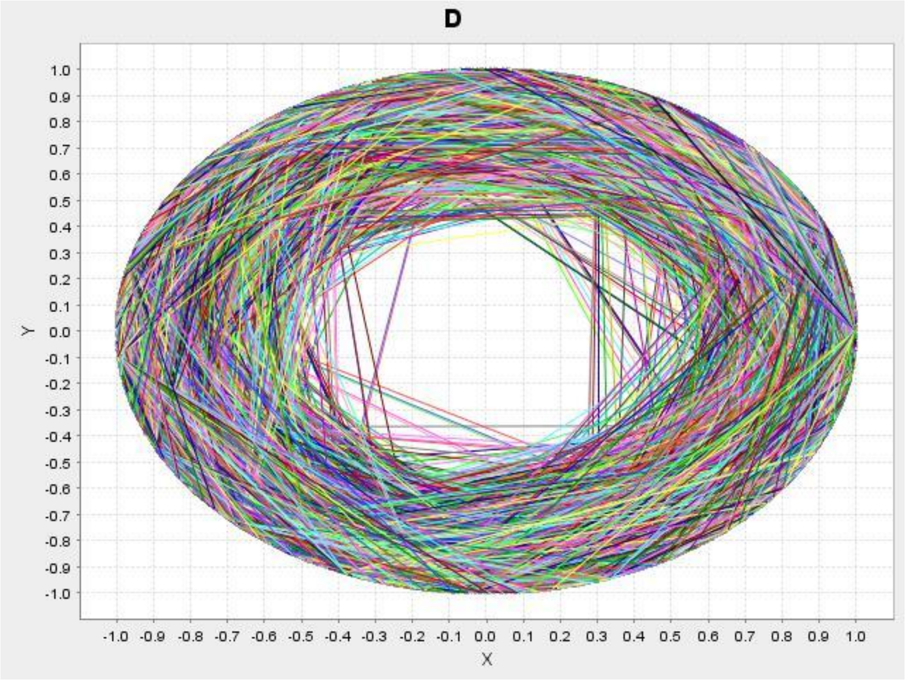
Homework-1

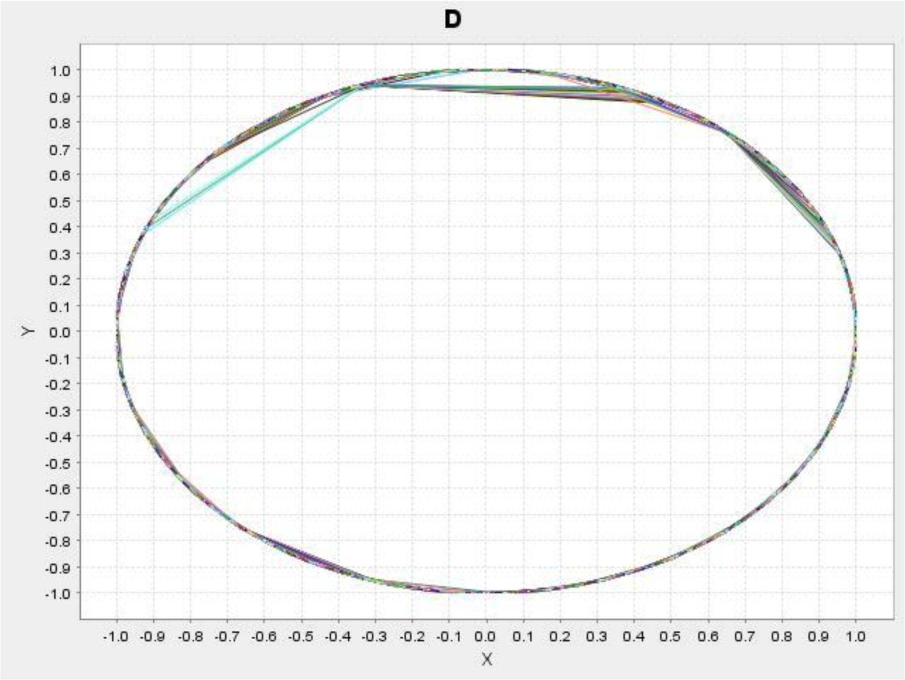
1)Node graphs for each of the three topologies:

a) Dynamic Ring:

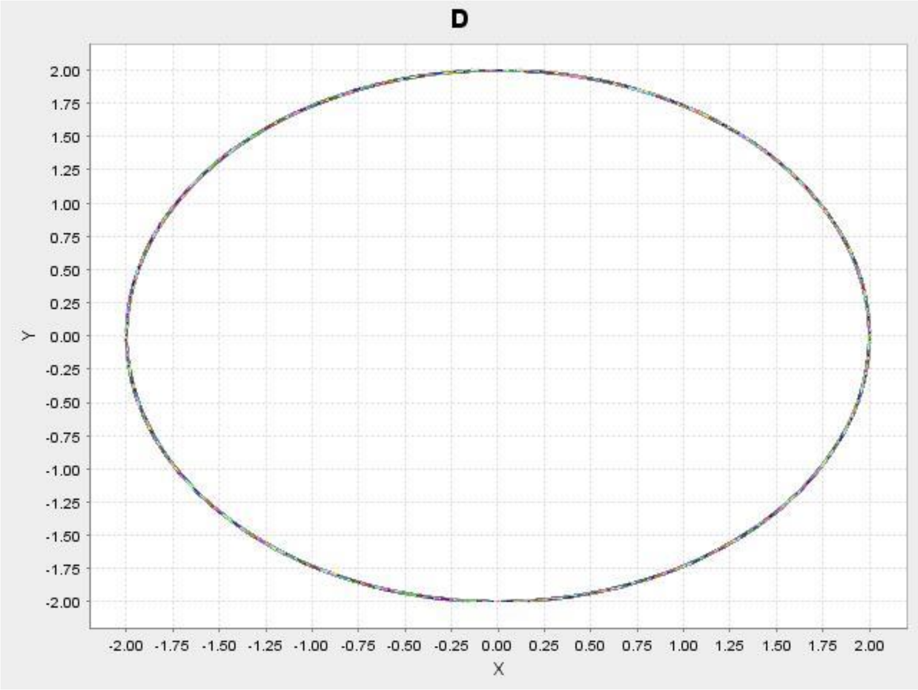
i. Cycle 1



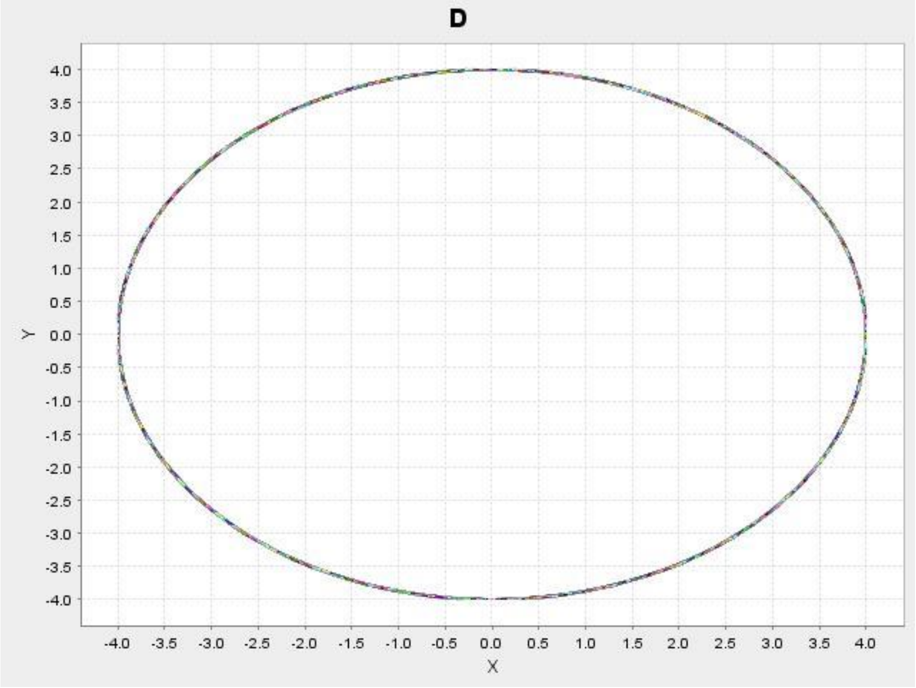
ii. Cycle 5



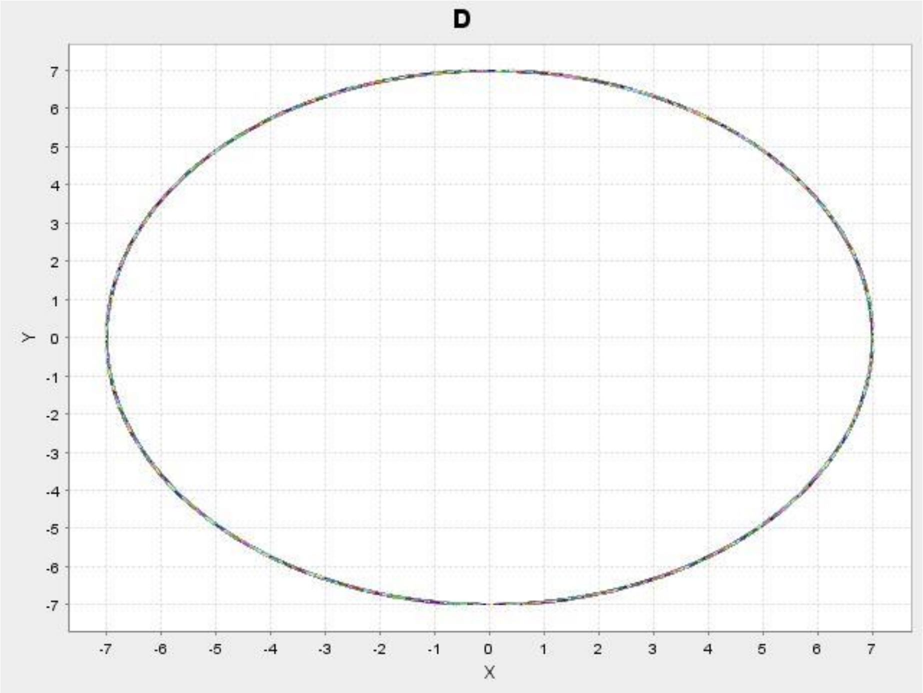
