_4_4_13_18_p" network partitions

2.2.3 "rb125" network

In the table 14 we report the modularity values of the network partitions, while in the table 15 we report the comparison measures between our partitions and the reference one(s).
In figure 8 the 6 partitions obtained after applying the algorithms to the "rb125" network.

Network name	Method	Modularity
nets\model\rb125	Reference1	0.60922656439419
nets\model\rb125	Reference2	0.5727258700875049
nets\model\rb125	Reference2	0.5677444951398535
nets\model\rb125	Greedy	0.6168970001542905
nets\model\rb125	Asyn	0.5688906522074546
nets\model\rb125	Louvain	0.6168970001542904
nets\model\rb125	Spinglass	0.599616478211995
nets\model\rb125	Walktrap	0.6121360400273315
nets\model\rb125	Girvan-Newman	0.6107694681390378

Table 14: "rb125" network network modularities

2.3 Networks in the "real" folder

2.3.1 "airports_UW" network

In the table 16 we report the modularity values of the network partitions.
In figure 9 the 6 partitions obtained after applying the algorithms to the "airports_UW" network.

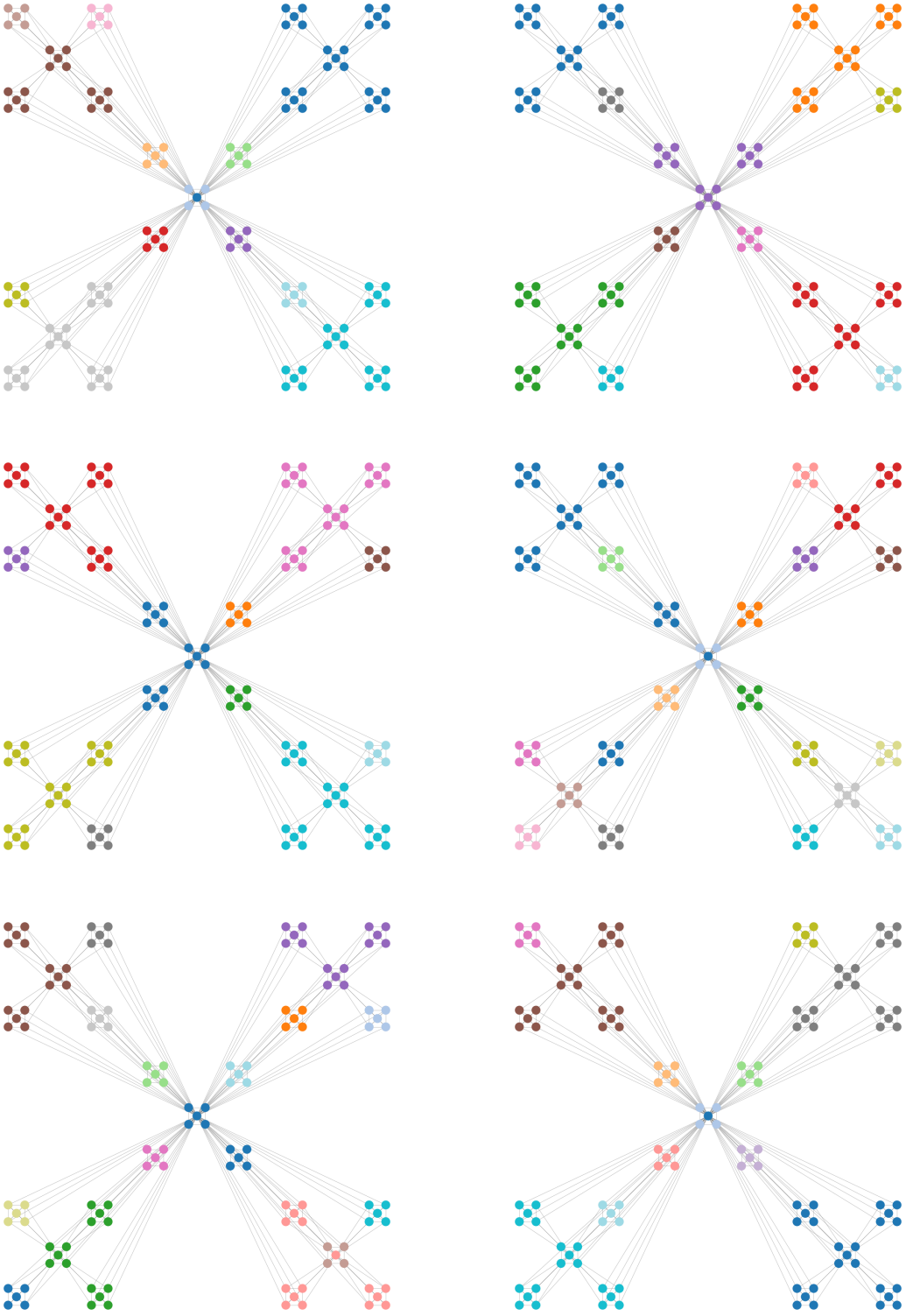


Figure 8: “rb125” network partitions

