Homework 3: Clustering Analysis for Complex Networks

CSE 601: Data Mining and Bioinformatics

AVIJEET MISHRA
(AVIJEETM@BUFFALO.EDU)
UB#:50169242

PRITHVI GOLLU INDRAKUMAR
(PGOLLUIN@BUFFALO.EDU)
UB#:50169089

DEPT OF COMPUTER SCIENCE,
UNIVERSITY AT BUFFALO

Contents

Contents

Introduction

Clustering is a process of grouping similar set of objects such that objects in the same cluster belong are more similar than with objects of other cluster. It can be used in image processing, pattern recognition, spatial data analysis, etc. as a preprocessing. We have implemented Markov Clustering Algorithm and applied it to the provided three datasets of AT&T Web network, physics collaboration network, and yeast metabolic network.

The given three datasets “attweb_net”, “physics_collaboration_net” and “yeast_undirected_metabolic” representing nodes and edges. All rows have 2 columns and each row represent an edge. In the above mentioned algorithms, we are grouping similar nodes (computer for AT&T Web network, physicist for physics collaboration network and biochemical compound for yeast metabolic network) into the same cluster based on the edges denoted in file. The main aim here is to generate .clu files containing cluster results by discovering clusters using MCL Algorithm.

Implementation

The MCL algorithm is implemented in C# using Microsoft Visual Studio as a platform. The data is read from the Excel files of the datasets. MCL is based on the concept of random walks. Concept of random walks states that in a graph, there will be many links within a cluster, and fewer links between clusters.

This means if we start at a node, and then randomly travel to a connected node, we are more likely to stay within a cluster than travel between different clusters. By doing random walks upon the graph, it may be possible to discover where the flow tends to gather, and therefore, where clusters are. Random Walks on a graph are calculated using "Markov Chains".

In this algorithm, Input is an un-directed graph, Expansion parameter e, and inflation parameter r. Initially associated matrix is created by assigning 1 to all the nodes which have edges in between. One of the issue observed here is that there is a strong effect that odd powers of expansion obtain their mass from simple paths of odd length, and likewise for even. So for this issue, self-loop has been used which removes the appearance of mass at odd powers of the matrix. So basically we have to add an edge at each node of value 1 to itself.

Code snippet for creating associated matrix and addition of self-loop.

```
double[,] final1 = new double[count, count];
//self loop
for (int i = 1; i < count; i++)
{
    for (int j = 1; j < count; j++)
    {
        if (i == j)
        {
            final[i, j] = 1;
        }
        else if (data[i, j] == null)
        {
            final[i, j] = 0;
        }
        else
        {
            final[i, j] = Convert.ToDouble(data[i, j]);
        }
    }

}
```

After that matrix is normalized across each column. Then comes the major two steps in this algorithm named as “Expansion” and “Inflation”. Expansion is just calculating the e^{th} power of matrix (calculated in previous step after adding self-loop) and after Inflation takes place.

Code snippet for expansion

```
//expand
            for (int i = 1; i < count; i++)
            {
                for (int j = 1; j < count; j++)
                {
                    final1[i, j] = 0;
                    for (int k = 1; k < count; k++)
                    {
                        final1[i, j] += final[i, k] * final[k, j];

                    }
                }

            }
            int p = power;
            while (p > 2)
            {
                double[,] temp = new double[count, count];
                for (int i = 1; i < count; i++)
                {
                    for (int j = 1; j < count; j++)
                    {
                        temp[i, j] = 0;
                        for (int k = 1; k < count; k++)
                        {
                            temp[i, j] += final1[i, k] * final1[k, j];
                        }
                    }
                }
                for (int i = 1; i < count; i++)
                {
                    for (int j = 1; j < count; j++)
                    {
                        final1[i, j] = temp[i, j];
                        final[i, j] = temp[i, j];
                    }
                }
                p--;

            }
```

Flow is easier within dense regions than across sparse boundaries, however, in the long run this effect disappears. During the earlier powers of the Markov Chain, the edge weights will be higher in links that are within clusters, and lower between the clusters. This means there is a correspondence between the distribution of weight over the columns and the clustering.

MCL deliberately boosts this affect by stopping partway in the Markov Chain and then adjusting the transitions by columns. For each vertex, the transition values are changed so that strong neighbors are further strengthened and less popular neighbors are demoted. This adjusting can be done by raising a single column to a non-negative power, and then re-normalizing. This operation is known as “Inflation”.

Code snippet for normalization

```
//normalize the matrix
            for (int i = 1; i < count; i++)
            {
                double sum = 0;
                for (int j = 1; j < count; j++)
                {
                    sum = sum + final[j, i];
                }
                for (int j = 1; j < count; j++)
                {
                    final[j, i] = final[j, i]/sum;
                }

            }
```

The inflation operator is responsible for both strengthening and weakening of current. (Strengthens strong currents, and weakens already weak currents). The inflation parameter, r, controls the extent of this strengthening / weakening which basically influences the granularity of clusters.

Code snippet for Inflation

```
// inflate
            int f = inflate;
            while (f != 1)
            {
                for (int i = 1; i < count; i++)
                {
                    for (int j = 1; j < count; j++)
                    {
                        final[i, j] = final1[i, j] * final[i, j];
                    }
                }
                f--;

            }
```

So Expansion and Inflation is repeated recursively until a steady state is achieved i.e. matrix is converged. Convergence is not proven in the thesis, however it is shown experimentally that it often does occur. In practice, the algorithm converges nearly always to a "doubly idempotent" matrix which is at a steady state and homogenous i.e. every value in a single column has the same number.

Code snippet for convergence

```
//Check for Convergence
            int t = 0;
            for (int i = 1; i < count; i++)
            {

                for (int j = 1; j < count; j++)
                {
                    if (final[i, j] == test[i, j])
                    {
                        t++;
                    }
                    else
                    {
                        test[i, j] = final[i, j];
                    }
                }
            }
            itr++;
            if (t != ((count - 1) * (count - 1)))
            {
                final = expand_inflate(final, test);
            }

            return final;
```

Now last step is to interpret matrix result to discover clusters. To interpret clusters, the vertices are split into two types. Attractors, which attract other vertices, and vertices that are being attracted by the attractors. Attractors have at least one positive flow value within their corresponding row (in the steady state matrix). Each attractor is attracting the vertices which have positive values within its row. Attractors and the elements they attract are swept together into the same cluster. So finally a cluster ".clu" file is generated as the output of the program which would contain the cluster number in each row corresponding to a node.

