CSE 601: Homework 3
Fall 2015

Graph cluster analysis for complex networks

Report by
Group #2
Ankit Kapur: 50133149
Samved Divekar: 50135204
Hrishikesh Sathe: 50134055

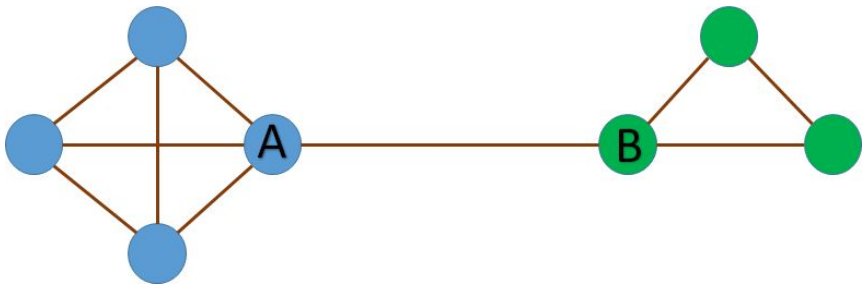
Part I: Understanding and implementation of MCL Algorithm

What is MCL Algorithm?

The Markov Cluster Algorithm is a fast and scalable unsupervised cluster algorithm for graphs, based on the simulation of flow in graphs. In a graph consisting of natural clusters, naturally there will be many links within nodes of same cluster and very few links amongst nodes from different clusters. That means, if we start at a node and randomly travel to a connected node, there is a high probability that we stay within the same cluster than travel between. MCL is based on the above principle.

How it works?

Consider the following graph. Node A is connected to 4 other vertices and node B is connected to 3 other vertices. Now, if we take a random walk starting at A, there is only ¼=0.25 probability that we will reach B. But the probability that it will reach other points is 0.75. Hence there is a high probability that node A belongs to the blue cluster than it belongs to Green cluster. Similar can be said for B. B belongs to green cluster because the probability that a random walk from B will reach to blue cluster is 0.33. MCL makes use of this probability to determine correct cluster for a node.



The algorithm starts with a given undirected weighted/unweighted graph, the expansion parameter and the inflation parameter. We first have to construct a matrix which represents the graph and contains weights of the edges(or 1's) between connected nodes as elements. Then the matrix is normalized. Then the normalized matrix is expanded using the expansion parameter. Expansion corresponds to computing random walks of higher length, which means random walks with many steps. It associates new probabilities with all pairs of nodes, where one node is the point of departure and the other is the destination.

This expanded matrix is then inflated using the given inflation parameter.Inflation will have the effect of boosting the probabilities of intra-cluster walks and will demote inter-cluster walks. These processes of expansion and inflation are repeated on the normalized matrix until the convergence is achieved.

Note: The expansion parameter M and Inflation parameter R can be varied and fined-tuned to achieve better results. In following sections, we have shared a few graphs which demonstrate how varying R and M affect modularity.

Why it works?

MCL works because it is based on the concept of Random Walks. When we consider a graph, we know that there will be many edges within a cluster and a few edges that connect this cluster to another cluster. Hence, if we choose a node and then randomly chose another connected node to travel to then the probability of us staying in the same cluster is more than probability of us traveling to another cluster. These are random walks and using these it is possible to find where the flow tends to gather and hence where the clusters are. These random walks are calculated using Markov Chains. This is the idea on that MCL uses to find clusters. MCL also boosts this effect by performing inflation - by taking the Hadamard power of a matrix i.e. taking powers entry wise. This inflation strengthens strong currents and weakens already weak currents.

Advantages:

- It is scalable and hence works with increasing graph size.
- Works with weighted and unweighted graphs.
- MCL is quite robust against noise in the graph data. Due to this, it is widely used in bioinformatics.
- It is easy to understand and implement.

Disadvantages:

- MCL cannot find overlapping clusters.
- It is very slow but the author claims it to be faster.
- Usually not suitable for clusters with a large diameter. This is because distributing the flow across the cluster needs long expansion and low inflation and hence it takes many iterations. This in turn causes MCL to be sensitive to small perturbations in the graph.
- It is also prone to output too many clusters

Pseudo-code:

1. Load edge data, and create an Adjacency matrix A.
2. Apply the MCL algorithm on the matrix, The function returns the clusters and the iteration count.

   applyMclAlgorithm(A, R, M, pruning_threshold, convergence_tolerance):
   1. Normalize the adjacency matrix A.
   2. initialize a counter to keep track of iterations.
   3. increment the counter.
   4. Expand the matrix using M (the matrix power(expansion) parameter).
   5. Inflate the matrix using R. Use following equation.

$$\Gamma_r(M_{ij}) = M_{ij}^r \,/\, \Sigma_{r,j}(M)$$

   6. Re-normalize the matrix.
   7. Prune out values that are close to zero. We used a pruning threshold for achieving this i.e. if any value is less than the pruning threshold, set the value to 0. This is done to improve performance.
   8. Check for convergence. If it has converged then go to step 9, else repeat steps 3 to 8.

9. Identify the clusters. For this, identify attractor node and attractor system. A node is an attractor node if it has non-zero values. If a node is an attractor then the set of its neighbours is called an attractor system. If a node is connected to any node of attractor system, the node belongs to the same cluster as that of the attractor system.
10. Return clusters and iteration count.

3. Calculate modularity using the clusters and the original Matrix.
4. Repeat steps 2 and 3 for all combinations of the R and M hyperparameters.