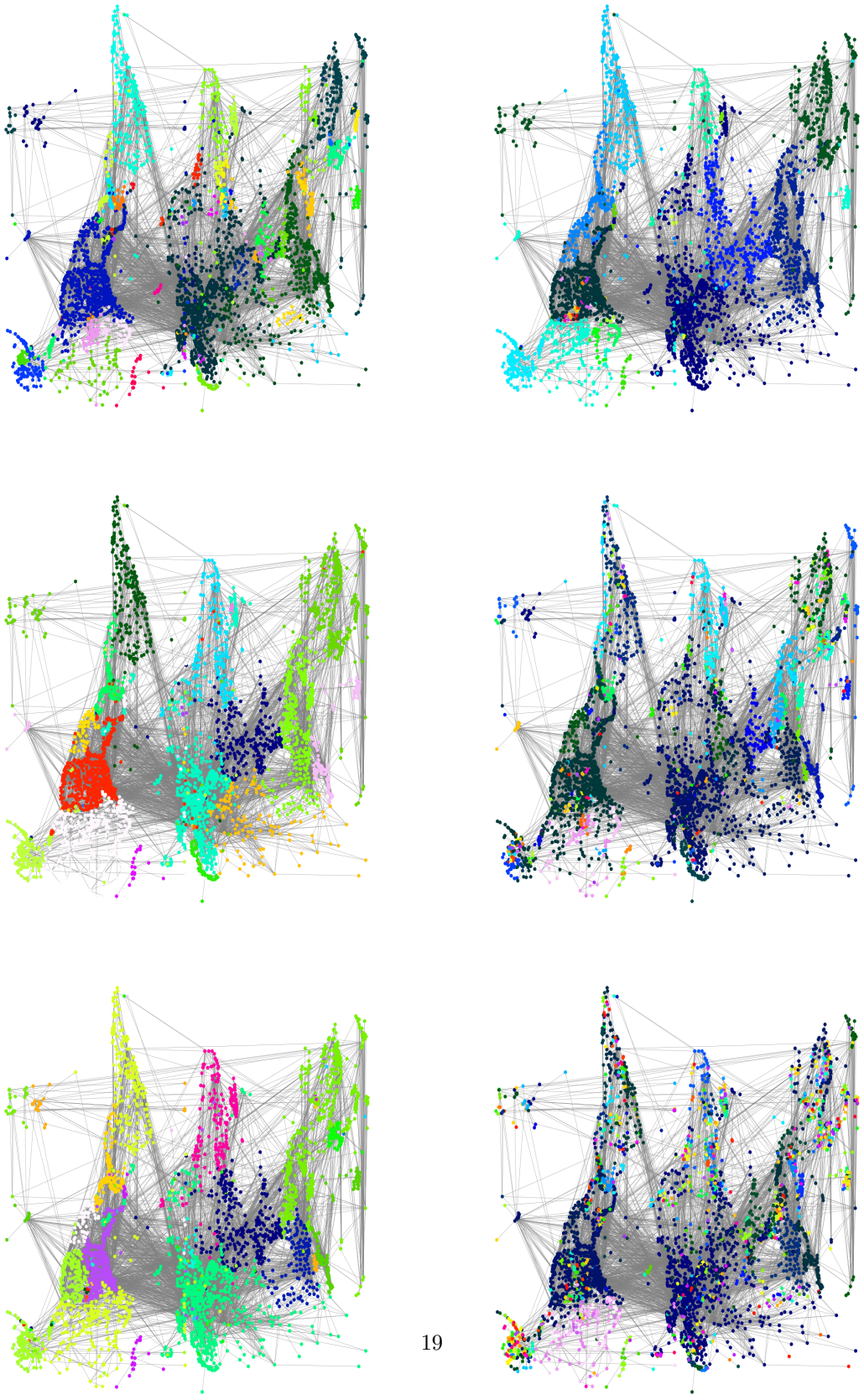


Figure 9: “airports_UW” network partitions

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
rb125-1.clu	rb125_asyn.clu	0.3215	0.6412	0.2434
rb125-1.clu	rb125_girvan-newman.clu	0.1587	0.8044	0.5711
rb125-1.clu	rb125_greedy.clu	0.1223	0.8450	0.6167
rb125-1.clu	rb125_louvain.clu	0.1223	0.8450	0.6167
rb125-1.clu	rb125_spinglass.clu	0.2264	0.7372	0.3781
rb125-1.clu	rb125_walktrap.clu	0.1400	0.8234	0.6203
rb125-2.clu	rb125_asyn.clu	0.1120	0.9089	0.3618
rb125-2.clu	rb125_girvan-newman.clu	0.1970	0.8279	0.2733
rb125-2.clu	rb125_greedy.clu	0.2111	0.8119	0.2703
rb125-2.clu	rb125_louvain.clu	0.2111	0.8119	0.2703
rb125-2.clu	rb125_spinglass.clu	0.1469	0.8771	0.3844
rb125-2.clu	rb125_walktrap.clu	0.2157	0.8085	0.2523
rb125-3.clu	rb125_asyn.clu	0.1079	0.9126	0.3639
rb125-3.clu	rb125_girvan-newman.clu	0.1929	0.8321	0.2746
rb125-3.clu	rb125_greedy.clu	0.2152	0.8089	0.2659
rb125-3.clu	rb125_louvain.clu	0.2152	0.8089	0.2659
rb125-3.clu	rb125_spinglass.clu	0.1510	0.8740	0.3781
rb125-3.clu	rb125_walktrap.clu	0.2115	0.8128	0.2533

Table 15: "rb125" network metrics

Network name	Method	Modularity
nets\real\airports_UW	Greedy	0.6392502296619001
nets\real\airports_UW	Asyn	0.6058792680322485
nets\real\airports_UW	Louvain	0.645839045568263
nets\real\airports_UW	Spinglass	0.6363445083252361
nets\real\airports_UW	Walktrap	0.6051379187679539
nets\real\airports_UW	Girvan-Newman	0.5958993304057855

Table 16: "airports_UW" network modularities

2.3.2 "cat_cortex_sim" network

In the table 17 we report the modularity values of the network partitions, while in the table 18 we report the comparison measures between our partitions and the reference one(s).
In figure 10 the 6 partitions obtained after applying the algorithms to the "cat_cortex_sim" network.

2.3.3 "dolphins" network

In the table 19 we report the modularity values of the network partitions, while in the table 20 we report the comparison measures between our partitions and the reference one(s).
In figure 11 the 6 partitions obtained after applying the algorithms to the "dolphins" network.

2.3.4 "football" network

In the table 21 we report the modularity values of the network partitions, while in the table 22 we report the comparison measures between our partitions and the reference one(s).

