Network Analysis Project

Presenting a clustering algorithm by Kim Kestenbaum and Esty Katzeff

Workshop on Analysis of Biological Networks

Supervision: Roded Sharan and Shani Jacobson

Date: Spring 2022

Keywords: Integer Linear Programming, Louvain, Newman, Clustering, Biological Networks

Introduction

In this paper, we will describe the clustering algorithm we have developed as part of the workshop on Analysis of Biological Networks. The aim of the project was to improve existing algorithms for clustering graphs. We compared the performance of our algorithm to baselines: Newman and Louvain algorithms. For testing purposes, we used graphs provided to us; benchmark graphs and protein-protein interaction networks.

The existing clustering algorithms based on maximum modularity, such as Newman and Louvain, are not optimal, because they use non-deterministic methods. We were interested in implementing a clustering algorithm that maximizes modularity in an optimal way – by using integer linear programming (ILP) to partition the graph (a.k.a *ILP-partition*). The main issue with ILP as an implementation is that it is very slow when there are too many variables and constraints. Meaning, when the input graph is too large, *ILP-partition* will have trouble running in a reasonable amount of time. To solve this, our algorithm is based on the idea of reducing the size of the input graph and then running *ILP-partition* on this new graph.

The general idea of the algorithm can be described according to the following steps:

- Step 1: Given an input graph, partially run an existing clustering algorithm (that is based on maximum modularity) on the graph until its size is small enough.
- Step 2: Create a new graph of a reduced size, based on the partition created in the first step.
- Step 3: Run *ILP-partition* on the new graph to create a partition based on maximizing modularity.

Our assumption: an algorithm that implements this idea will have better performance when compared to an existing maximum-modularity algorithm. This is because step 3 is done optimally, while the existing algorithm is not.

Specifications

ILP formulation: we will describe the formulation of *ILP-partition* we used [1]:

Let the binary variable x_{ij} indicate if nodes i and j belong to the same community or not. The value of x_{ij} is zero if nodes i and j belong to the same community, and one otherwise. Let:

- $I_{all} = \{(i,j) \in V^2 | i < j\}$
- d_i is the degree of node i
- $m = \sum_{i \in V} d_i$
- $q_{ij} = a_{ij} - \frac{d_i d_j}{2m}$, for each $(i,j) \in I_{all}$

The community detection problem of maximizing the modularity of the graph is formulated in terms of the following integer linear problem:

Objective function:

$$max \sum_{(i,j) \in I_{all}} q_{ij}(1 - x_{ij})$$

Constraints:

$$x_{ij} + x_{jk} - x_{ik} \geq 0 \qquad \forall i < j < k$$

$$x_{ij} - x_{jk} + x_{ik} \geq 0 \qquad \forall i < j < k$$

$$-x_{ij} + x_{jk} + x_{ik} \geq 0 \qquad \forall i < j < k$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in I_{all}$$

Implementation: To implement our algorithm, we needed to use the algorithms Newman, Louvain and *ILP-partition*. These algorithms were implemented in the following ways:

- Newman: we implemented the Newman algorithm in C, utilizing matrix properties such as representation using a sparse matrix.
- Louvain: we used the default implementation of networkx (python module), and modified it according to our needs.
- *ILP-partition*: we implemented this in python using Gurobi.

Sanity check: Before deep-diving into the implementation of the code we wanted to run a basic test to check the performance of the *ILP-partition* algorithm in comparison to the existing clustering algorithms Newman and Louvain.

The results of this sanity check are shown in the graph below.

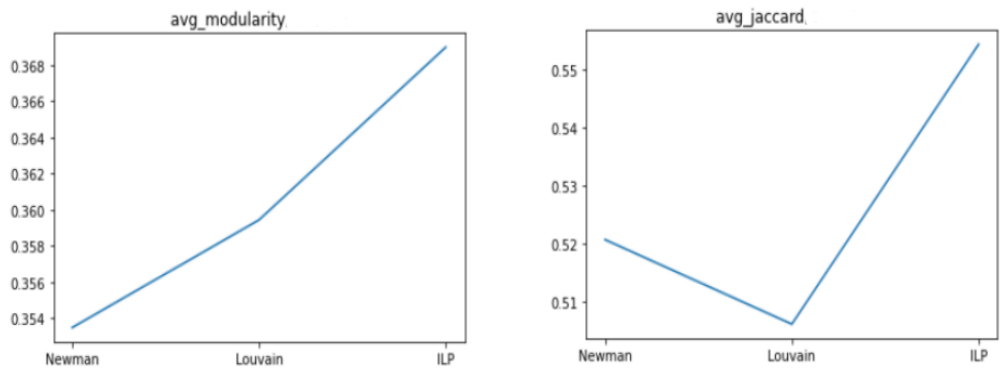

Figure 1: Comparing modularity scores of different algorithms; Newman, Louvain, ILP-partition. We generated 40 graphs for this test of size n=50 with parameter mu=0.2.