Rationale for using python:

Python offers libraries such as numpy for working efficiently with 2 dimensional matrices, and since we have used matrix operations extensively for our implementation of the algorithm, this helped in replacing loops with direct numpy matrix operations. Plotting graphs is also much easier with the matplotlib and networkx libraries.

You can take a look at our cumulative codebase on Github, or you can view `IP[y]: Notebook` our for a step-by-step understanding of the code.

Part II: Experiments and results

For our experiments we have varied the inflation parameter R to have the following values [1.1, 1.3, 1.5, 1.7, 1.9, 2], and the expansion parameter M to have the following values [2, 3, 4, 5, 6] in order to find an optimal tuning for the hyperparameters. The results are illustrated below.

For the purpose of evaluation we have used **Modularity** as the metric, which rewards higher edge weights within clusters (intracluster edges), and penalizes edges between clusters (intracluster edges). So, the **higher the modularity, the better the clustering**, as for a high modularity clustering the total weight of intracluster edges is large and the total weight of intercluster edges is small.

However for our case, even though we observed the highest modularity values for parameter configurations that produced **only 1 cluster** (refer to the results below), **we have ignored them** because a single cluster over a data set produces no meaningful information.

..:: **AT & T dataset** ::..
M = 2, R = 1.1: # of clusters = 01, iterations: 45, Modularity: 228 (intra 228, inter 0)
M = 2, R = 1.3: # of clusters = 03, iterations: 31, Modularity: 200 (intra 214, inter 14)
M = 2, R = 1.5: # of clusters = 07, iterations: 22, Modularity: 140 (intra 184, inter 44)
M = 2, R = 1.7: # of clusters = 10, iterations: 17, Modularity: 124 (intra 176, inter 52)
M = 2, R = 1.9: # of clusters = 55, iterations: 14, Modularity: 34 (intra 131, inter 97)
M = 2, R = 2.0: # of clusters = 55, iterations: 13, Modularity: 34 (intra 131, inter 97)

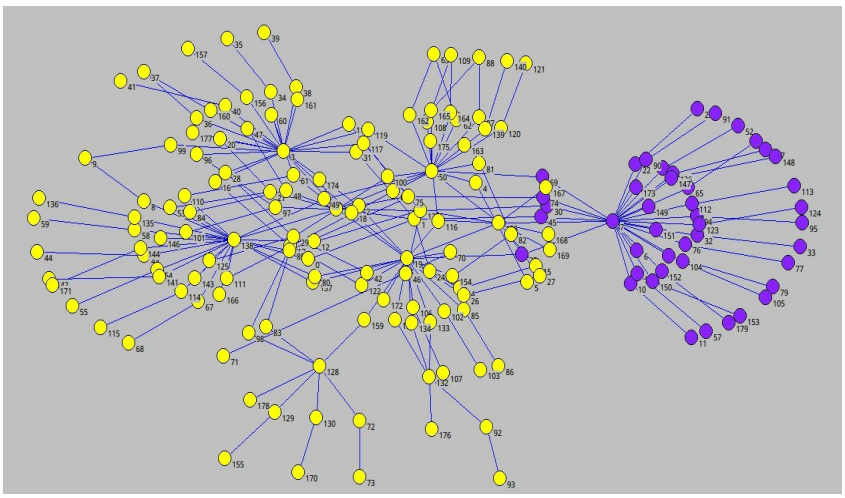
M = 3, R = 1.1: # of clusters = 01, iterations: 48, Modularity: 228 (intra 228, inter 0)
M = 3, R = 1.3: # of clusters = 01, iterations: 14, Modularity: 228 (intra 228, inter 0)
M = 3, R = 1.5: # of clusters = 03, iterations: 20, Modularity: 192 (intra 210, inter 18)
M = 3, R = 1.7: # of clusters = 06, iterations: 16, Modularity: 150 (intra 189, inter 39)
M = 3, R = 1.9: # of clusters = 06, iterations: 18, Modularity: 150 (intra 189, inter 39)
M = 3, R = 2.0: # of clusters = 08, iterations: 12, Modularity: 146 (intra 187, inter 41)

M = 4, R = 1.1: # of clusters = 01, iterations: 49, Modularity: 228 (intra 228, inter 0)
M = 4, R = 1.3: # of clusters = 01, iterations: 16, Modularity: 228 (intra 228, inter 0)
M = 4, R = 1.5: # of clusters = 01, iterations: 09, Modularity: 228 (intra 228, inter 0)
M = 4, R = 1.7: # of clusters = 02, iterations: 13, Modularity: 216 (intra 222, inter 6)
M = 4, R = 1.9: # of clusters = 06, iterations: 15, Modularity: 150 (intra 189, inter 39)
M = 4, R = 2.0: # of clusters = 06, iterations: 11, Modularity: 150 (intra 189, inter 39)

