Complex Social Networks - Lab 5

Afonso Ramos
Martin Dans

November 25, 2019

1 Introduction

The objective of this fifth lab project is to run and compare different community finding algorithms.

In the *igraph* package it is already possible to find many of these algorithms implemented, which we will take advantage of and make use of.

After gathering all the communities, we are tasked to calculate all of its metrics, such as **Triangle Partition Ratio (TPT)**, **modularity**, **conductance** and **expansion**. Afterwards we make use of a weighted average to calculate global values for the network.

As defined in the guide, we have used a total of 9 *igraph*'s algorithms to find communities in our data.

Before applying the different community algorithms, we have taken a brief look at each of their inner workings from the lecture slides and other sources. As such, these were the applied algorithms:

- In **edge.betweeness.community**, edges are removed in the decreasing order of their edge, recomputing every time an edge is removed.
- **fastgreedy.community** tries to optimize a quality function called modularity in a greedy manner. Modularity, as we will see in the next section, is the difference between the number of edges in community C and the expected number of expected edges $E[mc]$, *i.e.*, it tried to find zones in the network with greater density of edges.
- **label.propagation.community** tries to assign k labels the nodes of the network. The method then proceeds iteratively and re-assigns labels to nodes in a way that each node takes the most frequent label of its neighbors in a synchronous manner.
- **leading.eigenvector.community** also tries to optimize the modularity function through the evaluation of the eigenvector of the modularity matrix.
- In **multilevel.community** nodes are moved between communities such that each node makes a local decision that maximizes its own contribution to the modularity score.
- **optimal.community** which Works by maximizing the modularity measure over all possible partitions.
- **spinglass.community** simulates the physical *spin* property and at let the interactions between the nodes (particles) decide how to define a community.

- **walktrap.community** works based on random walks, concluding that some walks are more likely to stay within the same community because there are only a few edges that lead outside a given community. Walktrap runs short random walks and uses the results of these random walks to merge separate communities in a bottom-up manner like *fastgreedy.community* algorithm.

- **infomap.community** based on information theoretic principles, it tries to build a grouping which provides the shortest description length for a random walk on the graph, where the description length is measured by the expected number of bits per vertex required to encode the path of a random walk.

2 Results

We tested all the proposed algorithms with 4 different networks.

- Zachary: Famous small network that represents the relationship inside a Karate club. It's fast to compute community detection algorithms and provide a simple real case.

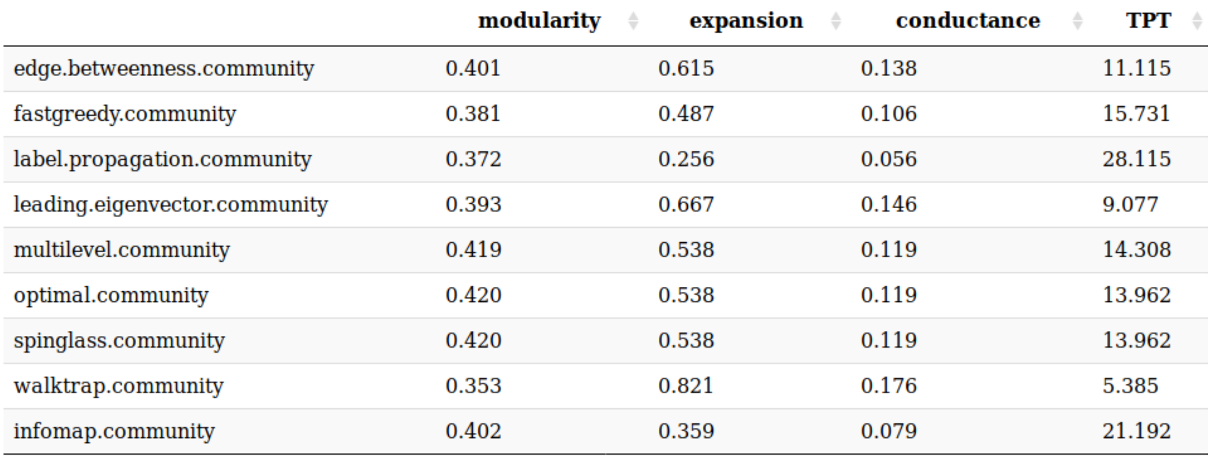
- Meredith: We wanted to try with a degenerated network. It's a 4-regular undirected graph with 70 vertices and 140 edges and it provides a counterexample to the conjecture that every 4-regular 4-vertex-connected graph is Hamiltonian.