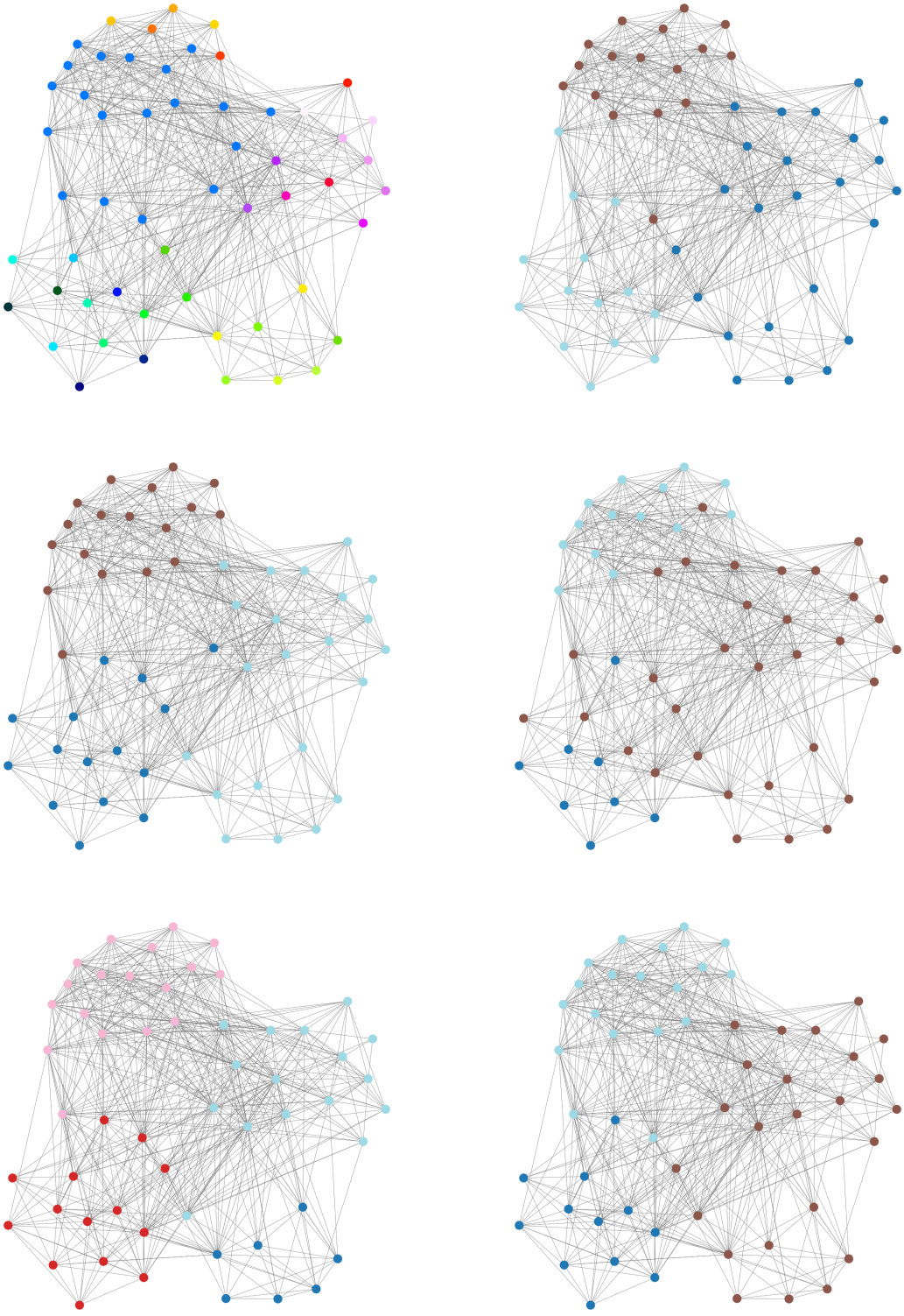


Figure 10: “cat_cortex_sim” network partitions

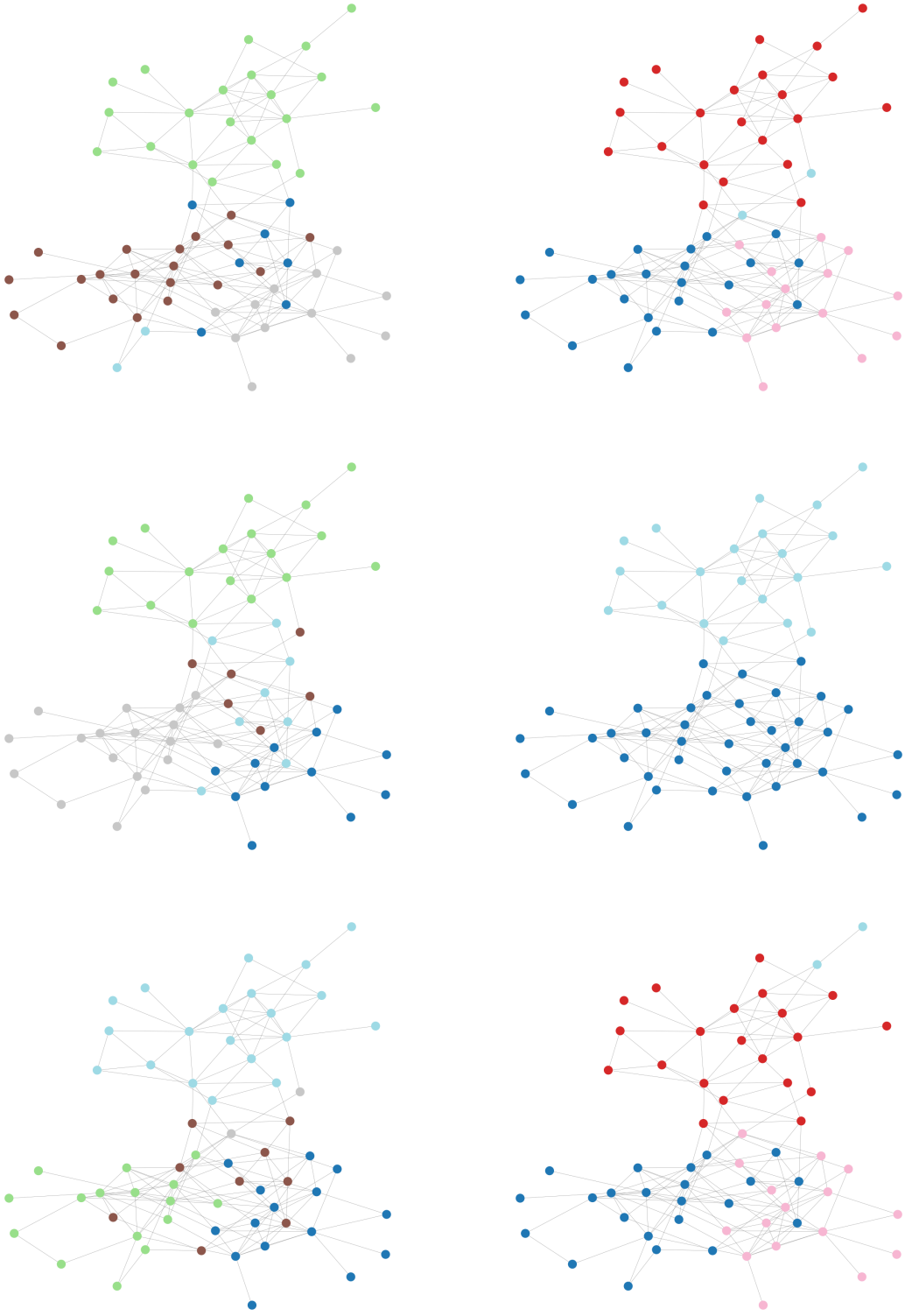


Figure 11: “dolphins” network partitions

Network name	Method	Modularity
nets\real\cat_cortex_sim	Reference	0.3624034600799502
nets\real\cat_cortex_sim	Greedy	0.36615284033327616
nets\real\cat_cortex_sim	Asyn	0.2911090108262251
nets\real\cat_cortex_sim	Louvain	0.36547582372290366
nets\real\cat_cortex_sim	Spinglass	0.3683153903021668
nets\real\cat_cortex_sim	Walktrap	0.3709244831417289
nets\real\cat_cortex_sim	Girvan-Newman	0.07093456636784069

Table 17: "cat_cortex_sim" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
cat_cortex_sim.clu	cat_cortex_sim_asyn.clu	0.2732	0.5185	0.3665
cat_cortex_sim.clu	cat_cortex_sim_girvan-newman.clu	0.5689	0.4733	0.1569
cat_cortex_sim.clu	cat_cortex_sim_greedy.clu	0.1882	0.6873	0.5804
cat_cortex_sim.clu	cat_cortex_sim_louvain.clu	0.1636	0.7297	0.6372
cat_cortex_sim.clu	cat_cortex_sim_spinglass.clu	0.0681	0.8979	0.8527
cat_cortex_sim.clu	cat_cortex_sim_walktrap.clu	0.1761	0.7060	0.5841

