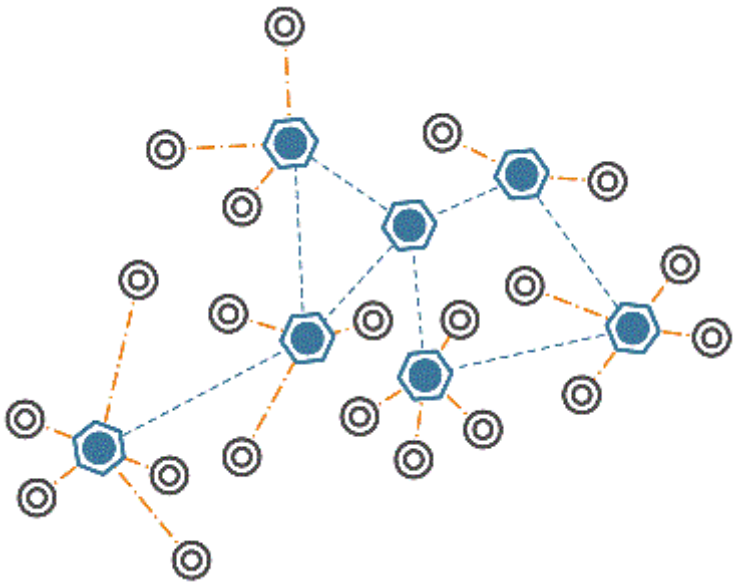
CSE 7350 - Algorithm Engineering

Fall 2017

Name - Azeem Merchant

SMU ID - 47513418

Wireless Sensor Networks



Contents

INTRODUCTION

The purpose of this project is to create the design and structure of a wireless sensor network using the concept of Random Geometric Graph (RGG). We then apply algorithms to find the optimal wireless sensor network structure for a given RGG that is generated. A few of the important terms and concepts for this project are described below.

Wireless Sensor Network [1]

Wireless Sensor Network (WSN), sometimes called wireless sensor and actuator networks (WSAN), are spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to other locations. WSN is built of several nodes where each is connected to one or many other.

WSNs are used in environmental tracking, military, health, transport, automated building, habitat monitoring. This project model's ad-hoc networks by creating random geometric graphs (RGGs) of various topologies, degree, number of vertices and connecting the vertices if they are in a range R. In the context of Wireless sensor networks R is the broadcast range of sensor (node) and is the maximum range in which a sensor can communicate with other sensors in the network.

Determining the connections between the vertices in the RGG is not a trivial task and can be a very expensive process if not done in efficient manner. This project makes use of various methods for decreasing the time and number of comparisons required to determine the adjacency list of all the vertices in the overall RGG. We use the smallest last ordering algorithm and a graph coloring algorithm to identify high quality sensors for backbones in the graph. A backbone is a bipartite subgraph of the RGG, or sensor network that covers all vertices that make up the entire network. The backbones are bipartite as this ensures that no two connected vertices share the same color which practically translates to sensor broadcast frequency.

Random Geometric Graph [2]

A random geometric graph is an undirected graph constructed by randomly placing N nodes in some metric space (in this project we make use of square and disk shaped 2D surfaces and a spherical 3D surface) and then connecting the nodes by a link only if the distance between them is less than a certain range.

Smallest Last Ordering

The smallest last ordering algorithm helps pre-process the node order before we implement the graph coloring algorithm. We keep the nodes with lesser number of neighbors towards the end of the list which helps is get a smaller color set. The vertices of any graph may always be ordered in such a way that the coloring algorithm produces an optimal coloring. For, given any optimal coloring in which the smallest color set is maximal, the second color set is maximal with respect to the first color set, etc., and one may order the vertices by their colors. Then when one uses a greedy algorithm with this order, the resulting coloring is automatically optimal.

Graph coloring [4]

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring. Similarly, an edge coloring assigns a color to each edge so that no two adjacent edges share the same color, and a face coloring of a planar graph assigns a color to each face or region so that no two faces that share a boundary have the same color.

Vertex coloring is the starting point of the subject, and other coloring problems can be transformed into a vertex version. For example, an edge coloring of a graph is just a vertex coloring of its line graph, and a face coloring of a plane graph is just a vertex coloring of its dual. However, non-vertex coloring problems are often stated and

studied as is. That is partly for perspective, and partly because some problems are best studied in non-vertex form, as for instance is edge coloring

Bipartite Graphs [5]

A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V. Vertex sets U and V are usually called the parts of the graph. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles.

The two sets U and V may be thought of as a coloring of the graph with two colors: if one colors all nodes in U blue, and all nodes in V green, each edge has endpoints of differing colors, as is required in the graph coloring problem. In contrast, such a coloring is impossible in the case of a non-bipartite graph, such as a triangle: after one node is colored blue and another green, the third vertex of the triangle is connected to vertices of both colors, preventing it from being assigned either color.

PROGRAMMING ENVIROMENT

The details for the programming environment, language, software and hardware are listed below.

Programming Environment
Netbeans IDE 8.2

Programming Language
Java 8
Java classes used:
Canvas (used for plotting)
File (used for data output, input and storage)

Hardware used
Laptop computer: Acer E15-573G
Processor: Intel i7 - 5500U 2.4GHz and 3.0GHz with TurboBoost
Memory: 12GB DDR3 RAM
Disk: 250GB SSD

Software used
Operating System: Windows 10 Home

Reasoning behind choosing Java
Java was used as a programming language for this project as Java is a universal programming language and is present in almost any industry and device that we can think of.

Java is a platform independent language as it runs on a Java Virtual Machine and not directly on the operating system. This means that the code written for this program would work exactly the same on any other operating system be it Mac OS, Windows or Linux. This gives Java an edge over other programming languages that may or may not have cross compatibility with almost all kinds of operating systems.

Java is also one of the most heavily used object oriented languages. This means that code can be easily modularized and structured into classes which makes it much easier to structure the entire project. As this project would be one with lots of different functions which produce different outputs with lots of repetitive computations.

Java is also one of the first programming languages that I have learned almost 13 years ago and I am more familiar with Java than other programming languages, making it a good option for me.