Code snippet for cluster discovery

```
//Cluster Discovery
            string[] clu = new string[count];
            int cluster = 1;
            for (int i = 1; i < count; i++)
            {
                int flag = 0;
                double temp = 999;
                int temp_flag = 0;
                for (int j = 1; j < count; j++)
                {
                    if (final[i, j] != 0 && temp_flag == 0)
                    {
                        temp = final[i, j];
                        temp_flag = 1;
                    }
                    if (final[i, j] == temp)
                    {
                        if (clu[j] == null)
                        {
                            clu[j] = cluster.ToString();
                            flag = 1;
                        }

                    }
                }
                if (flag == 1)
                {
                    cluster++;
                }

            }
clu[0] = "*Vertices " + (count - 1).ToString();

System.IO.File.WriteAllLines(@"E:\data mining\hw3\" + name + "power_" + power + "inflate_"
+ inflate + ".clu", clu);
```

Analysis of Algorithms

Each step of the algorithm except for expansion has a complexity of O(n^2). That is creating the associated matrix, adding self-loops to each node, normalizing the matrix and Inflation can be done in O (n^2) time. The number of steps to converge is on an average takes 10-50 iterations. Expansion of the matrix by taking the eth power of the matrix takes O(n^3) time. Here n is the number of vertices, therefore in matrix expansion n^3 is the cost of multiplying 2 matrices of n dimension. So the overall runtime complexity of MCL Algorithm is O(n^3) + O(n^2) + O(n^2) + O(n^2) + O(n^2). Therefore the asymptotic time complexity of MCL Algorithm is O(n^3).

Fine Tuning of parameters

For fine-tuning the algorithm and to get the most optimized solution, we have tested the algorithm for different combinations of Inflation and Expansion parameter. Below is the table which gives the number of clusters the network is divided in to, number of iterations taken and time taken of converge for different combinations of inflation and power parameters. Test cases with row in color are used to visualize using Pajek.

File	Inflation	Expansion	Number of Clusters	Number of Iterations	Time Taken(in ms)
attwb_net	2	2	55	19	5491
attwb_net	2	3	6	17	8437
attwb_net	2	4	1	12	9139
attwb_net	3	2	66	13	3828
attwb_net	3	3	9	13	5553
attwb_net	3	4	6	10	6853
attwb_net	4	2	66	10	2029
attwb_net	4	3	12	10	4193
attwb_net	4	4	6	10	6630
attwb_net	4	5	3	8	4614
attwb_net	5	2	66	9	1903
attwb_net	5	5	4	7	2238

attwb_net	6	5			
physics_collaboration_net	2	2	24	22	2319
physics_collaboration_net	2	3	9	17	3455
physics_collaboration_net	2	4	3	34	5169
physics_collaboration_net	2	6	1	14	7102
physics_collaboration_net	3	2	33	12	1200
physics_collaboration_net	3	3	14	23	2223
physics_collaboration_net	3	4	8	35	3682
physics_collaboration_net	3	5	4	12	5085
physics_collaboration_net	4	2	34	10	1009
physics_collaboration_net	4	3	20	9	1869
physics_collaboration_net	4	4	9	9	2798
physics_collaboration_net	4	5	6	10	2060
physics_collaboration_net	5	2	41	18	912
physics_collaboration_net	4	5	6	10	4133
physics_collaboration_net	4	6	3	11	3351
physics_collaboration_net	5	5	6	8	1567
physics_collaboration_net	6	6	4	9	2307
physics_collaboration_net	7	6	4	8	2558
yeast_undirected_metabolic	2	2	116	35	95726
yeast_undirected_metabolic	2	3	37	55	12874
yeast_undirected_metabolic	2	4	10	21	157931
yeast_undirected_metabolic	2	5	4	16	188866
yeast_undirected_metabolic	3	2	177	41	86706
yeast_undirected_metabolic	3	3	67	18	67226
yeast_undirected_metabolic	3	4	34	14	83916
yeast_undirected_metabolic	3	5	11	13	148879
yeast_undirected_metabolic	3	6	7	12	152299
yeast_undirected_metabolic	3	7	5	12	136625
yeast_undirected_metabolic	4	2	210	33	60506
yeast_undirected_metabolic	4	3	86	17	60562
yeast_undirected_metabolic	4	4	45	11	61117
yeast_undirected_metabolic	4	5	19	10	78677
yeast_undirected_metabolic	4	6	10	9	100879
yeast_undirected_metabolic	4	7	7	10	117458
yeast_undirected_metabolic	4	8	5	9	121771
yeast_undirected_metabolic	5	2	224	31	54279
yeast_undirected_metabolic	5	3	103	16	56885
yeast_undirected_metabolic	5	4	54	12	63352
yeast_undirected_metabolic	5	5	29	9	86306
yeast_undirected_metabolic	5	6	12	10	93939
yeast_undirected_metabolic	5	7	8	10	112780
yeast_undirected_metabolic	5	8	5	8	107422

We observe that when the value of power parameter increases, number of cluster decreases until it reaches a threshold value beyond which it does not decrease. Whereas in the case of inflation parameter, when we increase its value, number of clusters also increases, that is it increases cluster granularity up to a threshold value. The time taken to converge is inverse of number of clusters. That is increase in power parameter value increases the time, whereas increase in inflation parameter value decreases the time.