Table 18: "cat_cortex_sim" network metrics

Network name	Method	Modularity
nets\real\dolphins	Reference	0.37348206162730896
nets\real\dolphins	Greedy	0.4954906847039278
nets\real\dolphins	Asyn	0.3787033740753925
nets\real\dolphins	Louvain	0.5188283691309679
nets\real\dolphins	Spinglass	0.5258692298564138
nets\real\dolphins	Walktrap	0.48884537795182154
nets\real\dolphins	Girvan-Newman	0.5193821446936435

Table 19: "dolphins" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
dolphins.clu	dolphins_asyn.clu	0.0342	0.8888	0.9430
dolphins.clu	dolphins_girvan-newman.clu	0.2199	0.5542	0.4370
dolphins.clu	dolphins_greedy.clu	0.1883	0.5727	0.5041
dolphins.clu	dolphins_louvain.clu	0.2526	0.5162	0.3791
dolphins.clu	dolphins_spinglass.clu	0.1987	0.6053	0.4301
dolphins.clu	dolphins_walktrap.clu	0.2045	0.5372	0.4796

Table 20: "dolphins" network metrics

In figure 12 the 6 partitions obtained after applying the algorithms to the "football" network.

2.3.5 "zachary_unwh" network

In the table 23 we report the modularity values of the network partitions, and in the table 24 the comparison measures between our partitions and the reference one(s).
In figure 13 the 6 partitions obtained after applying the algorithms to the network.

