Comparison study on different heuristics and measures concerning the Louvain algorithm

Joes de Jonge (6228305) & Kieran van Gaalen (6437397)
June 2022

Abstract

TODO

1. Introduction

Most of the networks you can observe in the real world have certain communities, specific groups that share a lot of inner edges between the inner nodes. For example in a school there are a lot of friend groups. People inside a friend group have a lot of friends inside the friend group, and fewer friends outside of the friend group. With real world networks and with artificial ones, it is great practise to detect which communities there are. For example, with these conclusions you can determine where improvement for nodes and edges can be made by adding new nodes or edges if you see a bottle neck appear appearing. Add to that the value of being able to analyse a dataset of people for example, where you can actually use community detection for more targeted marketing.

The main issue with community detection is the amount of time it takes to calculate the communities. A good community detection algorithm will strike a good balance between delivering results quickly, but also making sure that the results are actually close to an optimal solution. The Louvain algorithm is a widely applied community detection algorithm. It gives proven good solutions and runs relatively quickly. An algorithm that looks a lot like the Louvain algorithm is the greedy community detection algorithm. The difference between the two is that the greedy algorithm stops after nodes have been moved to other communities and there is no new improvement detected, and the Louvain algorithm still continues then with repeating the movement of nodes but this time the nodes of the graph are the newly created communities. And thus with the Louvain algorithm you get networks of communities of communities etc. With this extra layer the Louvain algorithm gets better communities than the greedy algorithm, even though they are quite similar.

We were interested in which way a heuristic with its measure plays a role in the Louvain algorithm. The heuristic makes an algorithm work faster by making some shortcut solutions in the measure it plays on. Normally, you use the Louvain algorithm with a heuristic for the modularity of the graph. You want to maximize the modularity. But there are many heuristics with measures one can think of which all have a specific characteristic. And it's hard to compare most of the heuristics and measures, because they are mostly connected to a complete algorithm. With our research we tried to generalise the Louvain algorithm such that we could apply different functions which embody the heuristic and the measure.

2. Method

Our plan to test the different measures and heuristics was by implementing a community detection algorithm in such a way that the measures and heuristics are easily interchangable. This will give us results in which the only difference is the actual difference in measures and heuristics. We combined this with fixed seed LFR generators to remove the variance of graphs as well.

We started out by implementing the Louvain algorithm, which we generalised in such a way that it expects certain functions with certain structures. In these functions we could implement the specifics of the measure and heuristic we would use. One of the function is used when you want to get the difference in measure when a node is transferred from one community to another community. In that situation you only want the community transfer with the highest positive increase. This idea is implemented in the function getDelta. Furthermore, the algorithm also wants to combine nodes from one community to create one big node for the new iteration. For that, we implemented a totalMeasure function. You only want to iterate once more when the totalMeasure is increased compared to the previous step, or from the initial value of minus infinity.

We firstly wanted to focus ourselves on two measures:

modularity and the ratio between the amount of inner edges and the amount of inner nodes of a community, which we will refer to as edge ratio from now on. This ratio should be high, because a high amount of inner edges inside of a community compares to nodes means a high connectivity in the community. We also wanted to look at more kinds of measures, like average distance inside of communities, but these will be used in the end version of the final report.

Modularity is the standard measure used with the Louvain method and already a proven good measure. The ratio of edges and nodes as explained earlier is a much easier variant of modularity, because it only looks at the amount of inner edges and inner nodes of a community. That means this ratio would be calculated much faster than the modularity, and also for the difference calculation. This ratio follows the idea of what a community means, strongly connected nodes.

This more generalised version of the Louvain algorithm is used 20 times with randomly generated graphs using the LFR benchmark function of NetworkX. To make sure we can indeed compare these graphs and their outputs from the generalised Louvain algorithm, we also had to make sure these 20 different graphs were created in a similar way. Therefore, we used a fixed seed. This seed is used in the LFR benchmark function to randomize the creation of nodes and edges. We used the LFR benchmark function, because with the LFR benchmark you will get a graph that has certain amount of communities. And with all the parameters of LFR benchmark you can make it more precise how many communities there should be and how big the network should be. We also tried the experiment with a stochastic graph generator, but this resulted in graphs that were too large and we couldn't use that in our experiment to get rapid results.

The following LFR settings were used:

Variable	Value
Node Count (n)	250-2000
tau1	3
tau2	1.5
mu	0.03-0.75
average degree	20
max communities	0.1n
max degree	0.1n

Table 1. LFR variables

These settings took a bit of tweaking since LFR is quite temperamental.

To compare these communities, we need to determine how good their partition is on the whole network. We will use the end total measure, the run time, the amount of edges, the amount of nodes and the Jaccard coefficient. This last measure tells us how many pairs of nodes are correctly put together in communities according to the community labeling that results from the LFR benchmark function. The higher this result, how better is our partition.