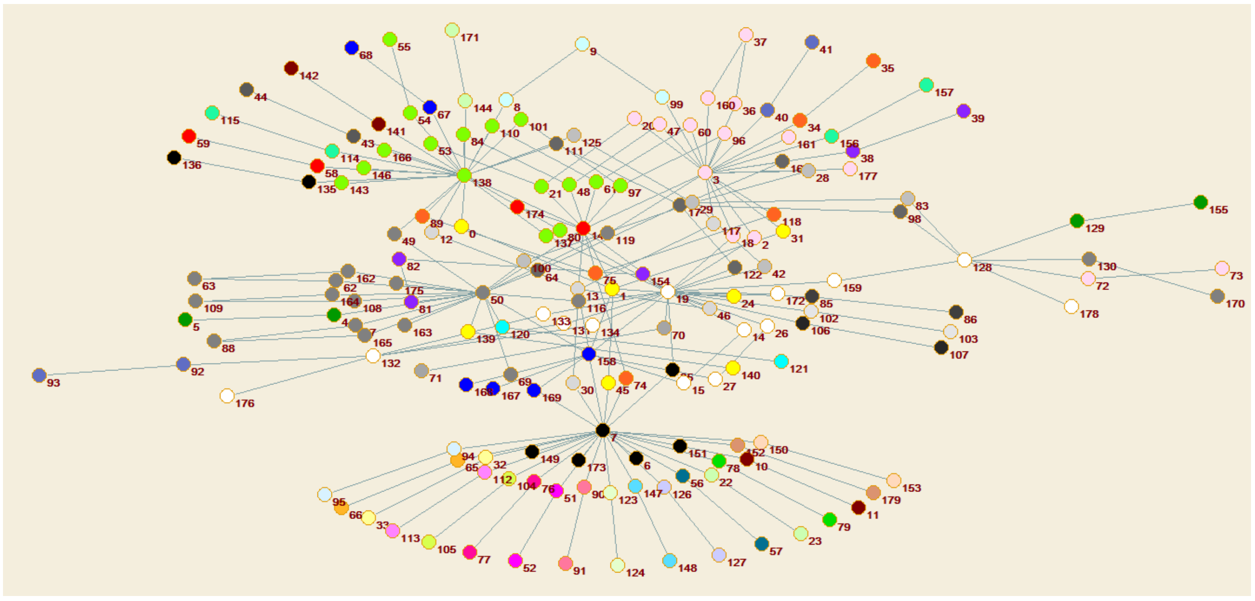
Visualization Using Pajek

Pajek is a Software package for analysis and visualization of huge networks. Here we are the provided ".net" file along with the ".clu" file which has the cluster allocation of each node are given as input. When we draw the network in ".net" file based on the partitions in the ".clu" file, we can visualize the clustering of nodes in the network and analyze it.

By comparing visualizations of different combinations of inflation and expansion parameters, we observe in image 3.1, 3.2 and 3.3 for yeast metabolic network that increasing the expansion from 2 to 4, granularity of cluster decreases from 224 to 54 and inversely, decreasing the inflation from 5 to 2, granularity of cluster increases from 54 to 10. One of the interesting observation between 3.4 and 3.5 indicates that if we are changing the parameters (inflation and expansion) and getting the same number of clusters then the assignment of nodes to a particular cluster is same for both the cluster networks. Same can be observed in other networks of AT&T Web Network and Physics collaboration network.

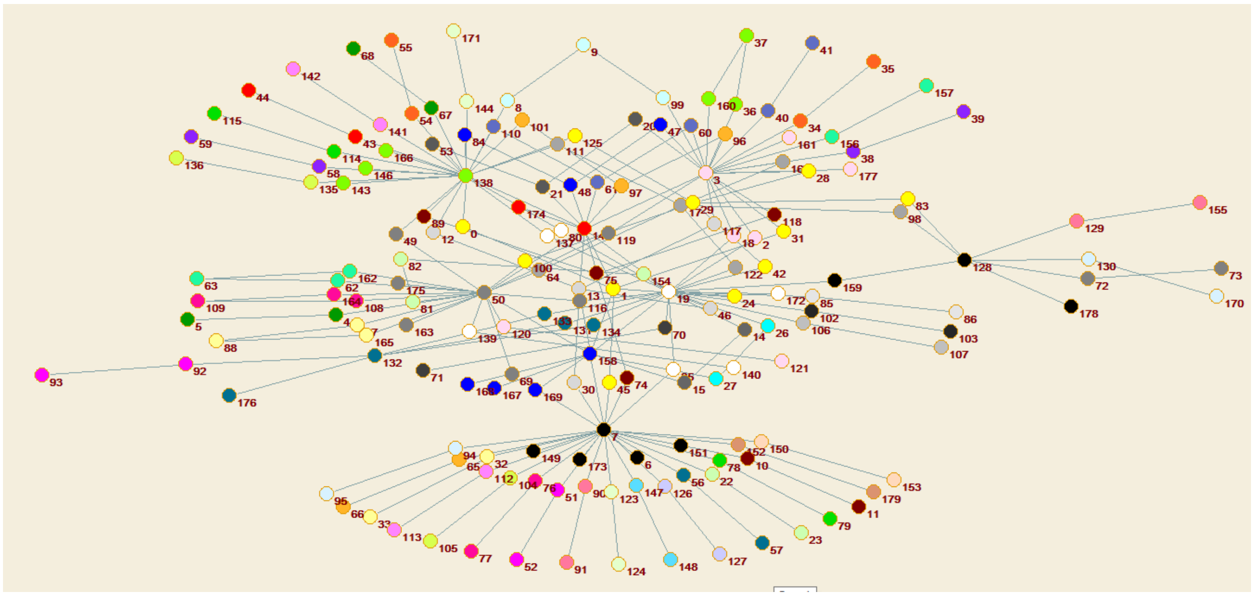
AT&T Web Network Visualization

Image 1.1



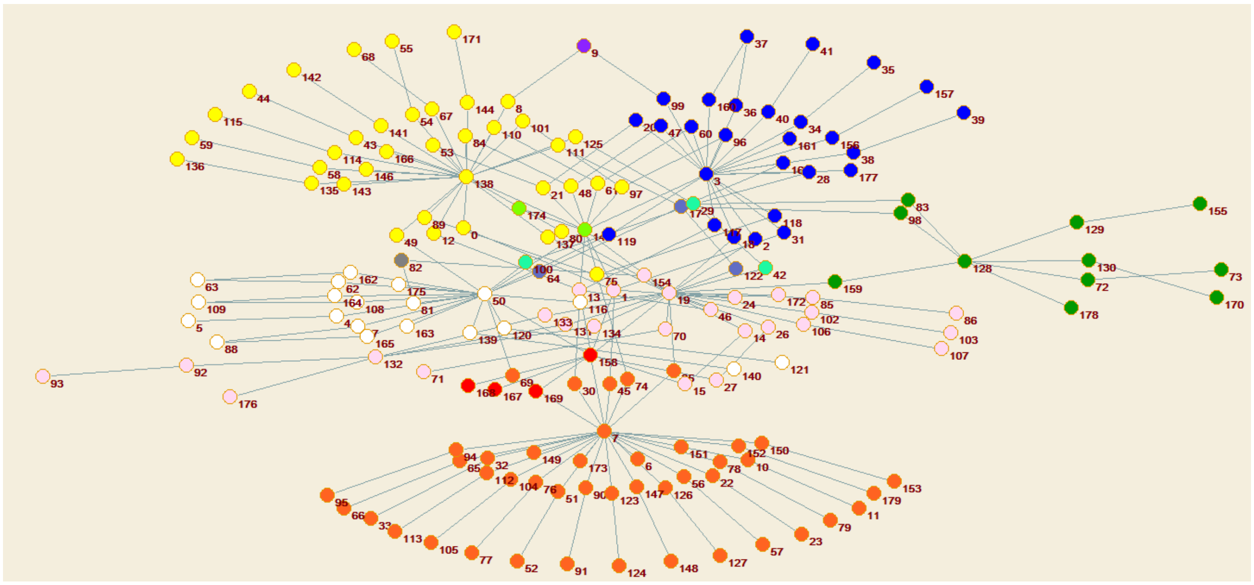
Inflation =2 Expansion=2 No. of Clusters=55 Iteration = 19 Time taken= 5491