Evaluating methods: we evaluated the performance of our methods using different scoring methods – modularity, Jaccard and accuracy [4].

The graphs we used to calculate these scores are ones that were provided to us as part of the project:

1. Benchmark graphs – there is a total of 60 benchmark graphs. The parameters that differentiate between the graphs is the graph size (n), and the mixing parameter (mu). The different mixing parameters are 0.4, 0.5, 0.6. There are 30 graphs of size 1000 with 10 graphs per mu, and 30 of size 10,000 with the same mu separation.
2. Yeast graph – this is a protein-protein interaction network.

Methods

Important term: *LP-critical*: the minimal size of a graph generated from the partition in step 1.

Method 1: Newman-ILP algorithm

In this method the Newman algorithm is used in step 1 of the algorithm.

Basic Idea: The clustering algorithm of Newman initially divides a group into the two groups that gives the highest increase in modularity, and iteratively continues to divide each group into two until there is no increase in modularity. In this method we run the Newman algorithm until the size of the groups is smaller than *LP critical*. Then we run the *ILP-partition* algorithm on each group and return the final partition.

The following diagram demonstrates the idea:

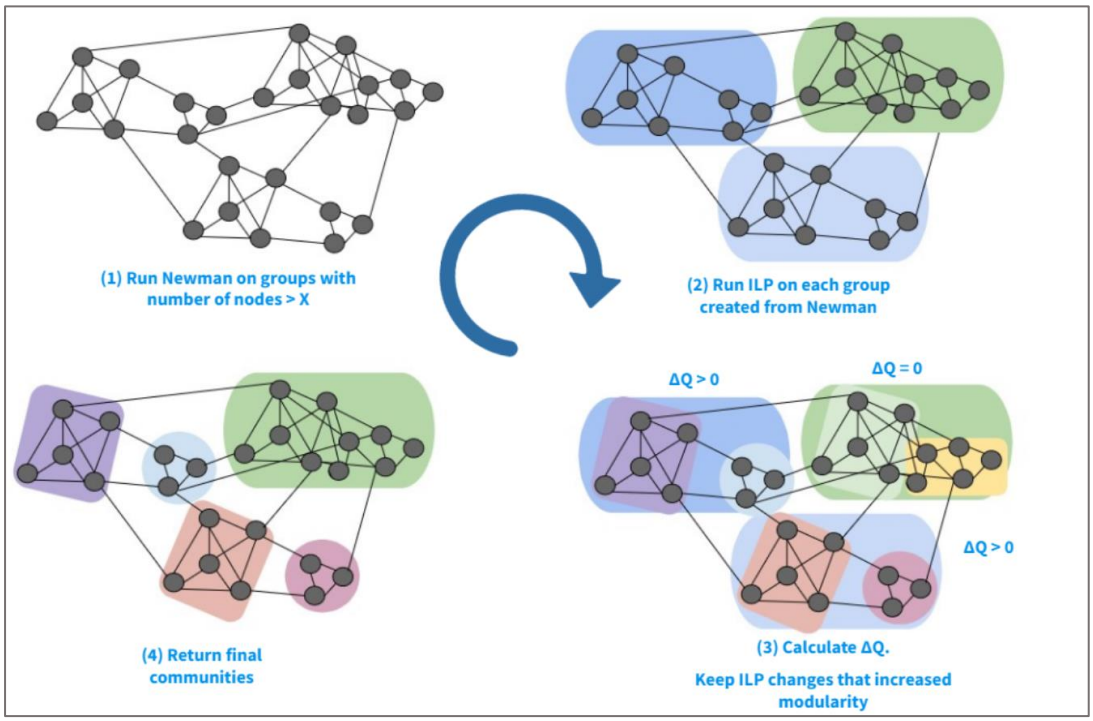

Figure 2: Demonstration of Method 1

Results: The graphs in *figure 2* show results of running the different algorithms on the benchmark graphs of size 1000. The x axis shows the results based on the different mu parameters. We were interested in comparing our algorithm (Newman-ILP) with Newman and Louvain algorithms in respect to different evaluation scores. In addition, we wanted to check the effect of different *LP-critical* values on the performance. We added a time limitation for the algorithm; 40 minutes.

