**Chart

Description automatically generated with medium confidence**

The Yemenite Step Method

**Workshop in Analysis of Biological Networks**

Talya Sapir,  
Bella Azaria

Prof. Roded Sharan

Tel Aviv University, 2022

Contents

[Summary 2](#_Toc113301235)

[1. Introduction 2](#_Toc113301236)

[Louvain 2](#_Toc113301237)

[Yemenite Step 2](#_Toc113301238)

[2. Yemenite Step Configurations 3](#_Toc113301239)

[Splitting functions 3](#_Toc113301240)

[Why we chose these methods 4](#_Toc113301241)

[Possible improvement options 4](#_Toc113301242)

[3. Choosing The Best Options for Each Method 5](#_Toc113301243)

[YemeniteStep with Newman 5](#_Toc113301244)

[YemeniteStep with Louvain 5](#_Toc113301245)

[YemeniteStep with GN Modularity 6](#_Toc113301246)

[YemeniteStep with GN Conductance 7](#_Toc113301247)

[4. Choosing The Best Method for Yemenite Step 7](#_Toc113301248)

[Modularity Results 8](#_Toc113301249)

[Conductance Results 8](#_Toc113301250)

[Jaccard Results 8](#_Toc113301251)

[5. Final Results 9](#_Toc113301252)

[Results on LFR Files 9](#_Toc113301253)

[Results on Arabidopsis Network 10](#_Toc113301254)

[Results on Yeast Network 11](#_Toc113301255)

[Results on ca-CondMatNetwork 11](#_Toc113301256)

[Comparing results to Louvain 11](#_Toc113301257)

[6. Conclusions 11](#_Toc113301258)

[Metrics performance 11](#_Toc113301259)

[Time performance 12](#_Toc113301260)

[7. Bonus 12](#_Toc113301261)

[8. Python Implementation 12](#_Toc113301262)

# Summary

We introduce Yemenite Step, a graph clustering method based on the widely used network clustering algorithms, Louvain[[1]](#footnote-1). The Yemenite Step python module can generally receive any clustering method as an input, along with different option flags, and runs an iterative Louvain algorithm where at each iteration- the given clustering method is applied to each community Louvain found.

In this Paper we test Yemenite Step with four different clustering methods: Louvain, Newman, Girvan-Newman, and a conductance-based variation of Girvan-Newman. We also introduce three possible improvement options, Random, Remerge, and Relative, and test them on each clustering method.

The metrices used to evaluate the clustering quality are Modularity, Conductance, Jaccard similarity, and Accuracy, and we compared each method with Louvain results.

The best method we found for using Yemenite Step is with Girvan-Newman as the splitting function argument, with the Random and Relative options. This method GAVE SOME RESULTS MAYBE.

# 1. Introduction

## Louvain

In the Louvain Method of community detection, small communities are found by optimizing modularity by locally moving all nodes between singleton communities, then each small community is grouped into one node and the first step is repeated iteratively until there is no gain in modularity.

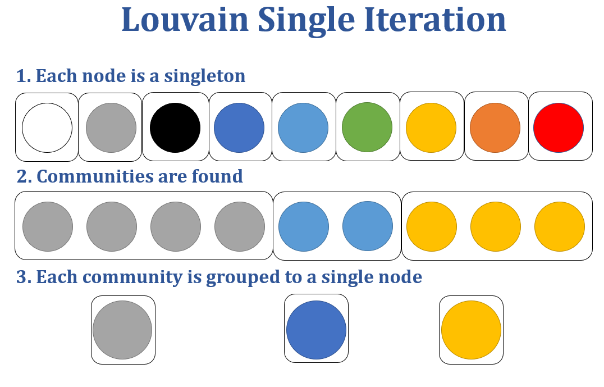
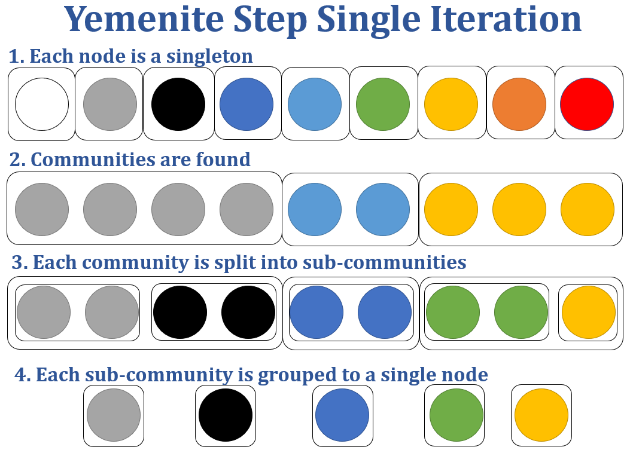
However, “Greedy” algorithms often don’t reach global maxima since they get stuck in lower local maxima. Giving the algorithm opportunities to “regret” some steps and explore new states can allow the algorithm to find new and sometimes better results.