Image 1.2



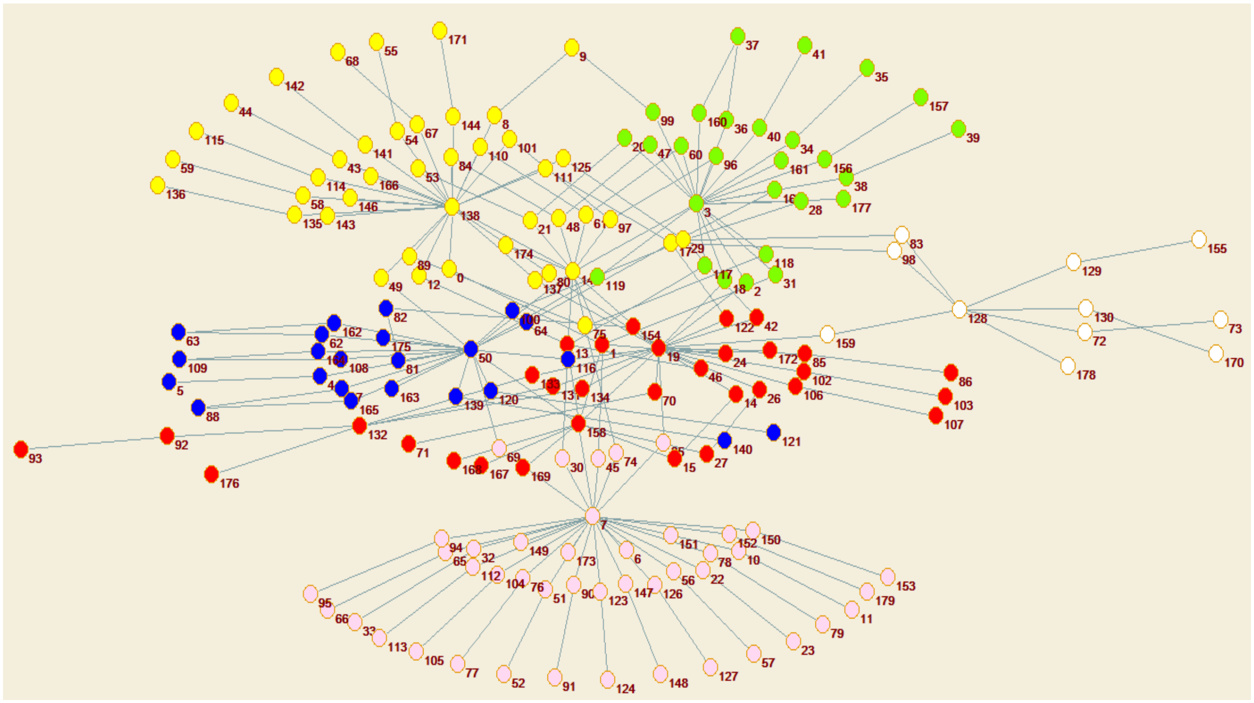
Inflation =4 Expansion=2 No. of Clusters=66 Iteration = 10 Time taken= 2029

Image 1.3



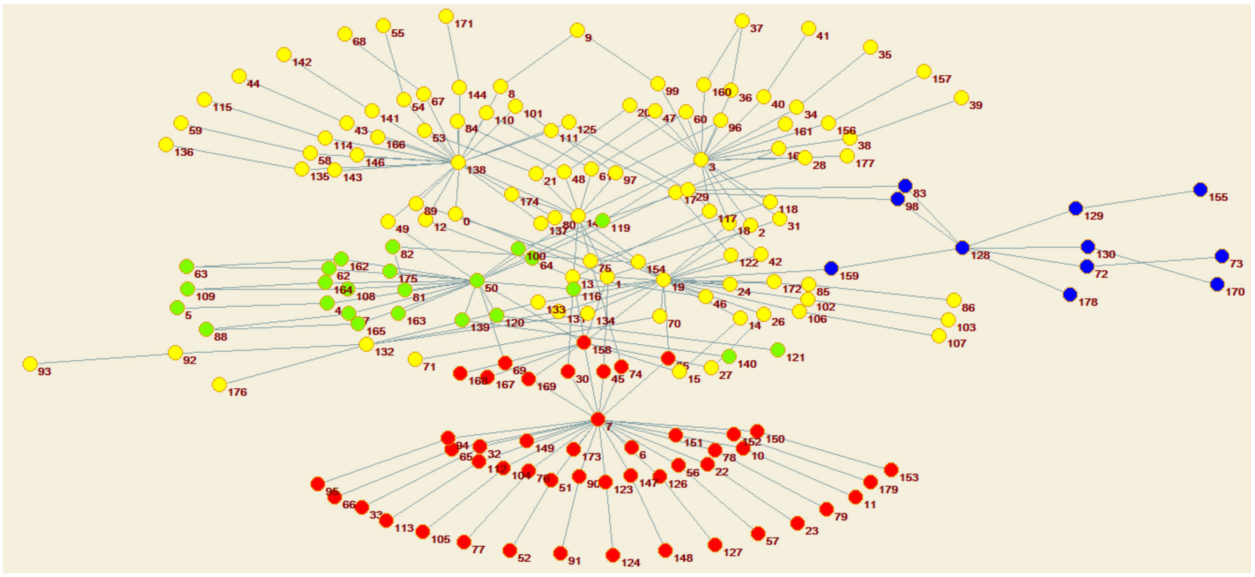
Inflation =4 Expansion=3 No. of Clusters=12 Iteration = 10 Time taken= 4193