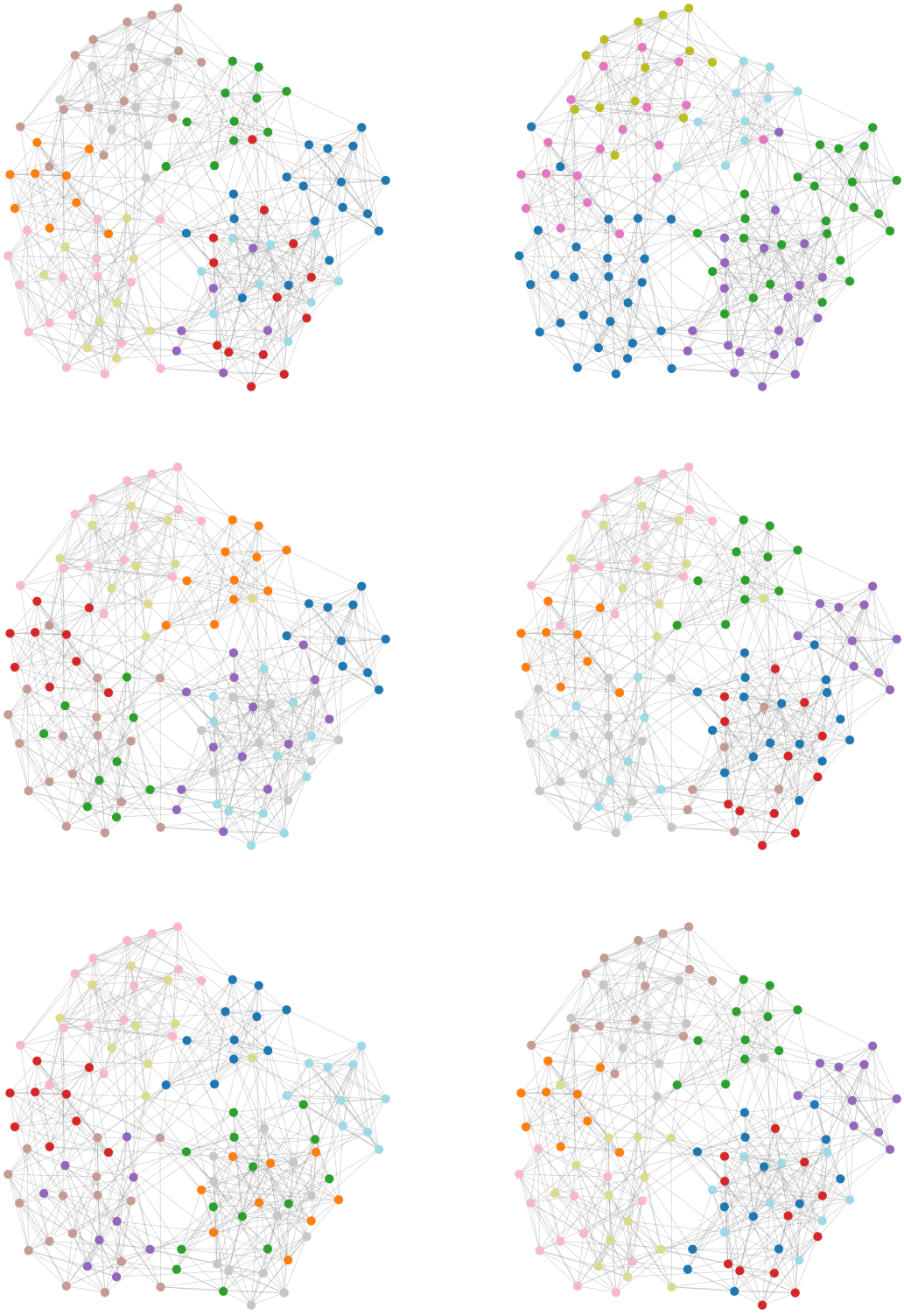


Figure 12: “football” network partitions

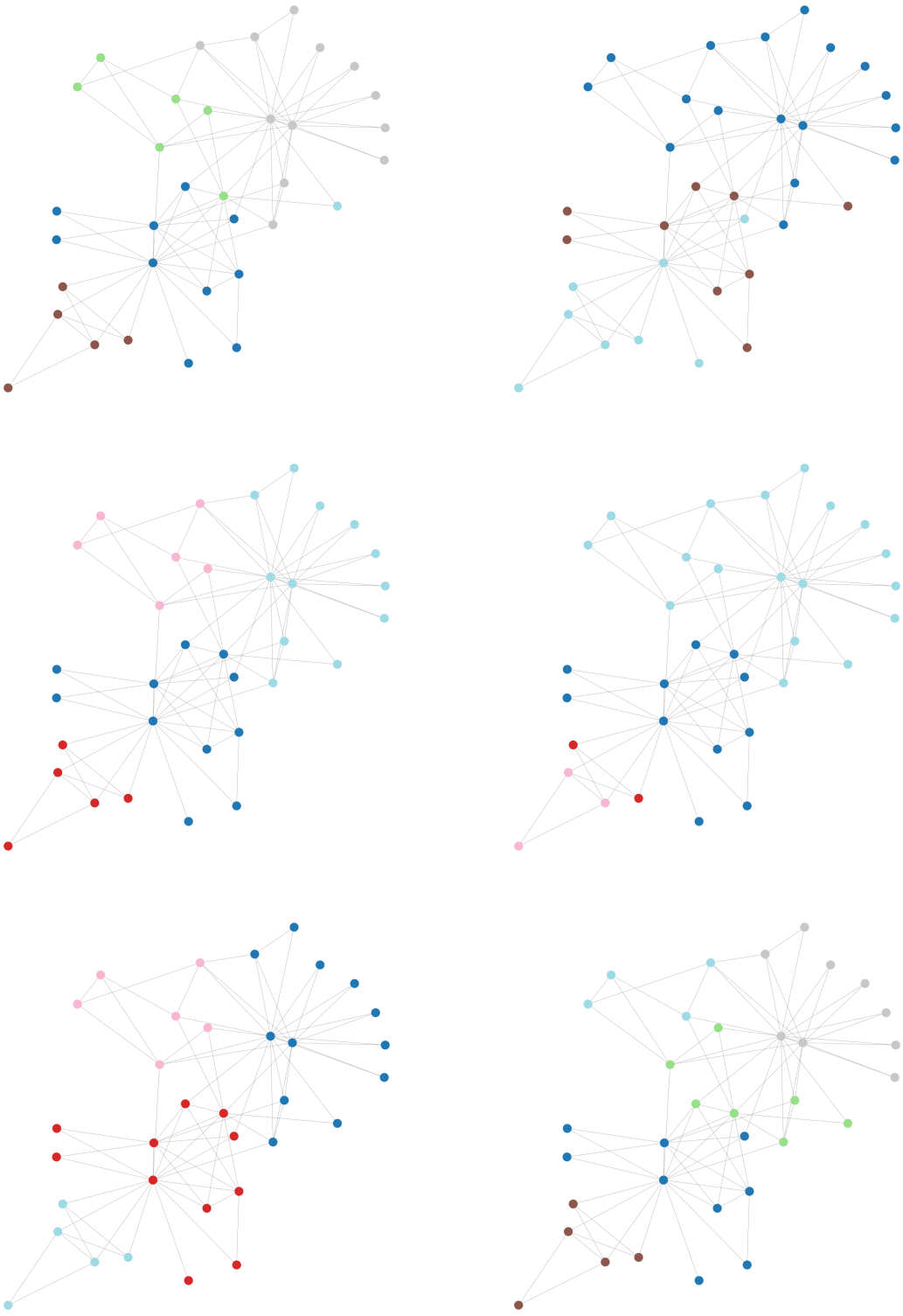


Figure 13: “zachary_unwh” network partitions

Network name	Method	Modularity
nets\real\football	Reference	0.553973318714423
nets\real\football	Greedy	0.5564043335134086
nets\real\football	Asyn	0.597786139889134
nets\real\football	Louvain	0.6043460210927457
nets\real\football	Spinglass	0.6045695626834571
nets\real\football	Walktrap	0.602914290428428
nets\real\football	Girvan-Newman	0.5996290274077959

Table 21: "football" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
football.clu	football_asyn.clu	0.1135	0.8858	0.6766
football.clu	football_girvan-newman.clu	0.1202	0.8789	0.6639
football.clu	football_greedy.clu	0.2528	0.7141	0.3709
football.clu	football_louvain.clu	0.1148	0.8850	0.6959
football.clu	football_spinglass.clu	0.1095	0.8903	0.7004
football.clu	football_walktrap.clu	0.1127	0.8874	0.7115

Table 22: "football" network metrics

Network name	Method	Modularity
nets\real\zachary_unwh	Reference	0.37146614069691
nets\real\zachary_unwh	Greedy	0.3806706114398422
nets\real\zachary_unwh	Asyn	0.37836949375410917
nets\real\zachary_unwh	Louvain	0.41978961209730437
nets\real\zachary_unwh	Spinglass	0.4197896120973044
nets\real\zachary_unwh	Walktrap	0.3532215647600263
nets\real\zachary_unwh	Girvan-Newman	0.40129848783694944

Table 23: "zachary_unwh" network modularities

Reference Name	Compared Name	Mutual info	Variance	Jaccard Index
zachary_unwh.clu	zachary_unwh_asyn.clu	0.1109	0.7795	0.7766
zachary_unwh.clu	zachary_unwh_girvan-newman.clu	0.2515	0.5798	0.4712
zachary_unwh.clu	zachary_unwh_greedy.clu	0.1509	0.6925	0.6833
