Optimizing Community Detection

New Mexico
Supercomputing Challenge
Final Report
April 3, 2012

Team 56
La Cueva High School

Team Members

Alexandra Porter
Stephanie Djidjev
Lauren Li

Teacher

Samuel Smith

Table of Contents

1.0 Executive Summary

Earth in itself is a large-scale network. It is a system connecting and overlapping the individual networks of daily life from communication systems to biological systems. In science, these networks are used to express connections between different nodes through arcs and paths. To understand and apply these structures, decomposition is of utmost importance. Decomposition allows for the breakdown of complex problems into smaller, more manageable ones, which adds an element of simplicity to the problem. With networks, this process results in the formation of communities which signifies the connectivity between distinct groups of nodes.

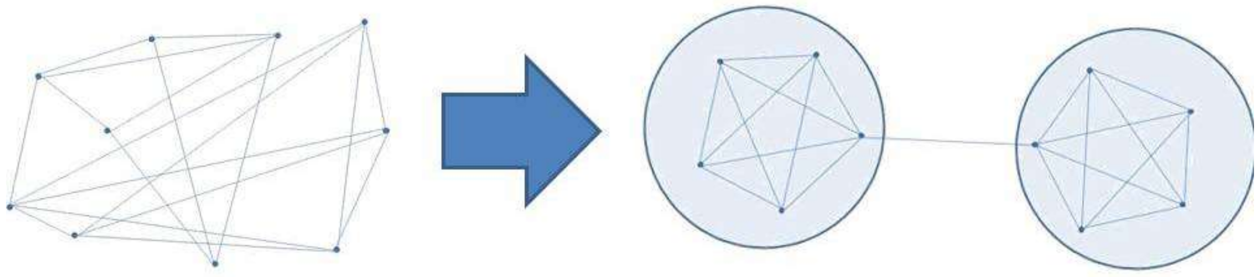
This project optimizes decomposition by finding the most efficient optimization method for partitioning of networks with maximum modularity. Java was used to implement and compare two main techniques: Girvan-Newman Algorithm and Ant Colony Optimization (ACO). Both algorithms were also tested with a variety of modifications. A Hierarchical Girvan-Newman Algorithm was implemented. Combinations between the ACO and Probability Summation as well as between the ACO and Power Iteration was used.

In discovering the most efficient optimization method, this project could be applied to most networks which naturally divide into communities or modules. Examples of such systems include transportation, manufacturing, communication, biological, and citation networks.

2.0 Problem Statement

What is the most efficient optimization method to maximize the modularity of a partitioned network?

The purpose of this project is to find an efficient and accurate method of maximizing the modularity of a partitioned network. Finding the maximum modularity of a network is extremely useful for any real-life situation the network may represent. Identifying groups makes it possible for someone to lay efficient connections or place the nodes in optimal locations.



3.0 Background Information

Networks are systems where sets of nodes are connected through links or edges. These systems appear in everyday life in the form of power grids, biological networks and pathways, metabolic networks, the Internet, social networks, and much more. Through the partitioning of networks in this project, systems are rearranged into communities allowing for further practical application of networks. For example, partitions within social networks can be used to represent social groupings based on common traits or interests. Citation network communities reveal related papers on certain topics, while the communities formed in the World Wide Web represent related sites. In economic networks, partitioning can reveal elements of vertical disintegration and groups of similar businesses. In a general sense, these partitions expose information which would be impossible to see in the big picture. With better understanding of networks through partitioning, correlations between network topology and function are revealed, and more efficient applications of networks can be discovered.

In the real world, the structure of networks is not random. Instead, various recurring patterns persist through networks. Sometimes patterns are obvious due to different contributing factors. In some cases, density of connections vary to form clusters or modules. Number of connections can also vary resulting in different costs of connection between nodes. Link capacity and structural patterns may also fluctuate. All of these factors involve the structural property of modularity, a significant characteristic of network topology.

Modularity is the qualitative measurement of the possible partitions within a network. A network with high modularity has dense connections between some nodes forming communities (modules) which are usually interconnected with other communities through sparse connections. Each module can be a dynamic community where there are more connections within modules and fewer between. Examples of modularity exist in many fields. For example, in engineering, modules form physical components with one-to-one functions. Modules can also represent platforms and individual content in products. Within social networks, high modularity is exhibited in cliques, groups (like in social networking sites), and association mutuality. In nature, biology involves modules in cell clusters, classification, and molecular reactions. Ecological modules include niches, pathways, and elements of food chains. High modularity is beneficial in that it allows for resistance to outside attacks, and individual problems are reduced. Substitutions can also occur with modules leading to flexibility and variety in networks. Finally, modularity also allows for parallelization. In this project, modularity is used as a measurement of the accuracy of a network partitioning. Modularity is the number of in-community edges minus the number of expected in-community edges in a random graph.

$$Q = \frac{1}{4m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)s_i s_j = \frac{1}{4m}\mathbf{s}^T\mathbf{B}\mathbf{s}.$$

4.0 Procedural Overview

4.1 Network Setup

In order to measure the algorithms used in this project, a customizable network creation system was made. This system allows the user to specify details about a network including the number of nodes, the number of communities in the network, and the probability of connections within or between communities. The network is then randomly generated based on these factors. This method creates networks with predetermined communities so that the accuracy of the optimization methods can be measured as well as the efficiency.

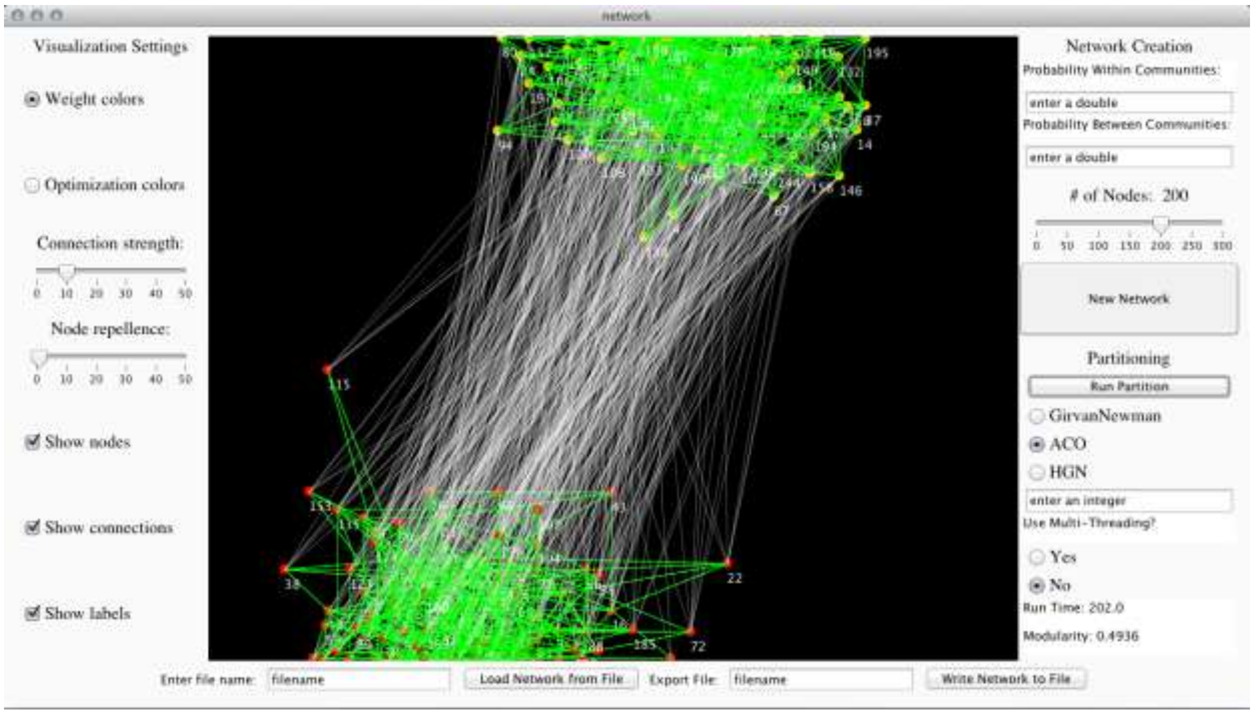
Within the network, each node and connection has a weighted value. This is important in using the network to represent a real situation. For example, in transportation methods such as roads, airplanes, and ships, each have a unique cost. Furthermore, physical distance is an important factor in many applications.

4.2 Visualization

The program developed in the project (shown in Figure 1) displays the network as a number of circles, each representing a node, and lines connecting them. Positions on the screen are initially random. A force-based layout algorithm then moves the nodes based on their proximity to other nodes and their connections.

In the force-based algorithm, each node is treated as an electron and each connection is treated as a spring. Nodes then repel each other while connections pull nodes together. The force on each node is calculated using Hooke's law, which relates to the springs, or connections, and using Coulomb's law, which relates to the electron charges. The weight of each node and connection are also factors in determining the forces in the network. A node with high weight, for instance, repels other nodes with greater force than one with a low weight.

Program users can customize the criteria used in the visualization. The values of the constants in Hooke's law and Coulomb's law can be adjusted in order to alter the layout. There are also two options for coloring the network's connections. Connection colors can be based on connection weights or the betweenness of the connection. Nodes are colored based on their weights. Finally, nodes, connections, and their names can be hidden for convenience in viewing the network. Users can generate random networks or create custom networks, as shown in Figure 2.



[✓ Weight colors] Optimization colors

✓ Show nodes

✓ Show connections

✓ Show labels

# of Nodes: 200

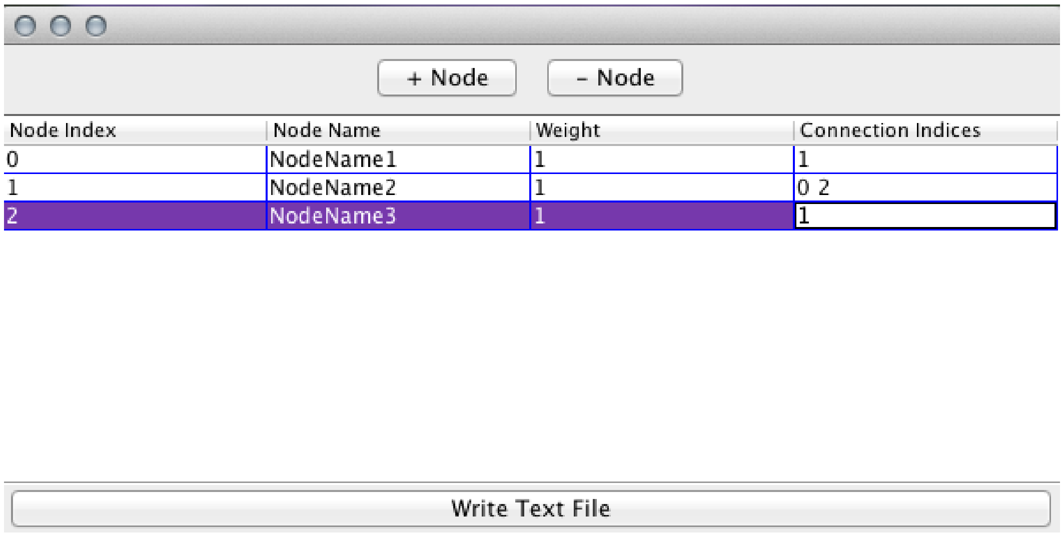
GirvanNewman [✓ ACO] HGN

Yes [✓ No]

Run Time: 202.0

Modularity: 0.4936

Figure 1



Node Index	Node Name	Weight	Connection Indices
0	NodeName1	1	1
1	NodeName2	1	0 2
2	NodeName3	1	1

Figure 2

4.3 Algorithm Setup

All the algorithms used in this project have a similar feature; they calculate weights of the edges in the network in order to calculate edge betweenness. Edges with a high betweenness have a high probability to lie within the shortest path that goes from one randomly selected node to another. All edges are equally weighted at the initiation of each algorithm implementation. As the algorithm runs, weights are deposited on the edges. Depending on the algorithm, an edge with either a large or small weight will have high betweenness. Edges with low betweenness are edges that are within communities, while edges with high betweenness are edges that link different communities.

The Floyd-Warshall Algorithm calculates shortest path by first assuming that a connection cost is infinite. It then iterates through possible intermediate connections in order to find the path with the lowest cost. When all connection weights are equal to one, the shortest path is the fewest number of connections. This algorithm was implemented in some of the other algorithms in order to detect communities.

The first algorithm used is the Girvan-Newman Algorithm. It first calculates the shortest path between all possible pair of nodes, using the Floyd-Warshall Algorithm. The number of times a connection is used as part of a shortest path is summed and the connections most used have the highest probability of existing between communities.

The second algorithm used is the Hierarchical Girvan-Newman Algorithm. Using the Floyd-Warshall Algorithm, it calculates the shortest path between all possible pairs of nodes. Then, it creates a hierarchy outlining all the edges each pair of nodes needs to go through in the shortest path. The weight of each edge is compounded into all of the components that comprise its shortest path. The edges that have the highest weight have the highest probability of existing between communities.

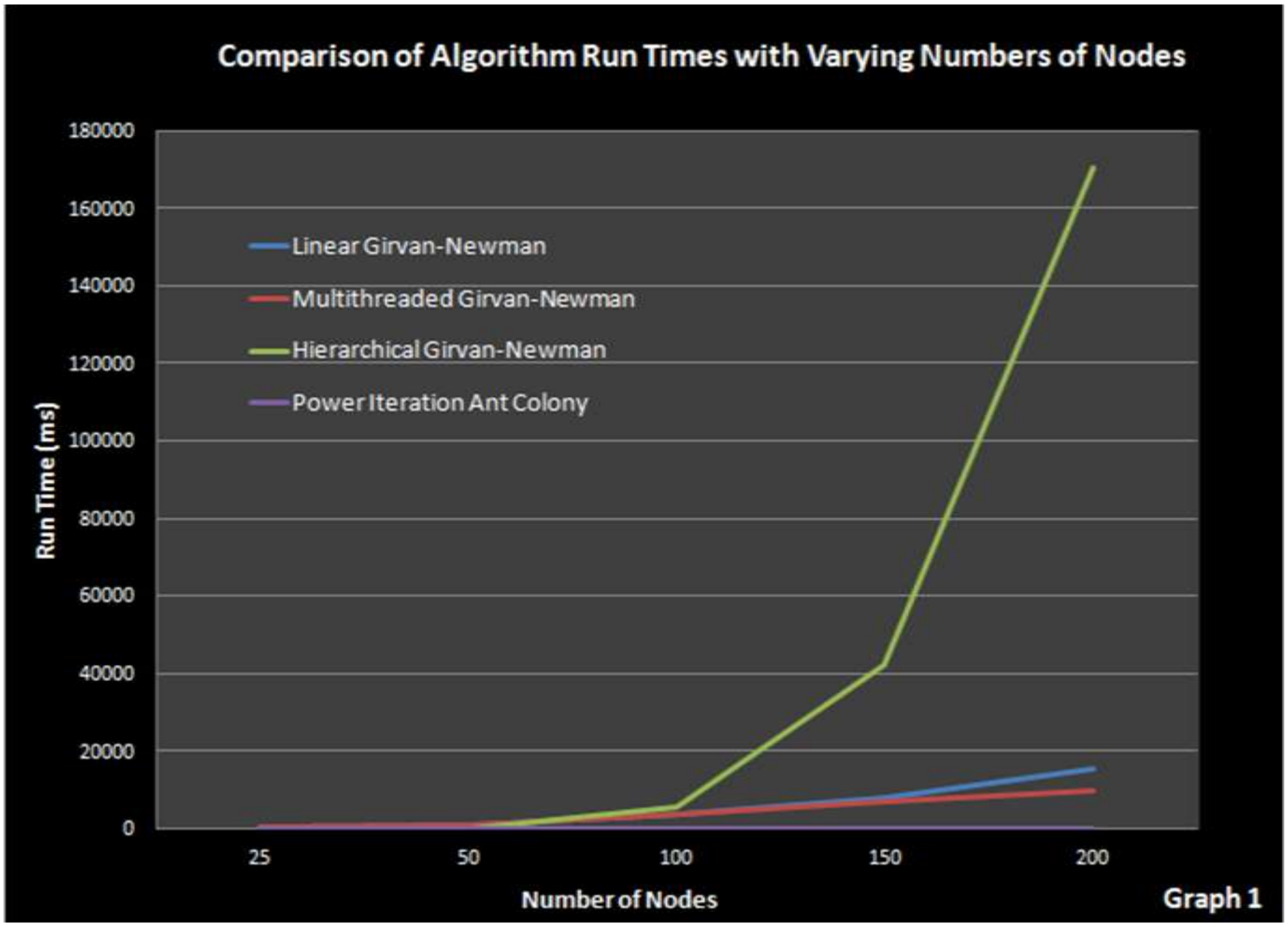
The final algorithm used is a novel combination between an Ant Colony Optimization (ACO) and Power Iteration, called the "Power Iteration Ant Colony" algorithm. ACOs are algorithms based on ant foraging behavior. Ants leave "pheromone" behind each of the nodes they travel on. The network is initialized with random values of pheromone on every node. Ants are placed randomly on the network and move from one node to another via the edges. On each node, it calculates an updated pheromone of that node based on the nodes that are directly connected to it using the Power iteration algorithm that given a network N, the algorithm will produce a number b_k and a pheromone level p, such that $Np = b_k p$. In the case of two communities, this algorithm results in either negative or positive pheromone levels. Nodes with negative pheromone values are in one community, while nodes with positive pheromone values are in another community. The Power Iteration Ant Colony algorithm can be recursively implemented to find community structures of more than only two modules, by treating each newly formed community of nodes as a new network, and dividing that into even more communities. The algorithm stops partitioning further when the new communities internally have a modularity of zero or less.