iii. Cycle 10



iv. Cycle 15

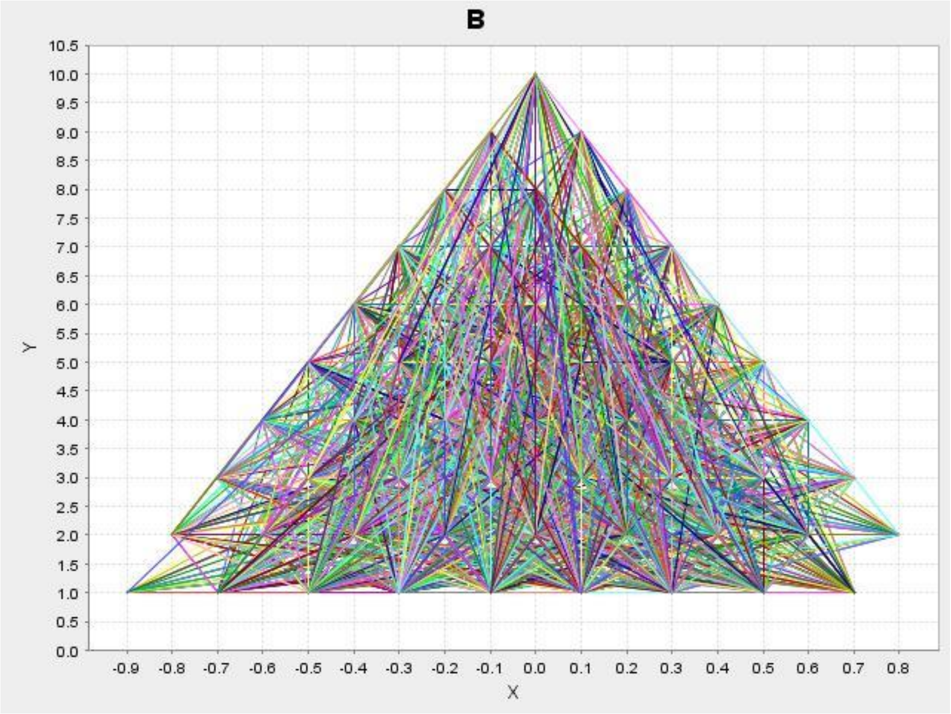


v. Cycle 50

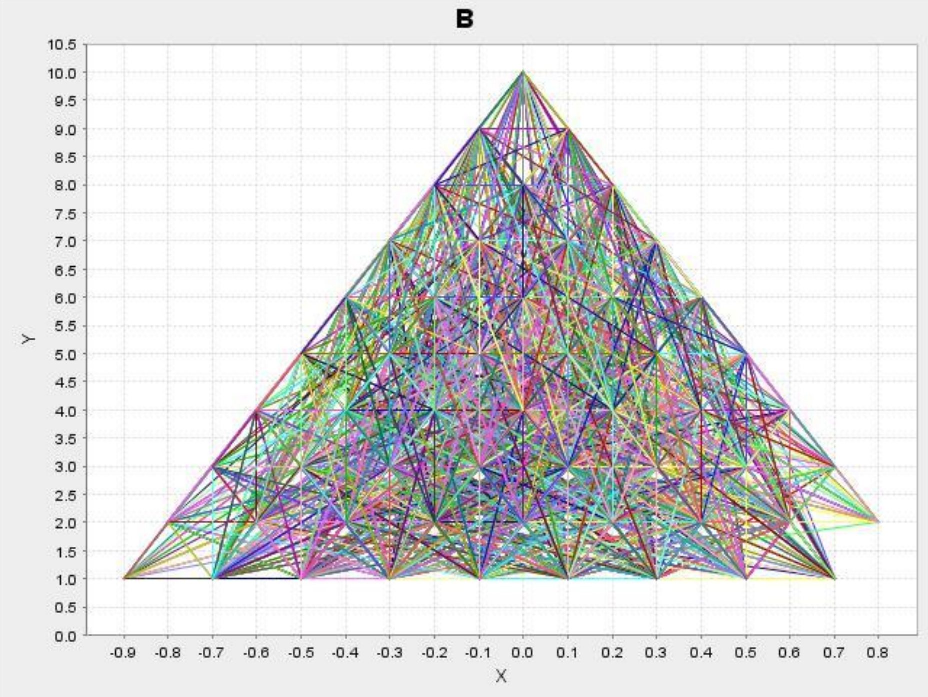


b) Binary Tree:

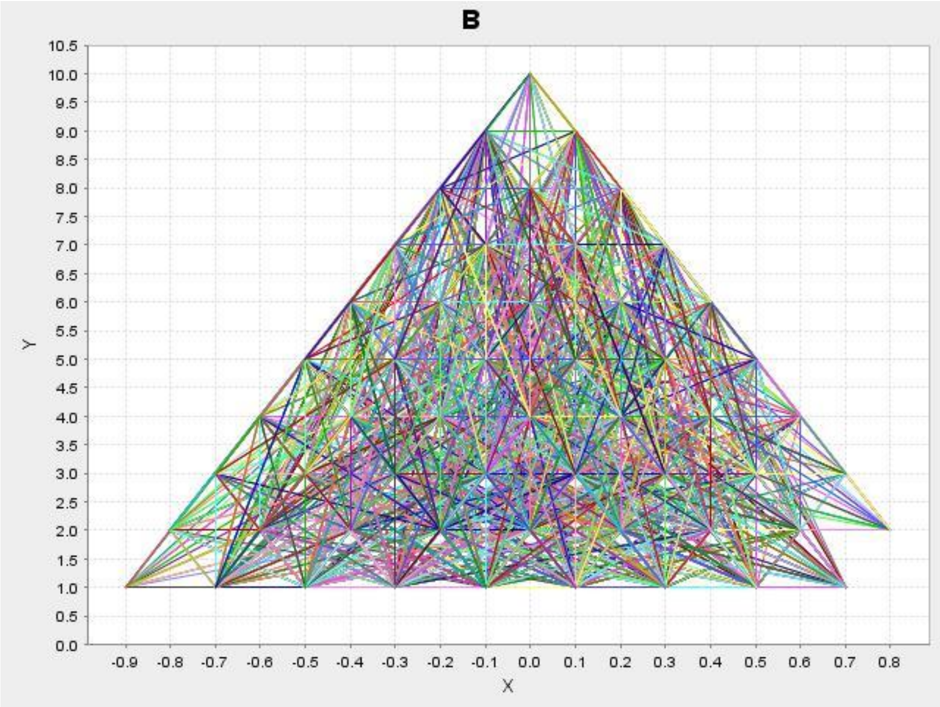
i. Cycle 1



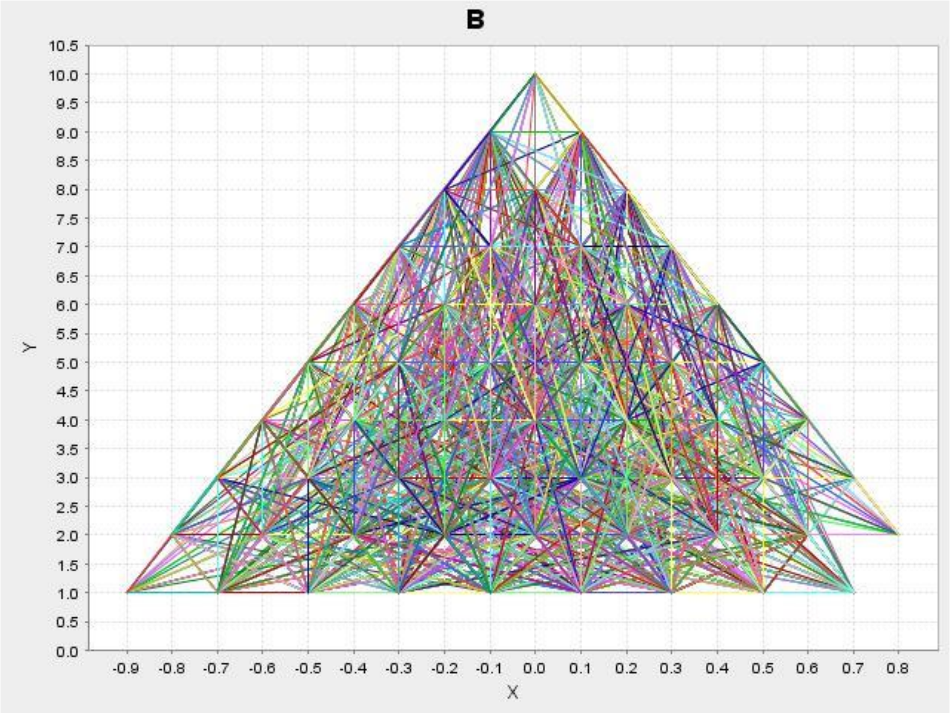