## Yemenite Step

The idea of Yemenite Step (YS) is to iteratively run Louvain, but at each step we allow it to regret and to fine-tune it's result by running a separate clustering algorithm on each community. Yemenite Step can be described in these three steps:

1. Repeatedly move nodes around until best communities are found (as normal Louvain does)
2. For each community C, run a clustering algorithm in the induced subgraph G[C], and find sub communities {c1, c2...}
3. Convert each sub-community to a single node in a super graph, and continue to next iteration

Fig. 1 – Demonstrating the differnce between YS and Louvain.



# 2. Yemenite Step Configurations

The Yemenite Step python module we implemented can generally receive any clustering method as an input, and has three additional running options.

## Splitting functions

For this workshop, we tested these known clustering algorithms as the splitting function for the Yemenite Step:

1. Louvain
2. Girvan-Newman[[2]](#footnote-2)
   1. Maximizing Modularity
   2. Maximizing Conductance
3. Newman[[3]](#footnote-3)

The result of the Girvan–Newman algorithm is a dendrogram representing the current clustering of each step. To choose a single clustering we calculate the modularity or the conductance of all levels of the dendrogram and return the clustering that maximizes it. we will refer to these two variations of Girvan-Newman as GN-Modularity and GN-Conductance.

## Why we chose these methods

All three methods are popular methods known to give pretty good results. For Louvain, Being the most popular of the three and the one that YS is based on, it is clear why we chose this method.

Girvan-Newman and Newman both have a top-down approach, being divisive methods that start with a single community and breaking it apart. This divisive approach might make more sense in this context, since we are given a community that we trust to be promising, and we just want to fine-tune it. It also mixes up the greedy aggregative course, allowing it to "explore" and break out of local maxima.

## Possible improvement options

Besides choosing the splitting function, we tested three additional options for how YS should run.

1. Random option

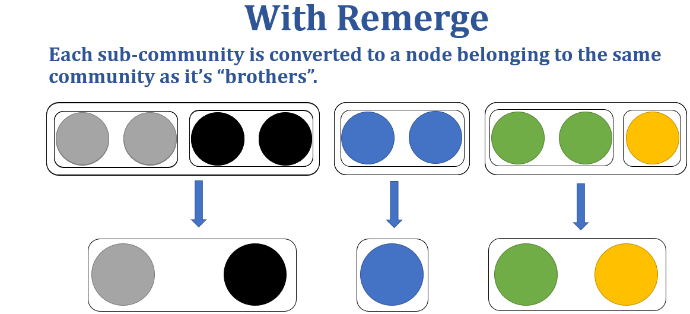
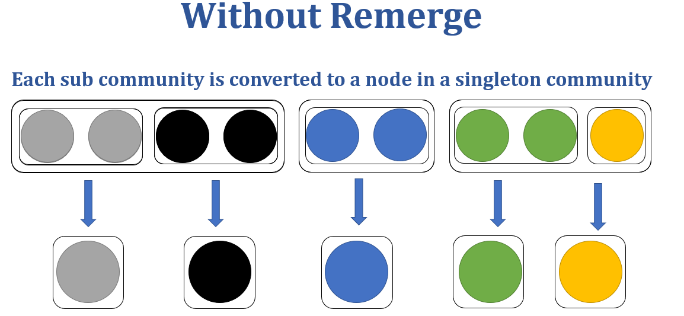
Yemenite Step uses Louvain's local moving heuristic to find communities in each iteration (before splitting each community). This is done by iterating over all nodes again and again in the same order. With this option on, the order of nodes changed randomly. This is a common variation used with Louvain.

1. Remerge option

After splitting a community into sub-communities, each sub-community is converted to a single node in a singleton community for the next iteration. However, it might make sense to "remember" which sub-communities came from the same super-community, assuming they are more likely to be part of the same cluster.

With this option on, each sub-community will be converted to a single node, but nodes that used to belong to same super-community will be in the same community in the next iteration.

Fig. 2- Demonstrating the Remerge option.



1. Relative option

When Yemenite Step uses a splitting algorithm on a community of nodes, it treats the community as a separate network, using an induced subgraph of the original network, thus ignoring edges to nodes from outside the community.

However, when calculating modularity, we use the nodes degree () and the total weight of edges (m) to try and capture how likely it is for to nodes to be connected in a random graph. Using an induced subgraph changes the degree and total amount of edges, so the modularity calculation may not be as helpful.

With this option on, the modularity of each sub-network will be calculated using the and m values from the original network.