3. Results

Graph	Modularity	Edge Ratio
1	0.1146119223	0.05540989338
2	0.08183265053	0.03961728814
3	0.03220903653	0.04171521543
4	0.03858338181	0.03881220361
5	0.2860360809	0.0506337531
6	0.04676959674	0.04774633736
7	0.09659937117	0.06145145193
8	0.04161416331	0.04712325498
9	0.02942829049	0.04975824662
10	0.3447528491	0.04934189327
11	0.05103125664	0.05090140659
12	0.04516072676	0.05893646652
13	0.2994181964	0.05608010062
14	0.08622355277	0.04535216735
15	0.1785008218	0.04857695416
16	0.3002578614	0.04344187446
17	0.0302730866	0.04190097507
18	0.03628343593	0.05448934218
19	0.1515739827	0.02922116639
20	0.3008413359	0.05811152676
Average	0.12960008	0.0484310759

Table 2. Jaccard coefficients of Modularity and Edge Ratio

Graph	Modularity Communities	Edge Ratio Communities	LFR Communities
1	26	36	19
2	30	84	46
3	159	51	34
4	63	100	50
5	37	75	33
6	1	93	43
7	31	26	21
8	42	37	30
9	50	8	25
10	83	24	25
11	32	25	21
12	22	22	18
13	43	64	29
14	79	68	35
15	29	51	29
16	261	98	47
17	46	66	44
18	91	59	33
19	52	92	53
20	81	25	21
Average	62.9	55.2	32.8

Table 3. Amount of communities in measures and correct amount of communities

Table 2 shows the Jaccard coefficient per graph for each measure, as well as the average Jaccard coefficient. Table 3 shows the amount of communities generated per measure and the average amount of communities. These can be compared to the community count set by the LFR generator.