Image 1.4



Inflation =2 Expansion=3 No. of Clusters=6 Iteration = 17 Time taken= 8437

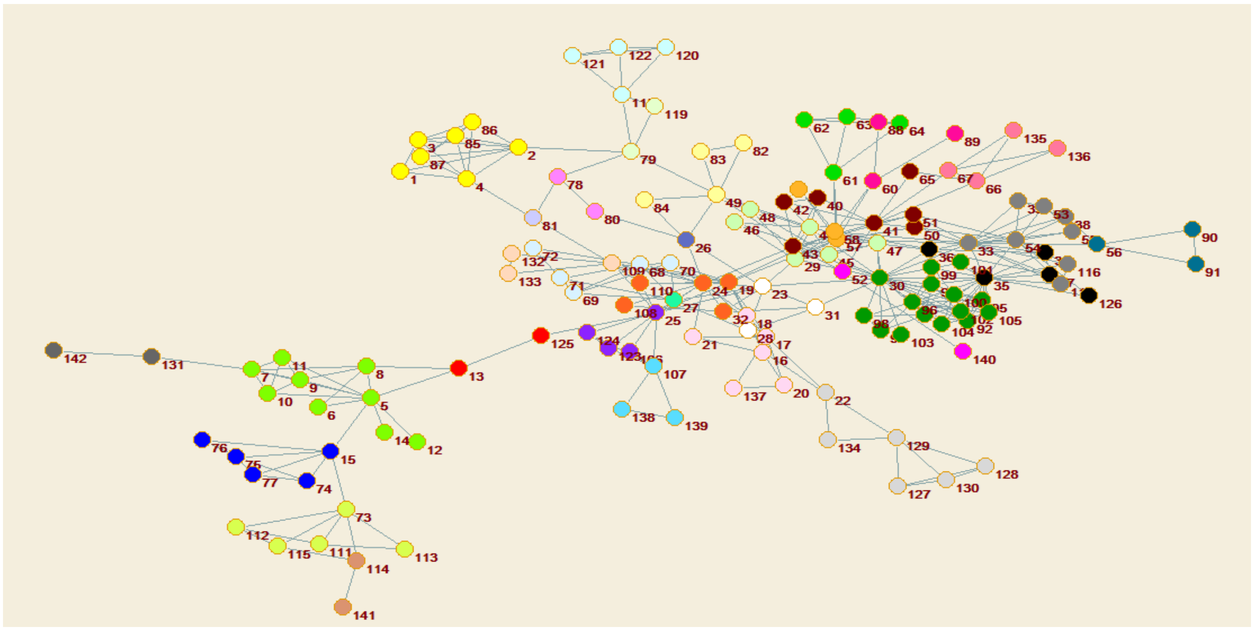
Image 1.5



Inflation =5 Expansion=5 No. of Clusters=4 Iteration = 7 Time taken= 2238

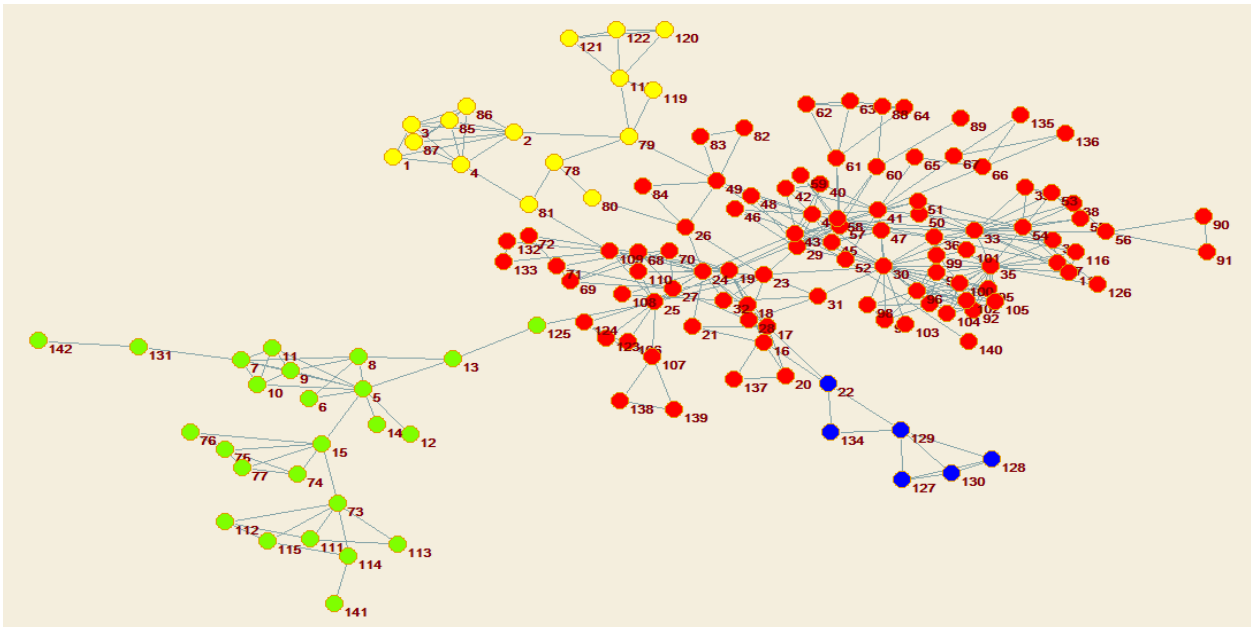
Physics Collaboration Network Visualization

Image 2.1



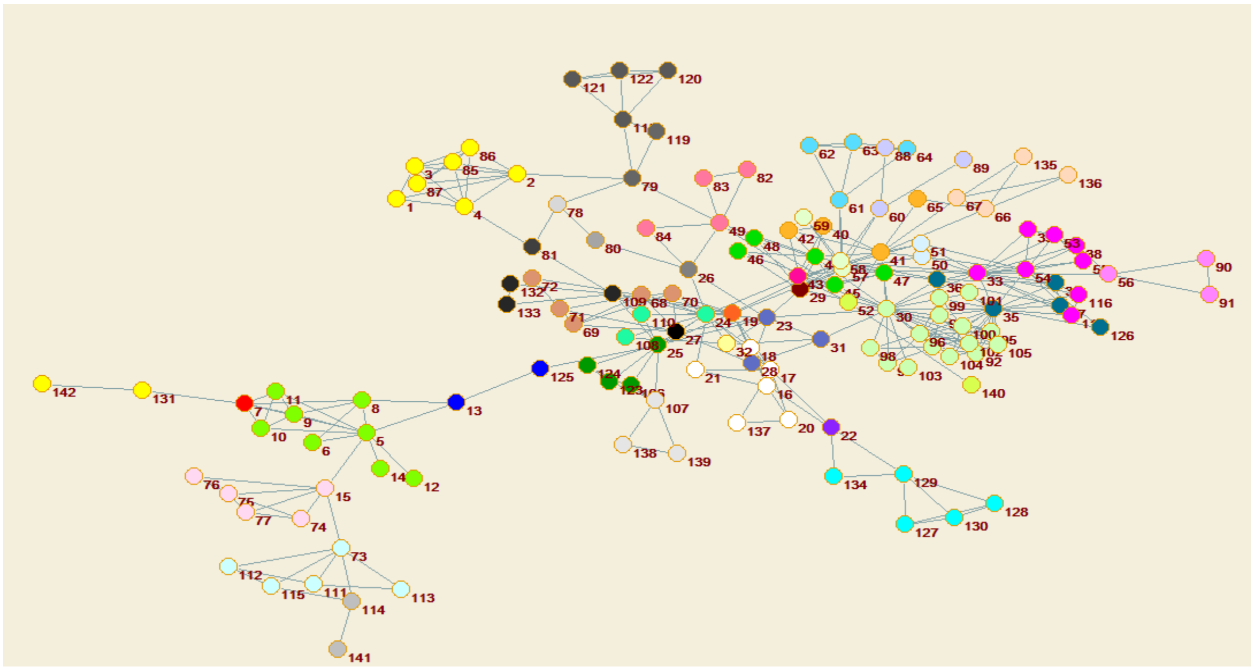
Inflation =3 Expansion=2 No. of Clusters=33 Iteration = 12 Time taken= 1200