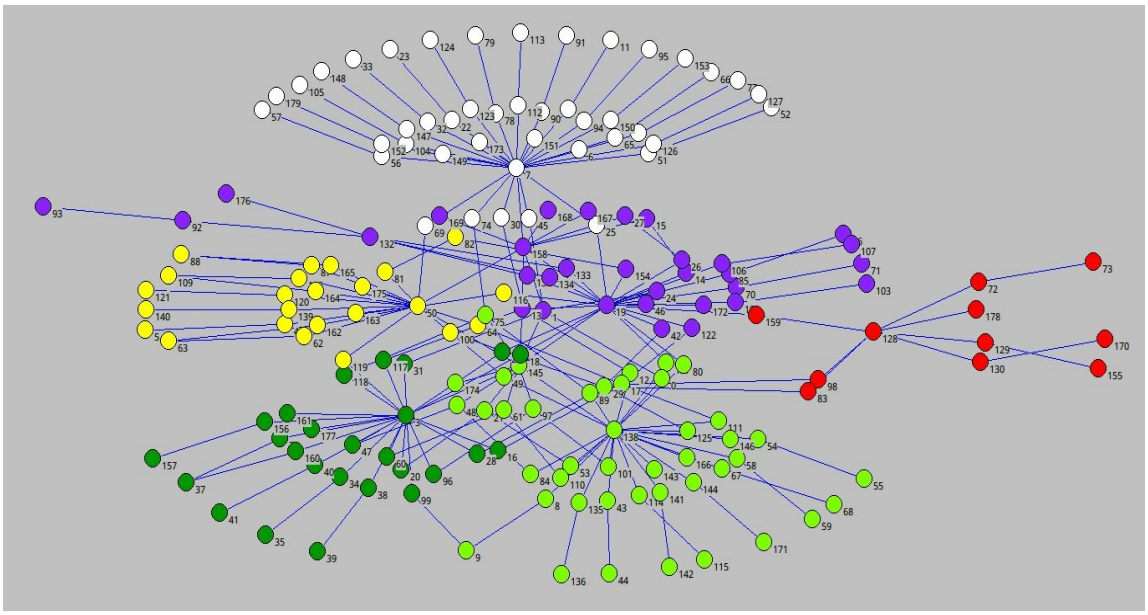
M = 5, R = 1.1: # of clusters = 01, iterations: 50 -- **No convergence**
M = 5, R = 1.3: # of clusters = 01, iterations: 17, Modularity: 228 (intra 228, inter 0)
M = 5, R = 1.5: # of clusters = 01, iterations: 09, Modularity: 228 (intra 228, inter 0)
M = 5, R = 1.7: # of clusters = 01, iterations: 08, Modularity: 228 (intra 228, inter 0)
M = 5, R = 1.9: # of clusters = 04, iterations: 11, Modularity: 186 (intra 207, inter 21)
M = 5, R = 2.0: # of clusters = 04, iterations: 15, Modularity: 180 (intra 204, inter 24)

M = 6, R = 1.1: # of clusters = 01, iterations: 50 -- **No convergence**
M = 6, R = 1.3: # of clusters = 01, iterations: 17, Modularity: 228 (intra 228, inter 0)
M = 6, R = 1.5: # of clusters = 01, iterations: 10, Modularity: 228 (intra 228, inter 0)
M = 6, R = 1.7: # of clusters = 01, iterations: 07, Modularity: 228 (intra 228, inter 0)
M = 6, R = 1.9: # of clusters = 01, iterations: 10, Modularity: 228 (intra 228, inter 0)
M = 6, R = 2.0: # of clusters = 02, iterations: 09, Modularity: 206 (intra 217, inter 11)

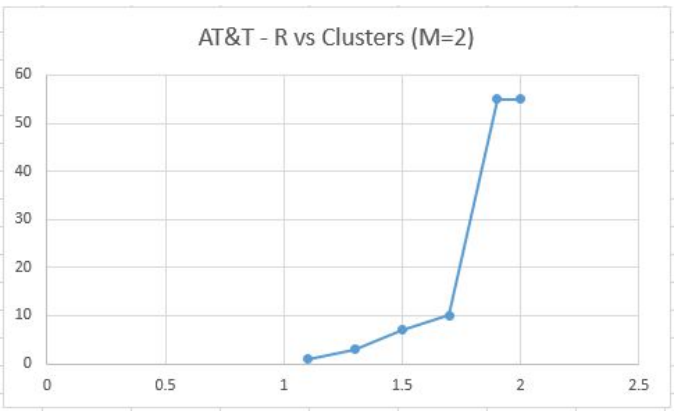
Most **optimal parameters** for the att dataset:
M = 4, R = 1.7: # of clusters = 02, iterations: 13, Modularity: 216 (intra 222, inter 6)

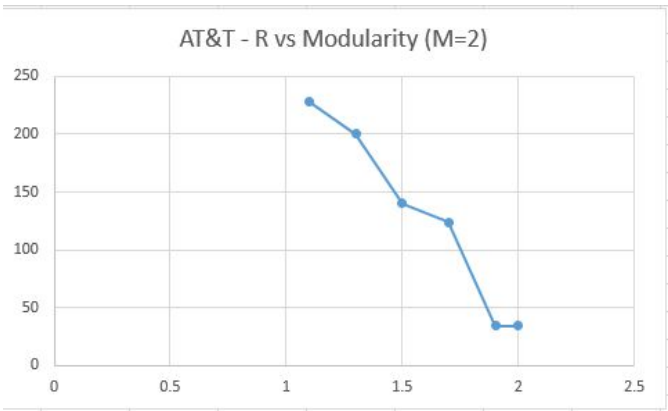


Visualization of the clusters for the **highest modularity value** (for M = 4, R = 1.7)