After running the tests, we saw that we did not manage to improve the modularity value of the graph, but we did manage to improve the Jaccard and accuracy values in respect to Newman. With that said, the Louvain algorithm has a higher performance in all evaluation parameters.

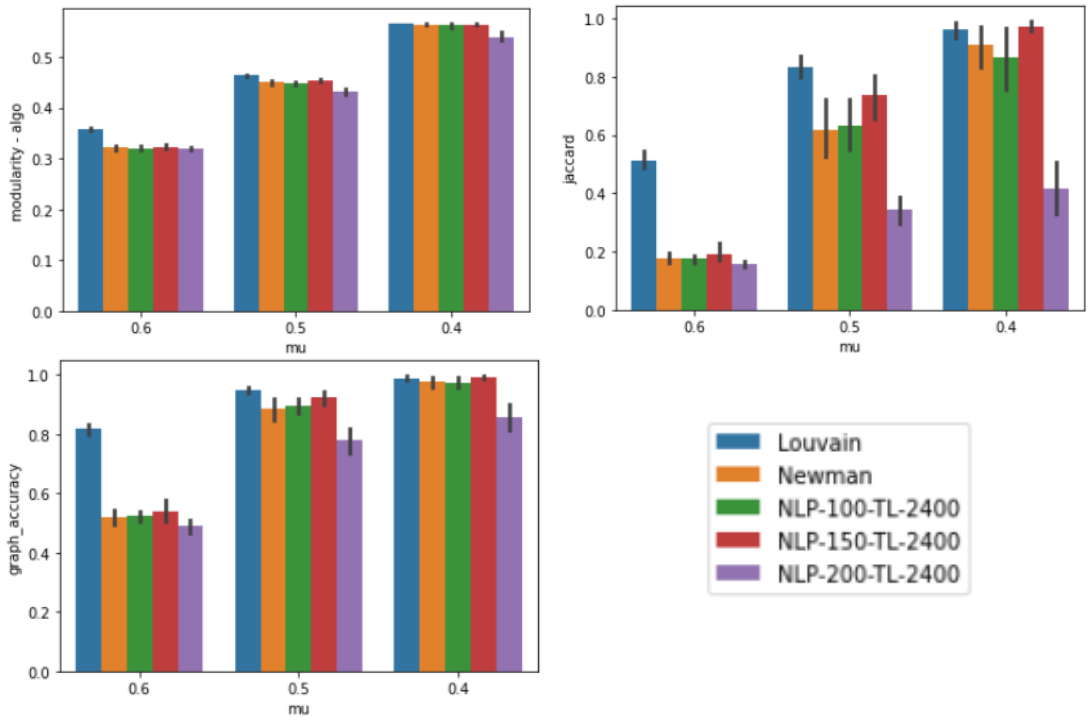

Figure 3: Comparing the different algorithms based on the benchmark graphs (size 1000). We tried different LP critical values that affected the run time of the algorithm; LP critical 100: 7 min, LP critical 150: 25 min, LP critical 200: 45 min (with time limit).

Conclusion and further research:

- The only *LP-critical* value that allows the algorithm to run in a reasonable amount of time is 100 (because this is the size the *ILP-partition* step can handle). We wanted to check if there is a way of reducing the running time with higher LP critical values, therefore we played with parameters provided by Gurobi – TimeLimit and IntFeasTol. Unfortunately, these did not improve the overall performance for higher LP critical values.
- The Louvain algorithm has higher performance on these graphs than any other algorithms that we tested (Newman, and Newman-ILP). Therefore, we decided that our next step will be to change the initial algorithm to be of Louvain instead of Newman.

Method 2: Louvain-ILP algorithm

Basic Idea: The Louvain clustering algorithm works by starting with putting each node of the graph in a separate group, and iteratively uniting the groups that bring to the highest increase in modularity. Each group that is united is represented as a mega node. In this method we run the Louvain algorithm on the initial graph until the number of mega nodes is smaller than *LP critical*. Then we run the mega graph with the *ILP-partition* algorithm and return the final partition.