- Generated: We also tried with a generated graph, were we put some clusters beforehand and then add random edges. Also if we created communities, we created them with different number of connections, from almost a full graph to a simple ring, to see when the algorithms differ from the predefined communities.

- Citations: Big graph representing citations in papers. The majority of the algorithms didn't finish the execution.

For each graph and algorithm we extracted the measures: Modularity, clousure, expansion and triangle partition ratio. In order to compute the metrics we used the fraction of nodes as the weight for the weighted average of each community.

2.1 Zachary



	modularity	expansion	conductance	TPT
edge.betweenness.community	0.401	0.615	0.138	11.115
fastgreedy.community	0.381	0.487	0.106	15.731
label.propagation.community	0.372	0.256	0.056	28.115
leading.eigenvector.community	0.393	0.667	0.146	9.077
multilevel.community	0.419	0.538	0.119	14.308
optimal.community	0.420	0.538	0.119	13.962
spinglass.community	0.420	0.538	0.119	13.962
walktrap.community	0.353	0.821	0.176	5.385
infomap.community	0.402	0.359	0.079	21.192

Figure 1: Measures for the Zachary graph

The best assignment seems to be the one generated by label propagation algorithm (Figure 2), it scores the best in expansion, conductance and TPT. Despite that, it's one with the lowest modularity.

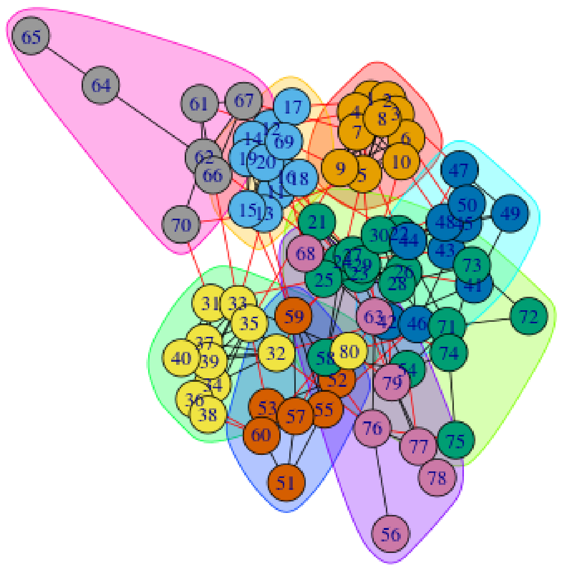


Figure 2: Label propagation Zachary

2.2 Meredith

It's interesting to see the plot of the Meredith graph to understand why it is degenerated and too see clearly what clusters should be selected (Figure 3).