5.0 Results

5.1 Efficiency

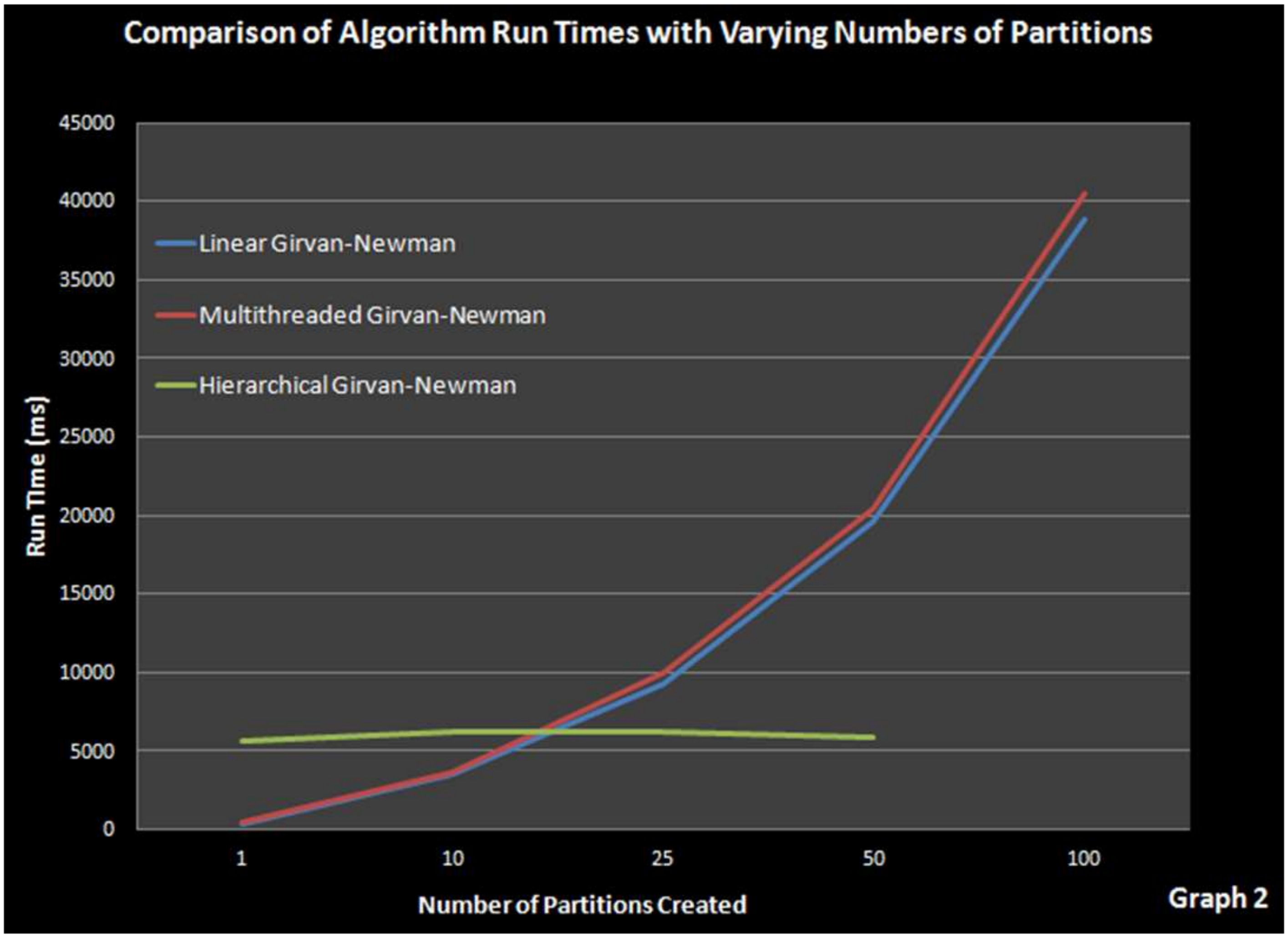
Run Times Varying Number of Nodes

Run times were recorded for partitioning a random network 10 times for increasing numbers of nodes. The probabilities of connection existence were 0.5 and 0.05 for all networks. All run times increase as the number of nodes increases. Due to the nature of the multithreading of the Girvan Newman Algorithm, the multithreaded version is more efficient after the number of nodes has passed a certain threshold, shown to be approximately 125 nodes for 10 partitions. The Hierarchical Girvan Newman is significantly slower and run times increase by a larger factor than for the Girvan Newman as the number of nodes increases. The Power Iteration Ant Colony Optimization was significantly faster than either of the other algorithms and run times did not visibly increase as the number of nodes increased.


Comparison of Algorithm Run Times with Varying Numbers of Nodes
Linear Girvan-Newman
Multithreaded Girvan-Newman
Hierarchical Girvan-Newman
Power Iteration Ant Colony
Run Time (ms)
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
25
50
100
150
200
Number of Nodes
Graph 1

Run Times Varying Number of Partitions

The run times were recorded for partitioning random networks of size 100 nodes with the same connection probabilities as in Graph 1. In this comparison the number of partitions was increased. Because 100 nodes is less than the threshold for multithreading to be more efficient than the linear Girvan Newman Algorithm, the multithreaded times are greater. This study shows that the run times diverge slightly, with the multithreaded run times becoming larger in relation to the linear times as the number of partitions increased. The Hierarchical Girvan Newman run times stayed approximately constant, even decreasing slightly. This is due to the fact the the bulk of the HGN run time occurs regardless of how many connections in the resulting hierarchy are actually removed. Based on these results, the HGN becomes more efficient for networks demanding a large number of partitions to maximize modularity.


Comparison of Algorithm Run Times with Varying Numbers of Partitions
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Run Time (ms)
Linear Girvan-Newman
Multithreaded Girvan-Newman
Hierarchical Girvan-Newman
1
10
25
50
100
Number of Partitions Created
Graph 2

5.2 Accuracy

Girvan-Newman Algorithm

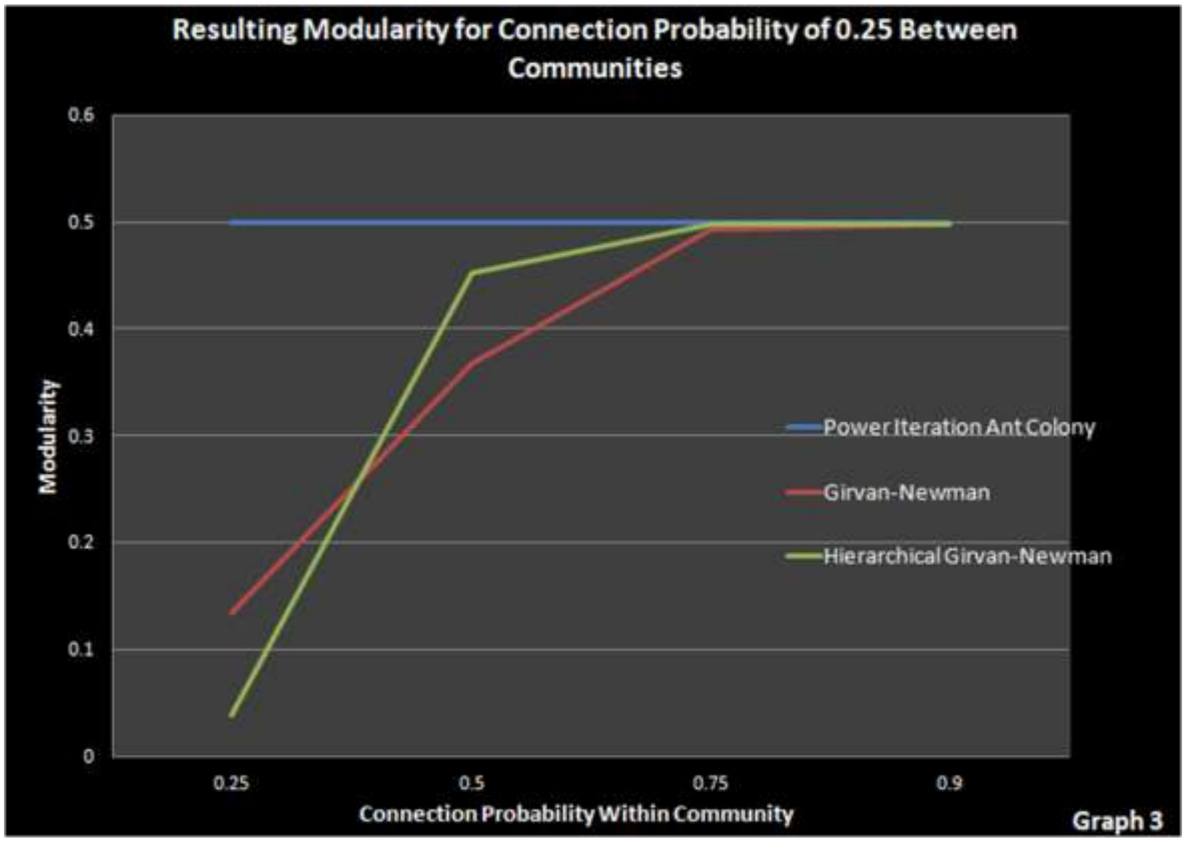
The Girvan-Newman Algorithm was able to reach maximum modularity for a network with 2 communities on almost all of the networks tested. As shown in Graphs 3 and 4, the modularity reached does drop off as communities become less distinct. The modularity reached is not as high because the algorithm removes connections within communities as well as between communities.

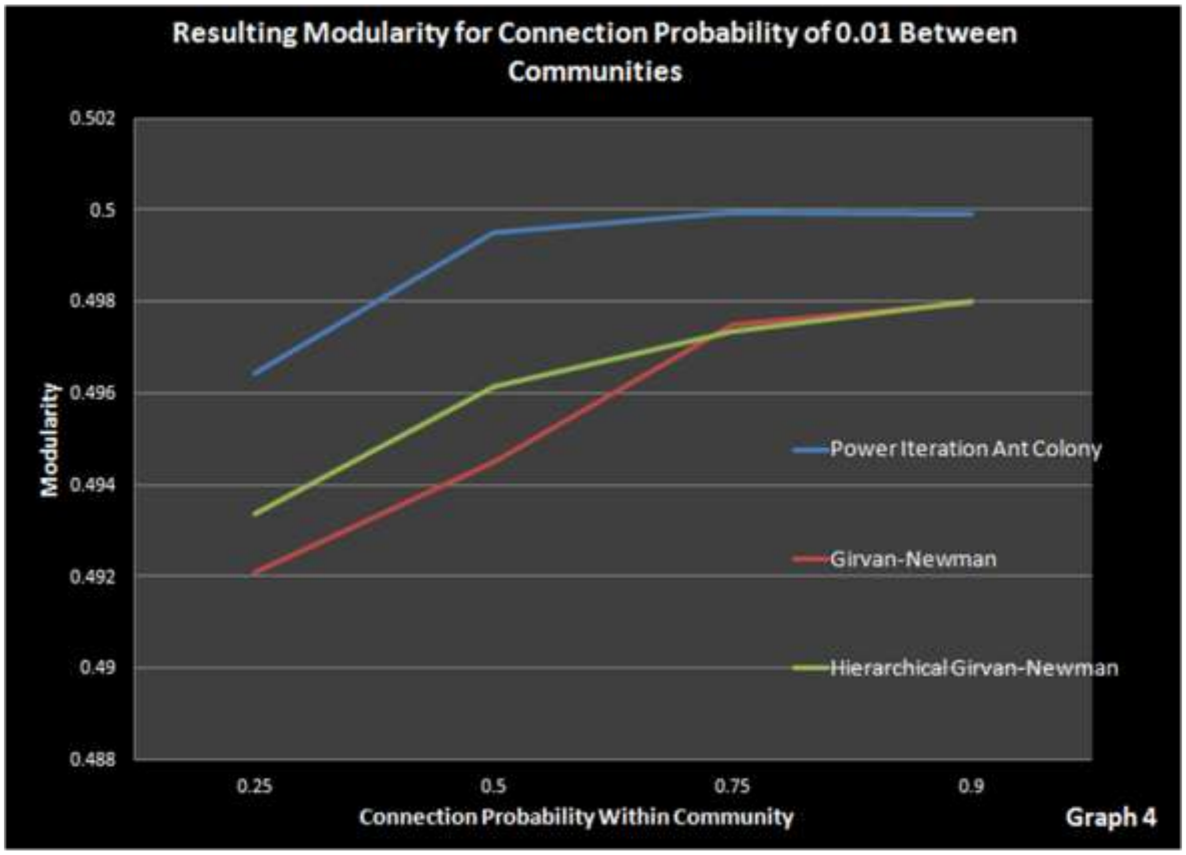
Hierarchical Girvan-Newman

The Hierarchical Girvan-Newman reached modularities similarly to the Girvan-Newman Algorithm but showed higher accuracy in the more convoluted networks. Between inside community probabilities of 0.25 and 0.75, especially, the HGN reached higher modularity than the Girvan-Newman.

Power Iteration Ant Colony Optimization

The Power Iteration ACO was extremely accurate and consistent. It also showed (on Graph 3) the unique capability of finding a partition with high modularity on a network that has uniform likelihood (0.25 probability) of a connection and therefore no predetermined communities. In Graph 3, the Power Iteration ACO reaches almost exactly a modularity of 0.5 (the modularity of an ideal network of 2 densely connected communities) for all probabilities tested.


Resulting Modularity for Connection Probability of 0.25 Between Communities
Modularity
0.6
0.5
0.4
0.3
0.2
0.1
0
0.25
0.5
0.75
0.9
Connection Probability Within Community
Power Iteration Ant Colony
Girvan-Newman
Hierarchical Girvan-Newman
Graph 3


Resulting Modularity for Connection Probability of 0.01 Between Communities
Modularity
0.502
0.5
0.498
0.496
0.494
0.492
0.49
0.488
0.25
0.5
0.75
0.9
Connection Probability Within Community
Power Iteration Ant Colony
Girvan-Newman
Hierarchical Girvan-Newman
Graph 4

Real-World Applications

The program was also tested on two real-world networks. The first is a social network, created with randomly selected “friends,” where connections represent friendships. The network was partitioned using the Power Iteration ACO. The communities identified were found to be closely related to factors including grade level and participation in band and orchestra. Node ‘M’, for instance is the only freshman included in the network. ‘A’, ‘B’, ‘C’, ‘H’, and ‘O’ are all seniors who are members of the band.