Image 2.2



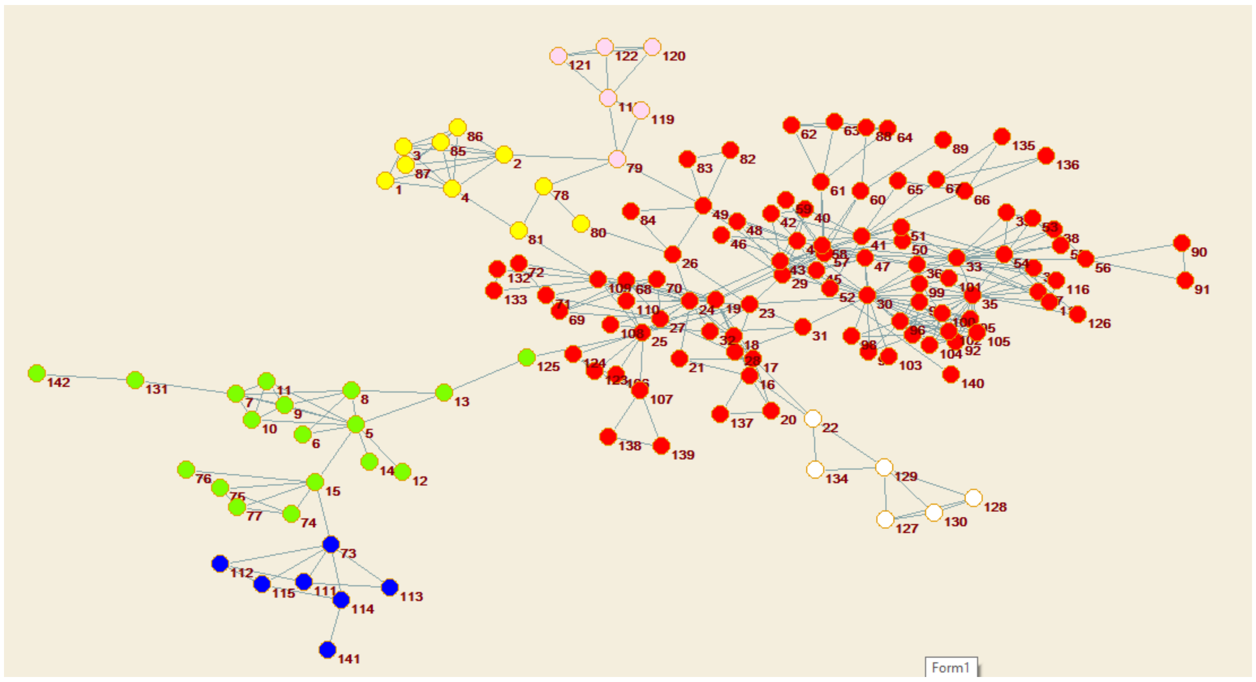
Inflation =3 Expansion=5 No. of Clusters=4 Iteration = 12 Time taken= 5085

Image 2.3



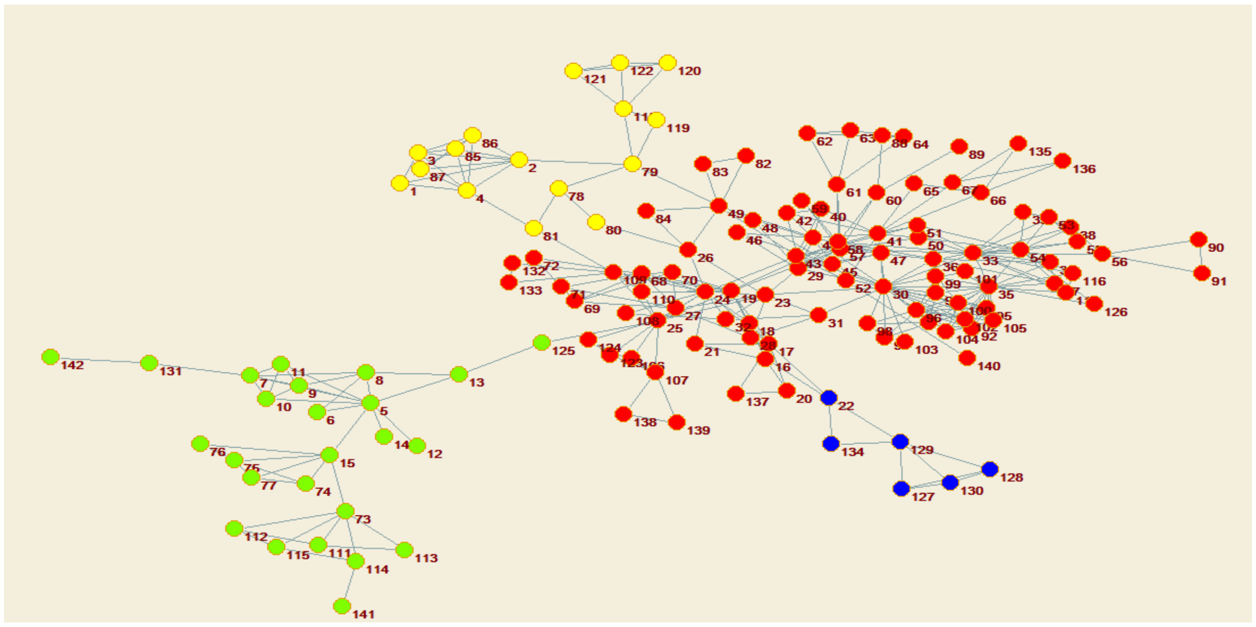
Inflation =5 Expansion=2 No. of Clusters=41 Iteration = 18 Time taken= 912