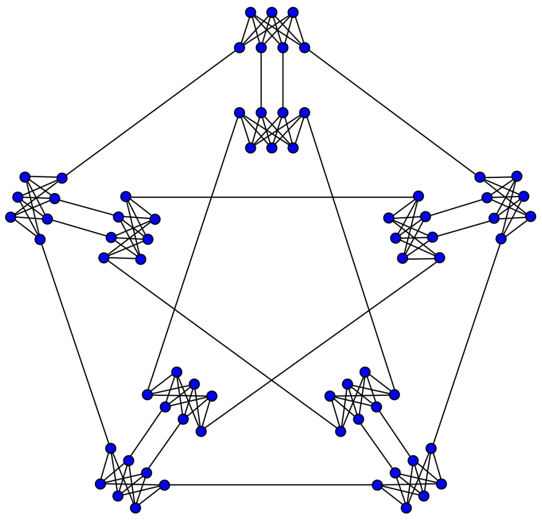
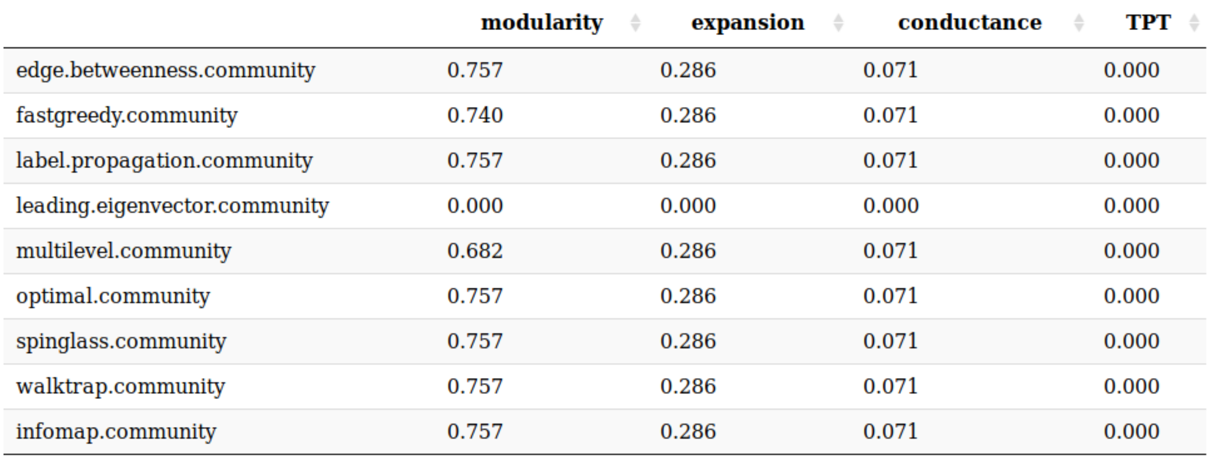


Figure 3: Meredith graph



	modularity	expansion	conductance	TPT
edge.betweenness.community	0.757	0.286	0.071	0.000
fastgreedy.community	0.740	0.286	0.071	0.000
label.propagation.community	0.757	0.286	0.071	0.000
leading.eigenvector.community	0.000	0.000	0.000	0.000
multilevel.community	0.682	0.286	0.071	0.000
optimal.community	0.757	0.286	0.071	0.000
spinglass.community	0.757	0.286	0.071	0.000
walktrap.community	0.757	0.286	0.071	0.000
infomap.community	0.757	0.286	0.071	0.000

Figure 4: Measures for the Meredith graph

Almost of all of the algorithms perform equally. We were expecting that output since the graph is strange and probably will never happen in a real graph. The interesting part of the results in this graph is not what performed better, instead that TPT is 0 for all the algorithms and why leading eigenvector failed to cluster it.

We can check the plot (Figure 5) to see what is the result produced by the leading eigenvector algorithm.

It only found one cluster which is all the graph! That was expected since it was the only way that expansion and conductance to be 0 in a connected graph, since at least one edge will have to have vertices in different clusters. The output of the others algorithms (Figure 6) match very well with the human identifiable communities.