The second network used for testing was a food web. The Power Iteration ACO was used initially, but too many connections were removed and the result was not useful. The Hierarchical Girvan-Newman, however, effectively identified two communities. The first, represented by red nodes, includes all of the birds and mammals. The orange community is comprised of snakes, spiders, and all of the insects included in the web.

Social Network Partitioned using HGN

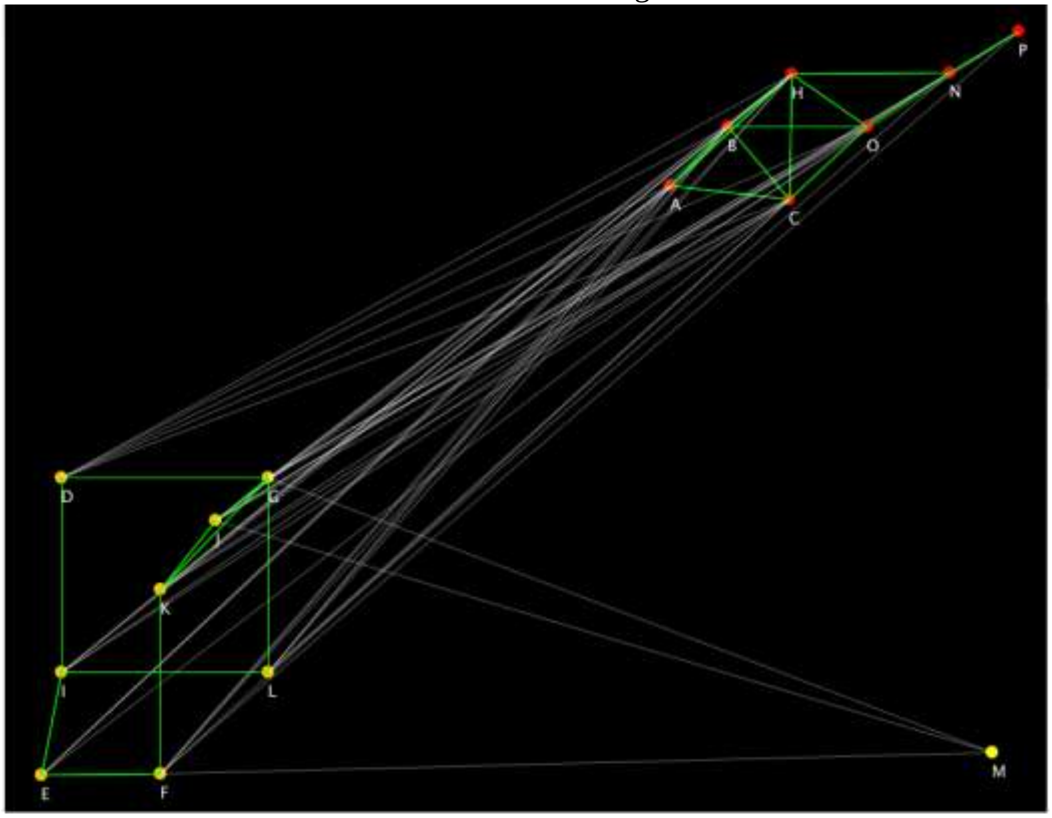


Figure 3

Food Web Partitioned with Power Iteration ACO

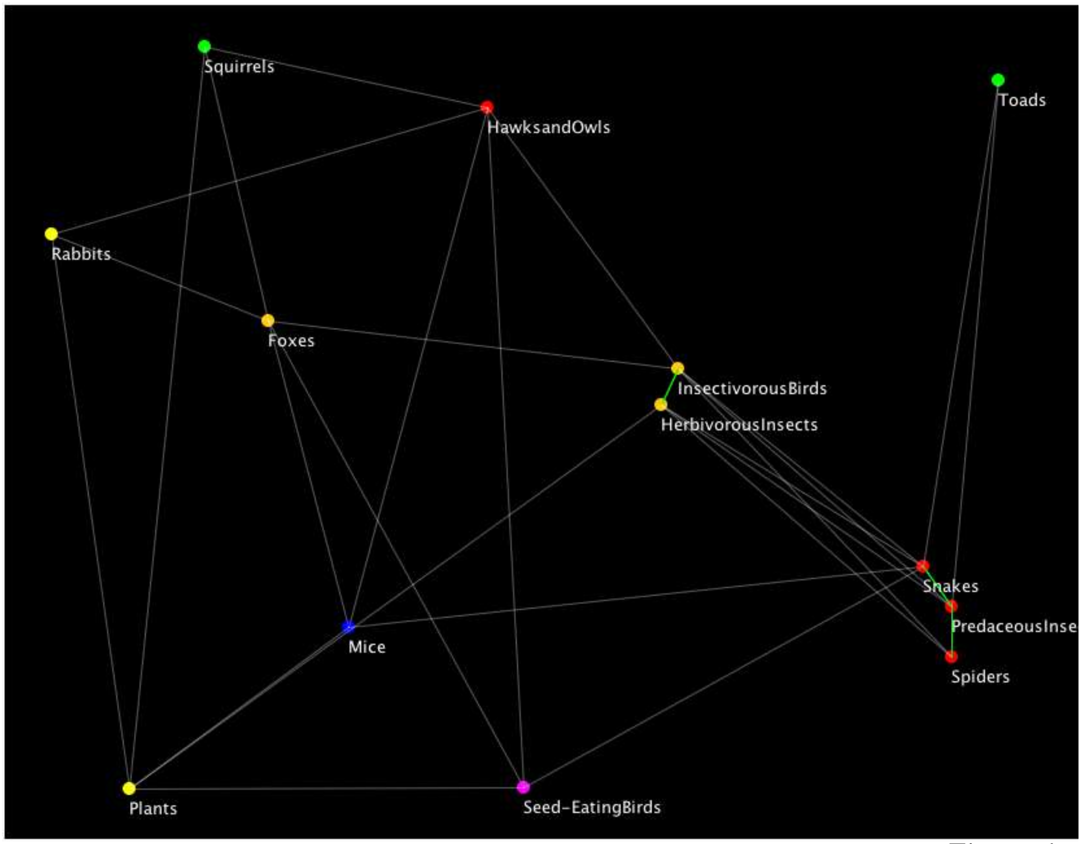


Figure 4

Food Web Partitioned with Hierarchical Girvan-Newman

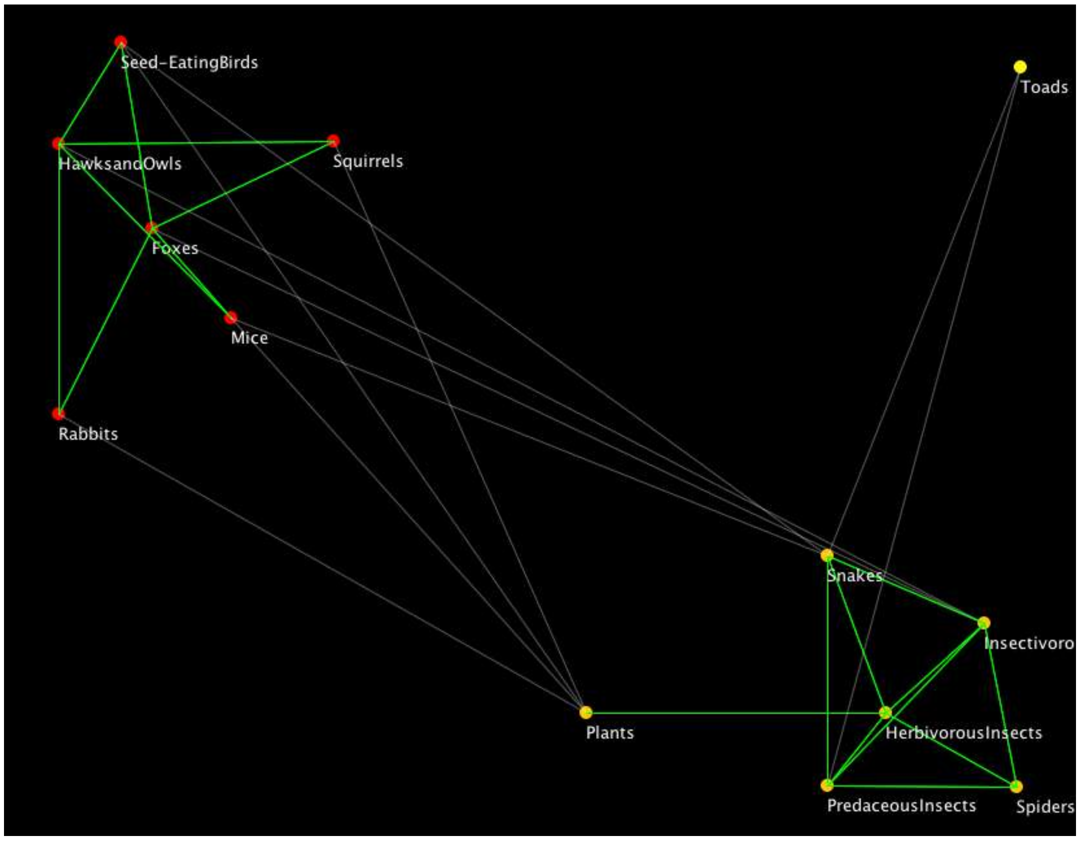


Figure 5

6.0 Conclusions

The multithreaded Girvan-Newman Algorithm was the fastest Girvan-Newman, as expected. However, the Hierarchical Girvan-Newman was slightly more accurate. The Power Iteration Ant Colony Optimization was the most accurate when applied to large, convoluted networks. It was also the fastest algorithm, so it is the best algorithm for partitioning larger networks. On smaller networks of approximately 15 nodes the HGN is actually the best algorithm because it reaches the highest modularity and in some instances the Power Iteration ACO partitions too many connections, therefore failing to effectively identify communities.

7.0 Significant Original Achievement

The main accomplishment of this project has been the development of the novel Power Iteration Ant Colony Optimization, a unique combination of two algorithms. The implementation of a Hierarchical Girvan-Newman algorithm is also original. In addition, a comprehensive program has been developed to allow users to input network data of any kind and identify a solution, using these algorithms.

8.0 Future Work

Next we plan to extend the Power Iteration Ant Colony algorithm to effectively partition more than two communities, which the Girvan-Newman and Hierarchical Girvan-Newman already do. To make this extension, the Power Iteration ACO will use a recursive method to partition each community into sub-communities. This also allows for parallelization, which we plan to add. In addition, we plan to add an algorithm to identify the preferable algorithm (HGN or Power Iteration ACO) for each individual situation.

9.0 Appendix

9.1 Acknowledgements

We would like to thank the organizers of the New Mexico Supercomputing Challenge for all of their assistance for this project. We would also like to thank David Rogers, Miguel Leyba, Janeen Anderson, Klaus Heinemann, and Bilal Shebaro for their helpful evaluations.

9.2 Works Cited

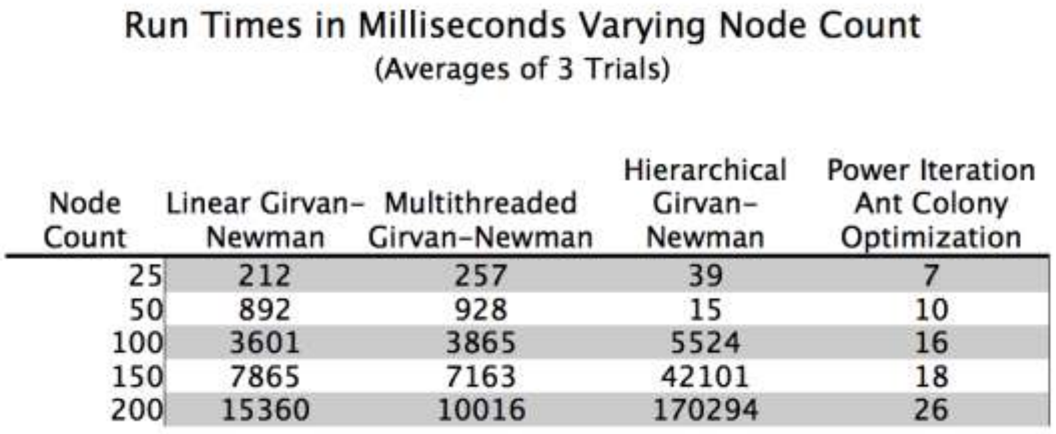
Capper, John, and Henrik Nilsson. "Static Balance Checking for First-Class Modular Systems of Equations." University of Nottingham, United Kingdom, 19 May 2010. Web. 1 Mar. 2012. <http://www.cs.ou.edu/tfp2010/files/09slides.pdf>.

C.L. Magee, *ESD 342 Class 14 Decomposition*, Spring 2010. (Massachusetts Institute of Technology: MIT OpenCouseWare), http://ocw.mit.edu (Accessed March 1, 2012). License: Creative Commons BY-NC-SA

Daniel E. Whitney, ESD 342 Network Representations of Complex Engineering Systems, Spring 2010. (Massachusetts Institute of Technology: MIT OpenCouseWare), http://ocw.mit.edu (Accessed March 1, 2012). License: Creative Commons BY-NC-SA

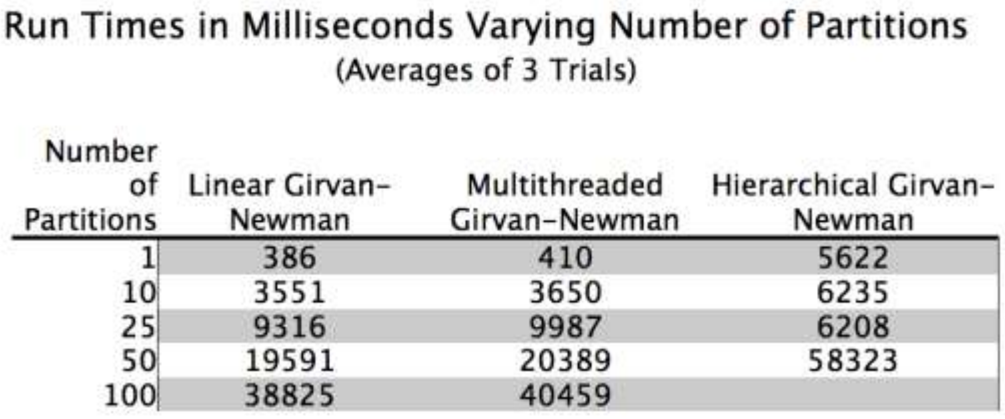
Girvan, M., and M. E. J. Newman. "Community Structure in Social and Biological Networks." *PNAS* 99.12 (2002): 7821-826. Web. 1 Mar. 2012. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC122977/>.