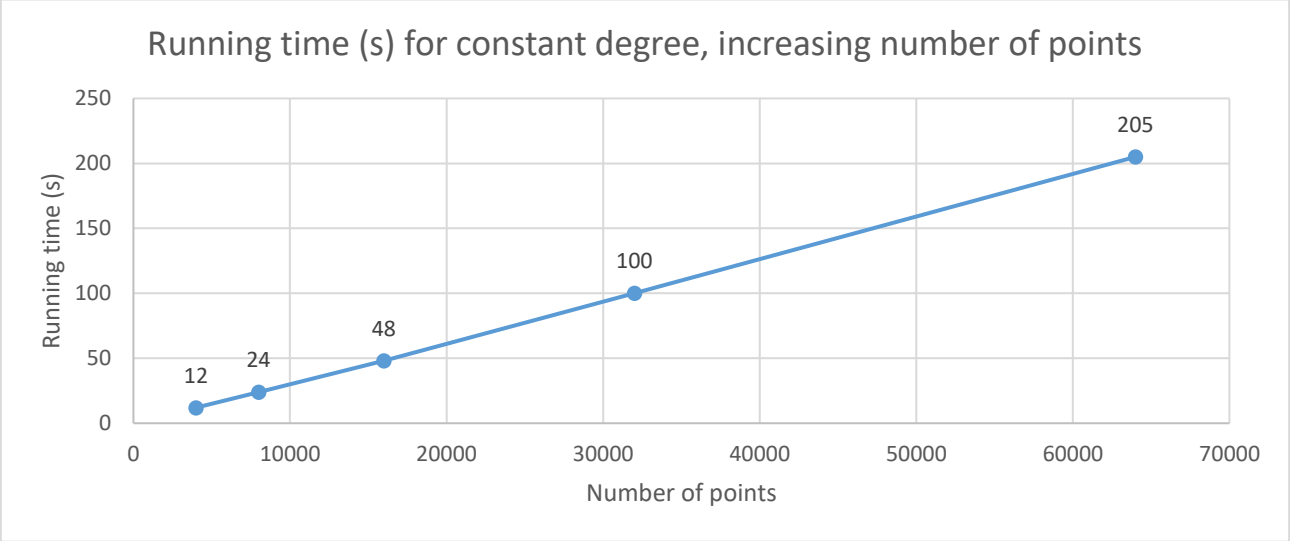
REDUCTION TO PRACTICE

Part I

Running time plots

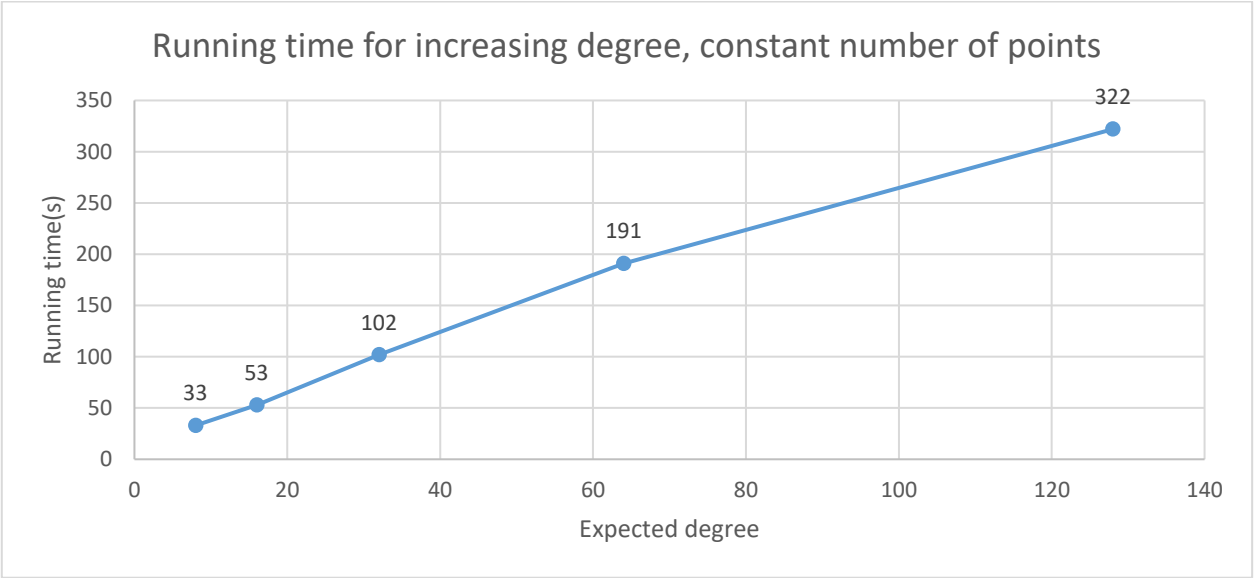
Keeping the input degree fixed at 32 for the square shaped RGG, the results were as follows:

No of Vertices (N)	Running Time (s)
4000	12
8000	24
16000	48
32000	100
64000	205



For the second plot, the no of vertices was kept fixed at 32000 and the degree was changed as follows:

Degree (input)	Running Time (s)
8	33
16	53
32	102
64	191
128	322



Implementation and Expected running time

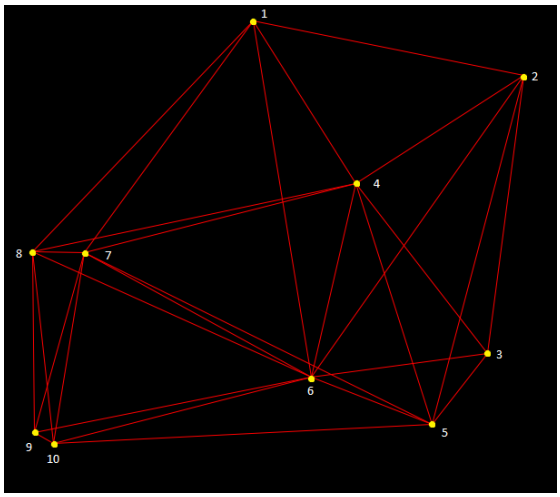
The algorithm used for generating the random geometric graph including the edges is described as follows:

1. Random points are generated using the Java pseudo random function.
2. These random points are stored in the form of a two dimensional array. X coordinate in the first column and Y coordinate in the second.
3. This two dimensional array is then copied and sorted by the X coordinate using an implementation of quick sort to increase efficiency.
4. To create edges, first R (adjacency limit) is calculated using the average expected degree and the formula.
5. We then iterate through the sorted by X coordinate array and check for adjacent vertices up to a value of (X+R) to increase efficiency.
6. The distance between the current point and the point we are checking is the calculated using the distance formula. If this distance is lesser than or equal R, an edge is drawn.
7. Iterate through steps 5 and 6 until we run through all the values in the sorted array.

After analyzing the algorithm, the running time should be $O(n^2)$ as there is use of a singly nested loop. Although the running time of the algorithm is negligible compared to the time taken for the drawing of the edges. This causes the running time complexity to appear as O(n). Or in other words, the running time of the program is directly proportional to the number of edges drawn.

Part II

Walkthrough of a small graph



Adjacency list for the graph shown		
Node	Adjacent nodes	Degree
1	2,4,6,7,8	5
2	1,3,4,5,6	5
3	2,4,5,6	4
4	1,2,3,5,6,7,8	7
5	2,3,4,6,7,10	6
6	1,2,3,4,5,7,8,9,10	9
7	1,4,5,6,8,9,10	7
8	1,4,6,7,9,10	6
9	6,7,8,10	4
10	5,6,7,8,9	5

For the smallest last ordering algorithm, we must:

1. Pick the node with the smallest degree from the adjacency list
2. Delete it and push it into the node order stack.
3. Update the degree of all the neighboring nodes after deletion
4. Repeat step 1, 2 and 3 until we have run through all the nodes graph

The deletion of the nodes in reverse order is then used as an input for our coloring algorithm.

1st iteration:
For the first step, we pick node 3 with a degree of 4 for deletion.

Node order: [3]

New adjacency list:

Node	Adjacent nodes	Degree
1	2,4,6,7,8	5
2	1,4,5,6	4
4	1,2,5,6,7,8	6
5	2,4,6,7,10	5
6	1,2,4,5,7,8,9,10	8
7	1,4,5,6,8,9,10	7
8	1,4,6,7,9,10	6
9	6,7,8,10	4
10	5,6,7,8,9	5

2nd iteration:
For the first step, we pick node 2 with a degree of 4 for deletion.