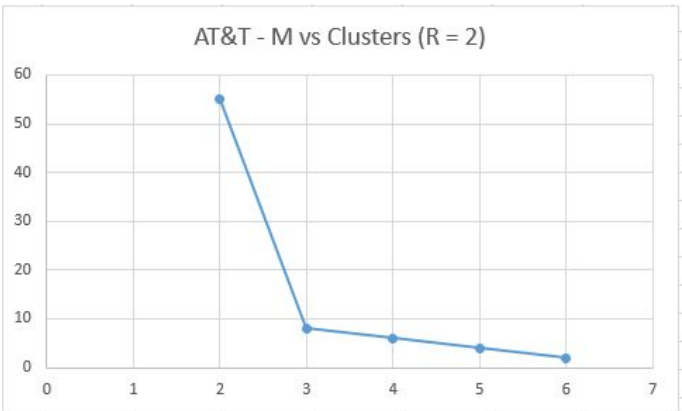


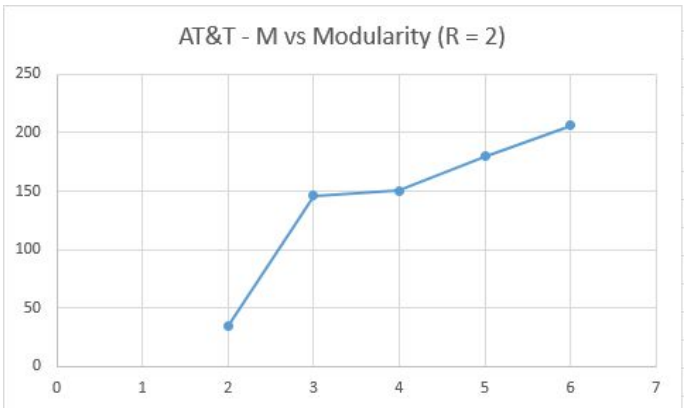
Visualization of the clusters for a moderately high modularity value, but **more clusters** (for M = 3, R = 1.7)





The number of clusters produced increases, while the modularity of the clusters produced decreases as the inflation hyperparameter R is increased, while keeping the expansion hyperparameter M constant.





The number of clusters produced reduces, while the modularity of the clusters produced increases as the expansion hyperparameter M is increased, while keeping the inflation hyperparameter R constant.

..:: **Physics collaboration dataset** ::..

M = 2, R = 1.1: # of clusters = 01, iterations: 47, Modularity: 339 (intra 339, inter 0)
M = 2, R = 1.3: # of clusters = 07, iterations: 26, Modularity: 305 (intra 322, inter 17)
M = 2, R = 1.5: # of clusters = 14, iterations: 20, Modularity: 247 (intra 293, inter 46)
M = 2, R = 1.7: # of clusters = 19, iterations: 14, Modularity: 209 (intra 274, inter 65)
M = 2, R = 1.9: # of clusters = 24, iterations: 16, Modularity: 169 (intra 254, inter 85)
M = 2, R = 2.0: # of clusters = 24, iterations: 16, Modularity: 169 (intra 254, inter 85)

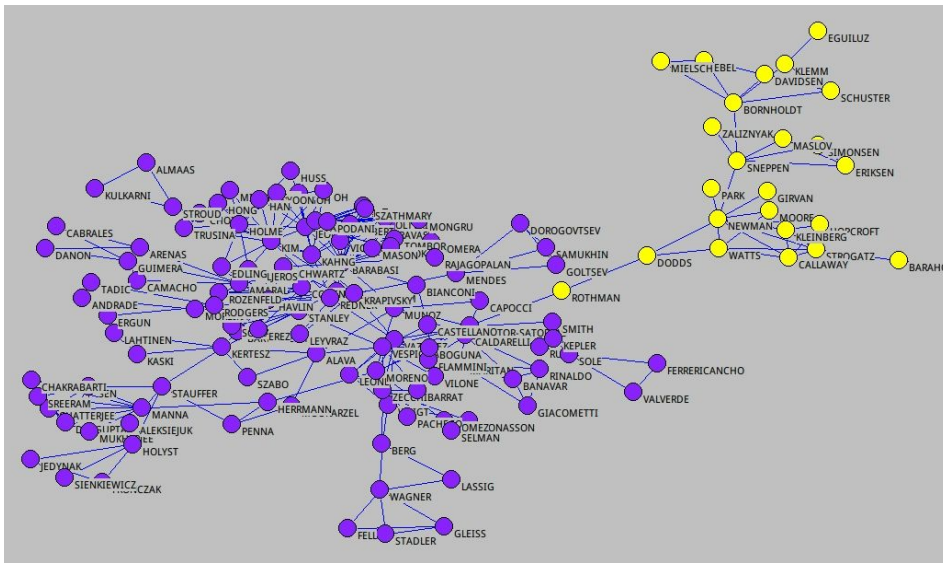
M = 3, R = 1.1: # of clusters = 01, iterations: 44, Modularity: 339 (intra 339, inter 0)
M = 3, R = 1.3: # of clusters = 02, iterations: 20, Modularity: 337 (intra 338, inter 1)
M = 3, R = 1.5: # of clusters = 06, iterations: 23, Modularity: 303 (intra 321, inter 18)
M = 3, R = 1.7: # of clusters = 10, iterations: 14, Modularity: 295 (intra 317, inter 22)
M = 3, R = 1.9: # of clusters = 12, iterations: 13, Modularity: 267 (intra 303, inter 36)
M = 3, R = 2.0: # of clusters = 14, iterations: 16, Modularity: 247 (intra 293, inter 46)

M = 4, R = 1.1: # of clusters = 01, iterations: 43, Modularity: 339 (intra 339, inter 0)
M = 4, R = 1.3: # of clusters = 01, iterations: 21, Modularity: 339 (intra 339, inter 0)
M = 4, R = 1.5: # of clusters = 03, iterations: 18, Modularity: 331 (intra 335, inter 4)
M = 4, R = 1.7: # of clusters = 06, iterations: 17, Modularity: 305 (intra 322, inter 17)
M = 4, R = 1.9: # of clusters = 09, iterations: 17, Modularity: 299 (intra 319, inter 20)
M = 4, R = 2.0: # of clusters = 09, iterations: 11, Modularity: 299 (intra 319, inter 20)