9.3 Data



Run Times in Milliseconds Varying Node Count
(Averages of 3 Trials)

Node Count	Linear Girvan-Newman	Multithreaded Girvan-Newman	Hierarchical Girvan-Newman	Power Iteration Ant Colony Optimization
25	212	257	39	7
50	892	928	15	10
100	3601	3865	5524	16
150	7865	7163	42101	18
200	15360	10016	170294	26



Run Times in Milliseconds Varying Number of Partitions
(Averages of 3 Trials)

Number of Partitions	Linear Girvan-Newman	Multithreaded Girvan-Newman	Hierarchical Girvan-Newman
1	386	410	5622
10	3551	3650	6235
25	9316	9987	6208
50	19591	20389	58323
100	38825	40459	

Resulting Modularity of the Hierarchical Girvan-Newman Algorithm
(2 Communities of Equal Size)

Connection Probability Between Communities	Connection Probabilities Within Communities 0.25	0.5	0.75	0.9
0.01	0.493379	0.496135	0.497346	0.498023
0.025	0.490364	0.495462	0.497591	0.498043
0.05	0.484028	0.495526	0.497677	0.498144
0.075	0.446055	0.492188	0.497233	0.495593
0.1	0.359165	0.494508	0.496849	0.498094
0.25	0.03833	0.453049	0.497399	0.498086

Resulting Modularity of the Girvan-Newman Algorithm
(2 Communities of Equal Size)

Connection Probability Between Communities	Connection Probabilities Within Communities 0.25	0.5	0.75	0.9
0.01	0.49209	0.494512	0.4975	0.497976
0.025	0.491718	0.495871	0.495992	0.497995
0.05	0.478819	0.492094	0.497472	0.497805
0.075	0.160303	0.490671	0.49782	0.498149
0.1	0.202503	0.484985	0.497596	0.497998
0.25	0.13469	0.368338	0.492724	0.497061

Resulting Modularity of the Power Iteration Ant Colony Optimization Algorithm
(2 Communities of Equal Size)

Connection Probability Between Communities	Connection Probabilities Within Communities			
	0.25	0.5	0.75	0.9
0.01	0.496437	0.499504	0.499935	0.499907
0.025	0.496876	0.498493	0.499913	0.49998
0.05	0.49719	0.499351	0.4991	0.499893
0.075	0.496078	0.498973	0.499545	0.499953
0.1	0.495338	0.498849	0.499217	0.499949
0.25	0.499882	0.49888	0.49917	0.499777

9.4 Program Code

Girvan-Newman Algorithm

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.util.*;
/**
 *
 * @author A Porter
 */
public class girvanNewman
{
   private int[][] network;
   private int[][] pathWeight;
   private int[][] path;
   private int[][] next;
   private int prevNode;
   private int nextNode;
   private int pathMax;
   int nodeCount;
   public girvanNewman(int[][] network, int nodeCount)
   {

      this.network = network;
      this.nodeCount= nodeCount;
      initGN();
      girvanNewman();
   }

   public int[][] getPathWeights()
   {
      return pathWeight;
   }

   private void initGN()
   {
      pathWeight=new int[nodeCount][nodeCount];
      path = new int[nodeCount][nodeCount];
      next = new int[nodeCount][nodeCount];
      prevNode=0;
      nextNode=0;
      pathMax=0;
      for (int i=0; i<nodeCount; i++)//rows of nodes
      {
         for(int j=0;j<nodeCount; j++)//cols of nodes
         {

            next[i][j]=-1;//initializing...
            pathWeight[i][j]=0;//initializing...
            if (network[i][j]>0 && i!=j)
            {
                path[i][j]=1;
                path[j][i]=1;
            }
            else if (i==j)
            {
                path[j][i]=0;
                path[j][i]=0;
            }
            else
            {
                path[i][j]=2*nodeCount;
                path[j][i]=2*nodeCount;
            }

        }

    }

}

public void floydWarshall()
{

    for (int k=0; k<nodeCount; k++)
    {
        for (int i=0; i<nodeCount; i++)
        {
            for (int j=0; j<nodeCount; j++)
            {
                //path[i][j] = Math.min(path[i][j], path[i][k]+path[k][j]);
                if ((path[i][k]+path[k][j])< path[i][j])
                {
                    path[i][j]=path[i][k]+path[k][j];
                    next[i][j]=k;

                }

            }
        }
    }

}

private void girvanNewman()
{
    floydWarshall();


    int prevNode=0;
    int nextNode=0;
    int pathMax=0;
    for (int i=0; i<nodeCount; i++)
    {
        for (int j=0; j<nodeCount; j++)
```

```
            {
               if (i!=j)
               {
                  Scanner pathScan=new Scanner(getPath(i,j));
                  prevNode = i;
                  while (pathScan.hasNextInt())
                  {
                     nextNode=pathScan.nextInt();
                     pathWeight[prevNode][nextNode]++;
                     pathWeight[nextNode][prevNode]++;
                     if (pathWeight[prevNode][nextNode]>pathMax)
                     {
                        pathMax=pathWeight[prevNode][nextNode];

                     }
                     prevNode=nextNode;
                  }
               }
            }
         }


   }

   private String getPath(int i, int j)
   {
      if (path[i][j]>nodeCount)
         return "no path";
      int intermediate=next[i][j];
      if (intermediate==-1)
      {
         return " ";

      }
      else
         return  getPath(i,intermediate) + ""+intermediate + ""+getPath(intermediate,j);
   }
}
```

Parallelized Girvan-Newman

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.awt.geom.*;
/**
 *
 * @author A Porter
 *
 * Multithreading algorithm based on Parallel Processing Tutorial provided
 * at the Supercomputing Challenge Kickoff Conference
 */
public class parallelGN  {

   private static final int NUM_THREADS = 4;

   private byte[] values;
private int count;
private long sum;
private double logSum;
private Worker[] workers;

private int[][] network;
private int[][] pathWeight;
private int[][] path;
private int[][] next;
private int prevNode;
private int nextNode;
private int pathMax;
int nodeCount;


public  parallelGN( int[][] network, int nodeCount)
{
   this.network = network;
   this.nodeCount=nodeCount;
   pathWeight=new int[nodeCount][nodeCount];
   path = new int[nodeCount][nodeCount];
   next = new int[nodeCount][nodeCount];
   prevNode=0;
   nextNode=0;
   pathMax=0;
   initializeData();
   floydWarshall();
   initializeThreads();
   process();
   //System.out.println("used");

   // return pathWeight;
}

public int[][] getPathWeights()
{
   return pathWeight;
}

private void initializeData() {
   long start = System.currentTimeMillis();
   for (int i=0; i<nodeCount; i++)//rows of nodes
   {
      for(int j=0;j<nodeCount; j++)//cols of nodes
      {

         next[i][j]=-1;//initializing...
         pathWeight[i][j]=0;//initializing...
         if (network[i][j]>0)
         {
            path[i][j]=network[i][j];
            path[j][i]=network[i][j];
         }
         else if (i==j)
         {
            path[j][i]=0;
```

```
                path[j][i]=0;
            }
            else
            {
                path[i][j]=2*nodeCount;
                path[j][i]=2*nodeCount;
            }

        }

    }
}

private void process() {
    long start = System.currentTimeMillis();
    //System.out.printf(PROCESS_PATTERN);
    processWithThreads();
    // System.out.printf(RESULT_PATTERN,
    //   count, sum / (double) count, Math.exp(logSum / count));
    //System.out.printf(TIME_PATTERN, System.currentTimeMillis() - start);
}

private void initializeThreads() {
    long start = System.currentTimeMillis();
    int sliceLength = nodeCount / NUM_THREADS;
    //System.out.printf(INIT_THREADS_PATTERN, NUM_THREADS);
    workers = new Worker[NUM_THREADS];
    for (int i = 0; i < workers.length; i++) {
        workers[i] = new Worker(sliceLength * i, sliceLength);
    }
    //  System.out.printf(TIME_PATTERN, System.currentTimeMillis() - start);
}

private void processWithThreads() {
    for (int i = 0; i < workers.length; i++) {
        workers[i].start();
    }
    for (int i = 0; i < workers.length; i++) {
        try {
            workers[i].join();
        }
        catch (InterruptedException e) {}
    }
}

public synchronized void update(byte value)
{
    int i= (int) value;
     for (int j=0; j<nodeCount; j++)
        {
            Scanner pathScan=new Scanner(getPath(i,j));
            prevNode = i;
            while (pathScan.hasNextInt())
            {
                nextNode=pathScan.nextInt();
                pathWeight[prevNode][nextNode]++;
                pathWeight[nextNode][prevNode]++;
                if (pathWeight[prevNode][nextNode]>pathMax)
                {
                    pathMax=pathWeight[prevNode][nextNode];
                }
                prevNode=nextNode;
            }

        }
    }
public void floydWarshall()
    {

        for (int k=0; k<nodeCount; k++)
        {
            for (int i=0; i<nodeCount; i++)
            {
                for (int j=0; j<nodeCount; j++)
                {
                    //path[i][j] = Math.min(path[i][j], path[i][k]+path[k][j]);
                    if ((path[i][k]+path[k][j])< path[i][j])
                    {
                        path[i][j]=path[i][k]+path[k][j];
                        next[i][j]=k;
                        /* pathWeight[i][k]++;
                        pathWeight[k][j]++;
                        pathWeight[i][j]--;*/
                    }

                }
            }
        }

    }

    private String getPath(int i, int j)
    {
        if (path[i][j]>nodeCount)
            return "no path";
        int intermediate=next[i][j];
        if (intermediate==-1)
        {
            return " ";

        }
        else
            return  getPath(i,intermediate) + ""+intermediate + ""+getPath(intermediate,j);
    }

    private class Worker extends Thread
    {

        private int rangeStart;
        private int rangeLength;

        public Worker(int rangeStart, int rangeLength) {
            this.rangeStart = rangeStart;
            this.rangeLength = rangeLength;
        }

        public void run() {
            for (int i = rangeStart; i < rangeStart + rangeLength; i++) {
                update((byte)i);
            }
        }

    }

}
```

Hierarchical Girvan-Newman

```
import acm.program.*;
import acm.graphics.*;
import java.awt.Color;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.lang.Object;
import java.io.*;
/**
 * Creates a graph with communities, based off probabilities.
 *
 * @author S Djidjev
 *
 * groupCount refers to how many communities are in the network
 * groupCountArray refers to the size of each community, and is set in an array.
 * p1 refers to the probability of making an edge within the community
 * p2 refers to the probability of making an edge between communities
 *
 * Example: If you want a network with five communities of node sizes 5, 10, 15, 20, and 20, with the probability
 * of making a edge within the community to be 0.8 and making an edge between communities 0.01, you go to the
 * testFreedomSetup() in the editor, and enter the following:
 *
 * int groupCount=5;
 * int[] groupCountArrray=new int[groupCount];
 * int[]groupCountArray = {5,10,15,20,20};
 * networkSetup(groupCount,groupCountArray,0.8,0.01);
 *
 * Then make a freedomGraph instance, and run testFreedomSetup()
 *
 * NOTE: p2 must be less than p1 in order for the network to have communities!
 */

public class HierarchalGirvanNewman
{
    public int[][] path;
    public int[][] next;
    public int[] listdegree0;
    int n1; //number of nodes in one group
    int n2; // number of nodes in the group after the first
    public int n0; //number of nodes
    public int[][] nodeWeight; //assigns a weight to each new node
    public int maxWeight;
    public int[]degNode;
    public int[][] network;
    public int[][] edgeRanks;

    /**
     * Creates a graph with communities, based off probabilities.
     */
    public arrayGraph networkSetup(int n)
    {

        arrayGraph g = new arrayGraph(n,n);

        g.n = n;
        g.n2 = n;
        for (int i =0; i<n; i++)
        {
            for(int j=0; j<n; j++)
            {
                g.edge[i][j]=network[i][j];
            }
        }
        return g;
    }

    public arrayGraph pathGraph(arrayGraph g)
    {

        arrayGraph pg = new arrayGraph(g.n*g.n,2);
        int newnode = 0; //later assigns a number to the node representing the pair of nodes [i][j]

        //create empty list
        for(int i = 0; i < pg.n; i++){
            for(int j = 0; j < 2; j++){
                pg.edge[i][j] = -1;
            }
        }

        //g.print("Initial graph for the shortest paths computation");
        for (int k=0; k<g.n; k++)
        {
            for (int i=0; i<g.n; i++)
            {
                for (int j=0; j<g.n; j++)
                {
                    if(i==j)
                    {
                        g.edge[i][j] =0;
                    }
                    else if((g.edge[i][k] > 0) && (g.edge[k][j] >0) &&(((g.edge[i][k]+g.edge[k][j])< g.edge[i][j]) ||
g.edge[i][j]==0) )
                    {

                        g.edge[i][j]=g.edge[i][k]+g.edge[k][j]; //the length of edge (i,j) is equal to the length of edges (i,k)
plus (k,j)
                        pg.edge[node(i,j, g.n)][0] = node(i,k, g.n);
                pg.edge[node(i,j, g.n)][1] = node(k,j, g.n);
                // System.out.println("i: " +i+"j: " + j+"k: "+k);
                // g.print("Iterative paths: (i: " +i+"j: " + j+"k: "+k+")");
            }
        }

      }

   }

   return pg;
}
public Edge computeHeaviestEdge(arrayGraph pg, arrayGraph g, int n0)
{
   // compute the counts on edges
   Edge E = new Edge(-1,-1);
   for(int i = 0; i <n0; i++)
      for(int j = i+1; j <n0; j++)
         if(g.edge[i][j]>0){
            E.Init(i,j);
            break;}
   arrayGraph dg = new arrayGraph(n0,n0);
   LinkedList degree0nodes = new LinkedList();

   int[] gWeights = new int [g.m];
   int r = 0;
   int maxWeight = 0;

   for(int i = 0; i <n0; i++)
      for(int j = i+1; j <n0; j++)
         dg.edge[i][j] = 1;

   // compute node indegrees
   int[] indegree = new int[pg.size()];
   for(int i = 0; i <pg.size(); i++){
      if(pg.get(i,0) >= 0){
         indegree[pg.get(i,0)]++;
         indegree[pg.get(i,1)]++;
      }
   }

   // put indegree 0 nodes in a list
   for(int i = 0; i < pg.size(); i++)
   {

      if(indegree[i] == 0)
      {
         if((i/n0) < (i - (i/n0)*n0))
         {
            degree0nodes.add(i);

         }
      }

   }
```

```
    ListIterator itr = degree0nodes.listIterator();

    while(itr.hasNext())
    {

        int k = (Integer)degree0nodes.removeFirst();

        for(int i = 0; i<=1; i++)
        {
            int kiEdge = pg.get(k,i);
            Edge e = new Edge(kiEdge,n0);
            Edge e2 = new Edge(k,n0);

            if (kiEdge != -1)
            {
                int j = --(indegree[kiEdge]);

                if (j == 0)
                {
                    degree0nodes.addLast(kiEdge);

                }
                dg.edge[e.getFirst()][e.getSecond()]+=dg.edge[e2.getFirst()][e2.getSecond()];
            }
        }

    }
    for(int i = 0; i<n0; i++)
        for(int j = i+1; j<n0;j++){
            if(g.edge[i][j] > 0){
                if(dg.edge[i][j] > maxWeight){
                    maxWeight = dg.edge[i][j];
                    E.setFirst(i);
                    E.setSecond(j);
                }
            }
    }
    return E;
}

public int node(int i, int j, int n)
{

    return (n*i)+j;
}

public double computeModularity(arrayGraph g, UnionFind u)
{
    double modularity=0;
    degNode=g.degreeCalculator();
    for(int i=0; i<g.n; i++){
        for(int j =0; j<g.n; j++) {

            if(u.compare(i,j)){

                modularity += g.edge[i][j]-(double)degNode[i]*degNode[j]/(2*g.m);
            }
        }

    }
    modularity/=(2*g.m);

    return modularity;
}

public UnionFind computeCommunities(Edge[] sortedEdges, arrayGraph g){
    UnionFind u= new UnionFind(g.n);
    UnionFind uBest = new UnionFind(g.n);
    double bestModularity=-1;

    for (int i=0;i<g.m;i++){
        Edge e=sortedEdges[g.m-i-1];
        int node1=e.getFirst();
        int node2=e.getSecond();

        if(!u.compare(node1,node2))
        {// if e is a between community edge

            u.unite(node1,node2);

            double modularity=computeModularity(g, u);

            // u.printID();
            if(modularity>bestModularity){
                bestModularity=modularity;
                uBest=u.copy(); // save the partition for output
            }
        }
    }

    //  uBest.printID();
    return uBest;
}

public void FreedomSetup(int[][]network, int nodeCount)
{
    edgeRanks = new int[nodeCount][nodeCount];
    this.network=network;
    int numbNodes=nodeCount;// todo: enter as input
    arrayGraph g = new arrayGraph(numbNodes,numbNodes);
    g = networkSetup(nodeCount);

    UnionFind u = new UnionFind(g.n);
    int[][]parent = new int[n0*n0][2]; //edges = matrix for each node it gives its two descendants

    //compute number of edges
    g.computeEdgeCount();
    Edge [] sortedEdges = new Edge[g.m];

    arrayGraph origGraph = g.copy(); // copy of the original graph (needed as g changes inside the next loop)
    arrayGraph SPGraph; // shortest path graph
```

```
        for(int i = 0; i < origGraph.m; i++){
           //  g.print("Next Graph "+i+ ": ");
           SPGraph = g.copy(); // need this as next step replaces orig. edge weight with distances
           arrayGraph pg = pathGraph(SPGraph);
           Edge E = computeHeaviestEdge(pg,SPGraph,SPGraph.n);
           sortedEdges[i] = E;

           g.edge[E.getFirst()][E.getSecond()] = 0;
           g.edge[E.getSecond()][E.getFirst()] = 0;
           edgeRanks[E.getSecond()][E.getFirst()] = i+1;

           g.m--; //decreases the number of edges by 1, after edge is removed
        }

        u=computeCommunities(sortedEdges,origGraph);
        int[] community= u.computeCommunityNumber();

        FileWriter fWriter = null;
        BufferedWriter writer = null;
        try
        {

           fWriter = new FileWriter("File6.txt");
           writer = new BufferedWriter(fWriter);
           writer.write(g.n+System.getProperty( "line.separator"));
           for (int i = 0; i < g.n; i++)
           {
              for (int j = 0; j < g.n; j++)
              {
                 writer.write(origGraph.edge[i][j]+" ");
              }
              writer.write(System.getProperty( "line.separator"));
           }
           for (int i=0; i<g.n; i++)
           {
              writer.write(community[i]+" ");
           }
           writer.close();
        }
        catch (Exception e)
        {
           System.out.println("file cannot be loaded");
        }


        computeModularity( origGraph, u);
      //  origGraph.print("origGraph:");

     }
     public int[][] getEdgesRanked()
     {
        return edgeRanks;
     }

  }
```