Node order: [3,2]

New adjacency list:

Node	Adjacent nodes	Degree
1	4,6,7,8	4
4	1,5,6,7,8	5
5	4,6,7,10	4
6	1,4,5,7,8,9,10	7
7	1,4,5,6,8,9,10	7
8	1,4,6,7,9,10	6
9	6,7,8,10	4
10	5,6,7,8,9	5

3rd iteration:
For the first step, we pick node 1 with a degree of 4 for deletion.

Node order: [3,2,1]

New adjacency list:

Node	Adjacent nodes	Degree
4	5,6,7,8	4
5	4,6,7,10	4
6	4,5,7,8,9,10	6

7	4,5,6,8,9,10	6
8	4,6,7,9,10	5
9	6,7,8,10	4
10	5,6,7,8,9	5

4th iteration:

For the first step, we pick node 4 with a degree of 4 for deletion

Node order: [3,2,1,4]

New adjacency list:

Node	Adjacent nodes	Degree
5	6,7,10	3
6	5,7,8,9,10	5
7	5,6,8,9,10	5
8	6,7,9,10	4
9	6,7,8,10	4
10	5,6,7,8,9	5

5th iteration:

For the first step, we pick node 5 with a degree of 3 for deletion

Node order: [3,2,1,4,5]

New adjacency list:

Node	Adjacent nodes	Degree
6	7,8,9,10	4
7	6,8,9,10	4
8	6,7,9,10	4
9	6,7,8,10	4
10	6,7,8,9	4

6th iteration:

For the first step, we pick node 6 with a degree of 4 for deletion

Node order: [3,2,1,4,5,6]

New adjacency list:

Node	Adjacent nodes	Degree
7	8,9,10	3
8	7,9,10	3
9	7,8,10	3
10	7,8,9	3

7th iteration:

For the first step, we pick node 7 with a degree of 3 for deletion

Node order: [3,2,1,4,5,6,7]

New adjacency list:

Node	Adjacent nodes	Degree
8	9,10	2
9	8,10	2
10	8,9	2

8th iteration:
For the first step, we pick node 8 with a degree of 2 for deletion

Node order: [3,2,1,4,5,6,7,8]

New adjacency list:

Node	Adjacent nodes	Degree
9	10	1
10	9	1

9th iteration:
For the first step, we pick node 9 with a degree of 1 for deletion

Node order: [3,2,1,4,5,6,7,8,9]

New adjacency list:

Node	Adjacent nodes	Degree
10	-	0

10th iteration:
We pick node 10 as that is the only node left

Node order: [3,2,1,4,5,6,7,8,9,10]

Coloring Algorithm

For the coloring algorithm, we take the node order stack in reverse

So the smallest last order is: [10,9,8,7,6,5,4,1,2,3]

And then we carry out the following steps:

1. Check the neighbors it is connected to and assign the least possible color
2. If none of the current colors are available, we create a new color
3. Repeat steps 1 and 2 until we have visited all the nodes of the graph in smallest last order

After running the algorithm on our smallest last order, we get the following colors assigned for the following nodes:

Node	Adjacent nodes	Color assigned
1	2,4,6,7,8	2
2	1,3,4,5,6	3
3	2,4,5,6	4
4	1,2,3,5,6,7,8	1
5	2,3,4,6,7,10	2
6	1,2,3,4,5,7,8,9,10	5
7	1,4,5,6,8,9,10	4
8	1,4,6,7,9,10	3
9	6,7,8,10	2

10	5,6,7,8,9	1

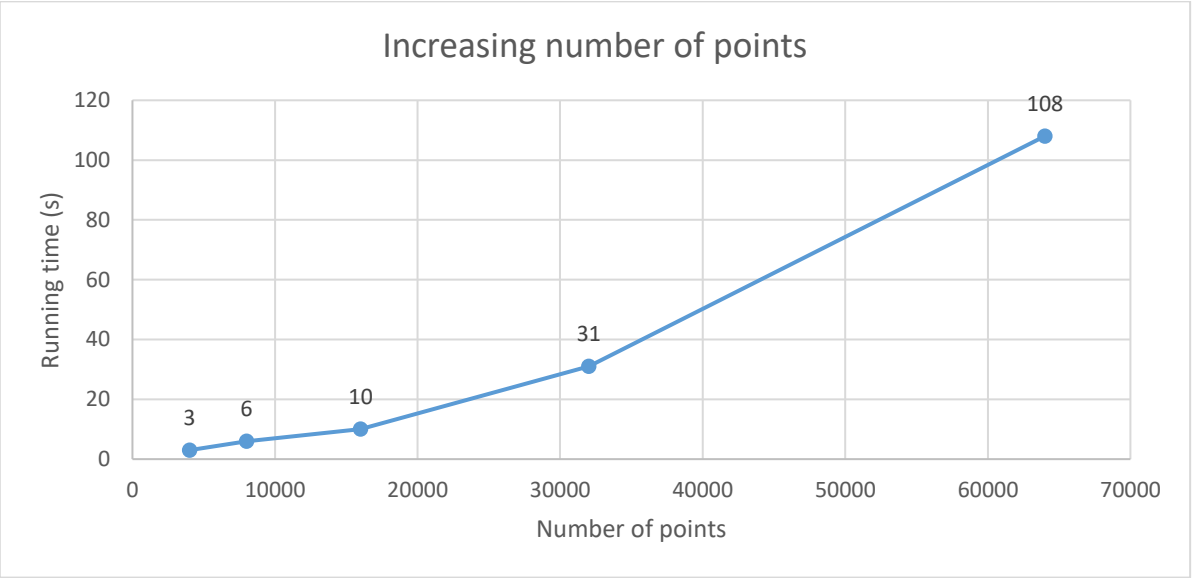
Color class distribution:

Color Number	Nodes	Frequency
1	4,10	2
2	1,5,9	3
3	2,8	2
4	3,7	2
5	6	1

Running time plots

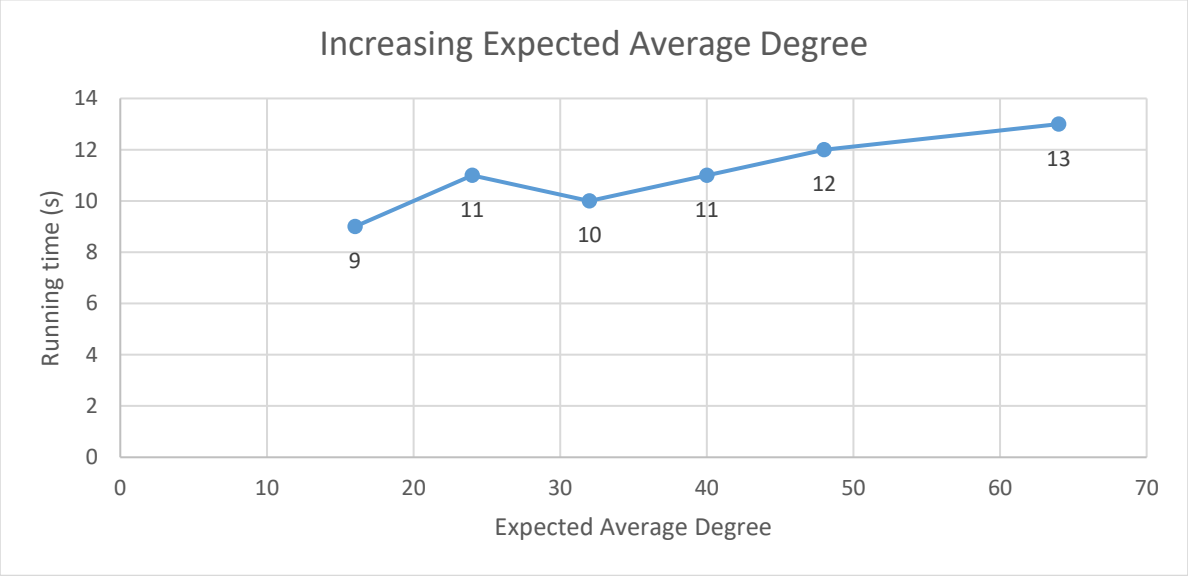
Shown below is a running time plot using number of points for Part II on the x-axis and the running time on the y-axis. The expected average degree was taken as 32.