The following diagram demonstrates the idea:

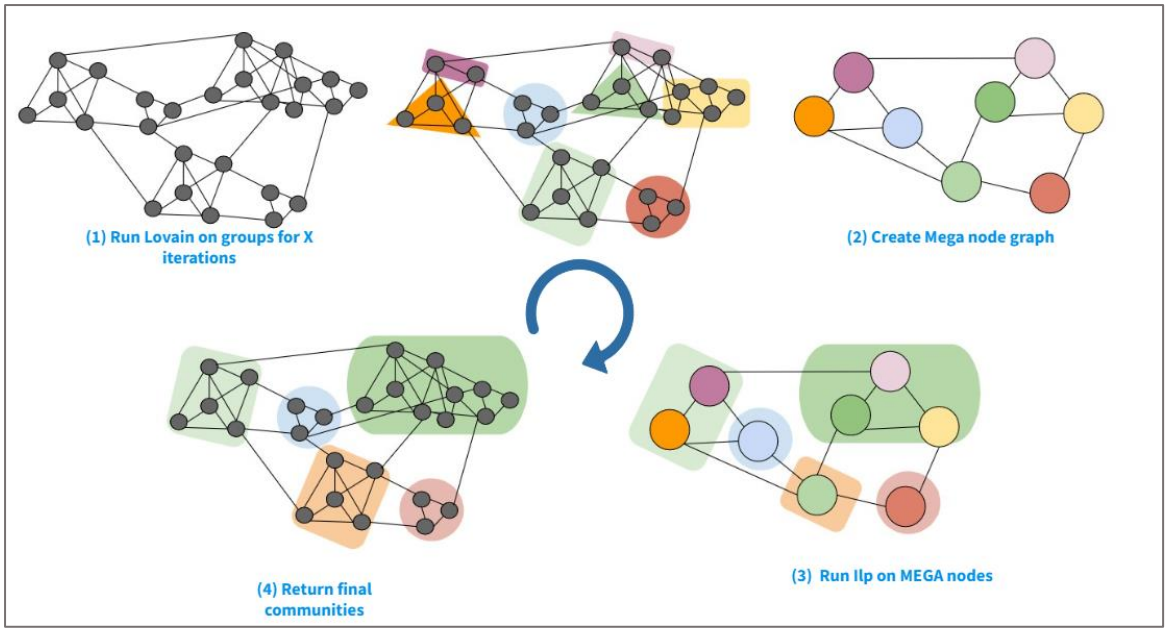

Figure 4: Demonstration of Method 2

Results: the following graphs are the results of running the algorithms on the benchmark graphs (of size 1000 and 10,000).