Edge

```
/**
 * @author S Djidjev
 *
 * converts index into node pair
 */

public class Edge
{
    private int first;
    private int second;
    private int n; // total number of nodes

    public Edge(int i, int n) {
        this.first = i/n;
        this.second = i-i/n*n;
        this.n=n;
    }

    public void Init(int i, int n) {
        this.first = i/n;
        this.second = i-i/n*n;
        this.n=n;
    }

    public int getFirst() {
        return first;
    }

    public int getSecond() {
        return second;
    }

    public void setFirst(int i) {
        first = i;
    }

    public void setSecond(int i) {
        second = i;
    }

    public String toString(){
           return "(" + first + ", " + second + ")";
    }

    public int index(){// convertes from Edge to int
        return this.first*n+this.second;
    }

}
```

UnionFind

```
/*************************************************************************
 *
 *  Compilation:  java UnionFind.java
 *
 *  This code is adapted from "Algorithms in Java, Third Edition",
 *  by Robert Sedgewick, Addison Wesley Longman, 2003.
 *
 *  Modified by Stephanie Djidjev
 *
 *  Program 4.13: weighted quick-union.
 *
 *************************************************************************/

public class UnionFind {
    private int[] parent;
    private int[] sz;
    private int components;

    // instantiate N isolated components 0 through N-1
    public UnionFind(int N) {
        parent = new int[N];
        sz = new int[N];
        components = N;
        for (int i = 0; i < components; i++) {
            parent[i] = i;
            sz[i] = 1;
        }
    }

    public void printID(){
        System.out.println("Printing UF IDs");
        for(int i =0; i<components; i++){
            System.out.print(i+": "+find(i)+", ");
        }
        System.out.println();
    }

    public int[] computeCommunityNumber(){
        int[]comNumber = new int[components];
        int[]community = new int[components]; //community of each node
        int nextNumber = 0;
        for(int i =0; i<components; i++){
            comNumber[i]= -1;
        }
        for(int i =0; i<components; i++){
            int k = find(i);
            if(comNumber[k] == -1)
                comNumber[k] = nextNumber++;
            community[i]=comNumber[k];

        }

        return community;
    }

    // return parent of component corresponding to element x
    public int find(int x) {
        //System.out.println(" x="+x+" parent[x]="+parent[x]);
        while (x != parent[x])
```

```
            x = parent[x];
        return x;
    }

    // return number of connected components
    public int components() {
        return components;
    }

    // are elements p and q in the same component?
    public boolean compare(int p, int q) {
        return find(p) == find(q);
    }

    public UnionFind copy(){
        UnionFind u = new UnionFind(components);
        for(int i = 0; i < components; i++)
        {
            u.parent[i] = parent[i];
            u.sz[i] = sz[i];
        }
        return u;
    }

    // merge components containing p and q
    public void unite(int p, int q) {
        int i = find(p);
        int j = find(q);
        if (i == j) return;
        if   (sz[i] < sz[j]) { parent[i] = j; sz[j] += sz[i]; }
        else                 { parent[j] = i; sz[i] += sz[j]; }
        //components--;
    }
}
```

arrayGraph

```
import acm.program.*;
import acm.graphics.*;
import java.awt.Color;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
/**
 *
 * @author S Djidjev
 */
public class arrayGraph // because edges are represented by a 2-d array
{
    public int n,n2,m;
    public int[][] edge;
    public int[] degNode;
    public int size()
    {
        return n;
```

```
}

public arrayGraph(int n, int n2)
{
   this.n=n;
   edge = new int[n][n2];
   degNode = new int[n];
}
public arrayGraph copy()
{
   arrayGraph cg = new arrayGraph(this.n,this.n2); //a copied graph
   cg.m=m;
   for(int i = 0; i <n; i ++)
      for(int j = 0; j<n2; j++)
         cg.edge[i][j] =  edge[i][j];
   return cg;
}

public int get(int i, int j)
{
   return edge[i][j];
}

public void computeEdgeCount()
{
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n2; j++)
      {
         if(edge[i][j] >= 1)
         {
            if(i>j) m++;
         }
      }
   }
}

public void printPG(String s,int n0)
{
   System.out.println();
   System.out.println(s);
   int m=0;
   for (int i = 0; i < n; i++)
   {
      Edge e = new Edge(i,n0);
      System.out.print(e.toString()+" -> ");
      if(edge[i][0]>=0){
         for (int j = 0; j < n2; j++)
         {
            int k = edge[i][j];
            Edge e2 = new Edge(k,n0);
            System.out.print(e2.toString()+" ");
         }
         System.out.println();
      }
      else System.out.println("!!!");
```

```
        }
    }

    public int[] degreeCalculator(){
        int[]nodeDegree = new int[n];
        for(int i = 0; i <n; i++){
            for(int j = 0; j <n; j++){
                nodeDegree[i]+=edge[i][j];
            }
        }
        return nodeDegree;
    }

    public void print(String s)
    {
        if(s!=":::")return;
        System.out.println();
        System.out.println(s);
        int m=0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n2; j++)
            {
                System.out.print(edge[i][j]+" ");
                if(edge[i][j] >= 1)
                {
                    if(i>j) m++;
                }
            }
            System.out.println();
        }
        System.out.print("Number of nodes: " + n);

        System.out.println("  Number of edges: " + m);

    }
}
```

Power Iteration Ant Colony Optimization

```
import acm.program.*;
import acm.graphics.*;
import java.awt.Color;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.lang.Object;
import java.io.*;
/**
 * Creates a graph with communities, based off probabilities.
 *
 * @author S Djidjev
 *
 * groupCount refers to how many communities are in the network
 * groupCountArray refers to the size of each community, and is set in an array.
 * p1 refers to the probability of making an edge within the community
 * p2 refers to the probability of making an edge between communities
 *
 * Example: If you want a network with five communities of node sizes 5, 10, 15, 20, and 20, with the probability
 * of making a edge within the community to be 0.8 and making an edge between communities 0.01, you go to the
 * testFreedomSetup() in the editor, and enter the following:
 *
 * int groupCount=5;
 * int[] groupCountArrray=new int[groupCount];
 * int[]groupCountArray = {5,10,15,20,20};
 * networkSetup(groupCount,groupCountArray,0.8,0.01);
 *
 * Then make a freedomGraph instance, and run testFreedomSetup()
 *
 * NOTE: p2 must be less than p1 in order for the network to have communities!
 */

public class ModifiedACO
{
    public int[][] path;
    public int[][] next;
    public int[] listdegree0;
    int n1; //number of nodes in one group
    int n2; // number of nodes in the group after the first
    public int n0; //number of nodes
    public int[][] nodeWeight; //assigns a weight to each new node
    public int maxWeight;
    newAntColony A ;
    int[][] network;

    int nodeCount;
    /**
     * Creates a graph with communities, based off probabilities.
     */
    public ACOArrayGraph networkSetup()
    {

        int n = nodeCount;

        ACOArrayGraph g = new ACOArrayGraph(n,n);

        g.n = n;
        g.n2 = n;
        for (int i =0; i<n; i++)
        {
            for(int j=0; j<n; j++)
            {
                g.edge[i][j]=network[i][j];
            }
        }

        return g;
    }

    public  ModifiedACO(int nodeCount, int[][]network)
    {
        this.network = new int[nodeCount][nodeCount];
        this.network = network;
        this.nodeCount = nodeCount;

        ACOArrayGraph g = new ACOArrayGraph(nodeCount,nodeCount);
        g = networkSetup();

        g.degreeCalculator();

        UnionFind u = new UnionFind(g.n);

        int[][]parent = new int[n0*n0][2]; //edges = matrix for each node it gives its two descendants

        g.computeEdgeCount();
        Edge [] sortedEdges = new Edge[g.m];
        ACOArrayGraph origGraph = g.copy(); // copy of the original graph (needed as g changes inside the next loop)

        A = new newAntColony(origGraph.edge, origGraph.n, 3,origGraph.degNode);
        A.runACOfast();

    }

    public int[][] getPartitionedNetwork()
    {
        int[] communities =  new int[nodeCount];
        communities = A.getBestCommunities();

        int[][] partitionedNetwork = new int[nodeCount][nodeCount];

        for (int i =0; i<nodeCount; i++)
        {
            for (int j =0; j<nodeCount; j++)
            {

                if (communities[i] != communities[j] && network[i][j]>0)
                {

                    partitionedNetwork [i][j]= -1;
                    partitionedNetwork[j][i]= -1;
                }
                else
                    partitionedNetwork[i][j] = network[i][j];
            }
        }
        return partitionedNetwork;
    }

}
```

newAntColony

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.util.*;
/**
 *
 * @author S Djidjev
 */
public class newAntColony
{
    private int[][] network;
    private int numAnts;
    private double [] pheromone;
    private double [] tempPheromone;
    int nodeCount;
    public int[] communityBest;
    double modularityACO;
    double bestModularityACO;

    public newAntColony(int[][] network, int nodeCount, int numAnts,int[] d)
    {
        this.network = network;
        this.nodeCount= nodeCount;
        this.numAnts=numAnts;
    }

    public void assignVectors(){
        pheromone = new double[nodeCount];
        tempPheromone = new double[nodeCount];
        Random generator = new Random();
        for(int i=0;i<nodeCount;i++){
            double p = generator.nextDouble();
            pheromone[i]=p-0.5;
            tempPheromone[i]=0;
        }
    }

    public void runACOfast()
    {
        bestModularityACO = -1;
        int numSteps=30;//nodeCount*4;
        int antLoc;
        int newLoc;
        double eps=0.00001;
        int maxIter = 1000;
        communityBest = new int[nodeCount];
        ACOArrayGraph g = new ACOArrayGraph(nodeCount,nodeCount,network);
        g.computeEdgeCount();
        g.degreeCalculator();

        for (int i=0; i<numAnts; i++)
        {
           // System.out.println("NEXT ANT==================================");
            assignVectors();
            LinkedList<Integer> unvisitedNodes=new LinkedList<Integer>();
            int iterationNumber =0;
            double error=1;
```

```
while(error>eps&&iterationNumber<maxIter){
   iterationNumber++;
   double averPheromone=0;
   for (int k=0; k<nodeCount; k++) averPheromone+=pheromone[k]*g.degNode[k];
   averPheromone/=2*g.m;
   //System.out.println("averPheromone="+averPheromone);
   for (int k=0; k<nodeCount; k++){
      unvisitedNodes.push(k);// todo randomize order
      tempPheromone[k]=-averPheromone*g.degNode[k];;
   }
   //antLoc=(int)(Math.random()*nodeCount);
   while(unvisitedNodes.size()>0){
      antLoc=unvisitedNodes.pop();
      for (int k=0; k<nodeCount; k++)//search for connected nodes
      {
         if (network[antLoc][k]>0)
         {
            // connectedNodes.add(k);
            tempPheromone[antLoc]+= pheromone[k];
         }
      }
      // normalize vector t
   }
   double sum=0;
   for(int m =0;m<nodeCount;m++){
      sum+=tempPheromone[m]*tempPheromone[m];
   }

   sum=Math.sqrt(sum);
   if(sum<0.00000001) {
      System.exit(1);
   }
   for(int m =0;m<nodeCount;m++){
      tempPheromone[m]/=sum;
   }
   // compute the difference with the previous iteration
   for(int m =0;m<nodeCount;m++){
      error = 0;
      error += ((pheromone[m]-tempPheromone[m])*(pheromone[m]-tempPheromone[m]));
   }
   error=Math.sqrt(error);
   error/=nodeCount;
   for(int r=0;r<nodeCount;r++){
      pheromone[r] = tempPheromone[r];
   }
}
double sum =0;
for(int m =0;m<nodeCount;m++) {
   sum+=pheromone[m];

}
// compute the modularity
communityStructure c=new communityStructure(nodeCount);
for(int m =0;m<nodeCount;m++){
   if( pheromone[m] > 0)c.community[m]=0;
   else c.community[m]=1;
```

```
            }

            modularityACO = c.computeModularityACO(g);
         //   System.out.println("MODULARITY ACO="+modularityACO);
            if(modularityACO > bestModularityACO){
                bestModularityACO = modularityACO;
                for(int m =0;m<nodeCount;m++){
                    communityBest[m]=c.community[m];
                }
            }
        }
      //  System.out.println("BEST MODULARITY="+bestModularityACO);
    }


    public void print(){
        System.out.print("communityBest[i]: ");
        for(int m =0;m<nodeCount;m++){
            System.out.print(communityBest[m]+" ");
        }
    }
    public int[] getBestCommunities()
    {
        return communityBest;
    }
}
```

communityStructure

```
/**
 *
 * @author S Djidjev
 */
public class communityStructure
{
    public int[] community;
    public double modularity;
    public double computeModularityACO(ACOArrayGraph g)
    {
        modularity=0;
        for(int i=0; i<g.n; i++){
            for(int j =0; j<g.n; j++) {
                if(community[i]==community[j]){
                    modularity += g.edge[i][j]-(double)g.degNode[i]*g.degNode[j]/(2*g.m);

                }
            }
        }
        modularity/=(2*g.m);
        return modularity;
    }

    public communityStructure(int n){
        community = new int[n];
    }
}
```

ACOarrayGraph

```
import acm.program.*;
import acm.graphics.*;
import java.awt.Color;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
/**
 *
 * @author S Djidjev
 */
public class ACOArrayGraph // because edges are represented by a 2-d array
{
    public int n,n2,m;
    public int[][] edge;
    public int[] degNode;

    public int size()
    {
        return n;
    }

    public ACOArrayGraph(int n, int n2)
    {
        this.n=n;
        this.n2=n2;
        edge = new int[n][n2];
        degNode = new int[n];
    }

    public ACOArrayGraph(int n, int n2, int[][]edgeArray)
    {
        this.n=n;
        this.n2=n2;
        edge = new int[n][n2];
        degNode = new int[n];
        for(int i = 0; i<n;i++)
            for(int j = 0;j<n2;j++)
                this.edge[i][j]=edgeArray[i][j];
    }

    public ACOArrayGraph copy()
    {
        ACOArrayGraph cg = new ACOArrayGraph(this.n,this.n2); //a copied graph
        cg.m=m;
        for(int i = 0; i <n; i ++)
            for(int j = 0; j<n2; j++)
                cg.edge[i][j] =  edge[i][j];
        return cg;
    }

    public int get(int i, int j)
    {
        return edge[i][j];
    }
```

```
public void computeEdgeCount()
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n2; j++)
        {
            if(edge[i][j] >= 1)
            {
                if(i>j) m++;
            }
        }
    }
}

public void printPG(String s,int n0)
{
    System.out.println();
    System.out.println(s);
    int m=0;
    for (int i = 0; i < n; i++)
    {
        Edge e = new Edge(i,n0);
        System.out.print(e.toString()+" -> ");
        if(edge[i][0]>=0){
            for (int j = 0; j < n2; j++)
            {
                int k = edge[i][j];
                Edge e2 = new Edge(k,n0);
                System.out.print(e2.toString()+" ");
            }
            System.out.println();
        }
        else System.out.println("!!!");
    }
}

public void degreeCalculator(){
    for(int i = 0; i <n; i++){
        for(int j = 0; j <n; j++){
            degNode[i]+=edge[i][j];
        }
    }
}

public void print(String s)
{
    if(s!=":::")return;
    System.out.println();
    System.out.println(s);
    int m=0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n2; j++)
        {
            System.out.print(edge[i][j]+" ");
```