No of Points	Running time (s)
4000	3
8000	6
16000	10
32000	31
64000	108



Shown below is the running time plot graph using the expected average degree as the x-axis and the running time in seconds as the y-axis. The number of points was taken as 16000.

Expected Average degree	Running time (s)
16	9
24	11
32	10
40	11
48	12
64	13



Part III

Building upon the same colored graph from the example above, below is a walkthrough for Part III.

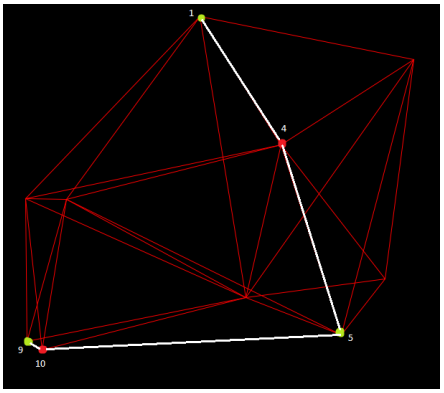
We now select the four largest color classes to create our bipartite backbones. We assign colors to each of the color classes as red, green, blue and yellow.

The four largest color classes are as follows:

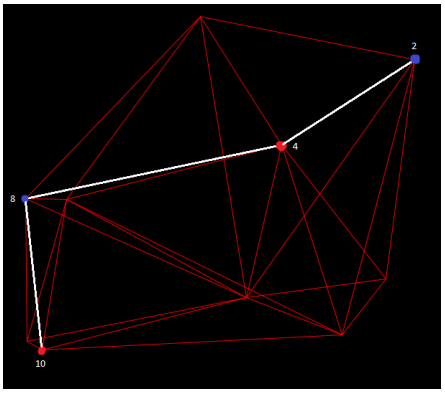
Color Number	Nodes	Frequency
1 - Red	4,10	2
2 - Green	1,5,9	3
3 - Blue	2,8	2
4 - Yellow	3,7	2

Since we have 4 color classes, we pick two at a time to create the bipartite backbones, this gives us 6 different combinations.

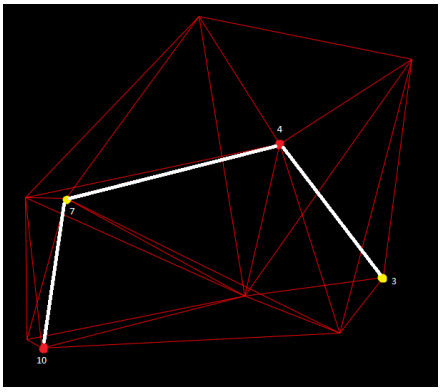
#	Color combination	Collective Nodes	No of Nodes	No of Edges	Coverage
1	Red (1) and Green (2)	1,4,5,9,10	5	4	100%
2	Red (1) and Blue (3)	2,4,8,10	4	4	100%
3	Red (1) and Yellow (4)	3,4,7,10	4	4	100%
4	Green (2) and Blue (3)	1,2,5,8,9	5	4	100%
5	Green (2) and Yellow (4)	1,3,5,7,9	5	3	100%
6	Blue (3) and Yellow (4)	2,3,7,8	4	2	100%



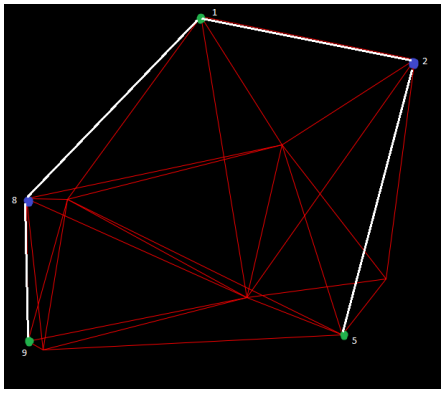
Backbone 1 - Red and Green



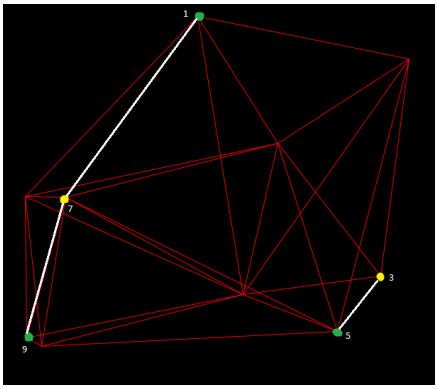
Backbone 3 - Red and Blue



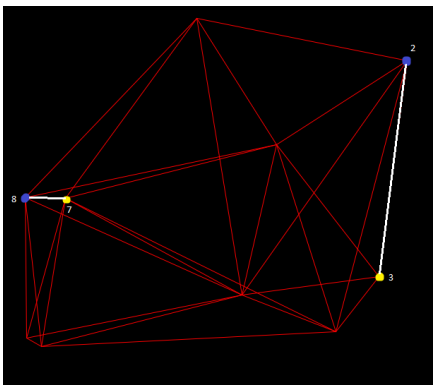
Backbone 3 - Red and Yellow



Backbone 4 - Green and Blue



Backbone 5 - Green and Yellow



Backbone 6 - Blue and Yellow

In this example, all the six produced backbones have 100% coverage of nodes. We can reach any of the 10 nodes from the nodes on the backbones in all graphs.

We must now select the two best bipartite backbones out of the six.

Since the coverage for all is the same, we select the backbones which have the most number of edges and nodes in them. We thus conclude that backbones 1 and 4 are the optimal bipartite backbones for this example graph.

SUMMARY

Benchmark ID	1	2	3	4	5	6	7	8	9	10
N	4000	8000	16000	32000	64000	64000	64000	128000	8000	8000
R	32	64	64	64	32	64	128	64	64	128
M	61704	246217	497783	1002991	1011479	2015980	4011729	4053613	1199005	2304623
Min Degree	8	12	16	21	7	17	35	16	127	276
Avg Degree	30.85	61.55	62.22	62.69	31.61	63	125.37	63.34	299.75	576.16
Max Degree	52	96	91	95	58	96	171	98	374	713
Running time (Part 1)	12	42	92	196	205	355	657	762	228	408
Number of colors	25	42	41	45	27	47	75	45	154	296
Max Color class size	279	302	625	1245	4528	2479	1327	4995	70	40
Running time (Part 2)	3	6	13	43	108	195	241	678	9	14
Edges Backbone 1	600	703	1498	3018	9818	5992	3492	12198	180	112
Vertices Backbone 1	557	603	1243	2487	9016	4957	2649	9973	138	78
Backbone 1 coverage (%)	100	100	100	99.99	99.99	100	100	100	100	100
Edges Backbone 2	578	698	1478	3000	9772	5989	3486	12194	178	104
Vertices Backbone 2	552	598	1273	2467	8959	4957	2650	9963	138	77
Backbone 2 coverage (%)	100	99.96	100	99.99	99.99	100	100	100	100	100
Running time (Part 3)	2	2	3	4	13	9	7	21	2	3

Shown below is the prediction table which estimates the running time for the same algorithm if the number of points was increased to 256,000 and 1,000,000,000 nodes with a degree of 64.