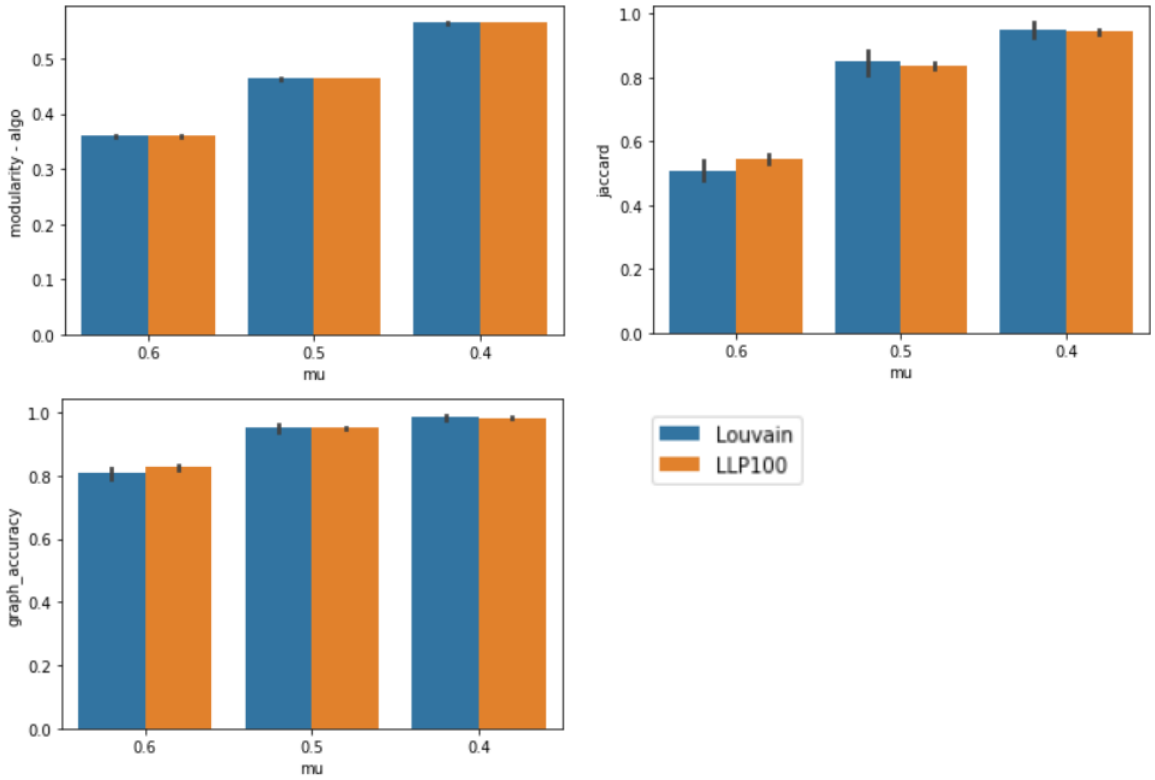

Figure 5: Comparing evaluation scores of this method; Louvain-ILP (a.k.a LLP) with the original Louvain algorithm. The LP critical value is 100. The graphs used for this test were the benchmark graphs of size 1000 with the different mu parameters.

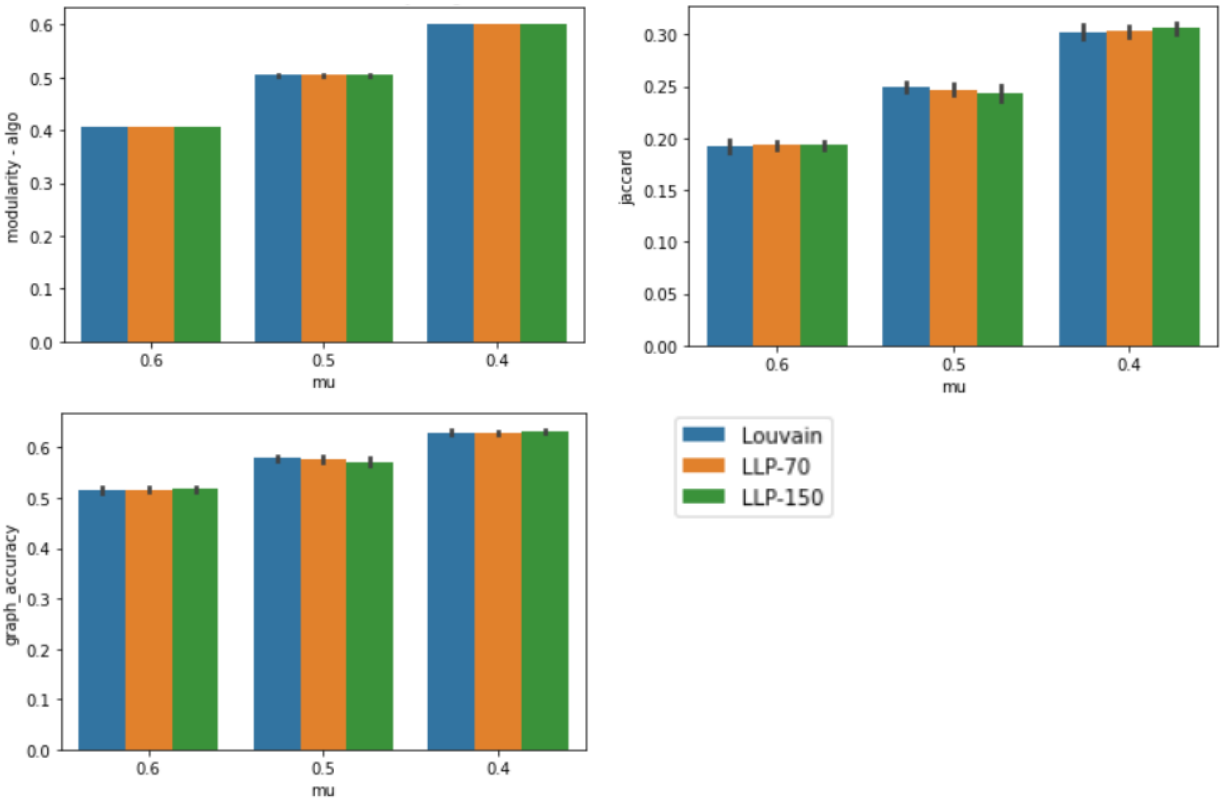

Figure 6: Comparing evaluation scores of this method; Louvain-ILP (a.k.a LLP) with the original Louvain algorithm. We tried different LP critical values; 70, 150. The graphs used for this test were the benchmark graphs of size 10,000.