Image 2.4



Inflation =4 Expansion=5 No. of Clusters=6 Iteration = 10 Time taken= 4133

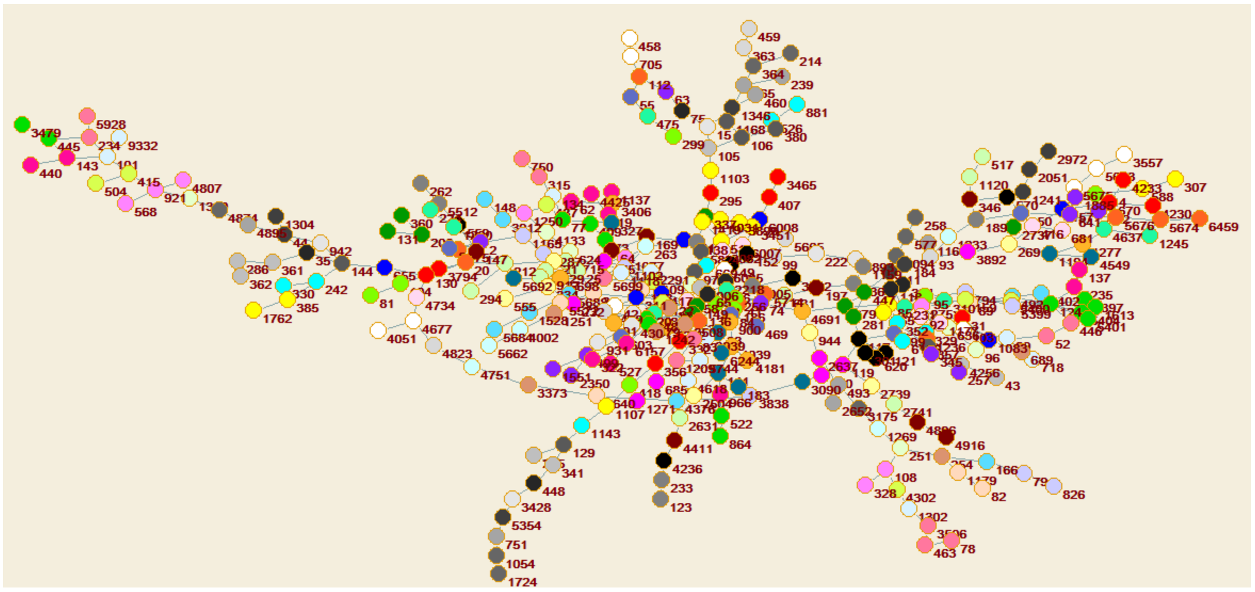
Image 2.5



Inflation =7 Expansion=6 No. of Clusters=4 Iteration = 8 Time taken= 2558

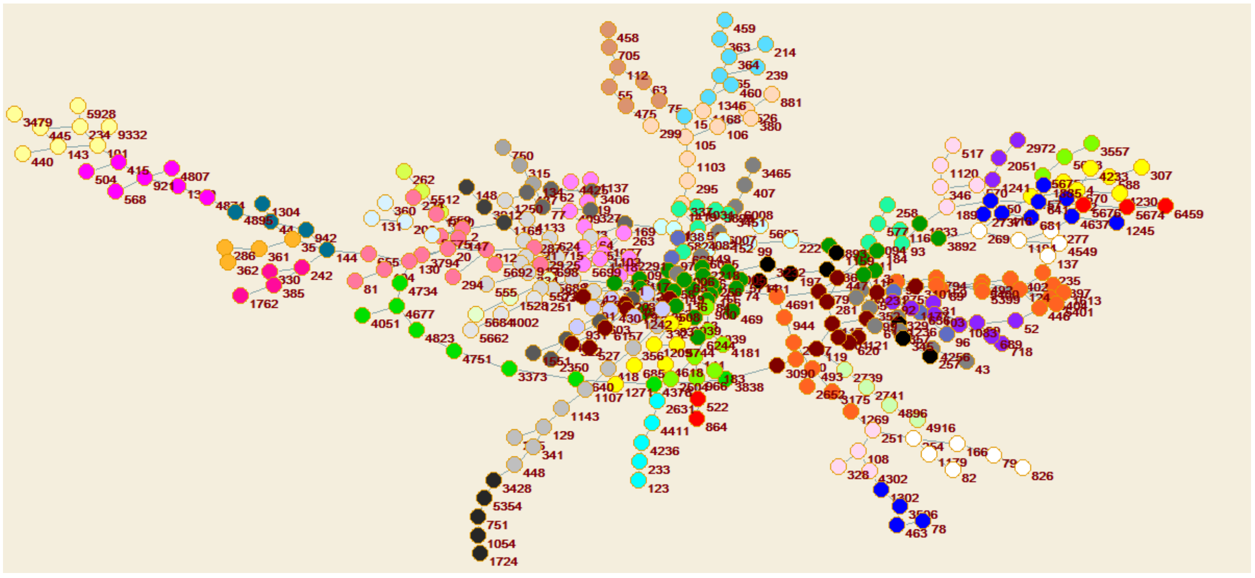
Yeast Metabolic Network Visualization

Image 3.1



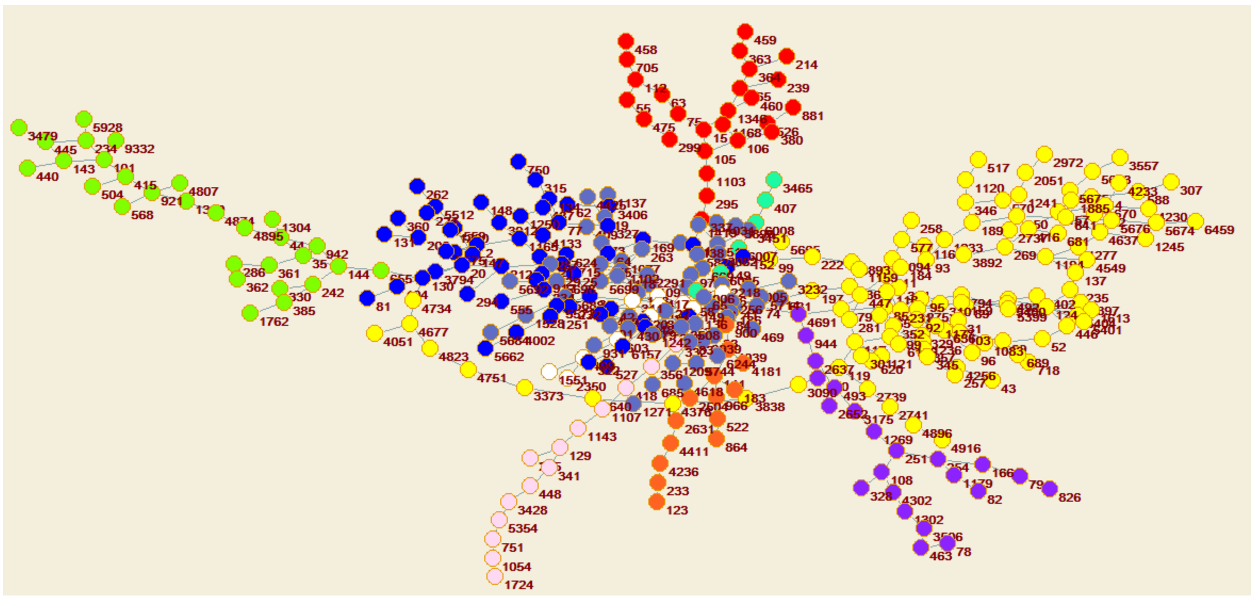
Inflation =5 Expansion=2 No. of Clusters=224 Iteration = 31 Time taken= 54279