This option is only useful for splitting methods that use modularity (Louvain, GN Modularity)

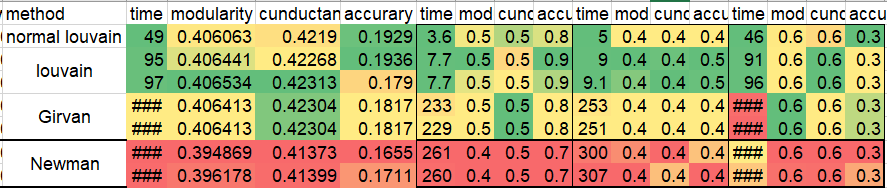
# 3. Choosing the Best Options for Each Method

Our goal was to find what is the best way to use Yemenite Step for the clustering problem, by choosing a splitting function and the additional options for it.

First, since the different additional options may have different affects on different methos, we tested each option on each method, using LFR[[4]](#footnote-4) graphs (sizes=1000, 10000, mixing parameter ()=0.4,0.5,0.6). Then after finding the best option combination per method, we compared the methods to each other and to the regular Louvain clustering algorithm, to choose the best one. The best preforming method was then tested on the Yeast, Arabidopsis, and ca-CondMat networks[[5]](#footnote-5).

## Yemenite Step with Newman

We noticed early on that using Newman as the splitting method gave the worst result in every metric, and took a very long time to run, so we decided not to use it, and not to test it with any additional option

  
Fig. 3 - For Illustration purposes: Newman metric scores on different networks (of sizes 100,1000, compared to the other methods

## Yemenite Step with Louvain

These are the results for how the three different options effected the evaluation metrices, when using Louvain as the splitting function.

For each option (Random, Remerge, Relative) and for each graph size (1000, 10000) we calculated the affect on each metric (Modularity, Conductance, Jaccard). The *affect* is the relative change of the average value:

|  |  |  |
| --- | --- | --- |
| Adding **Random** Option | **1000** | **10000** |
| Modularity | -0.06%▼ | -0.01%▼ |
| Conductance | -0.05%▼ | -0.01%▼ |
| Jaccard | 0.04%▲ | -0.07%▼ |

|  |  |  |
| --- | --- | --- |
| Adding **Remerge** Option | **1000** | **10000** |
| Modularity | -11.12%▼ | -0.02%▼ |
| Conductance | -11.65%▼ | -0.07%▼ |
| Jaccard | -16.16%▼ | 1.41%▲ |

|  |  |  |
| --- | --- | --- |
| **Adding Relative Option** | **1000** | **10000** |
| Modularity | -0.01%▼ | 0%▼ |
| Conductance | -0.01%▼ | -0.04%▼ |
| Jaccard | -0.28%▼ | 0.59%▲ |

We conclude that when using Louvain as the splitting function, it is best to use it with all the options off.

## Yemenite Step with GN Modularity

These are the results for how the three different options effected the evaluation metrices, when using GN Modularity as the splitting function.

For each option (Random, Remerge, Relative) and for each graph size (1000, 10000) we calculated the affect on each metric (Modularity, Conductance, Jaccard). The *affect* is calculated as explained in Splitting with Louvain.

|  |  |  |
| --- | --- | --- |
| **Adding Random Option** | **1000** | **10000** |
| Modularity | -0.02%▼ | 0.02%▲ |
| Conductance | 0.02%▲ | -0.03%▼ |
| Jaccard | -0.21%▼ | 1.6%▲ |

|  |  |  |
| --- | --- | --- |
| **Adding Remerge Option** | **1000** | **10000** |
| Modularity | -10.48%▼ | -0.46%▼ |
| Conductance | -59.78%▼ | -22.36%▼ |
| Jaccard | -16.73%▼ | 9.1%▲ |

|  |  |  |
| --- | --- | --- |
| **Adding Relative Option** | **1000** | **10000** |
| Modularity | 0.03%▲ | 0.5%▲ |
| Conductance | -0.17%▼ | 0.52%▲ |
| Jaccard | 1.88%▲ | 0.25%▲ |

We conclude that when using GN Modularity as the splitting function, it is best to use it with the "Relative" option on, and while results on the "Random" option aren't as clear, we might still want to use it on large graphs.

## Yemenite Step with GN Conductance

These are the results for how the three different options effected the evaluation metrices, when using GN Conductance as the splitting function.