```
                if(edge[i][j] >= 1)
                {
                    if(i>j) m++;
                }
            }
            System.out.println();
        }
        System.out.print("Number of nodes: " + n);

        System.out.println("  Number of edges: " + m);

    }
}
```

Network

```
/**
 * model of basic network - includes the visualization of the network
 * also calls partitioning methods upon itself
 *
 * @author A Porter
 *
 */

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.awt.geom.*;
public class network
{

    public static int nodeCount;
    public int[][] network;
    public int[][] bindStatus;//1==  bound/
    public int[][] status;//column 0=infcetion column1=has out connections
    public double[][] xy;
    public double[][] xygoTo;
    public double[][] nodeStep;
    public int[][] path;
    public int[][] next;
    public int[][] pathWeight;// tallies how often the connection is used.
    public int[] weights;
    public String[] names;
    int[] community;
    public int maxNetValue;// maximum value in the network
    public  freedomGraph fgraph;
    public static int WEIGHT1; //girvan newman weights
    public static int WEIGHT2;
    public static int WEIGHT3;
    public static int WEIGHT4;

    public static int ANTSCALE1;
    public static int ANTSCALE2;
    public static int ANTSCALE3;
    public static int ANTSCALE4;
    private final int SCREEN_WIDTH = (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
private final int  SCREEN_HEIGHT = (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();

private double innerProb;
private double betweenProb;
boolean OWLcolor;
int panelWidth;
int panelHeight;
int pathMax;
int pathSecondMax=10;
NetworkPanel netPan;
NetworkFrame netFrame;
int comCount;
double mod;
double prevMod;

double ck;//k for coulumb's law
double hk;//k for hooke's lawy
// private int[][] connectionWeight;

public boolean stopDraw;
private boolean nodesOn;
private boolean connectionsOn;
private boolean labelsOn;
private boolean communitiesAssigned;

public network( int nodesIn, NetworkPanel netPan, NetworkFrame netFrame, double p1, double p2)
{
   nodeCount=nodesIn;

   WEIGHT1=nodeCount/10; //girvan newman weights
   WEIGHT2=nodeCount/5;
   WEIGHT3=nodeCount*3/10;
   WEIGHT4=nodeCount*2/5;

   ANTSCALE1=nodeCount*2/5;
   ANTSCALE2=nodeCount*3/10;
   ANTSCALE3=nodeCount/5;
   ANTSCALE4=nodeCount/10;
   this.netPan=netPan;
   this.netFrame=netFrame;
   hk=10;
   ck=10000;
   OWLcolor = false;
   nodesOn = true;
   connectionsOn = true;
   labelsOn = true;
   innerProb = p1;
   betweenProb = p2;
   communitiesAssigned = false;

   // setup();
}

public void setup()
{
   network=new int[nodeCount][nodeCount];
```

```
        bindStatus=new int[nodeCount][nodeCount];
        status= new int[5][nodeCount];//status[4][i]==use in location indexing
        xy=new double[nodeCount][2];
        xygoTo=new double[nodeCount][2];
        nodeStep=new double[nodeCount][2];
        path=new int[nodeCount][nodeCount];
        next=new int[nodeCount][nodeCount];
        pathWeight=new int[nodeCount][nodeCount];
        weights=new int[nodeCount];

        // randNetwork();
        makeFreedomGraph();

        nodeLocation(xy);//use for intitial node location

    }

    public void makeFreedomGraph()
    {
        randNetwork();
        fgraph=new freedomGraph();

        int groupCount=2;
        int[] groupCountArrray=new int[groupCount];

        int[] groupCountArray = new int[groupCount];// = {10,10,5};
        groupCountArray[0]= nodeCount/2;
        groupCountArray[1]=nodeCount-groupCountArray[0];
        //groupCountArray[2]=nodeCount-groupCountArray[0]-groupCountArray[1];

        network= fgraph.networkSetup( nodeCount,groupCount, groupCountArray, innerProb, betweenProb);
        nameNodes();
    }

    public void setPanelSize(int x, int y)
    {
        panelWidth = x;
        panelHeight = y;
        if (panelWidth!= SCREEN_WIDTH || panelHeight != SCREEN_HEIGHT)// if has been resized
        {
            setup();
            netPan.startAnimation();
        }

    }

    private void randNetwork()
    {
        int randConnect;
        maxNetValue = 0;
        for (int i=0; i<nodeCount; i++)//rows of nodes
        {
            weights[i]=1;
            for(int j=0;j<nodeCount; j++)//cols of nodes
            {
```

```
            randConnect=(int)(Math.random()*nodeCount);//-1,0,1

            if (randConnect!=1)
               randConnect=0;

            // //System.out.println(i+" "+j+" "+ randConnect);
            if (network[i][j]==0 )//no intersection at i and j
            {
               network[i][j]=randConnect*(int)(Math.random()*5);//randConnect;
               network[j][i]=randConnect*(int)(Math.random()*5);//!!reverse direction for opposite connection
               if (network[i][j]>maxNetValue)
                  maxNetValue= network[i][j];
               status[1][i]=1;//true i has an out connection

            }
            if (i==j)
            {
               network[i][j]=0;
               path[i][j]=0;

            }

         }
      }
      nameNodes();
   }

   public  void nodeLocation(double[][] arrayIn)
   {        // System.out.println("nodeLocation");

      for (int i=0; i<nodeCount; i++)
      {
         xy[i][0]=Math.random()*panelWidth;
         xy[i][1]=Math.random()*panelHeight;
      }
   }

   public void draw(Graphics2D g2)
   {

      if (!stopDraw)
      {
         g2.setPaint(Color.BLACK);
         g2.fillRect(0,0,panelWidth, panelHeight);

         for (int i=0; i<nodeCount; i++)
         {
            if (communitiesAssigned)
               g2.setPaint(getNodeColor(community[i]));
            else
               g2.setPaint(Color.WHITE);
            if (nodesOn)
               g2.fillOval((int)xy[i][0]-5,(int)xy[i][1]-5,10,10);

            for (int j=0; j<nodeCount; j++)//go down nodes column looking for 1s=outlines
            {
```

```
            // //System.out.println(status[0][j]);

            if (network[i][j]>0 && connectionsOn  )//existing connection
            {

               g2.setPaint( getColor(i, j, pathWeight[i][j], network[i][j], nodeCount));
               g2.drawLine((int)xy[i][0],(int)xy[i][1], (int)xy[j][0], (int)xy[j][1]);
            }
            else if (network[i][j]<0 && connectionsOn)//former connection
            {
               g2.setPaint(new Color(255,255,255,50));
               g2.drawLine((int)xy[i][0],(int)xy[i][1], (int)xy[j][0], (int)xy[j][1]);
            }

         }
         g2.setPaint(Color.WHITE);
         if(labelsOn)
            g2.drawString(names[i]+"", (int)xy[i][0], (int)xy[i][1]+20);
      }
   }
   else
   {
      g2.setPaint(Color.WHITE);
      g2.drawString("loading....", 100, 100);
   }
}

private  Color getColor(int i, int j, int weight,int networkValue, int nodeCount)
{

   if (OWLcolor == true)// based on optimizationWeight
   {

      if (weight>0 && weight<pathSecondMax/5)
         return new Color(148,0,211);
      if (weight<pathSecondMax * 2/5)
         return Color.BLUE;
      if (weight<pathSecondMax * 3/5)
         return Color.GREEN;
      if (weight<pathSecondMax*4/5)
         return Color.ORANGE;
      else
         return Color.RED;
   }
   else
   {

      if (networkValue>0 && networkValue<maxNetValue/5)
         return new Color(148,0,211);
      if (networkValue<maxNetValue * 2/5)
         return Color.BLUE;
      if (networkValue<maxNetValue * 3/5)
         return Color.GREEN;
      if (networkValue<maxNetValue*3/5)
         return Color.ORANGE;
      else
```

```
            return Color.RED;
        }
    }

    public void move()
    {
        // System.out.println("move");
        if (!stopDraw)
        {
            double kineticE=0.0;
            double[] force =new double[2];
            double[] nodeVelocity =new double[2];

            double timestep=0.1;
            double damping=.01;

            // double eMin
            //  while (kineticE> eMin)
            // {

            for (int i=0; i<nodeCount; i++)
            {

                force[0]=0;
                force[1]=0;
                nodeVelocity[0]=0;
                nodeVelocity[1]=0;
                for (int j=0; j<nodeCount; j++)
                {
                    if (i!=j)
                    {
                        // System.out.println(getDistance(i,j));
                        force[0] -=ck*weights[i]*weights[j]/getXDistance(i,j);
                        force[1] -=ck*weights[i]*weights[j]/getYDistance(i,j);

                        if (network[i][j]>0)//existing connection
                        {

                            force[0] +=hk*getXDistance(i,j);
                            force[1] += hk*getYDistance(i,j);
                        }
                    }

                }

                force[0] += ck*weights[i]*weights[i]/(xy[i][0]-panelWidth);
                force[1] += ck*weights[i]*weights[i]/(xy[i][1]-panelHeight);

                force[0] += ck*weights[i]*weights[i]/(xy[i][0]);
                force[1] += ck*weights[i]*weights[i]/(xy[i][1]);

                nodeVelocity[0]=(nodeVelocity[0]+timestep*force[0])*damping;
                nodeVelocity[1]=(nodeVelocity[1]+timestep*force[1])*damping;
                // System.out.println(i+" "+force[0]+ " "+force[1]);
                //if (xy[i][0]+timestep*nodeVelocity[0]<panelWidth-100 && xy[i][0]+timestep*nodeVelocity[0]>0)
                xy[i][0]+=timestep*nodeVelocity[0];
                // if (xy[i][1]+timestep*nodeVelocity[1]<panelHeight-100 && xy[i][1]+timestep*nodeVelocity[1]>0)
                xy[i][1]+=timestep*nodeVelocity[1];
                // kineticE+=weights[i]*(nodeVelocity[1]*nodeVelocity[1] + nodeVelocity[0]*nodeVelocity[0]);
                //System.out.println(kineticE);
            }
        }

    }

    private double getXDistance (int i, int j)
    {
        //  System.out.println(xy[i][0]+" "+xy[i][1]);
        if (xy[i][0]< xy[j][0])
            return Math.sqrt((xy[i][0]-xy[j][0])*(xy[i][0]-xy[j][0])+(xy[i][1]-xy[j][1])*(xy[i][1]-xy[j][1]));
        else
            return -1*Math.sqrt((xy[i][0]-xy[j][0])*(xy[i][0]-xy[j][0])+(xy[i][1]-xy[j][1])*(xy[i][1]-xy[j][1]));

    }

    private double getYDistance (int i, int j)
    {
        if (xy[i][1]<xy[j][1])
            return Math.sqrt((xy[i][0]-xy[j][0])*(xy[i][0]-xy[j][0])+(xy[i][1]-xy[j][1])*(xy[i][1]-xy[j][1]));
        else
            return -1*Math.sqrt((xy[i][0]-xy[j][0])*(xy[i][0]-xy[j][0])+(xy[i][1]-xy[j][1])*(xy[i][1]-xy[j][1]));
    }

    public void setHookeK(double kin)
    {
        hk=kin;
    }

    public void setCoulombK(double kin)
    {
        ck=kin;
    }

    public void gnPartition(boolean threadSwitch, boolean checkMod)
    {
        if( !checkMod)
        {
            if (threadSwitch)
            {
                parallelGN pgn= new parallelGN(network, nodeCount);
                pathWeight= pgn.getPathWeights();
                removeMostTravelled();
            }
            else
            {

                girvanNewman linearGn = new girvanNewman(network, nodeCount);
                pathWeight= linearGn.getPathWeights();

                int pMax = removeMostTravelled();

            }
        }
        else
        {
            if (threadSwitch)
            {

                prevMod = -100;
                mod = -100;
                while (prevMod<=mod || mod< 0)
                {
                    parallelGN pgn= new parallelGN(network, nodeCount);
                    pathWeight= pgn.getPathWeights();
                    removeMostTravelled();

                    prevMod = mod;
                    mod = calcModularity();
                    if (mod<prevMod && mod> 0)
                    {
                        mod = prevMod;
                        break;
                    }

                }
            }
            else
            {
                prevMod = -100;
                mod = -100;
                while (prevMod<=mod || mod< 0)
                {
                    girvanNewman linearGn = new girvanNewman(network, nodeCount);
                    pathWeight= linearGn.getPathWeights();

                    removeMostTravelled();
                    prevMod = mod;
                    mod = calcModularity();
                    if (mod<prevMod && mod> 0)
                    {
                        mod = prevMod;
                        break;
                    }

                }
            }
        }

    }

    public void hgnPartition(boolean checkMod, int loopCount)
    {

        prevMod = -100;
        mod = -100;
        int  idealMod = Math.abs(loopCount);
        if (checkMod)
        {
            HierarchalGirvanNewman  hgn= new HierarchalGirvanNewman();//network,nodeCount);
            hgn.FreedomSetup(network,nodeCount);
            pathWeight = hgn.getEdgesRanked();
            while (prevMod<=mod || mod< 0 || comCount < idealMod)
            {

                removeHighestRank();
                prevMod = mod;
                mod = calcModularity();

                if (mod<prevMod && mod> 0 && comCount==idealMod)
                {
                    mod = prevMod;
                    break;
                }

            }
        }
        else
        {
            HierarchalGirvanNewman  hgn= new HierarchalGirvanNewman();//network,nodeCount);
            hgn.FreedomSetup(network,nodeCount);
            pathWeight = hgn.getEdgesRanked();
            for (int i=0; i<loopCount; i++)
                removeHighestRank();
        }

    }

    public void gnLocPartition( )
    {

        //        LocationGN locGN = new LocationGN(network, xy,nodeCount);
        //        pathWeight= locGN.getPathWeights();
        //        // removeMostTravelled();
        //        distWeightRemoveHighest();

    }

    public void distWeightRemoveHighest()
    {//distances between nodes added (or multiplied into?) weight

        int pathMax;
        int maxj;
        int maxk;
        //      girvanNewman();
        for (int i=0; i<1; i++)//arbitrary loop, how many levels to divide
        {
            pathMax=0;
            maxj=-1;
            maxk=-1;
            for (int j=0; j<nodeCount; j++)//begin search for max used path
            {
                for (int k=0; k<nodeCount; k++)
                {
                    if (pathWeight[j][k]>0)
                        pathWeight[j][k] += Math.sqrt((xy[j][0]-xy[k][0])*(xy[j][0]-xy[k][0])  + (xy[j][1]-
    xy[k][1])*(xy[j][1]-xy[k][1]));//distance between j and k
                    //  System.out.println(j+" " +k+" "+ pathWeight[j][k]+" "+Math.sqrt(xy[j][0]*xy[k][0] + xy[j][1]
    *xy[k][1]));
                    if (pathWeight[j][k]>pathMax && network[j][k]>0)
                    {
                        pathMax=pathWeight[j][k];
                        maxj=j;
                        maxk=k;
                        pathSecondMax=pathMax;

                    }
                }

            }
            //    System.out.println("pathMax: "+pathMax+" k: "+maxk+" j: "+maxj);
            if (maxj!=-1 && maxk!=-1)
            {

                network[maxj][maxk]=-network[maxj][maxk];
                network[maxk][maxj]=-network[maxk][maxj];

                pathWeight[maxj][maxk]=0;
                pathWeight[maxk][maxj]=0;
            }
        }