	No of Nodes	Degree	Running time (s) for Part I	Running time (s) for Part II	Running time (s) for Part III
1	256,000	64	1524	2712	42
2	1,000,000,000	64	6242304	11108352	172032

REFERENCES

1. https://en.wikipedia.org/wiki/Wireless_sensor_network
2. https://en.wikipedia.org/wiki/Random_geometric_graph
3. https://www.eurandom.tue.nl/events/workshops/2011/YEPVIII/Presentations/YepVIII.pdf
4. https://en.wikipedia.org/wiki/Graph_coloring
5. https://en.wikipedia.org/wiki/Greedy_coloring
6. https://en.wikipedia.org/wiki/Bipartite_graph
7. Java Canvas drawing tool. https://books.trinket.io/thinkjava/appendix-b.html
8. Java Files. https://docs.oracle.com/javase/7/docs/api/java/io/File.html
9. Bipartite Grid Partitioning of a Random Geometric Graph (Zizhen Chen and David Matula)
10. Employing Dominating Set Partitions as Backbones in Wireless Sensor Networks (Dhia Mahjoub and David Matula)
11. Random Geometric Graphs as Model of Wireless Sensor Networks (Hichem Kenniche and Vlady Ravelomananana)
12. Smallest Last Ordering and Clustering and Graph Coloring Algorithms (David Matula and Leland Beck)

APPENDIX A: SOURCE CODE (Part 1) IN JAVA

```
import java.util.*;
import java.io.*;
import java.nio.file.*;
import java.awt.*;
import java.awt.geom.Line2D;
import javax.swing.*;
public class RGG extends Canvas{
   public String m="d"; //Change to s for square, d for disk
   public int s=600; //Size of square side in pixels
   public int n=8000; //No of nodes
   public double exp=128;
   public double r=Math.sqrt(exp/(n*Math.PI))*s; //Adjacency in unit * pixels
   public double points[][]=new double[n][2];
   public double pointsx[][]=new double[n][2];
   public double pointsy[][]=new double[n][2];
   public int degrees[]=new int[5000];
   public long edges=0;
   public String adjlist="";
   public String ptslist="";
   boolean draw=false; //true for graph, false no graph
   long time = System.currentTimeMillis();
   public void quickSort(double[][] arr, int low, int high)
   {
                  if (arr == null || arr.length == 0) return;
                  if (low >= high) return;
                  int middle = low + (high - low) / 2;
                  double pivot = arr[middle][0];
                  int i = low, j = high;
                  while (i <= j)
          {
                        while (arr[i][0] < pivot) i++;
```

```
                while (arr[j][0] > pivot) j--;
                if (i <= j)
            {
                        double temp = arr[i][0];
                        arr[i][0] = arr[j][0];
                        arr[j][0] = temp;
              temp=arr[i][1];
              arr[i][1]=arr[j][1];
              arr[j][1]=temp;
                        i++;
                        j--;
                }
            }
            if (low < j) quickSort(arr, low, j);
            if (high > i) quickSort(arr, i, high);
}
public double Dist(double x1,double y1, double x2, double y2)
{
   return ((x1-x2)*(x1-x2))+((y1-y2)*(y1-y2));
}
public void saveToFile(String adjlist, String ptslist) throws IOException
{
   String filename="adjlistoutput_"+n+"_"+(int)exp+"_"+m+".txt";
   Files.write(Paths.get(filename), adjlist.getBytes());
   filename="ptslistoutput_"+n+"_"+(int)exp+"_"+m+".txt";
   Files.write(Paths.get(filename), ptslist.getBytes());
}
public void paint(Graphics g) {
   Graphics2D g2 = (Graphics2D) g;
   super.paint(g);
   g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
   g.setColor(Color.blue);
   int i=0;
```

```
System.out.println("r= "+(r/s));
if(m.equals("d"))
{
   s/=2;
   r*=2;
   while(i<n)
   {
      double xtemp=Math.random()*s*2;
      double ytemp=Math.random()*s*2;
      if(Dist(xtemp,ytemp,s,s) <= s*s)
      {
         points[i][0]=xtemp;
         points[i][1]=ytemp;
         pointsx[i][0]=points[i][0];
         pointsx[i][1]=points[i][1];
         pointsy[i][0]=points[i][0];
         pointsy[i][1]=points[i][1];
         i++;
      }
   }
}
else if(m.equals("s"))
{
   while(i<n)
   {
      points[i][0]=Math.random()*s;
      points[i][1]=Math.random()*s;
      pointsx[i][0]=points[i][0];
      pointsx[i][1]=points[i][1];
      pointsy[i][0]=points[i][0];
      pointsy[i][1]=points[i][1];
      i++;
   }
```

```
}

quickSort(pointsx,0,n-1);

for(i=0; i<n; i++)
{
   //ptslist+=i+" => "+pointsx[i][0]+", "+pointsx[i][1]+"\n";
   System.out.println(i+" => "+pointsx[i][0]+", "+pointsx[i][1]);
}
if(draw)
for(i=0;i<n;i++)
{
   g2.draw(new Line2D.Double(pointsx[i][0],pointsx[i][1],pointsx[i][0],pointsx[i][1]));
}
g.setColor(Color.red);

double x1,x2,y1,y2;
int x1in,x2in,y1in,y2in;

int degree=0;
boolean comma=false;
int jmin=0;

for(i=0; i<n; i++)
{
   //adjlist+=i+" => ";
   System.out.print(i+" => ");
   comma=false;

   for(int j=jmin;j<n;j++)
   {
      if(pointsx[i][0]-pointsx[j][0]>r)
      {
```

```
        jmin=j+1;
        //System.out.println("Continue at i "+i+" and j "+j);
        continue;
      }
      double dist=Dist(pointsx[j][0],pointsx[j][1],pointsx[i][0],pointsx[i][1]);
      if(dist<= r*r && dist > 0)
      {
        if(j>=i)
        {
          if(draw)
          g2.draw(new Line2D.Double(pointsx[j][0],pointsx[j][1],pointsx[i][0],pointsx[i][1]));
          edges++;
          degree++;
        }
        else
        {
          degree++;
        }
        if(comma)
          System.out.print(", "+j);
         // adjlist+=", "+j;
        else
        {
          System.out.print(j);
          //adjlist+=""+j;
          comma=true;
        }

      }
      if(pointsx[j][0]-pointsx[i][0]>r)
        break;
    }
    degrees[degree]++;
```

```
   degree=0;
   System.out.println();
   //adjlist+="\n";
}
for(i=0;i<300;i++)
{
   if(degrees[i]>0)
   System.out.println(i+"="+degrees[i]+",");
}


int mindegree=0,maxdegree=0;
for(i=0;i<5000;i++)
{
   if(degrees[i]>0)
   {
      mindegree=i;
      break;
   }
}
for(i=4999;i>=0;i--)
{
   if(degrees[i]>0)
   {
      maxdegree=i;
      break;
   }
}
System.out.println("Average degree= "+(float)edges*2/n);
System.out.println("Edges="+edges);
System.out.println("Minimum degree="+mindegree);
System.out.println("Maximum degree="+maxdegree);
System.out.println("Done");
```

```
        try{
            saveToFile(adjlist,ptslist);
        }
        catch(IOException e)
        {

        }
        time=(System.currentTimeMillis()-time)/1000;
        System.out.println(time);
    }
    public static void main(String args[])
    {
        RGG canvas = new RGG();
        JFrame frame = new JFrame();
        frame.setSize(1000,1000);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(canvas);
        frame.getContentPane().setBackground(Color.black);
        frame.setVisible(true);
    }
}
```