Image 3.2



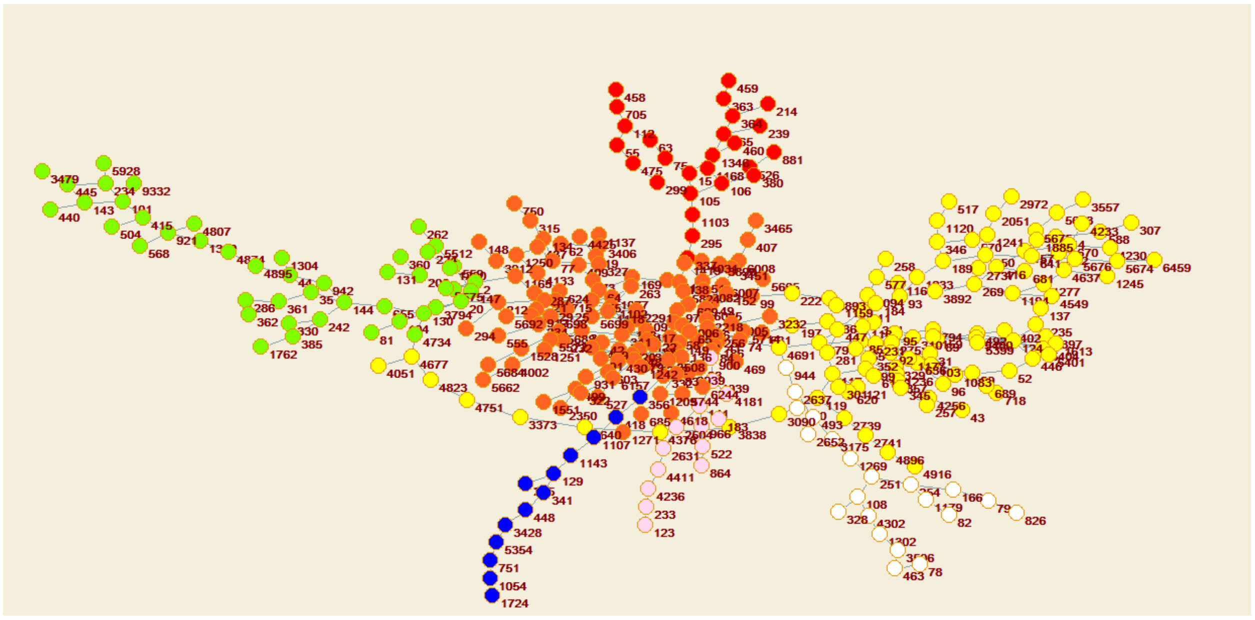
Inflation =5 Expansion=4 No. of Clusters=54 Iteration = 12 Time taken= 63352

Image 3.3



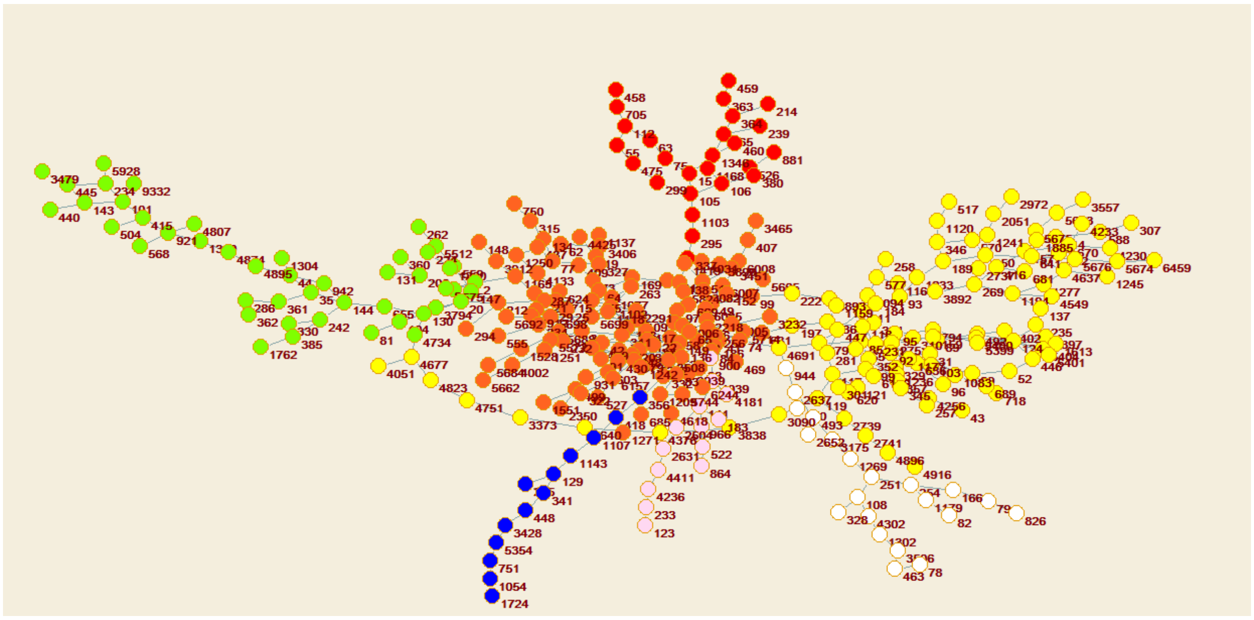
Inflation =2 Expansion=4 No. of Clusters=10 Iteration = 21 Time taken= 157931

Image 3.4



Inflation =3 Expansion=6 No. of Clusters=7 Iteration = 12 Time taken= 152299

Image 3.5



Inflation =4 Expansion=7 No. of Clusters=7 Iteration = 10 Time taken= 117458

Conclusion

We have successfully implemented the "Markov Cluster Algorithm". This algorithm partitions the given network into clusters based on the concept of random walks which was explained earlier. The MCL algorithm was implemented on 3 networks, AT&T Web network, physics collaboration network, and yeast metabolic network. Then by visualizing the clusters formed in the network using Pajek, we observed that the number of links inside a cluster are more as compared to the number of links in between different clusters. We also analyzed the effect of parameters inflation and expansion (power) on clustering.