zachary_unwh.clu	zachary_unwh_louvain.clu	0.1784	0.6873	0.5348
zachary_unwh.clu	zachary_unwh_spinglass.clu	0.1784	0.6873	0.5348
zachary_unwh.clu	zachary_unwh_walktrap.clu	0.3169	0.5042	0.3498

Table 24: "zachary_unwh" network metrics

3 Conclusions

By simply looking at the modularity values of the obtained partitions, we can conclude that the *Spinglass* community detection algorithm is the one that, on average, performs very well on all the analyzed networks, since the partitions obtained through it achieve almost always one of the highest modularity value. However, by testing the execution time on our python script and by

using the network "airports_UW", we noticed that the algorithm is pretty slow compared to the others (around 206 seconds on our reference network, only *Girvan-Newman* took more execution time).

What we can also see is that *Asynchronous label propagation* is the algorithm with the worst performances in terms of modularity, but it's execution is very fast on the "airports_UW" network.

For what regards the other algorithms, they provide very similar performances on all the analyzed networks, so we cannot conclude much about them.

We would like to remark that, during our analysis, we noticed (by using the "time" module of python which allows to calculate the computational time of a function) that the *Girvan-Newman* algorithm was pretty slow compared to the others (even if it's implemented in "igraph" which relies on C).

While *Louvain* provides good scalability properties, being able to achieve reasonable computation time for all the considered networks, indeed it has a complexity of $O(n \log n)$.