APPENDIX B: SOURCE CODE (Part 2) IN JAVA

```
import java.io.*;
import java.util.*;
import java.nio.file.*;
import java.awt.*;
import java.awt.geom.Line2D;
import javax.swing.*;
public class RGG2 extends Canvas{
   public static String m="d"; //Change to s for square, d for disk
   public int s=600; //Size of square side in pixels
   public static int n=4000; //No of nodes
   public static double exp=32; //Expected average degree
   public double r=Math.sqrt(exp/(n*Math.PI))*s; //Adjacency in unit * pixels
   public double points[][]=new double[n][2];
   public int degrees[]=new int[5000];
   public long edges=0;
   public String adjlist="";
   public String ptslist="";
   boolean draw=true; //true for graph, false no graph
   long time = System.currentTimeMillis();
   public static void quickSort(int[][] arr, int low, int high)
   {
                    if (arr == null || arr.length == 0) return;
                    if (low >= high) return;
                    int middle = low + (high - low) / 2;
                    int pivot = arr[middle][1];
                    int i = low, j = high;
                    while (i <= j)
           {
                            while (arr[i][1] < pivot) i++;
                            while (arr[j][1] > pivot) j--;
                            if (i <= j)
```

```
            {
                                int temp = arr[i][0];
                                arr[i][0] = arr[j][0];
                                arr[j][0] = temp;
                  temp=arr[i][1];
                  arr[i][1]=arr[j][1];
                  arr[j][1]=temp;
                                i++;
                                j--;
                        }
                  }
                  if (low < j) quickSort(arr, low, j);
                  if (high > i) quickSort(arr, i, high);
}
public static int[][] getAdj() //Getting the adjacency lists from the text file
{
   String filename="adjlistoutput_"+n+"_"+(int)exp+"_"+m+".txt";
   int adj[][]= new int[129000][1000];
   Scanner sc=new Scanner(System.in);
   File fileadj = new File(filename);
   int count=0;
   try
   {
      sc= new Scanner(fileadj);
      while(sc.hasNextLine())
      {
         int i=0;
         String line=sc.nextLine();
         adj[count][0]=count+1;
         for(i=0;i<line.length();i++)
         {
            if(line.charAt(i)=='>')
            {
```

```
                break;
            }
        }
        line=line.substring(i+2);
        String vals[]=new String[500];
        vals=line.split(",");
        for(i=0; i<vals.length; i++)
        {
            adj[count][i+1]=Integer.parseInt(vals[i].trim())+1;
        }
        count++;
      }
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   return adj;
}
public static double[][] getPts() //Getting the points from the text file
{
   double pts[][]=new double[129000][2];
   String filename="ptslistoutput_"+n+"_"+(int)exp+"_"+m+".txt";
   Scanner sc=new Scanner(System.in);
   File filepts = new File(filename);
   int count=0,i=0;
   try
   {
      sc= new Scanner(filepts);
      while(sc.hasNext())
      {
         String line=sc.nextLine();
         for(i=0;i<line.length();i++)
```

```
            {
                if(line.charAt(i)=='>')
                    break;
            }
            line=line.substring(i+2);
            String vals[]=new String[2];
            vals=line.split(",");
            pts[count][0]=Double.parseDouble(vals[0].trim());
            pts[count][1]=Double.parseDouble(vals[1].trim());
            count++;
        }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    return pts;
}
public static int getSmallestDeg(int degreelist[][],int adj[][],int rem) //smallest last ordering function
{
    int smallest=100000;
    int node=-1;
    int found=-1;
    for(int i=0;i<n; i++)
    {
        if(degreelist[i][1]<smallest && degreelist[i][1]>=0)
        {
            node=degreelist[i][0];
            smallest=degreelist[i][1];
            found=i;
        }
    }
    degreelist[found][1]=-1;
```

```
    int current=1;
    while(adj[node][current]>0)
    {
        int x=adj[node][current]-1;
        for(int i=1;i<n;i++)
        {
            if(degreelist[i][0]==(x+1))
            {
                degreelist[i][1]--;
                break;
            }
        }
        for(int i=1;;i++)
        {
            if(adj[x][i]==node)
            {
                //System.out.println("Deleted"+adj[x][i]);
                adj[x][i]=-1;
                break;
            }
            if(adj[x][i]==0)
            {
                break;
            }
        }
        current++;
    }

    return node;
}
public static void main(String args[]) throws IOException
{
```

```
 int adj[][]=new int[129000][1000];
 int adj1[][]=new int[129000][1000];
 int color[]=new int[129000];
 double pts[][]=new double[129000][2];
 int degreelist[][]=new int[129000][2];
 int order[][]=new int[129000][1000];
//  int slast[][]=new int[129000][100];
 int sorder[]=new int[129000];
 int sordercnt=0;
 int colorfreq[]=new int[500];
 int rem=n;
 pts=getPts();
 adj=getAdj();

 for(int i=0;i<n;i++)
 {
    for(int j=0;j<2;j++)
    {
      // System.out.print(pts[i][j]+" ");
    }
  //  System.out.println();
 }

 int i=0,j=0,pos=1; boolean flag=false;

 for(i=0;i<n;i++)
 {
    for(j=0;j<1000;j++)
    {
       if(adj[i][j]==0)
           break;
    }
    degreelist[i][0]=i+1;
```

```
   degreelist[i][1]=j-1;
}

quickSort(degreelist, 0,n);
for(i=0;i<n;i++)
{
   System.out.println(degreelist[i][0]+"\t"+degreelist[i][1]);
}
int maxdeg=degreelist[i-1][1];
System.out.println("Max degree= "+maxdeg);

System.out.println("Adj");
for(i=0;i<=n;i++)
{
   for(j=0;j<1000;j++)
   {
      if(adj[i][j]==0)
      {
         adj[i][j]=-2;
         //System.out.print("Count="+ (j-1));
         break;
      }
      //System.out.print(adj[i][j]+"\t");
   }
   //System.out.println();
}

//System.out.println("Adjacency list copied:");
for(i=0;i<n;i++)
{
  // System.out.print("Index"+i+"\t");
   for(j=0;j<2000;j++)
   {
```

```
        if(adj[i][j]==0) break;
        adj1[i][j]=adj[i][j];
        //System.out.print(adj1[i][j]+"\t");
    }
 //  System.out.println();
}

System.out.println("Degrees listed");
int x=0;
for(;rem>0;rem--)
{
    //System.out.println(getSmallestDeg(degreelist,adj,rem));
    sorder[rem-1]=getSmallestDeg(degreelist,adj,rem);
}
System.out.println("Smallest last order");
for(i=0;i<n;i++)
    System.out.println(sorder[i]);


//Coloring graph
String colors="";
int current=0;
int neighbour=0;
boolean colored;
System.out.println("Colored nodes");
for(i=0;i<n-1;i++) //Iterating through smallest last order
{
    current=sorder[i];
    colored=false;
    if(color[current-1]==0)
    {
        for(j=1;j<=1000;j++)//Iterating colors
        {
```

```
        for(int k=1;k<1000;k++) //Iterating neighbors
        {
           neighbour=adj1[current-1][k];
           if(adj1[current-1][k]==-2)
           {
              color[current-1]=j;
              colorfreq[j]++;
              System.out.println(current+"\t"+j);
              colored=true;
              break;
           }

           if(color[neighbour-1]==j)
           {
              break;
           }
        }
        if(colored)
        {
           break;
        }
     }
  }
}

int topclr1=0, topclr2=0, topclr3=0,topclr4=0,temp;
for(i=1;colorfreq[i]>0;i++)
{
  if(colorfreq[i]>=colorfreq[topclr1])
  {
     topclr4=topclr3;
     topclr3=topclr2;
     topclr2=topclr1;
```

```
            topclr1=i;
        }
        else if(colorfreq[i]>=colorfreq[topclr2])
        {
            topclr4=topclr3;
            topclr3=topclr2;
            topclr2=i;
        }
        else if(colorfreq[i]>=colorfreq[topclr3])
        {
            topclr4=topclr3;
            topclr3=i;
        }
        else if(colorfreq[i]>=colorfreq[topclr4])
        {
            topclr4=i;
        }
        if(colorfreq[i]==0)
            break;
        System.out.println("\t"+i+"\t"+colorfreq[i]);
    }
    for(i=0;i<n;i++)
    {
        if(color[i]==topclr1||color[i]==topclr2||color[i]==topclr3||color[i]==topclr4)
        colors+=Integer.toString(i)+"\t"+Integer.toString(color[i])+"\n";
    }
    //Writing to txt file
    String filename="colorsoutput_"+n+"_"+(int)exp+"_"+m+".txt";
    Files.write(Paths.get(filename), colors.getBytes());

    System.out.println("TOP COLORS \n1. "+topclr1+"\n2. "+topclr2+"\n3. "+topclr3+"\n4. "+topclr4);
  }
}
```