Conclusion and further research:

- The running time issue we had with Newman-ILP is solved in this method.
- There is still no significant improvement of our algorithm in this method compared to the original Louvain algorithm; the evaluation scores are almost identical. Our assumption is that the first part of our algorithm, where Louvain creates a mega graph, is too dominant. Meaning, we are not giving enough weight to the *ILP-partition* part of the algorithm, therefore it does not have a chance to make the improvement.

Method 3: Louvain-ILP with added splits

Basic idea: Try to improve the algorithm of method 2 by adding another step (a.k.a *splitting step)*; after Louvain algorithm creates the mega graph, run another algorithm to partially split the mega graph nodes creating a mega graph with a larger number of nodes. Only then, run the ILP algorithm on this new and larger mega graph. The idea is that the *splitting step* of the algorithm will reverse part of the work done by Louvain and allow *ILP-partition* to have a more significant effect on the overall result.

The following diagram demonstrates this add-on to the initial method 2:

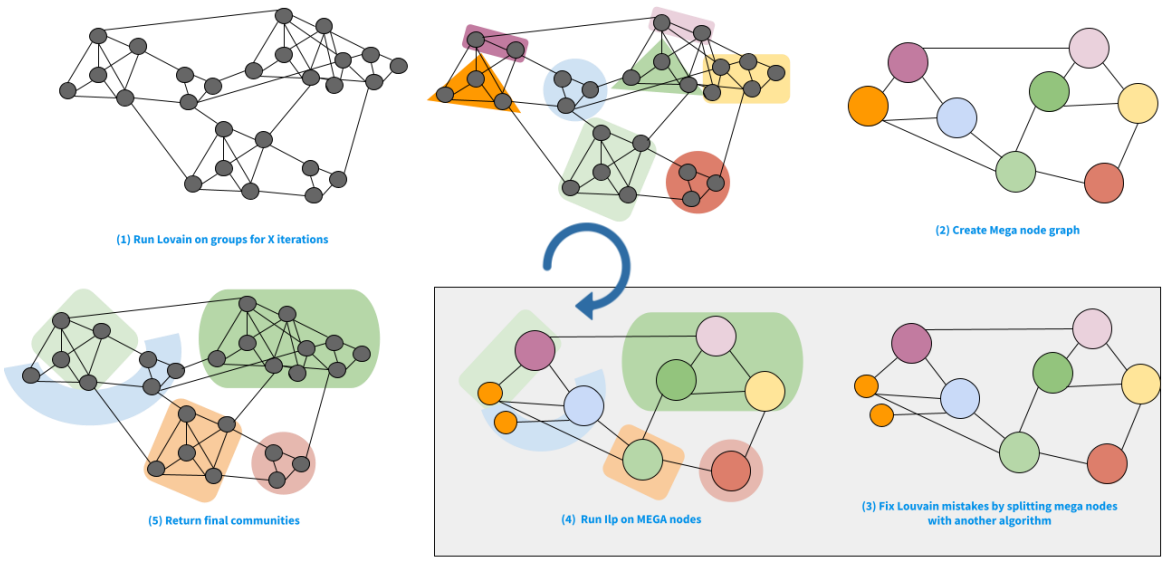

Figure 7: Demonstration of Method 3

Split methods: we tried different methods for splitting the mega graph

- Random split: we tried a basic method to see if there would already be an improvement at this point. We randomly split each mega node in the mega graph into two groups – doubling the size of the mega graph.
- Greedy modularity: in this method we created a subgraph from each mega node and split the subgraph using Clauset-Newman-Moore greedy modularity maximization function of networkX.
- Min-cut split: similarly to greedy modularity, here we also created a subgraph from each mega node and then chose 2 nodes randomly and used min-cut of networkX on it.
- Maximum modularity: we tried implementing this using two different methods:
  - ILP: we rephrased our original *ILP-partition* formulation for it to split the given graph into two groups only. The formulation we used:

    Let the binary variable x_i indicate which group node i belongs to. The value of x_i is zero if node i belongs to community zero, and one if belongs to community one.