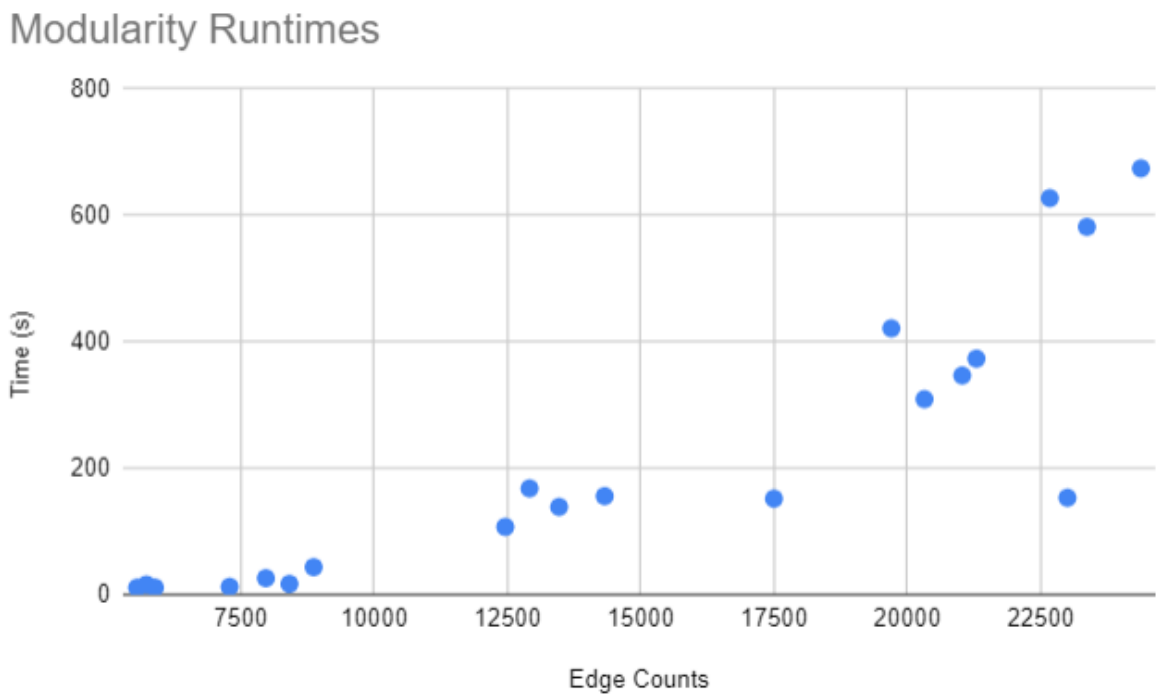

Figure 1. Modularity runtimes

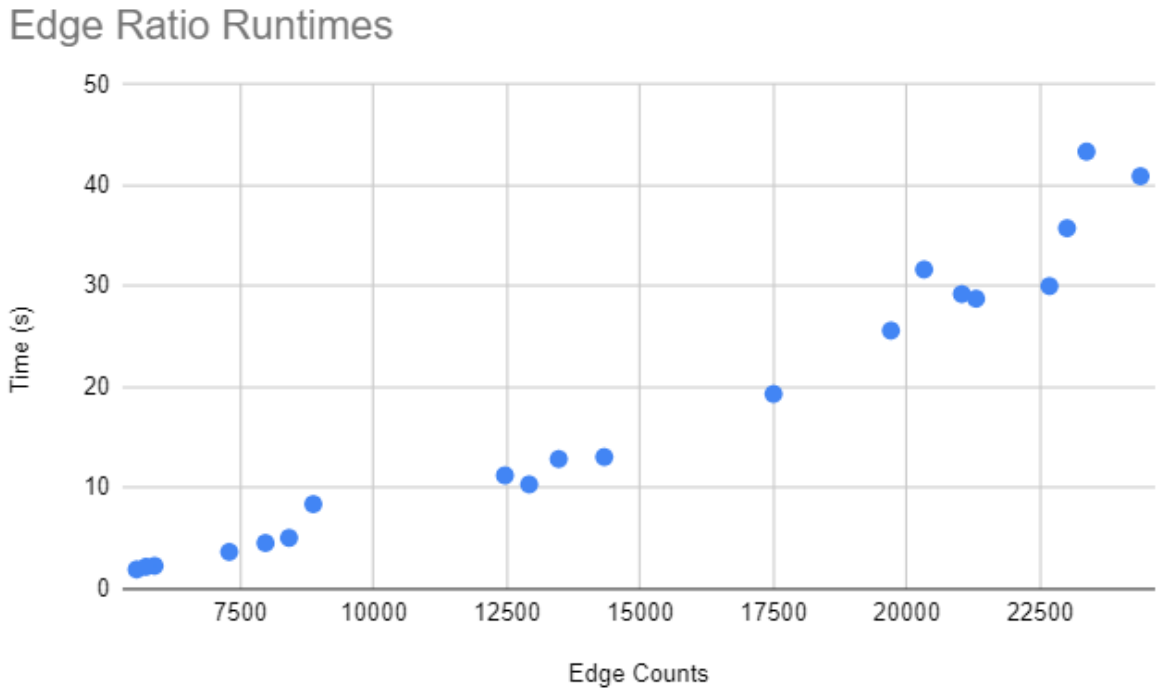

Figure 2. Edge ratio runtimes

Figures 1 and 2 show the runtimes in seconds for each measure with on the x axis the size of the graph defined by the amount of edges (since most measures calculate the amount of edges). This doesn't only show the difference between the measures but also the time complexity of the function.

4. Discussion

To properly evaluate our measures we need to look at both runtime and correctness.

4.1. Runtime

If we evaluate the runtime of both measures we can clearly see in Figure 1 that the runtime can quickly balloon to more than 10 minutes for larger graphs. Considering our graphs aren't enormous, with the node count being a maximum of 2000 nodes, in more real world applications this can quickly become much larger. The edge ratio seen in Figure 2 however, doesn't even go above a rutime of sixty seconds, which is way better. Another thing of importance is the shape of the graph. The shape of these graphs indicate the increase in runtime when the graph size is increased, which is also called time complexity. Interestingly enough, even though the modularity takes a lot longer to calculate, the curve is the same. This indicates that modularity calculations have a longer base time it takes to calculate, but said calculation does not have a larger time complexity than the edge ratio.

4.2. Correctness

To evaluate the correctness of a solution we have to compare it to some ground truth. The LFR generator we used generates a graph with every node having an attribute which contains a list of the nodes in its community. We can compare this community structure to community structure generated by our Louvain algorithms to find out the correctness of these results. To grade the correctness we used the Jaccard coefficient. This will give us a result between zero and one, zero meaning absolutely nothing in common and one meaning the communities are identical. In Table 2 we can see that the average Jaccard coefficient for modularity is way higher than for the edge ratio. Since the graphs these algorithms have been run on are identical, we can run a paired t-test on these results. Running this t-test will result in a P-value of 0.0042. This means that the difference between these two results is statistically significant. This means that modularity has a signficantly better result than the edge ratio. Interestingly enough, the standard deviation for modularity (0.113) than the standard deviation for edge ratio (0.008), which suggests that edge ratio delivers a consistently bad result, whereas modularity differs more, with some results being very bad and others a lot better.
Another thing to look at for the correctness is the amount of communities generated by each algorithm. This can be found in table 3. Both community counts are higher on average than the correct community counts generated by the LFR generator. Modularity actually has a higher difference between the community count generated and the correct community count. This would make it seem that the communities generated by modularity are either very close to the correct communities, or fragmented into multiple smaller communities.

5. Conclusion

From the results the measure with the best runtime is clearly the edge ratio, however runtime doesn't mean much when the results are of a low quality, which is the case for the edge ratio. The correctness results for edge ratio are not good enough to be worth it for the decrease in runtime. Modularity however does have a good result, however the runtime is not ideal. In general however, modularity does appear to be the better option of the two. This is probably also the reason modularity is the standard for the Louvain algorithm.