APPENDIX C: SOURCE CODE (Part 3) IN JAVA

```
import java.io.*;
import java.util.*;
import java.nio.file.*;
import java.awt.*;
import java.awt.geom.Line2D;
import javax.swing.*;
public class RGG3 extends Canvas{
   public static String m="s"; //Change to s for square, d for disk
   public int s=600; //Size of square side in pixels
   public static int n=64000; //No of nodes
   public static double exp=64*2; //Expected average degree
   public double r=Math.sqrt(exp/(n*Math.PI))*s; //Adjacency in unit * pixels
   public static double pts[][]=new double[n][2];
   public int degrees[]=new int[5000];
   public long edges=0;
   public String adjlist="";
   public String ptslist="";
   public static int adj[][]=new int[130000][1000];
   public static int visited[][]= new int[130000][2];
   public static int clr1[]=new int[20000];
   public static int clr2[]=new int[20000];
   public static int clr3[]=new int[20000];
   public static int clr4[]=new int[20000];
   public static int clr1cnt=0,clr2cnt=0,clr3cnt=0,clr4cnt=0;
   public static int bestbackbone=0;
   public static int clra=0,clrb=0;

   public static int[][] getAdj() //Getting the adjacency lists from the text file
   {
      String filename="adjlistoutput_"+n+"_"+(int)exp+"_"+m+".txt";
      int adj[][]= new int[130000][1000];
```

```
Scanner sc=new Scanner(System.in);
File fileadj = new File(filename);
int count=0;
try
{
   sc= new Scanner(fileadj);
   while(sc.hasNextLine())
   {
      int i=0;
      String line=sc.nextLine();
      adj[count][0]=count+1;
      for(i=0;i<line.length();i++)
      {
         if(line.charAt(i)=='>')
         {
            break;
         }
      }
      line=line.substring(i+2);
      String vals[]=new String[500];
      vals=line.split(",");
      for(i=0; i<vals.length; i++)
      {
         adj[count][i+1]=Integer.parseInt(vals[i].trim())+1;
      }
      count++;
   }
}
catch(Exception e)
{
   e.printStackTrace();
}
return adj;
```

```
}
public static double[][] getPts() //Getting the points from the text file
{
   double pts[][]=new double[130000][2];
   String filename="ptslistoutput_"+n+"_"+(int)exp+"_"+m+".txt";
   Scanner sc=new Scanner(System.in);
   File filepts = new File(filename);
   int count=0,i=0;
   try
   {
      sc= new Scanner(filepts);
      while(sc.hasNext())
      {
         String line=sc.nextLine();
         for(i=0;i<line.length();i++)
         {
            if(line.charAt(i)=='>')
               break;
         }
         line=line.substring(i+2);
         String vals[]=new String[2];
         vals=line.split(",");
         pts[count][0]=Double.parseDouble(vals[0].trim());
         pts[count][1]=Double.parseDouble(vals[1].trim());
         count++;
      }
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   return pts;
}
```

```
public static int[][] getClr()
{
   String filename="colorsoutput_"+n+"_"+(int)exp+"_"+m+".txt";
   String line;
   int i=0;
   String[] temp=new String[2];
   int[][] clr=new int[130000][2];
   Scanner sc=new Scanner(System.in);
   File fileclr=new File(filename);
   try
   {
      sc= new Scanner(fileclr);
      while(sc.hasNext())
      {

         line=sc.nextLine();
         //System.out.println(line);

         temp=line.split("\t");
         clr[i][0]=Integer.parseInt(temp[0]);
         clr[i][1]=Integer.parseInt(temp[1]);
         // break;
         i++;
      }
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }

   return clr;
}
public static boolean CheckEdge(int a, int b)
```

```
{
   boolean edge=false;
   for(int i=1;i<1000;i++)
   {
      if(adj[a][i]==0)
      {
         //System.out.println("No at\t"+a+"\t"+b);
         return false;
      }
      if((adj[a][i])==b)
      {
         //System.out.println("Yes at\t"+a+"\t"+b);
         edge=true;
         break;
      }
   }
   return edge;
}
public static int CountEdges(int[] a, int b[], int na, int nb)
{
   int ne=0;
   for(int i=0;i<na;i++)
   {
      for(int j=0;j<nb;j++)
      {
         if(CheckEdge(a[i],b[j]))
            ne++;
      }
   }
   return ne;
}
public static int[] copyArray(int[] a,int n)
{
```

```
    int b[]=new int[20000];
    for(int i=0;i<n;i++)
    {
       b[i]=a[i];
    }
    return b;
}
public void paint(Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    super.paint(g);
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

    int pntno;
    //Plotting color 1
    g.setColor(Color.blue);
    for(int i=0;i<clr1cnt;i++)
    {
       pntno=clr1[i];
       g2.draw(new Line2D.Double(pts[pntno][0],pts[pntno][1],pts[pntno][0],pts[pntno][1]));
    }
    //Plotting color 2
    g.setColor(Color.red);
    for(int i=0;i<clr2cnt;i++)
    {
       pntno=clr2[i];
       g2.draw(new Line2D.Double(pts[pntno][0],pts[pntno][1],pts[pntno][0],pts[pntno][1]));
    }
    //Plotting color 3
    g.setColor(Color.yellow);
    for(int i=0;i<clr3cnt;i++)
    {
       pntno=clr3[i];
```

```
    g2.draw(new Line2D.Double(pts[pntno][0],pts[pntno][1],pts[pntno][0],pts[pntno][1]));
}
//Plotting color 4
g.setColor(Color.green);
for(int i=0;i<clr4cnt;i++)
{
    pntno=clr4[i];
    g2.draw(new Line2D.Double(pts[pntno][0],pts[pntno][1],pts[pntno][0],pts[pntno][1]));
}

//Drawing selected bipartite backbone