For each option (Random, Remerge) and for each graph size (1000, 10000) we calculated the affect on each metric (Modularity, Conductance, Jaccard). The *affect* is calculated as explained in Splitting with Louvain.

|  |  |  |
| --- | --- | --- |
| **Adding Random Option** | **1000** | **10000** |
| Modularity | 0.17%▲ | 0%▲ |
| Conductance | -0.08%▼ | -0.04%▼ |
| Jaccard | 3.41%▲ | 1.49%▲ |

|  |  |  |
| --- | --- | --- |
| **Adding Remerge Option** | **1000** | **10000** |
| Modularity | -6.82%▼ | -0.22%▼ |
| Conductance | -33.58%▼ | -14.78%▼ |
| Jaccard | -7.25%▼ | 4.48%▲ |

We conclude that when using GN Conductance as the splitting function, it is best to use it with the "Random" option on, and while results on the "Remerge" option have a surprising jump in Jaccard metric in large graph, we decide to not use this option.

# 4. Choosing the Best Method for Yemenite Step

The methods we are comparing are:

1. Yemenite Step with Louvain – with no additional options
2. Yemenite Step with GN Modularity – with Random and Relative options
3. Yemenite Step with GN Conductance – with Random option

We are comparing all methods to the results of running the regular Louvain and Newman algorithms.

## Modularity Results



## Conductance Results



## Jaccard Results



The best preforming method in nearly every metric is **Yemenite Step with GN Modularity – with Random and Relative options.** How ever, we were unable to run Yemenite Step with GN Modularity on the larger networks (Yest, ca-CondMat), since running time took over 30 hours. So, in the final results we will also present **Yemenite Step with Louvain – with no additional options.**

# 5. Final Results

Here are results for YS with GN and Louvain, in detail for each value, and show its results on the Yeast, Arabidopsis, and ca-CondMat networks.

Notice we do not have the results for Newman on all networks. We were unable to run Newman on networks of size bigger than 100 due to the time it took to run, so we used the Newman data from one of our classmates, which were also partial.

## Results on LFR Files

**Modularity** on size=1000, for



Modularity

**Modularity** on size=10,000, for



Modularity

**Conductance** on size=1000, for



Conductance

**Conductance** on size=1000, for



Conductance

**Jaccard** on size=1000, for



Jaccard

**Jaccard** on size=1000, for



Jaccard

## Results on Arabidopsis Network



## Results on Yeast Network



## Results on ca-CondMat Network



## Comparing results to Louvain

We represent the same data from the graphs above in table form. For each network and for each method we calculated the relative difference compared to Louvain method:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Network** | **Method** | **Modularity** | **Conductance** | **Accuracy** |
| **Arabidopsis** | **YS(GN\_mod)** | -0.06%▼ | -0.23%▼ | 0%▼ |
| **Arabidopsis** | **YS(Louvain)** | -0.36%▼ | 1.69%▲ | 0%▼ |
| **Yeast** | **YS(Louvain)** | -0.9%▼ | 0.56%▲ | 0%▼ |
| **ca-CondMat** | **YS(GN\_mod)** | 0.09%▲ | -0.03%▼ |  |
| **ca-CondMat** | **YS(Louvain)** | 0.01%▲ | 0.12%▲ |  |

# 6. Conclusions

## Metrics performance

Both YS with GN-mod and with Louvain gave better results running on small networks (1000), or with a low mixing parameter (0.4), however for bigger networks including the three natural networks. On the ca-CondaMat Network YS with Louvain worked surprisingly well, but only improved modularity by 0.01%, while using GN-mod improved modularity by 0.09% (compared to Louvain).

## Time performance

Time performance was not the goal of this workshop, and when implementing the methods in code we did not try to optimize time performance or code efficiency in any way. Though we did record running times, they are not very informative since we ran them on different computers, sometimes using different threading methods.

However, even though running on a designated server with adder memory and computation resources, using YS with Girvan-Newman on Yeast took over 30 hours, and we did not wait for results. This is probably because Girvan-Newman is sensitive to the number of edges, and Yeast has much more edges than the other networks (~550,00 compared to a maximum of ~77,000 in the LFR networks).

# 7. Bonus

On a Google Cloud Linux Server (c2-standard-4) with 4 vCPUs and 16Gib memory, running YS with Louvain as a splitting function runs in less than 1h on DBLP network.

# 8. Python Implementation

A Python Implementation for Yemenite Step can be found here: <https://github.com/GertieGer/BioNetworkClustering>

Please see README file for further explanations on how to use it.

1. 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 [↑](#footnote-ref-1)
2. Girvan M. and Newman M. E. J., Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002) [↑](#footnote-ref-2)
3. Modularity and Community Structure in Networks M.E.J Newman, PNAS 2006 [↑](#footnote-ref-3)
4. Benchmark graphs for testing community detection algorithms, Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi (2008) [↑](#footnote-ref-4)
5. Yeast and are from the <https://thebiogrid.org/> database. ca-condMat is from the Collaboration network from Stanford Network Analysis Project [↑](#footnote-ref-5)