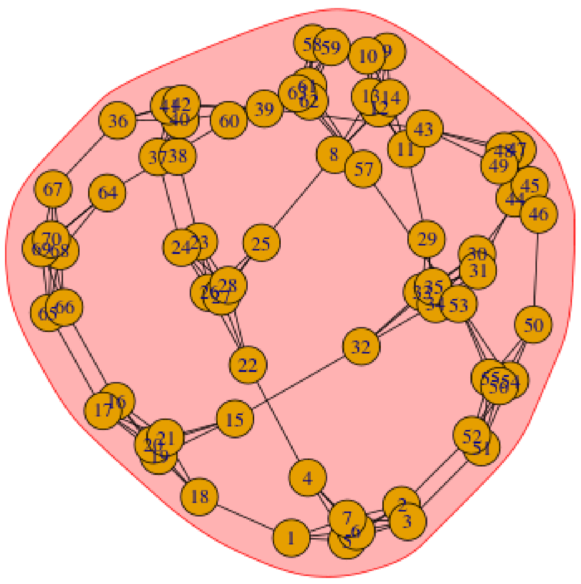


Figure 5: Leading eigenvector Meredith

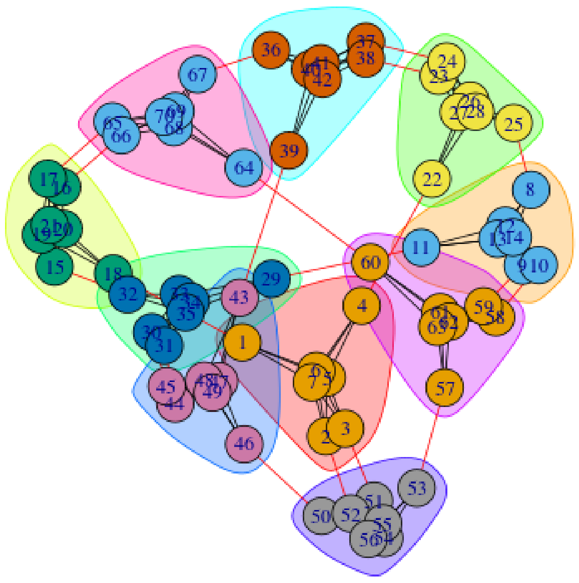
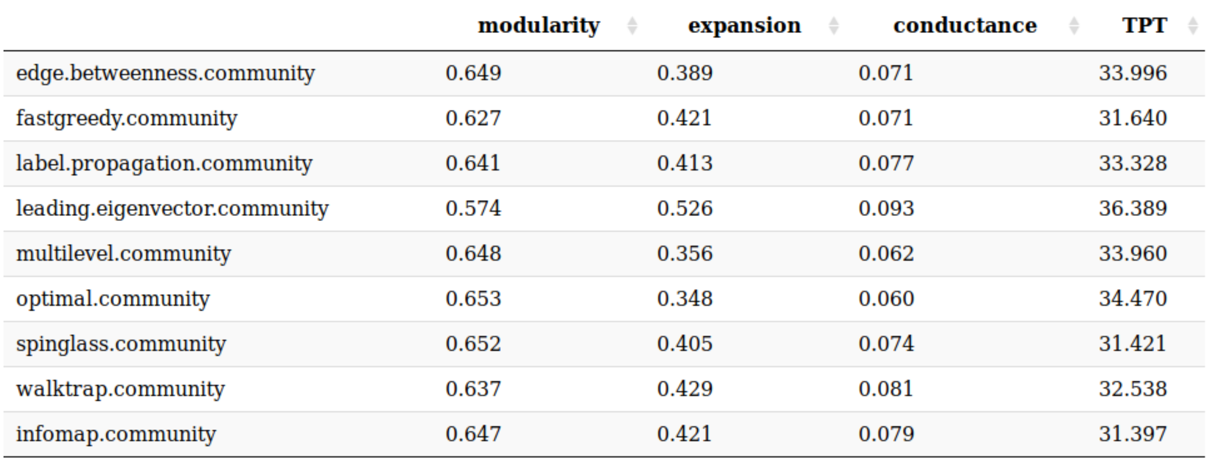


Figure 6: Edge betweenness Meredith

2.3 Generated

In order to test how the algorithms work and how well they find the communities we generated a graph (Figure 7) that have some communities defined by us. The communities generated are of size 10 and there are 8 communities. Nodes from 1 to 10 belong to the first one, from 11 to 20 to the second, etc. Lower indices have stronger communities (more interedges between them than higher indices.