g.setColor(Color.white);
int clr1arr[]=new int[20000];
int clr2arr[]=new int[20000];
int clr1arrcnt=0,clr2arrcnt=0;
if(clra==1)
{
    clr1arr=copyArray(clr1,clr1cnt);
    clr1arrcnt=clr1cnt;
}
else if(clra==2)
{
    clr1arr=copyArray(clr2,clr2cnt);
    clr1arrcnt=clr2cnt;
}
else if(clra==3)
{
    clr1arr=copyArray(clr3,clr3cnt);
    clr1arrcnt=clr3cnt;
}
if(clrb==2)
{
    clr2arr=copyArray(clr2,clr2cnt);
```

```
        clr2arrcnt=clr2cnt;
    }
    else if(clrb==3)
    {
        clr2arr=copyArray(clr3,clr3cnt);
        clr2arrcnt=clr3cnt;
    }
    else if(clrb==4)
    {
        clr2arr=copyArray(clr4,clr4cnt);
        clr2arrcnt=clr4cnt;
    }
    for(int i=0;i<clr1arrcnt;i++)
    {
        for(int j=0;j<clr2arrcnt;j++)
        {
            int pt1=clr1arr[i];
            int pt2=clr2arr[j];
            if(CheckEdge(pt1,pt2))
            {
              g2.draw(new Line2D.Double(pts[pt1][0],pts[pt1][1],pts[pt2][0],pts[pt2][1]));
            }
        }
    }
}
public static double coverage(int a[], int b[], int cnta, int cntb)
{
    double cov=0;
    for(int i=1;i<=n;i++)
    {
        visited[i][0]=i;
        visited[i][1]=0;
    }
```

```
    for(int i=0;i<cnta;i++)
    {
        int current=a[i];
        visited[current][1]=1;
        for(int j=1;adj[current][j]>0;j++)
        {
            visited[adj[current][j]][1]=1;
        }
    }
    for(int i=0;i<cntb;i++)
    {
        int current=b[i];
        visited[current][1]=1;
        for(int j=1;adj[current][j]>0;j++)
        {
            visited[adj[current][j]][1]=1;
        }
    }
    for(int i=0;i<=n;i++)
    {
        if(visited[i][1]==1)
            cov++;
    }
    cov=cov/n*100.0;
    return cov;
}
public static void main(String args[]) throws IOException
{
    //double pts[][]=new double[130000][2];
    int clr[][]=new int[130000][2];
    int set[]=new int[4];
    int i=0;
    int cfound=0;
```

```
 boolean found=false;
 int currcolor;
 pts=getPts();
 adj=getAdj();
 clr=getClr();

// System.out.println("colors");
 for(i=0;i<n;i++)
 {
    found=false;
    currcolor=clr[i][1];
    if(cfound<4)
    for(int j=0;j<=cfound;j++)
    {
       if(currcolor==set[j])
       {
          found=false;
          break;
       }
       if(!found && j==cfound)
       {
          set[cfound]=clr[i][1];
          cfound++;
          break;
       }
    }
    if(clr[i][0]==0&&clr[i][1]==0)
       break;
   // System.out.println(clr[i][0]+"\t"+clr[i][1]);
 }

 for(i=0;i<n;i++)
 {
```

```
        if(clr[i][0]==0&&clr[i][1]==0)
            break;
        if(clr[i][1]==set[0])
        {
            clr1[clr1cnt]=clr[i][0];
            clr1cnt++;
        }
        if(clr[i][1]==set[1])
        {
            clr2[clr2cnt]=clr[i][0];
            clr2cnt++;
        }
        if(clr[i][1]==set[2])
        {
            clr3[clr3cnt]=clr[i][0];
            clr3cnt++;
        }
        if(clr[i][1]==set[3])
        {
            clr4[clr4cnt]=clr[i][0];
            clr4cnt++;
        }
    }
    System.out.println("Cfound\tColor 1: "+clr1cnt+"\tColor 2: "+clr2cnt+"\tColor 3: "+clr3cnt+"\tColor 4: "+clr4cnt);

    int nedges1=CountEdges(clr1,clr2,clr1cnt,clr2cnt);
    int nedges2=CountEdges(clr1,clr3,clr1cnt,clr3cnt);
    int nedges3=CountEdges(clr1,clr4,clr1cnt,clr4cnt);
    int nedges4=CountEdges(clr2,clr3,clr2cnt,clr3cnt);
    int nedges5=CountEdges(clr4,clr4,clr2cnt,clr4cnt);
    int nedges6=CountEdges(clr3,clr4,clr3cnt,clr4cnt);
```

```
if(nedges1>=bestbackbone)
{
   bestbackbone=nedges1;
   clra=1;clrb=2;
}
if(nedges2>=bestbackbone)
{
   bestbackbone=nedges2;
   clra=1;clrb=3;
}
if(nedges3>=bestbackbone)
{
   bestbackbone=nedges3;
   clra=1;clrb=4;
}
if(nedges4>=bestbackbone)
{
   bestbackbone=nedges4;
   clra=2;clrb=3;
}
if(nedges5>=bestbackbone)
{
   bestbackbone=nedges5;
   clra=2;clrb=4;
}
if(nedges6>=bestbackbone)
{
   bestbackbone=nedges6;
   clra=3;clrb=4;
}

System.out.println("Backbone 1 coverage\t"+coverage(clr1,clr2,clr1cnt,clr2cnt));
System.out.println("Backbone 2 coverage\t"+coverage(clr1,clr3,clr1cnt,clr3cnt));
```

```
        System.out.println("Backbone 3 coverage\t"+coverage(clr1,clr4,clr1cnt,clr4cnt));
        System.out.println("Backbone 4 coverage\t"+coverage(clr2,clr3,clr2cnt,clr3cnt));
        System.out.println("Backbone 5 coverage\t"+coverage(clr2,clr4,clr2cnt,clr4cnt));
        System.out.println("Backbone 6 coverage\t"+coverage(clr3,clr4,clr3cnt,clr4cnt));
        System.out.println("Backbone 1: Color 1 and 2: "+nedges1+" / "+(clr1cnt+clr2cnt));
        System.out.println("Backbone 2: Color 1 and 3: "+nedges2+" / "+(clr1cnt+clr3cnt));
        System.out.println("Backbone 3: Color 1 and 4: "+nedges3+" / "+(clr1cnt+clr4cnt));
        System.out.println("Backbone 4: Color 2 and 3: "+nedges4+" / "+(clr2cnt+clr3cnt));
        System.out.println("Backbone 5: Color 2 and 4: "+nedges5+" / "+(clr2cnt+clr4cnt));
        System.out.println("Backbone 6: Color 3 and 4: "+nedges6+" / "+(clr3cnt+clr4cnt));

        RGG3 canvas = new RGG3();
        JFrame frame = new JFrame();
        frame.setSize(1000,1000);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(canvas);
        frame.getContentPane().setBackground(Color.black);
        frame.setVisible(true);
    }
}
```