    Objective function:

    $$\sum_{(i,j)\in I_{all}} \left(a_{ij} - \frac{k_i k_j}{m}\right) \cdot (y_{ij} + 1 - z_{ij})$$

    Constraints:

    $$y_{ij} \leq x_i$$

    $$y_{ij} \leq x_j$$

    $$y_{ij} \geq x_i + x_j - 1$$

    $$z_{ij} = x_i + x_j - y_{ij}$$

  - Newman: we changed our original implementation of Newman in the following way:

- In the original implementation the first step of the algorithm is: given an input graph create a group containing a group of all nodes of the graph, and then continue trying to divide this initial group.
- Instead, we changed the first step to create a group that contains several groups, based on a given partition (the partition was created according to the mega graph of Louvain).
- We ran Newman on these groups in the purpose of creating more subgroups from the initial partition.

Results: the splitting method according to Newman had the best performance and best running time, therefore we will present these results only. Nevertheless, for further research the results of running the other methods are presented here.

We saw that the Newman splitting method does not make a significant change to the number of partitions of the input graph (size of the mega graph). It is likely that this is because a further division does not give a significant increase of the modularity of the graph. This is based on the results on the benchmark graphs of size 1000 – where the Newman splitting method divides between 1-3 existing partitions, and on the benchmark graphs of 10,000 the Newman splitting method does not create any further divisions of the existing partitions.

Therefore, the graphs presented are of the tests that were run on the benchmark graphs of size 1000 only.

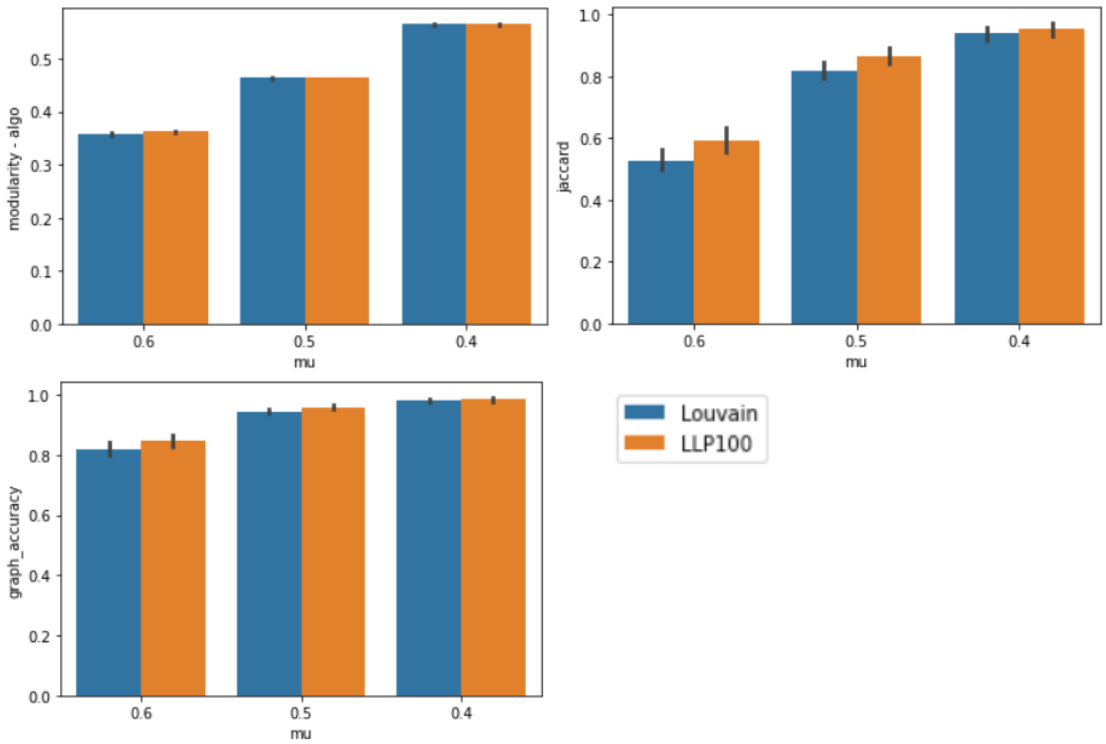

Figure 8: Comparing evaluation scores of this method; Louvain-ILP with splits with the original Louvain algorithm, on the benchmark graphs of size 1000.

Conclusion: There is an improvement in the Jaccard score in this method compared to the previous method, but not a significant improvement in the modularity score.