    }

    public void removeHighestRank()
    {
        int rankBest=nodeCount;
        int iBest=-1;
        int jBest=-1;
        for (int i =0; i< nodeCount; i++)
        {
            for (int j=0; j<nodeCount; j++)
            {
                if (pathWeight[i][j]>0 && pathWeight[i][j]<rankBest)
                {
                    rankBest = pathWeight[i][j];
                    iBest = i;
                    jBest = j;
                }
            }
        }

        network[iBest][jBest]=-network[iBest][jBest];
        network[jBest][iBest]=-network[jBest][iBest];

        pathWeight[iBest][jBest]=0;
        pathWeight[jBest][iBest]=0;

    }
```

```
public int removeMostTravelled()
{
   int pathMax=0;
   int maxj;
   int maxk;

   //       girvanNewman();
   for (int i=0; i<1; i++)//arbitrary loop, how many levels to divide
   {
      pathMax=0;
      maxj=-1;
      maxk=-1;
      for (int j=0; j<nodeCount; j++)//begin search for max used path
      {
         for (int k=0; k<nodeCount; k++)
         {
            if (pathWeight[j][k]>pathMax && network[j][k]>0)
            {
               pathMax=pathWeight[j][k];
               maxj=j;
               maxk=k;
               pathSecondMax=pathMax;

            }
         }

      }
      //    System.out.println("pathMax: "+pathMax+" k: "+maxk+" j: "+maxj);
      if (maxj!=-1 && maxk!=-1)
      {

         network[maxj][maxk]=-network[maxj][maxk];
         network[maxk][maxj]=-network[maxk][maxj];

         pathWeight[maxj][maxk]=0;
         pathWeight[maxk][maxj]=0;
      }
   }
   // nodeLocation();
   return pathMax;
}

public void acoPartition(boolean threadSwitch)
{

   ModifiedACO mACO = new ModifiedACO(nodeCount, network);
   network = mACO.getPartitionedNetwork();
}


public void removeLeastTravelled()
{
   int pathMin;
   int minj;
   int mink;
```

```
pathMin=Integer.MAX_VALUE-1;
minj=-1;
mink=-1;
// ArrayList<Integer> equalMins = new ArrayList<Integer>;
ArrayList<Integer> jlist = new ArrayList<Integer>();
ArrayList<Integer> klist = new ArrayList<Integer>();

for (int j=0; j<nodeCount; j++)//begin search for min used path
{
   for (int k=0; k<nodeCount; k++)
   {
      if(k!=j)
      {
         if (pathWeight[j][k]<pathMin  && network[j][k]>0)
         {
            pathMin=pathWeight[j][k];
            minj=j;
            mink=k;
            jlist = new ArrayList<Integer>();// clear for new path weight f
            klist = new ArrayList<Integer>();
            //System.out.println("removed: "+j+" "+k);
            // //System.out.println("pathMin:"+pathMin);
         }

         if (pathWeight[j][k]==pathMin  && network[j][k]>0)
         {

            jlist.add(j);
            klist.add(k);
            //System.out.println("multi!" + pathWeight[j][k]);
         }
      }
   }

}

//         minj= jlist.get(randMin);//randomly choose from those that are equal
//         mink = klist.get(randMin);
//    //System.out.println("pathMax: "+pathMax+" k: "+maxk+" j: "+maxj);

if (jlist.size()>0)
{
   int randMin =(int)( Math.random()*jlist.size());
   // System.out.println("randMin"+randMin);
   minj = jlist.get(randMin);
   mink = klist.get(randMin);
   network[minj][mink]=-network[minj][mink];
   network[mink][minj]=-network[mink][minj];
   pathWeight[minj][mink]=0;
   pathWeight[mink][minj]=0;
}

else  if (minj!=-1 && mink!=-1)
{
   network[minj][mink]=-network[minj][mink];
```

```
        network[mink][minj]=-network[mink][minj];
        pathWeight[minj][mink]=0;
        pathWeight[mink][minj]=0;

    }

    //nodeLocation(xy);

}

public void loopPartition(int oi, int numLoops, boolean threadSwitch)//optimization index in
{

    if (numLoops<0)
    {
        switch(oi)
        {
            case 0:
            gnPartition(threadSwitch, true);
            break;

            case 2:
            gnLocPartition();
            case 3:
            hgnPartition(true, Math.abs(numLoops));
        }
    }
    else
    {
        for (int i=0; i< numLoops; i++)
        {
            switch(oi)
            {
                case 0:
                gnPartition(threadSwitch, false);
                break;
                //case 1:

                case 2:
                gnLocPartition();

            }
        }
        if (oi ==3)
            hgnPartition(false, numLoops);
       if (oi ==1)
            acoPartition(threadSwitch);
    }

}

public void setNetwork(int[][] network, int nodeCount, String[] names, int[] nodeWeights)
{
    this.network= network;
    this.nodeCount= nodeCount;
    this.names=names;
    weights= nodeWeights;
    stopDraw = false;
}

public void newNetwork( int nodeCount)
{

    netPan.running =false;
    this.nodeCount = nodeCount;
    netPan.nodeCount = nodeCount;
    setup();
    netPan.running =true;
    stopDraw = false;
}

public void setLineStyle (boolean opWeightLineColor)
{
    OWLcolor=opWeightLineColor;
}

public void nameNodes()
{
    names= new String[nodeCount];
    for (int i=0; i<nodeCount; i++)
    {
        names[i]=i+" ";
    }

}

public String[] getNames()
{ return names; }

public double calcModularity()
{

    double modularity=0;
    int[]degNode = new int[nodeCount]; //length
    int edgeCount = 0;
    community = new int[nodeCount];
    assignCommunities(nodeCount);
    for(int i = 0; i <nodeCount; i++)
    {
        for(int j = 0; j <nodeCount; j++) // all edges degNode[i] += network[i][j]
        {
            if (network[i][j]>0)
            {
                degNode[i]+=network[i][j]; //can't just add network because weights vary
                edgeCount++;
            }
        }
    }
    edgeCount = edgeCount/2;//all are added twice in above method

    for (int i = 0 ; i<nodeCount; i++)
    {
        for (int j= 0; j<nodeCount; j++)
        {
            if (community[i]==community[j])
            {
                modularity +=  network[i][j] - (double)degNode[i]*degNode[j]/(2*edgeCount);
            }
        }
    }

    modularity/=(2*edgeCount);

    return modularity;
    /* writer.newLine();
    for (int i= 0; i<communityCount; i++)
    {
    writer.write(partitionedIn[i]+" ");
    }*/

}

public void assignCommunities(int nodeCount)
{

    comCount = 1;
    //System.out.println("center: "+imaxDegree);

    for (int i=0; i<nodeCount; i++)
        community[i]=-1;
    community[0] = 0;
    int nodesAssigned=1;//starting center was assigne
    int iCurrent=0;//current center index
    String indexStack=0+"";//stack of centers;
    String is2="";//index stack # 2
    int is2size=0;

    Scanner indexStackScan= new Scanner(indexStack);

    while (nodesAssigned<nodeCount-1)
    {

        //System.out.println(indexStack);

        while (indexStackScan.hasNextInt())//go through stack of centers
        {
            iCurrent=indexStackScan.nextInt();

            // nodesAssigned+=1;
            for (int i=0; i<nodeCount; i++)//check for nodes around current center
            {
                if (network[i][iCurrent]>0 && community[i]==-1)//if match with center and status is "unused"
                {
                    is2=is2+" "+i;
                    is2size++;
                    community[i]= community[iCurrent];
                    nodesAssigned++;
                }
              }

              //       System.out.println("node assigned: "+circleIndex);
              //assign indices aroudn circle
            }
            //  System.out.println("center locations: "+centerLocation);
            indexStack=is2;//centers tracks all new centers, not just those for one previous center
            is2="";  //clear is2
            is2size=0;

            //indexStack=centers;
            indexStackScan=new Scanner(indexStack);//update scanner to add new centers

            if (nodesAssigned<nodeCount && !indexStackScan.hasNextInt())//if no more connected to current branch
group
            {
              int nextNode=1;

              while (community[nextNode]!=-1 && nextNode<(nodeCount-1))
              {
                nextNode++;

              }
              if (nextNode==(nodeCount-1) && community[nextNode]!=-1)
                nodesAssigned=nodeCount;

              else
              {
                indexStack=""+nextNode;
                community[nextNode] = community[iCurrent]+1;
                comCount++;

                indexStackScan=new Scanner(indexStack);//update scanner to add new centers
              }
            }
          }
          communitiesAssigned = true;
        }

        private Color getNodeColor(int com)
        {

          if (com % 6 ==0)
            return Color.RED;
          if (com % 6 == 1)
            return Color.ORANGE;
          if (com % 6==2)
            return Color.YELLOW;
          if (com % 6== 3)
            return Color.GREEN;
          if (com % 6== 4)
            return Color.BLUE;
          if (com % 6 ==5)
            return Color.MAGENTA;
        return Color.WHITE;
    }

    public void setNodeVis(boolean nodesOn)
    {
        this.nodesOn = nodesOn;
    }