	modularity	expansion	conductance	TPT
edge.betweenness.community	0.649	0.389	0.071	33.996
fastgreedy.community	0.627	0.421	0.071	31.640
label.propagation.community	0.641	0.413	0.077	33.328
leading.eigenvector.community	0.574	0.526	0.093	36.389
multilevel.community	0.648	0.356	0.062	33.960
optimal.community	0.653	0.348	0.060	34.470
spinglass.community	0.652	0.405	0.074	31.421
walktrap.community	0.637	0.429	0.081	32.538
infomap.community	0.647	0.421	0.079	31.397

Figure 7: Measures for the Generated graph

Interestingly not all the algorithms performed equally, neither output the same communities, in fact, when you check the plots generated, you can visually see very significant differences. We realized that for this graph and this measures it's difficult to say what is the better algorithm for some instances. For this execution particularly, we think that the best one is produced by optimal community algorithm (Figure 8).

We can see how the lower indices are all grouped! The 10 first nodes are clearly separated into a community, following with the next 10. We can continue until some point the community is barely there. Not all the communities where detected though, the last ones are spliced and nodes are randomly attached to others. Of course that's because the noise that we added.

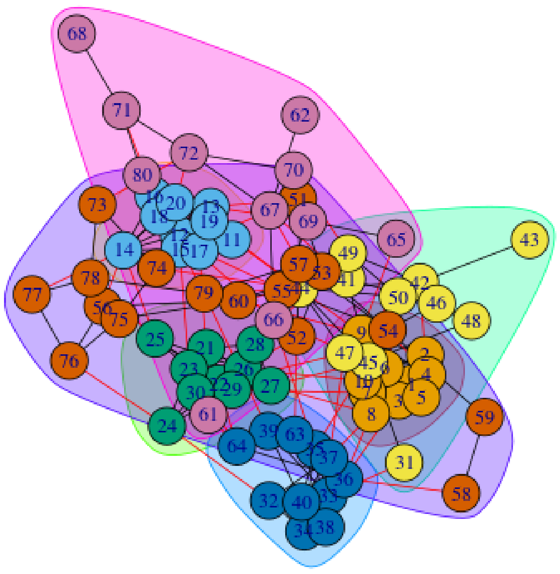


Figure 8: Optimal Meredith

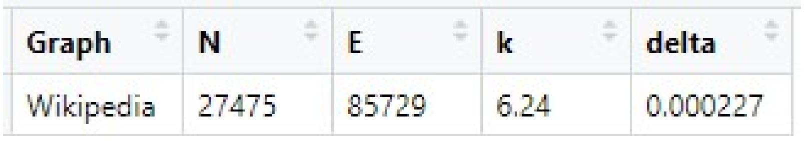
2.4 Citations

We couldn't process all the algorithms with the graph, but one that gave results very fast was the fastgreedy. There are no plots of the graph since we couldn't neither the results of the metrics because computing them it's expense itself, but we can see the sizes of the biggest communities found.

```
Communities size:  1083 3512 3120 3082 333 300 164 83 73 68 45...
```

2.5 Task 2

We are now going to proceed to detect the communities on the Wikipedia dataset provided in *'wikipedia.gml'*. We can clearly observe that this network is really large even though it is not that dense. It seems that as of the average, each article is connected to 6 other articles.



Graph	N	E	k	delta
Wikipedia	27475	85729	6.24	0.000227

Figure 9: Wikipedia summary table.

As we can clearly see, this is a very big community, and, due to its size, **edge betweeness** as a community detection algorithm is completely out of the question, being completely discarded for the performance needed. Looking at the tables in the chapters above, we can see that the fastest community detection algorithms are walktrap, multilevel, label, infomap and fastgreedy. Multilevel cannot be used because it needs the graph to be undirected and we are dealing, in fact, with a directed graph.