Yeast: We will present the results of running our final method (method 3) on the yeast protein-protein interaction network. Not all the runs on yeast succeeded in the *splitting step* (using the Newman splitting method), but the runs that did succeed in this step (that divided at least one mega graph) gave higher performance, and this can be seen in the graph below.

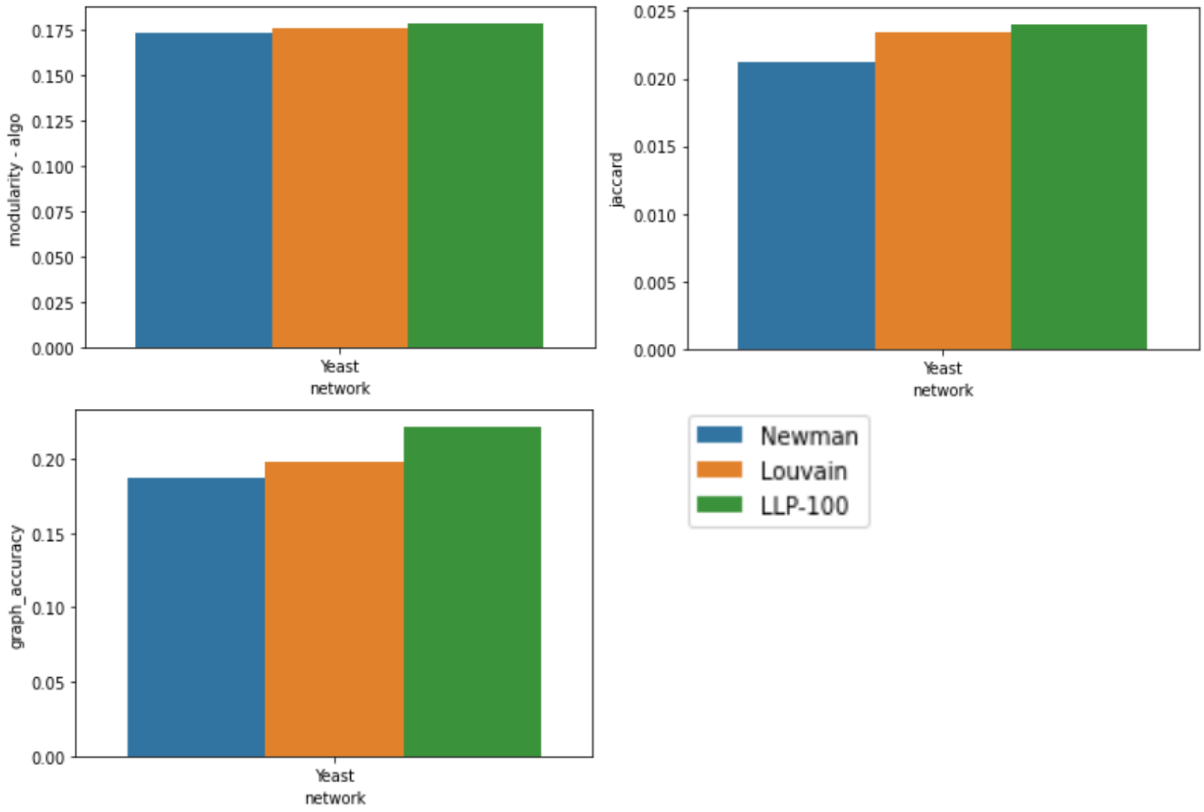

Figure 9: Comparison of all algorithms on yeast protein-protein interaction network. The ILP-partition algorithm is of Method 3, and these are results of a specific run that succeeded in "splitting step".

References

[1] Discovering Communities in Networks: A Linear Programming Approach Using Max-Min Modularity, Arman Ferdowsi & Alireza Khanteymoori, 2021

[2] Newman, M. E. J. (2006). Modularity and community structure in networks. Proceedings of the National Academy of Sciences of the United States of America, 103(23), 8577–82. https://doi.org/10.1073/pnas.0601602103

[3] Blondel, Vincent D; Guillaume, Jean-Loup; Lambiotte, Renaud; Lefebvre, Etienne (9 October 2008). "Fast unfolding of communities in large networks". Journal of Statistical Mechanics: Theory and Experiment. 2008 doi:10.1088/1742-5468/2008/10/P10008

[4] Evaluation of clustering algorithms for protein-protein interaction networks, Brohee & van Helden, BMC Bioinformatics, 2006