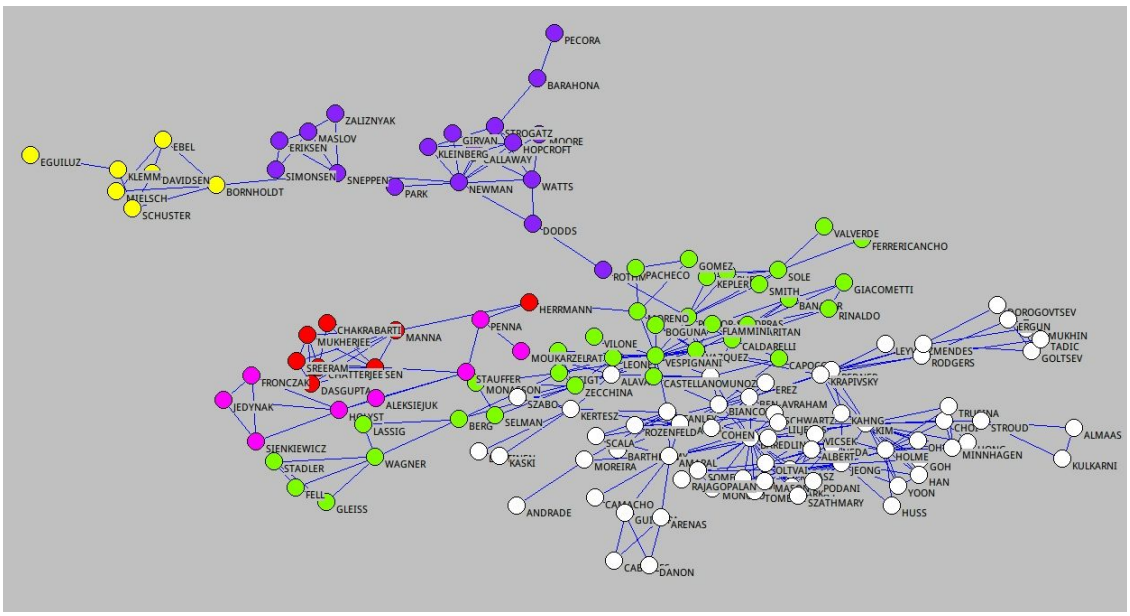
M = 5, R = 1.1: # of clusters = 01, iterations: 43, Modularity: 339 (intra 339, inter 0)
M = 5, R = 1.3: # of clusters = 01, iterations: 20, Modularity: 339 (intra 339, inter 0)
M = 5, R = 1.5: # of clusters = 02, iterations: 13, Modularity: 337 (intra 338, inter 1)
M = 5, R = 1.7: # of clusters = 03, iterations: 14, Modularity: 331 (intra 335, inter 4)
M = 5, R = 1.9: # of clusters = 06, iterations: 18, Modularity: 305 (intra 322, inter 17)
M = 5, R = 2.0: # of clusters = 08, iterations: 11, Modularity: 297 (intra 318, inter 21)

M = 6, R = 1.1: # of clusters = 01, iterations: 43, Modularity: 339 (intra 339, inter 0)
M = 6, R = 1.3: # of clusters = 01, iterations: 18, Modularity: 339 (intra 339, inter 0)
M = 6, R = 1.5: # of clusters = 02, iterations: 13, Modularity: 337 (intra 338, inter 1)
M = 6, R = 1.7: # of clusters = 03, iterations: 14, Modularity: 331 (intra 335, inter 4)
M = 6, R = 1.9: # of clusters = 05, iterations: 16, Modularity: 315 (intra 327, inter 12)
M = 6, R = 2.0: # of clusters = 05, iterations: 11, Modularity: 313 (intra 326, inter 13)

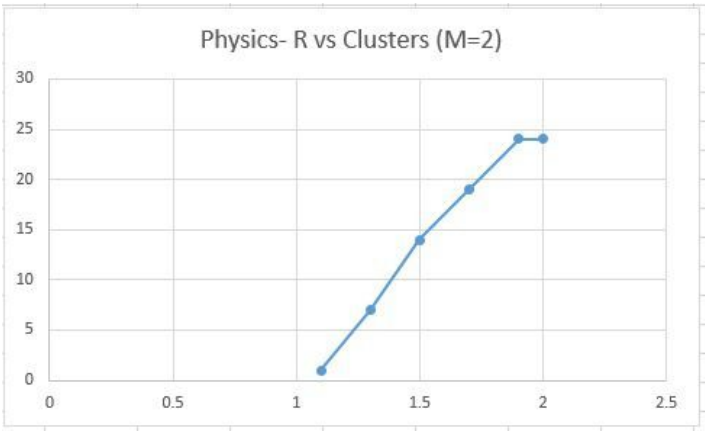
Most **optimal parameters** for the physics dataset:
M = 3, R = 1.3: # of clusters = 02, iterations: 20, Modularity: 337 (intra 338, inter 1)