ii. Cycle 5



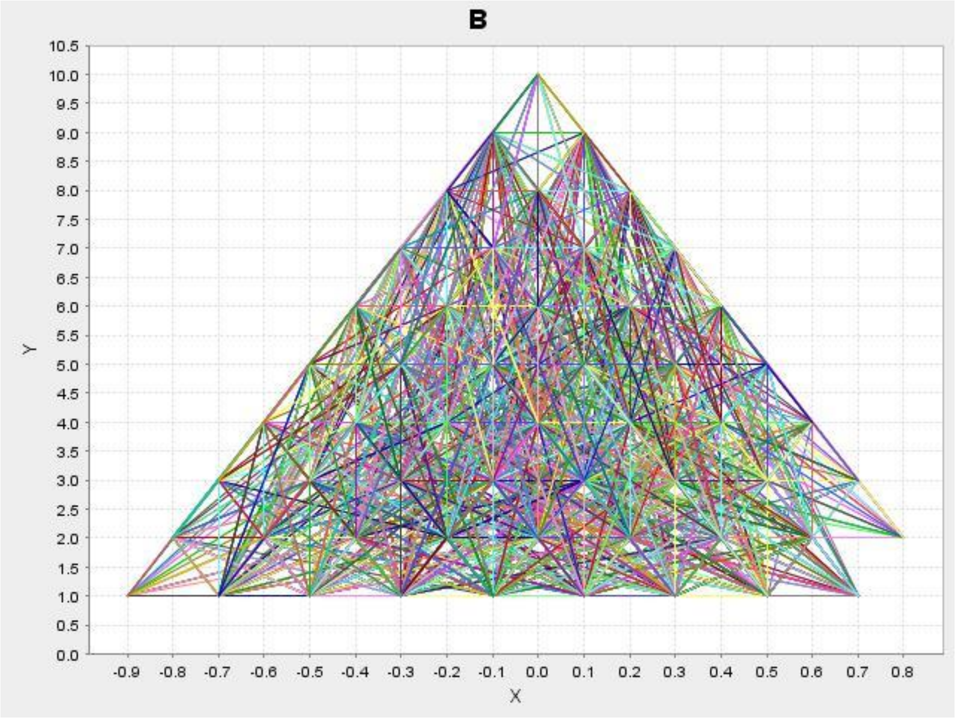
iii. Cycle 10



iv. Cycle 15