In order to perform the community detection, we opter to test it with Walktrap because it works based on random walks. Conluding that some walks are more likely to stay within the same community, since there are few edges that lead outside of one.

Furthermore, we have created a function that retrieves the total number of communities found, which is then used as a maximum number to generate, randomly, a graph of a community like the following:

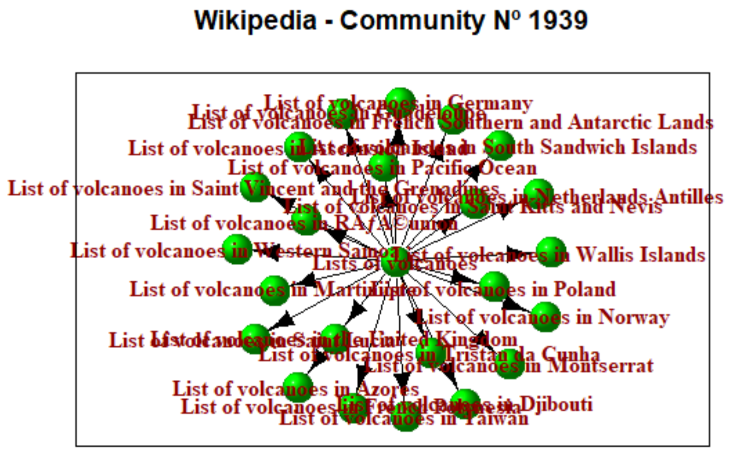

Wikipedia - Community Nº 1939
List of volcanoes in Germany
List of volcanoes in Guadeloupe
List of volcanoes in French Southern and Antarctic Lands
List of volcanoes in South Sandwich Islands
List of volcanoes in Pacific Ocean
List of volcanoes in Saint Vincent and the Grenadines
List of volcanoes in Netherlands Antilles
List of volcanoes in Saint Kitts and Nevis
List of volcanoes in RÃ©union
List of volcanoes in Western Samoa
Lists of volcanoes
List of volcanoes in Wallis Islands
List of volcanoes in Martinique
List of volcanoes in Poland
List of volcanoes in Norway
List of volcanoes in the United Kingdom
List of volcanoes in Tristan da Cunha
List of volcanoes in Montserrat
List of volcanoes in Azores
List of volcanoes in French Polynesia
List of volcanoes in Djibouti
List of volcanoes in Taiwan

Figure 10: Wikipedia summary table.

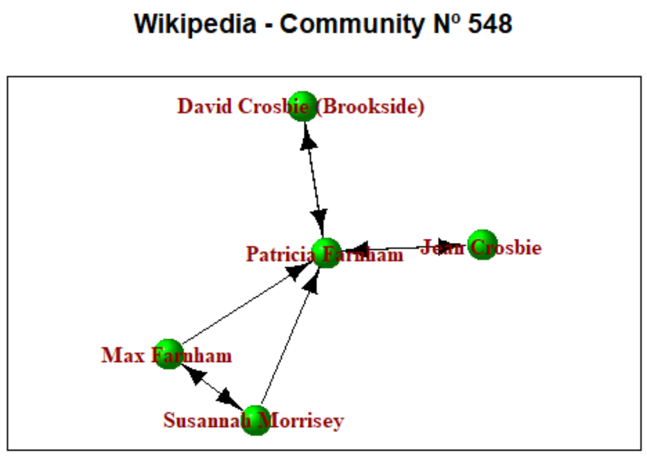


Figure 11: Wikipedia summary table.