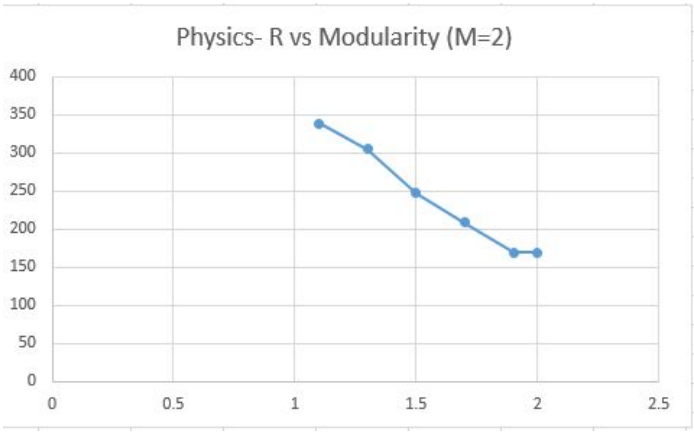


Visualization of the clusters for the **highest modularity** value (for M = 3, R = 1.3)

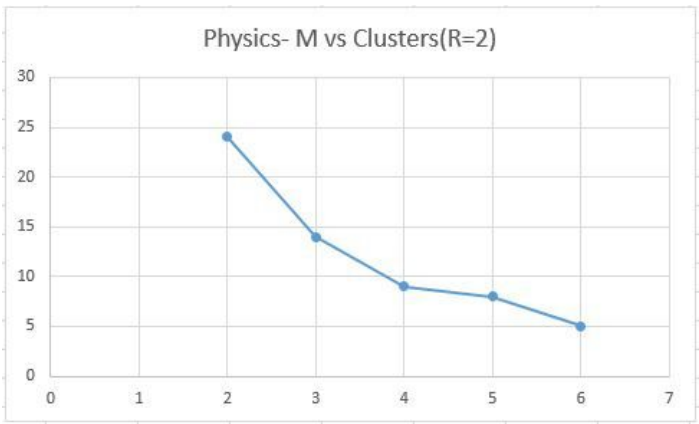


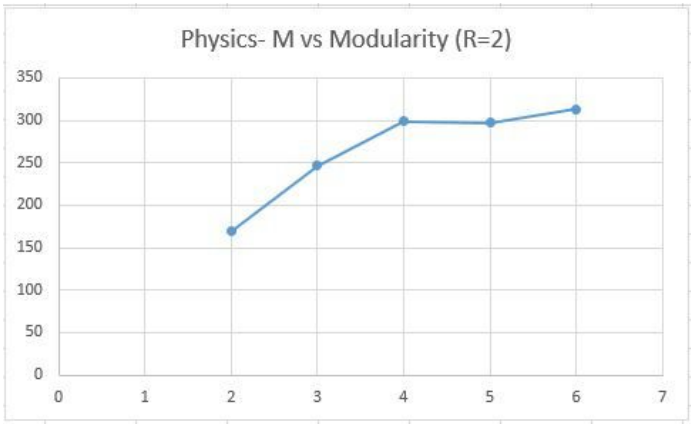
Visualization of the clusters for a moderately high modularity value, but **more clusters** (for M = 3, R = 1.5)





The number of clusters produced increases, while the modularity of the clusters produced decreases as the inflation hyperparameter R is increased, while keeping the expansion hyperparameter M constant.





The number of clusters produced reduces, while the modularity of the clusters produced increases as the expansion hyperparameter M is increased, while keeping the inflation hyperparameter R constant.

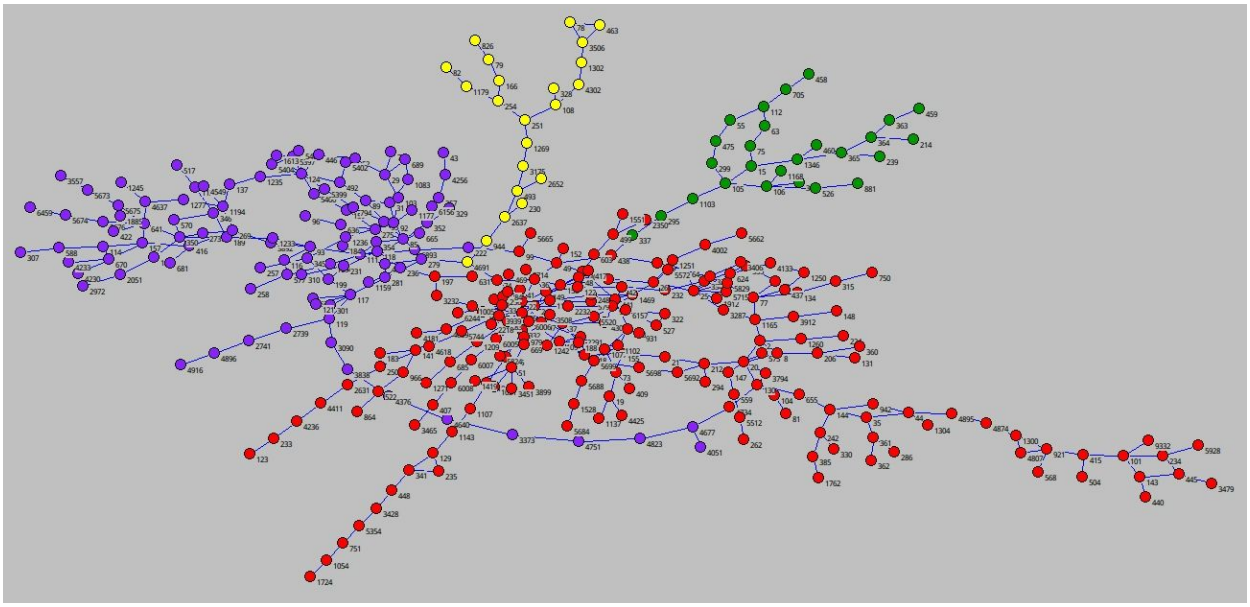
..:: **Yeast metabolic dataset** ::..
M = 2, R = 1.1: # of clusters = 01, iterations: 48, Modularity: 435 (intra 435, inter 0)
M = 2, R = 1.3: # of clusters = 22, iterations: 34, Modularity: 341 (intra 388, inter 47)
M = 2, R = 1.5: # of clusters = 52, iterations: 35, Modularity: 251 (intra 343, inter 92)
M = 2, R = 1.7: # of clusters = 84, iterations: 27, Modularity: 175 (intra 305, inter 130)
M = 2, R = 1.9: # of clusters = 105, iterations: 29, Modularity: 127 (intra 281, inter 154)
M = 2, R = 2.0: # of clusters = 116, iterations: 29, Modularity: 101 (intra 268, inter 167)
M = 3, R = 1.1: # of clusters = 01, iterations: 51 -- **No convergence**
M = 3, R = 1.3: # of clusters = 04, iterations: 44, Modularity: 421 (intra 428, inter 7)
M = 3, R = 1.5: # of clusters = 19, iterations: 32, Modularity: 341 (intra 388, inter 47)
M = 3, R = 1.7: # of clusters = 42, iterations: 19, Modularity: 267 (intra 362, inter 95)
M = 3, R = 1.9: # of clusters = 50, iterations: 19, Modularity: 255 (intra 345, inter 90)
M = 3, R = 2.0: # of clusters = 53, iterations: 20, Modularity: 247 (intra 341, inter 94)