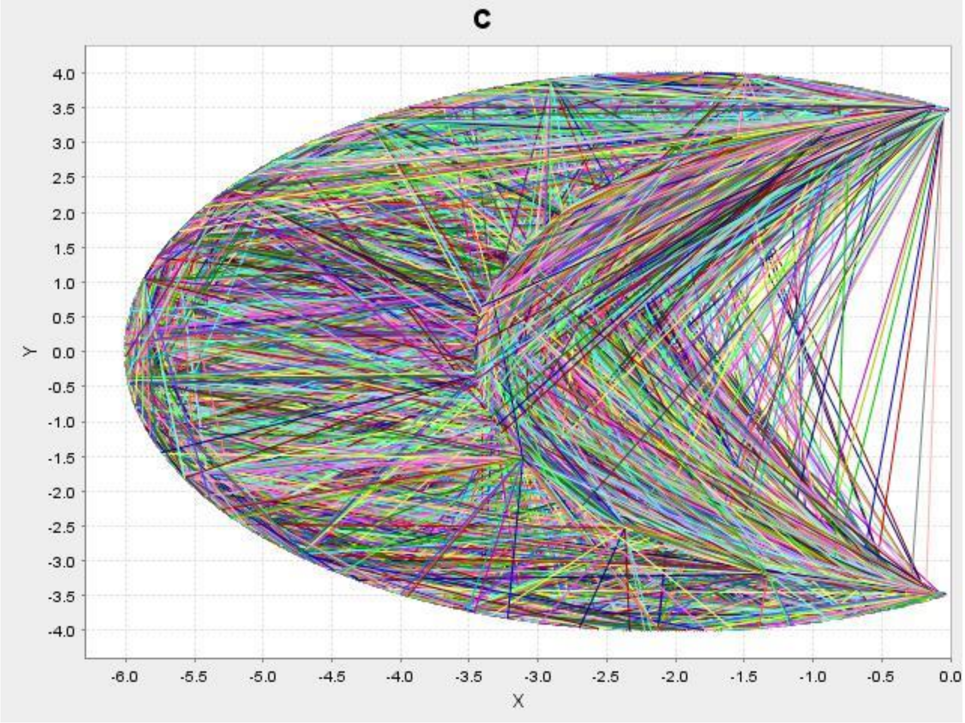


v. Cycle 50

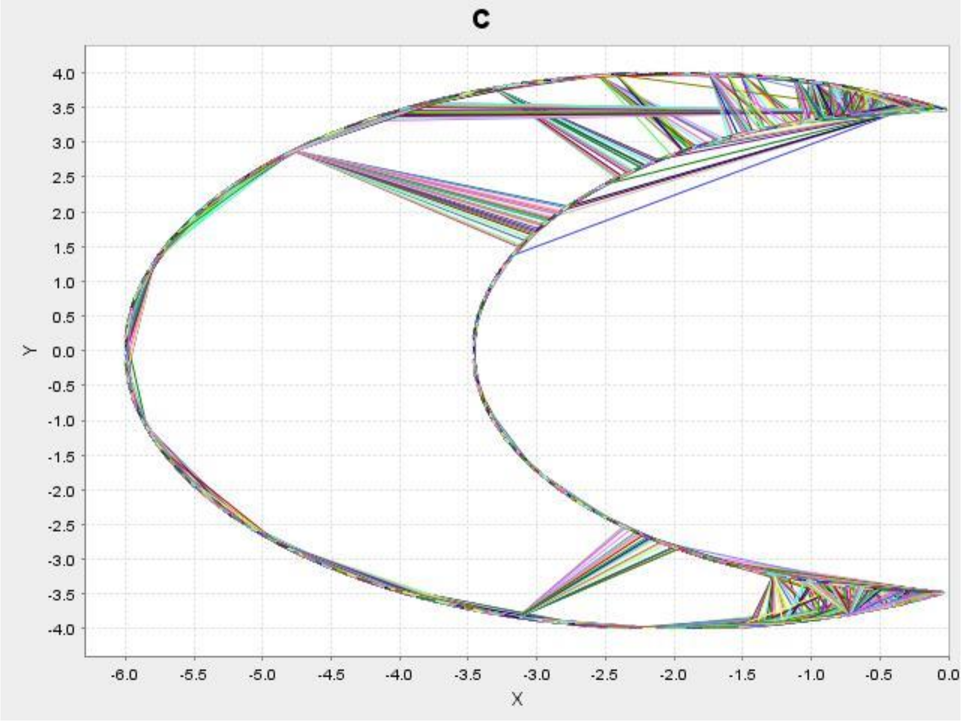