    public void setConnectionVis(boolean in)
    {
        connectionsOn = in;
    }

    public void setLabelVis(boolean in)
    {
        labelsOn = in;
    }

    public void setInnerProb (double pIn)
    {
        innerProb = pIn;
    }
}
```

Network Frame

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.util.*;
/**
 *
 * @author A Porter
 * Creates the interactive frame containing buttons, sliders, textfields, and the network panel
 */
public class NetworkFrame {

    private static final int FRAME_HEIGHT = (int)Toolkit.getDefaultToolkit().getScreenSize().getHeight() - 80;

    //(fractalPanel + settingsPanel + content.border + frame.border) = fractalPanel.width + 210 + 20 + 16 = width + 
246;
    private static final int FRAME_WIDTH = FRAME_HEIGHT - 58 + 246;

    //default settings
    private static int optimizationIndex=3;
    private static int nodeCount = 200;
    private static int loopCount=10;
    private static double runTime=0.0;
    private static long startTime;
    private static boolean threadSwitch=false;
private static boolean opWeightLineColor= false;
private double innerProb = 0.1;
private double betweenProb = 0.05;

private static String fileName;
private static BufferedImage backgroundImage;
static Border myBorder = BorderFactory.createLineBorder(new Color(255,255,255));
static Font myFont = new Font("Serif", Font.PLAIN, 18);
NetworkPanel networkP;
dataRecorder dataRec;
NetworkFrame()
{
   createAndShowGUI();
}

private  void createAndShowGUI()
{

   JToolBar toolbar = new JToolBar();
   final JFrame frame = new JFrame("network");
   networkP= new NetworkPanel(this, nodeCount, optimizationIndex, innerProb, betweenProb);
   dataRec= new dataRecorder();

   final  JTextArea lineStyle = new JTextArea();
   lineStyle.setEditable(false);
   lineStyle.setText("Line Color Style: ");

   final JRadioButton weightLineColors = new JRadioButton("Weight colors");
   weightLineColors.setSelected(true);
   weightLineColors.setFont(myFont);

   weightLineColors.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {

             opWeightLineColor = false;
             networkP.net.setLineStyle(opWeightLineColor);
          }
      });

   final JRadioButton colorOptimizeWeight = new JRadioButton("Optimization colors");
   colorOptimizeWeight.setSelected(false);
   colorOptimizeWeight.setFont(myFont);
   colorOptimizeWeight.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {

             opWeightLineColor = true;
             networkP.net.setLineStyle(opWeightLineColor);
          }
      });
   ButtonGroup lineColorType = new ButtonGroup();
   lineColorType.add(weightLineColors);
   lineColorType.add(colorOptimizeWeight);

   final JSlider hookesKSlider = new JSlider(0, 50, 10);
   hookesKSlider.setMajorTickSpacing(10);
   hookesKSlider.setPaintTicks(true);
hookesKSlider.setPaintLabels(true);
hookesKSlider.setBorder(new TitledBorder(myBorder, "Connection strength:",
        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
        myFont, Color.BLACK));
hookesKSlider.setOpaque(false);
hookesKSlider.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            if(!hookesKSlider.getValueIsAdjusting()) {

                networkP.net.setHookeK(hookesKSlider.getValue());

            }
        }
    });

final JSlider coulombKSlider = new JSlider(0, 50, 1);
coulombKSlider.setMajorTickSpacing(10);
coulombKSlider.setPaintTicks(true);
coulombKSlider.setPaintLabels(true);
coulombKSlider.setBorder(new TitledBorder(myBorder, "Node repellence:",
        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
        myFont, Color.BLACK));
coulombKSlider.setOpaque(false);
coulombKSlider.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            if(!coulombKSlider.getValueIsAdjusting()) {
                networkP.net.setCoulombK(coulombKSlider.getValue()*5000);
            }
        }
    });
final JCheckBox showNodes = new JCheckBox("Show nodes", true);
showNodes.setFont(myFont);
showNodes.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            networkP.net.setNodeVis(showNodes.isSelected());

        }
    });
final JCheckBox showConnections = new JCheckBox("Show connections", true);
showConnections.setFont(myFont);
showConnections.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {

            networkP.net.setConnectionVis(showConnections.isSelected());
        }
    });
final JCheckBox showLabels = new JCheckBox("Show labels", true);
showLabels.setFont(myFont);
showLabels.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            networkP.net.setLabelVis(showLabels.isSelected());
        }
    });

JPanel stylePanel = new JPanel();
stylePanel.setLayout(new GridLayout(0, 1, 10, 10));
stylePanel.setBorder(new TitledBorder(myBorder, "Visualization Settings",
        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
        myFont, Color.BLACK));
stylePanel.add(weightLineColors);
stylePanel.add(colorOptimizeWeight);

stylePanel.add(hookesKSlider);
stylePanel.add(coulombKSlider);
stylePanel.add(showNodes);
stylePanel.add(showConnections);
stylePanel.add(showLabels);

stylePanel.setOpaque(false);

//  stylePanel.add(lineStyle);
final JSlider nodeCountSlider = new JSlider(0, 300, nodeCount);
nodeCountSlider.setMajorTickSpacing(50);
nodeCountSlider.setPaintTicks(true);
nodeCountSlider.setPaintLabels(true);
nodeCountSlider.setBorder(new TitledBorder(myBorder, "# of Nodes:  "+nodeCount,
        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
        myFont, Color.BLACK));
nodeCountSlider.setOpaque(false);
nodeCountSlider.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            if(!nodeCountSlider.getValueIsAdjusting()) {
                nodeCount = nodeCountSlider.getValue();
                nodeCountSlider.setBorder(new TitledBorder(myBorder, "# of Nodes:  "+nodeCount,
                        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
                        myFont, Color.BLACK));
            }
        }
    });

final  JTextArea withinProbText = new JTextArea();
withinProbText.setEditable(false);
withinProbText.setText("Probability Within Communities: ");

final JTextField withinProbField= new JTextField(5);
withinProbField.setText("enter a double");
withinProbField.selectAll();
withinProbField.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            Scanner readDouble= new Scanner( withinProbField.getText());
            innerProb = readDouble.nextDouble();
        }
    });

final  JTextArea betweenProbText = new JTextArea();
betweenProbText.setEditable(false);
betweenProbText.setText("Probability Between Communities: ");

final JTextField betweenProbField= new JTextField(15);
betweenProbField.setText("enter a double");
betweenProbField.selectAll();
betweenProbField.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            Scanner readDouble= new Scanner( betweenProbField.getText());
            betweenProb = readDouble.nextDouble();
        }
    });

JButton newNetButton=new JButton("New Network");
newNetButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            //                  networkP.net.stopDraw = true;
            //                  networkP.net.newNetwork(nodeCount);
            //                  weightLineColors.setSelected(true);
            //                  colorOptimizeWeight.setSelected(false);
            //                  opWeightLineColor= false;
            //                  nodeCountSlider.setValue(nodeCount);
            networkP.newNetwork(nodeCount, innerProb, betweenProb );
            networkP.net.setCoulombK(coulombKSlider.getValue()*5000);
            networkP.net.setHookeK(hookesKSlider.getValue());
        }
    });

JPanel sliderPanel =  new JPanel();
sliderPanel.setLayout(new GridLayout(0, 1));
sliderPanel.add(nodeCountSlider);
sliderPanel.add(newNetButton);
sliderPanel.setOpaque(false);
JPanel newNetPanel =  new JPanel();
newNetPanel.setLayout(new GridLayout(0, 1));

newNetPanel.add(withinProbText);
newNetPanel.add(withinProbField);
newNetPanel.add(betweenProbText);
newNetPanel.add(betweenProbField);
//  newNetPanel.add(newNetButton);
newNetPanel.setBorder(new TitledBorder(myBorder, "Network Creation",
        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
        myFont, Color.BLACK));
newNetPanel.setOpaque(false);

JButton partitionButton=new JButton("girvan newman partition");
partitionButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            //  networkP.net.gnPartition(threadSwitch);
        }
    });

JButton ACOpartitionButton=new JButton("ACO partition");
ACOpartitionButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            //   networkP.net.acoPartition(threadSwitch);
        }
    });
```

```
        final  JTextArea displayTime = new JTextArea();
        displayTime.setEditable(false);
        displayTime.setText("Run Time: ");

        final  JTextArea modularityText = new JTextArea();
        modularityText.setEditable(false);
        modularityText.setText("Modularity: ");

        JButton runPartitionButton=new JButton("Run Partition");
        runPartitionButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    //  networkP.net.acoPartition();
                    // System.out.println(loopCount);
                    startTime = System.currentTimeMillis();
                    networkP.net.loopPartition(optimizationIndex,loopCount, threadSwitch);

                    runTime = System.currentTimeMillis() - startTime;
                    displayTime.setText("Run Time: "+runTime);
                    dataRec.recordRun("name", networkP.net.nodeCount, optimizationIndex, threadSwitch, loopCount,
runTime);
                    modularityText.setText("Modularity: "+roundDecimals(networkP.net.calcModularity()));
                }
            });

        JRadioButton ACORadio = new JRadioButton("ACO");

        ACORadio.setSelected(false);
        ACORadio.setFont(myFont);
        ACORadio.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    // networkP.setParameters(nodeCount,0);

                    optimizationIndex=1;
                    //  networkP.reset(nodeCount,optimizationIndex);
                }
            });

        JRadioButton gnRadio = new JRadioButton("GirvanNewman");
        gnRadio.setSelected(false);
        gnRadio.setFont(myFont);
        gnRadio.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    // networkP.setParameters(nodeCount,1);
                    // networkP.net.loopPartition(0,10);
                    optimizationIndex=0;
                    //  networkP.reset(nodeCount,optimizationIndex);
                }
            });

        JRadioButton hgnRadio = new JRadioButton("HGN");
        hgnRadio.setSelected(true);
        hgnRadio.setFont(myFont);
        hgnRadio.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
```

```
            // networkP.setParameters(nodeCount,1);
            // networkP.net.loopPartition(0,10);
            optimizationIndex=3;
            //  networkP.reset(nodeCount,optimizationIndex);
        }
    });

JRadioButton gnLocRadio = new JRadioButton("GN Loc");
gnLocRadio.setSelected(false);
gnLocRadio.setFont(myFont);
gnLocRadio.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            // networkP.setParameters(nodeCount,1);
            // networkP.net.loopPartition(0,10);
            optimizationIndex=2;
            //  networkP.reset(nodeCount,optimizationIndex);
        }
    });

ButtonGroup optimizationType = new ButtonGroup();
optimizationType.add(ACORadio);
optimizationType.add(gnRadio);
//  optimizationType.add(gnLocRadio);
optimizationType.add(hgnRadio);

final JTextField opRuns= new JTextField();
opRuns.setText("enter an integer");
opRuns.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            Scanner countScan =new Scanner(opRuns.getText());
            if (countScan.hasNextInt())
            {
                loopCount=countScan.nextInt();

            }

        }
    });

final  JTextArea threadText = new JTextArea();
threadText.setEditable(false);
threadText.setText("Use Multi-Threading? ");

JRadioButton multiOn = new JRadioButton("Yes");
multiOn.setSelected(false);
multiOn.setFont(myFont);
multiOn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            threadSwitch= true;
        }
    });
```

```
JRadioButton multiOff = new JRadioButton("No");
multiOff.setSelected(true);
multiOff.setFont(myFont);
multiOff.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            threadSwitch=false;

        }
    });

ButtonGroup multiThread = new ButtonGroup();
multiThread.add(multiOn);
multiThread.add(multiOff);

JPanel oTypePanel = new JPanel();
oTypePanel.setLayout(new GridLayout(0, 1));
oTypePanel.add(runPartitionButton);
oTypePanel.add(gnRadio);
oTypePanel.add(ACORadio);
// oTypePanel.add(gnLocRadio);
oTypePanel.add(hgnRadio);
oTypePanel.add(opRuns);
oTypePanel.add(threadText);
oTypePanel.add(multiOn);
oTypePanel.add(multiOff);
oTypePanel.add(displayTime);
oTypePanel.add(modularityText);

oTypePanel.setBorder(new TitledBorder(myBorder, "Partitioning",
        TitledBorder.CENTER, TitledBorder.ABOVE_TOP,
        myFont, Color.BLACK));
oTypePanel.setOpaque(false);
JPanel buttonPanel = new JPanel(new GridLayout(0, 1, 10, 10));
buttonPanel.setLayout(new FlowLayout());
buttonPanel.add(partitionButton, BorderLayout.WEST);
buttonPanel.add(ACOpartitionButton, BorderLayout.EAST);

buttonPanel.setOpaque(false);

JPanel settingsPanel = new JPanel(new BorderLayout(10, 10));
settingsPanel.add(newNetPanel, BorderLayout.NORTH);
settingsPanel.add(sliderPanel, BorderLayout.CENTER);
settingsPanel.add(oTypePanel, BorderLayout.SOUTH);
// settingsPanel.add(oTypePanel, BorderLayout.CENTER);
//settingsPanel.add(buttonPanel, BorderLayout.CENTER);
settingsPanel.setOpaque(false);
try {
    backgroundImage = ImageIO.read(new File("background.jpg"));
} catch (IOException e) {
    System.out.println("fail");
}
JPanel content = new JPanel(new BorderLayout()) {
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.setColor(new Color (250, 250, 250));
            g.fillRect(0, 0, getWidth(), getHeight());
            // g.drawImage(backgroundImage,
            //   0, 0, getWidth(), getHeight(),
            //   0, 0, 450, 315,
            //   null);
          }

        };

    //text field, button to import and display
    final  JTextArea filePrompt = new JTextArea();
    filePrompt.setEditable(false);
    filePrompt.setText("Enter file name: ");

    final JTextField fileField= new JTextField(15);
    fileField.setText("filename");
    fileField.selectAll();
    fileField.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {

            fileName = fileField.getText();

            // networkP.net.setNodeWeights(fileIO.getNodeWeights());
            // networkP.net.setNodeNames(fileIO.getNodeNames());
            //  fileField.setText("");
            fileField.selectAll();
          }
        });

    JButton loadNetworkButton=new JButton("Load Network from File");
    loadNetworkButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            txtFileInterface fileIO = new txtFileInterface(fileName);
            networkP.net.stopDraw= true;
            nodeCount = fileIO.getNodeCount();
            networkP.net.setNetwork(fileIO.getNetwork(), fileIO.getNodeCount(), fileIO.getNames(),
fileIO.getNodeWeights());
            nodeCountSlider.setValue(nodeCount);
          }
        });

    final  JTextArea outFilePrompt = new JTextArea();
    outFilePrompt.setEditable(false);
    outFilePrompt.setText("Export File: ");
    final JTextField outFileField= new JTextField(15);
    outFileField.setText("filename");
    outFileField.selectAll();
    outFileField.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {

            fileName = outFileField.getText();
            txtFileInterface fileIO = new txtFileInterface(fileName, networkP.net);
            outFileField.selectAll();
          }
        });

    JButton writeDataButton=new JButton("Write Network to File");
```

```
writeDataButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
             dataRec.writeToFile("dataOut");

        }
    });

       JButton openNetworkMaker=new JButton("Open Network Maker");
openNetworkMaker.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            RowAdder ra = new RowAdder();
          //  ra.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            ra.setSize(600, 300);
            ra.setVisible(true);
        }
    });

JPanel importPanel =  new JPanel(new FlowLayout());// BorderLayout(10,10));
importPanel.add(filePrompt);
importPanel.add(fileField);
importPanel.add(loadNetworkButton);
importPanel.add(outFilePrompt);
importPanel.add(outFileField);
importPanel.add(writeDataButton);
importPanel.add(openNetworkMaker);

importPanel.setOpaque(false);

content.add(networkP, BorderLayout.CENTER);
content.add(settingsPanel, BorderLayout.EAST);
content.add(importPanel, BorderLayout.SOUTH);

//  toolbar.add(importPanel);
content.add(stylePanel, BorderLayout.WEST);

content.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
content.setOpaque(false);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setExtendedState(Frame.MAXIMIZED_BOTH);
//  frame.add(toolbar, BorderLayout.SOUTH);//PAGE_START);

// JToolBar toolbar2 = new JToolBar();
// toolbar2.add(stylePanel);

//  frame.add(toolbar2, BorderLayout.WEST);

frame.add(content);
// frame.pack();
//  frame.add(buttonPanel, BorderLayout.SOUTH);
frame.setVisible(true);

frame.addWindowListener(new
    WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            Window win = e.getWindow();
            win.setVisible(false);
```

```
                win.dispose();
                System.exit(0);
            }
        });

        frame.addComponentListener(new ComponentAdapter() {
            public void componentResized(ComponentEvent e)
            {
                networkP.net.setPanelSize(networkP.getWidth(), networkP.getHeight());
                // networkP.net.setup();
            }
        });

    }

    double roundDecimals(double d) {
        int ix = (int)(d * 10000.0); // scale it
        double dbl2 = ((double)ix)/10000.0;
        return dbl2; //dbl=12.3456, dbl2=12.34i
    }

    public static void main(String args[])
    {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new NetworkFrame();
            }
        });
    }
}
```

Network Panel

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
/**
 *
 * @author A Porter
 */
public class NetworkPanel extends JPanel  implements Runnable
{
    private int PANEL_WIDTH, PANEL_HEIGHT;

    private static final int FRAME_HEIGHT = (int)Toolkit.getDefaultToolkit().getScreenSize().getHeight();
    private static final int FRAME_WIDTH = FRAME_HEIGHT ;

   public  int nodeCount;

    private static final int DEFAULT_FPS = 60;
    private static final int DEFAULT_PAUSE_TIME = 1000; // in ms
    private static final int NO_DELAYS_PER_YIELD = 16;
    /* Number of frames with a delay of 0 ms before the
    animation thread yields to other tunning threads. */
```

```
private static int MAX_FRAME_SKIPS = 5;
/* Number of frames that can be skipped in any one animation loop
i.e the game state is updated but not rendered*/
public Thread animator;

public long period; // in ns
private long pauseTime; // in ms
private Graphics2D dbg2;
private BufferedImage dbImage = null;

private int optimizationIndex;
network net;
int movementFramesCount;
int numberOfSteps;
public volatile boolean running;
int initHeight;
int initWidth;
NetworkFrame frame;
NetworkPanel(NetworkFrame frame, int nodeCountIn, int optimizationIndex, double p1, double p2)    {
   this.frame= frame;
   this.optimizationIndex=optimizationIndex;

   nodeCount=nodeCountIn;
   running=true;
   setSize(Toolkit.getDefaultToolkit().getScreenSize());

   net = new network(nodeCount, this, frame, p1, p2);
   net.setPanelSize(getWidth(), getHeight());
   //net.setPanelSize(getWidth, getHeight);
}

public void newNetwork(int nodeCount, double p1, double p2)
{
   running = false;

   this.nodeCount = nodeCount;
   net = new network(nodeCount, this, frame, p1, p2);
   net.setPanelSize(getWidth(), getHeight());
   running = true;

}

public void reset(int nodeCountIn, int optimizationIndex)
{
   this.optimizationIndex=optimizationIndex;
   nodeCount=nodeCountIn;
   running=false;

   //       /if (optimizationIndex==0 || optimizationIndex==1)
   //       {
   //          net=new GirvanNetwork();
   //       }
   //       else if (optimizationIndex==2)
   //       {
   //          lNet=new localOpNet(nodeCount);
   //          lNet.setPanelSize(getWidth(), getHeight());
```

```
    //            lNet.setup();
    //        }
    //        else if(optimizationIndex ==3)
    //        {
    //            cNet=new circleNet(nodeCount);
    //            cNet.setPanelSize(getWidth(), getHeight());
    //            cNet.setup();
    //        }
    //startAnimation();
}

public void setSize(int width, int height)
{
    super.setSize(width, height);

}

public void resizeNetwork()
{
    net.setPanelSize(getWidth(), getHeight());
    //        if (optimizationIndex==0 || optimizationIndex==1)
    //            net.setPanelSize(getWidth(), getHeight());
    //        else if (optimizationIndex==2)
    //            lNet.setPanelSize(getWidth(), getHeight());
    //        else if (optimizationIndex==3)
    //            cNet.setPanelSize(getWidth(), getHeight());
}

public void startAnimation()
{
    if (animator == null || !animator.isAlive()) {
        animator = new Thread(this);
        animator.setName("Animations Thread");
        animator.start();
        // System.out.println(getWidth() +" "+getHeight());
    }

}

private long beforeTime;
public void run()
/* Repeatedly: update, render, sleep so loop takes close
to period nsecs. Sleep inaccuracies are handled.*/
{

    long afterTime, timeDiff, sleepTime;
    long overSleepTime = 0L;
    int noDelays = 0;
    long excess = 0L;

    beforeTime = System.nanoTime();

    movementFramesCount = 0;

    while (true) {
        // System.out.println("in while:"+getWidth()+" "+getHeight());
        //if (optimizationIndex==0 || optimizationIndex==1)
        //     net.getNodeSteps();

        update();
        render();
        paintPanel();

        afterTime = System.nanoTime();
        timeDiff = afterTime - beforeTime;
        sleepTime = (period - timeDiff) - overSleepTime;  // time left in this loop

        if (sleepTime > 0) {    // some time left in this cycle
            try {
                Thread.sleep(sleepTime/1000000L);   // nano -> ms
            } catch (InterruptedException ex) { }

            overSleepTime = (System.nanoTime() - afterTime) - sleepTime;
        } else {    // sleepTime <= 0; frame took longer than the period
            excess -= sleepTime;    // store excess time value
            overSleepTime = 0L;

            if (++noDelays >= NO_DELAYS_PER_YIELD) {
                Thread.yield();     // give another thread a chance to run
                noDelays = 0;
            }
        }

        beforeTime = System.nanoTime();

        /* If frame animation is taking too long, update the game state
        without rendering it, to get the updates/sec nearer to
        the required FPS. */
        int skips = 0;
        while (excess > period && skips < MAX_FRAME_SKIPS) {
            excess -= period;
            update();   // update state but don't render
            skips++;
        }
    }
}

private void update()
{
    if(running )
    {
        net.move();

    }
}

private void render()
// draw the current frame to an image buffer
{
    //System.out.println("(run)"+getHeight() + " "+getWidth());

    if (dbImage == null){//System.out.println(getWidth()+"----"+getHeight());
```

```
            dbImage = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE_INT_ARGB);
            if (dbImage == null) {
                System.out.println("dbImage is null");
                return;
            }
            else
                dbg2 = dbImage.createGraphics();
        }

        // use anti-aliasing when possible
        dbg2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        // anti-alias text too
        dbg2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
RenderingHints.VALUE_TEXT_ANTIALIAS_ON);

        // clear the background
        dbg2.setPaint(Color.BLACK);
        dbg2.fillRect(0, 0, 2*getWidth(), getHeight());
        //  System.out.println(getWidth()+"---"+getHeight());
        /** draw elements -----------------------------
         * Nodes and Lines
         */

        if (running)
        net.draw(dbg2);

        // --------------------------------------------
    }

    private void paintPanel()
    {
        Graphics g;
        try {
            g = this.getGraphics();
            if (g != null && dbImage != null)
                g.drawImage(dbImage, 0, 0, null);
            Toolkit.getDefaultToolkit().sync();
            g.dispose();
        } catch (Exception e) {
            System.out.println("Graphics context error: " + e);
            System.exit(0);
        }
    }

    public void startLocalOp()
    {
        running=true;
    }

}
```