M = 4, R = 1.1: # of clusters = 01, iterations: 51 -- **No convergence**
M = 4, R = 1.3: # of clusters = 01, iterations: 15, Modularity: 435 (intra 435, inter 0)
M = 4, R = 1.5: # of clusters = 09, iterations: 21, Modularity: 385 (intra 410, inter 25)
M = 4, R = 1.7: # of clusters = 20, iterations: 21, Modularity: 333 (intra 384, inter 51)
M = 4, R = 1.9: # of clusters = 32, iterations: 15, Modularity: 305 (intra 370, inter 65)
M = 4, R = 2.0: # of clusters = 38, iterations: 14, Modularity: 291 (intra 363, inter 72)

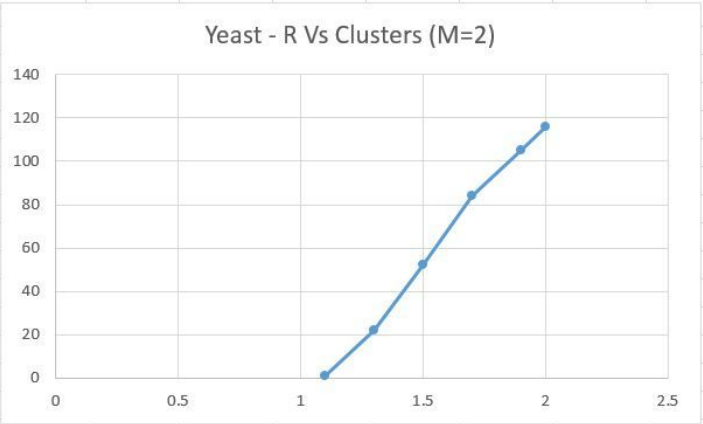
M = 5, R = 1.1: # of clusters = 01, iterations: 51 -- **No convergence**
M = 5, R = 1.3: # of clusters = 01, iterations: 19, Modularity: 435 (intra 435, inter 0)
M = 5, R = 1.5: # of clusters = 05, iterations: 19, Modularity: 419 (intra 427, inter 8)
M = 5, R = 1.7: # of clusters = 10, iterations: 22, Modularity: 373 (intra 404, inter 31)
M = 5, R = 1.9: # of clusters = 20, iterations: 22, Modularity: 333 (intra 384, inter 51)
M = 5, R = 2.0: # of clusters = 23, iterations: 15, Modularity: 341 (intra 388, inter 47)

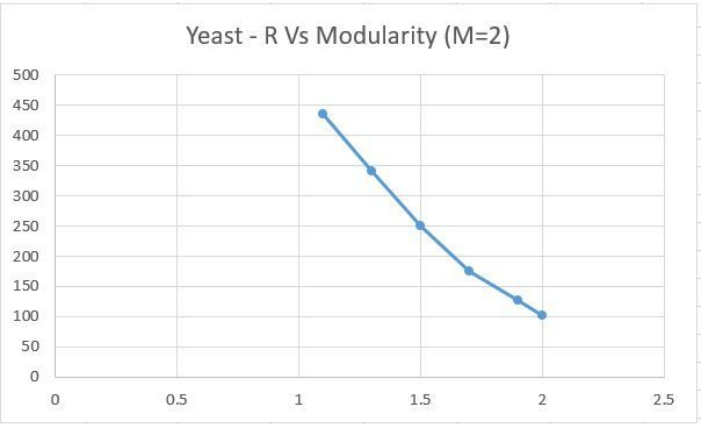
M = 6, R = 1.1: # of clusters = 01, iterations: 48, Modularity: 435 (intra 435, inter 0)
M = 6, R = 1.3: # of clusters = 01, iterations: 21, Modularity: 435 (intra 435, inter 0)
M = 6, R = 1.5: # of clusters = 04, iterations: 16, Modularity: 423 (intra 429, inter 6)
M = 6, R = 1.7: # of clusters = 09, iterations: 18, Modularity: 385 (intra 410, inter 25)
M = 6, R = 1.9: # of clusters = 14, iterations: 20, Modularity: 363 (intra 399, inter 36)
M = 6, R = 2.0: # of clusters = 19, iterations: 13, Modularity: 343 (intra 389, inter 46)