c) <u>Cresent Moon:</u>

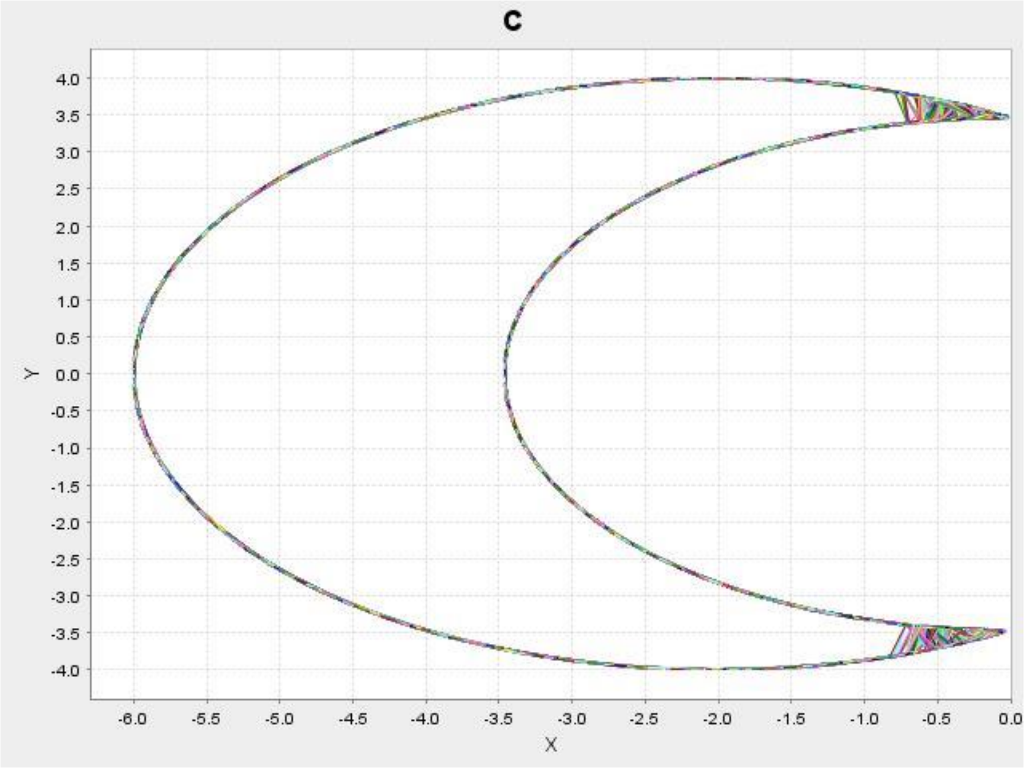
i. Cycle 1