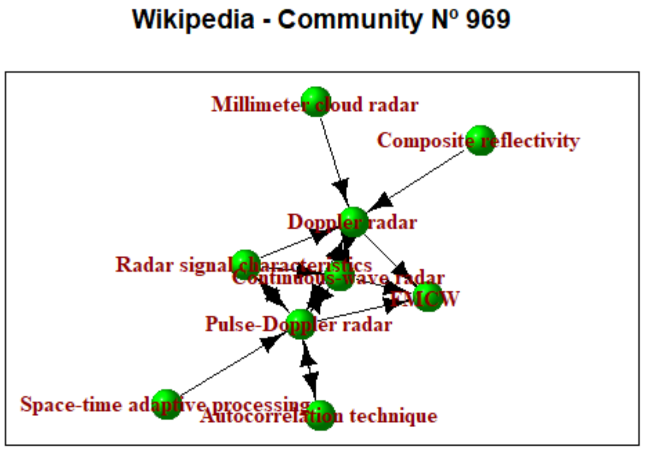

Wikipedia - Community Nº 969
Millimeter cloud radar
Composite reflectivity
Doppler radar
Radar signal characteristics
Continuous-wave radar
FMCW
Pulse-Doppler radar
Space-time adaptive processing
Autocorrelation technique

Figure 12: Wikipedia summary table.

The number of communities found is quite large, as is the whole network, and, as such, plotting the whole graph is unfeasible, as it would be too crowded.

Looking at the communities, we can see that the first one is very well-populated and the last two communities do not have as many nodes, however, this graphs were selected, as there as much more with just 1 to 3 nodes. And, from there we can assume that most communities found are articles totally unrelated to others. On the first we can see that there is a list that links to other lists, while on the second there is much more biderectional relations. While on the last one we can observe two main nodes that are the central topics.

3 Discussion

First of all, comment that we tested how well the communities were detected, we found that the algorithms correctly find the strongest communities that we set in our generated graph and that when the community gets weaker, the algorithms start to struggle to find them. We have seen that almost all the algorithms detected very well communities with at least 60% of the edges when having a 33% noise edges added. In our example until node 50 almost of the nodes where correctly put into their a priory set community.

The results above show us that we can't stick with only one measure to evaluate how good are the communities found and we have also seen that some algorithms are better than the others for some graphs and vice-versa.

We can't get one measure because we observed that the best algorithm given one measure was not the same if we pick another measure. For example, some algorithms have low TPT value but score very good in the other measures.

Even taking that into account, we showed that measure can fail themselves to show the quality of the communities found. We got 0 TPT for all the algorithms with the Meredith graph, since the graph doesn't have a single triangle, and we got 0 for all the metrics with the leading eigenvector algorithm when a priory 0 seems a perfect score for some of them we need to double check the solution to see that in fact it only found one community and that it's a very poor community finding.

Finally, add that on top of that there could be some others measures, as we have seen with the big citation graph, while almost all the algorithms take a lot of time to finish, the fastgreedy finds a solution in few seconds in a end-user laptop. While it didn't make it to the top in any of our tests, it was always giving good solutions and when time is a problem, it can provide solutions where other more accurate algorithms can't.

We can discard the idea of picking one algorithm and one measure and use it universally because we showed that even if normally both give acceptable results sometimes they can fail completely for some other.

As for the wikipedia communities found with Walktrap, the articles grouped by it seem to be related. Thus the algorithm has worked well, at least in our opinion and having seen some communities randomly.

4 Methods

For doing this project we wanted to find a graphs that had the following properties:

- Easy for human to find clusters
- Some metrics fail to evaluate them
- An algorithm fails to find any community

In order to do that, we looked into the famous graphs and when we found Meredith and saw it's results we thought it was perfect for our propose.

Another thing that we wanted to do was to generate our graph where actually the algorithms differ on finding the communities but that also we knew exactly what communities were there. An extra experiment we added later was making the communities weaker to see when the algorithms started to fail to find them. To accomplish that we did the following: Generate complete graphs and remove a percentages of the edges, each time we removed more edges, the values went from 10% to 80% edges removed. After that we joined all the graphs and added 33% more random edges to add some noise. With that we had noise enough to make algorithms differ in their result. We notice that strong communities were almost always detected not matter how much noise edges we added until very high values.