Most **optimal parameters** for the yeast dataset:
M = 6, R = 1.5: # of clusters = 04, iterations: 16, Modularity: 423 (intra 429, inter 6)

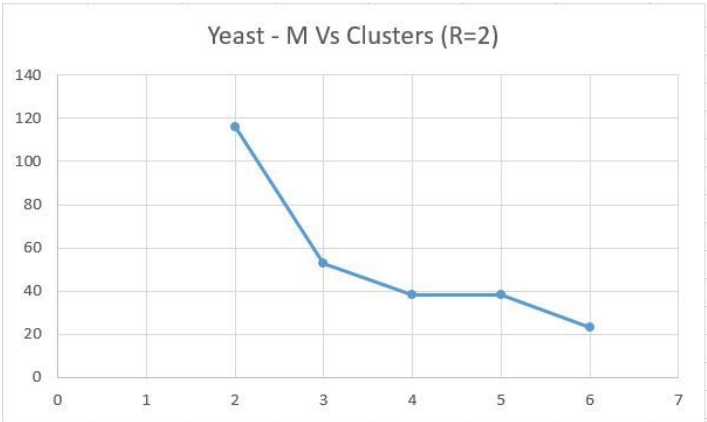


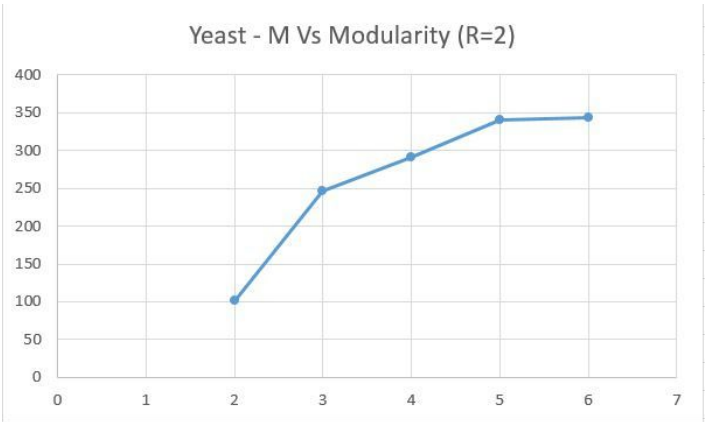
Visualization of the clusters for the **highest modularity value** (for M =6, R = 1.5)





The number of clusters produced increases, while the modularity of the clusters produced decreases as the inflation hyperparameter R is increased, while keeping the expansion hyperparameter M constant.





The number of clusters produced reduces, while the modularity of the clusters produced increases as the expansion hyperparameter M is increased, while keeping the inflation hyperparameter R constant.

Analysis of experiment results

Observation for hyper-parameter R

As observed in the graphs above, as R is increased from 1.1 to 2.0, while keeping the expansion hyperparameter M constant, it was observed for every dataset that the size of the clusters formed became **smaller**, and the number of clusters formed grew larger. The modularity on the other hand was observed to be decreasing.

Observation for hyper-parameter M

When the expansion hyperparameter M is increased, while keeping the inflation hyperparameter R constant, two observations were made for all of the datasets:

1. The number of clusters produced reduces,

2. The modularity of the clusters produced increases

Overlapping clusters

For M = 3 , R = 1.7 it was observed that 2 of the clusters had overlapping values:
[9, 10, 11, 15, 16, 17, 19, 20], and
[12, 13, 14, 15, 16, 17, 19, 20]

For such an overlap scenario, we have simply chosen one of the clusters to assign the points, since for these simple datasets, overlaps should not be a problem.

Selecting hyperparameter values

Dataset size plays a significant role in deciding the hyper-parameter, as was evident with the yeast dataset which was of the largest size. After tuning the parameters we observed that higher values of M can give higher modularity clusters, while lower R values can have the same effect. But it must be kept in mind that only a balanced mixture of both values can give meaningful results, depending on the dataset size.

Performance improvements

We have performed **pruning** on the matrix in order to improve the performance of MCL.
We inspect the matrix and set values below a certain threshold to 0. This is because such small values would eventually become zero. This has improved the speed of the algorithm significantly as demonstrated below.

Data Set	Without Pruning (secs)	With Pruning (secs)	% Improvement
AT&T	25.603	20.2578	20.88%
PHYSICS	15.799	11.941	24.42%
YEAST	221.409	198.140	10.51%

In addition to pruning, we have also used a **tolerance** measure while checking for convergence. We have done this in the *hasConverged* function with a configurable tolerance so if we observed the transitional matrix M to be changing only by a very minute margin, we declare it as converged.

This helps cut down on the number of iterations it takes for the algorithm to converge.

Conclusion

After implementing the Markov clustering algorithm we have understood its ability to simulate random walks to gradually learn partitions in a graph. The algorithm was applied on the 3 datasets, and the impact of the hyper parameters R and M was observed to be quite similar for all datasets. The number of clusters produced increases, while the modularity of the clusters produced decreases as the inflation hyperparameter R is increased, while keeping the expansion hyperparameter M constant. And, the number of clusters produced increases, while the modularity of the

clusters produced decreases as the inflation hyperparameter R is increased, while keeping the expansion hyperparameter M constant. We also saw an improvement in the performance when we implemented pruning.