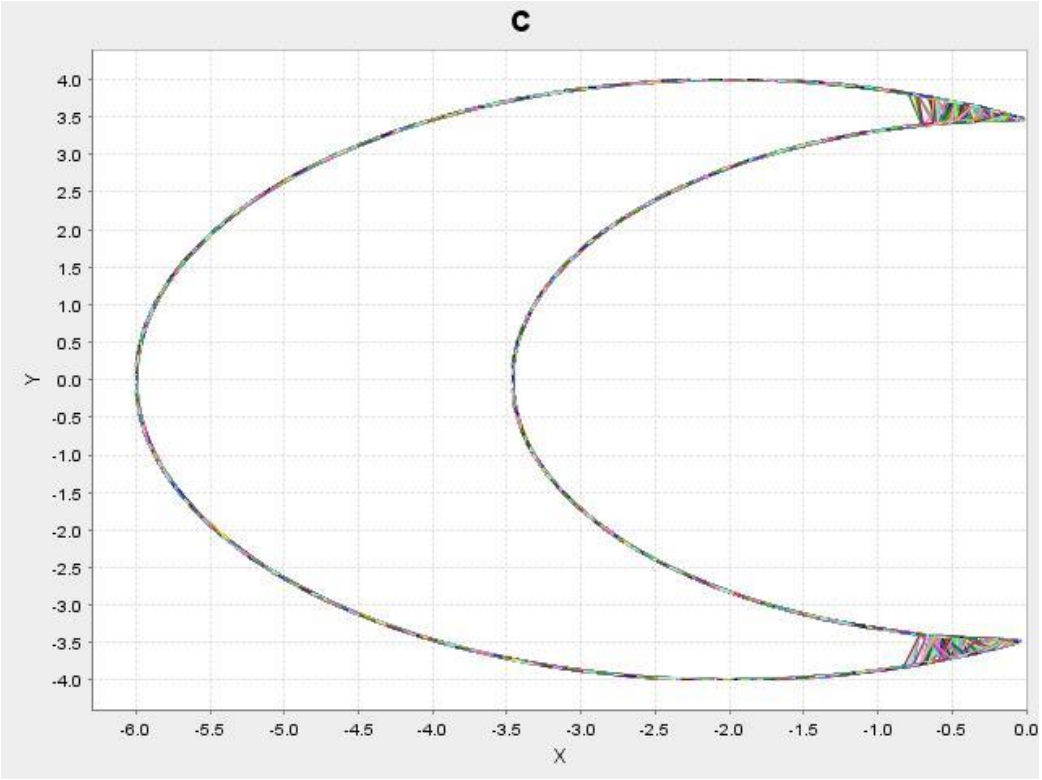
ii. Cycle 5



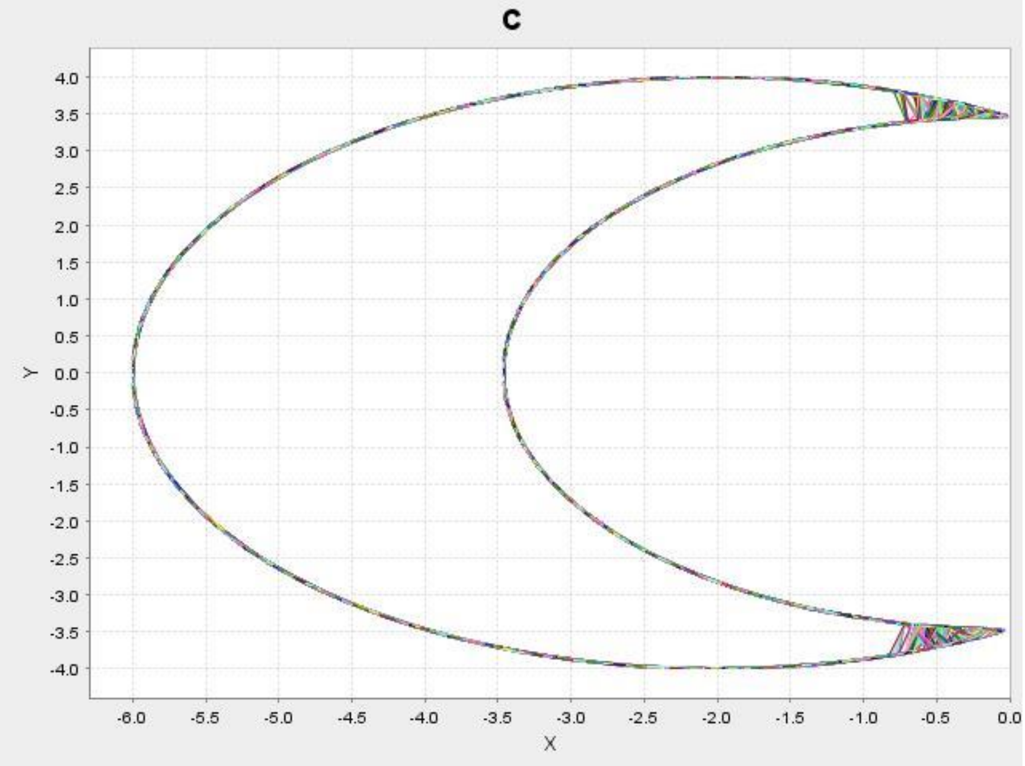
iii. Cycle 10



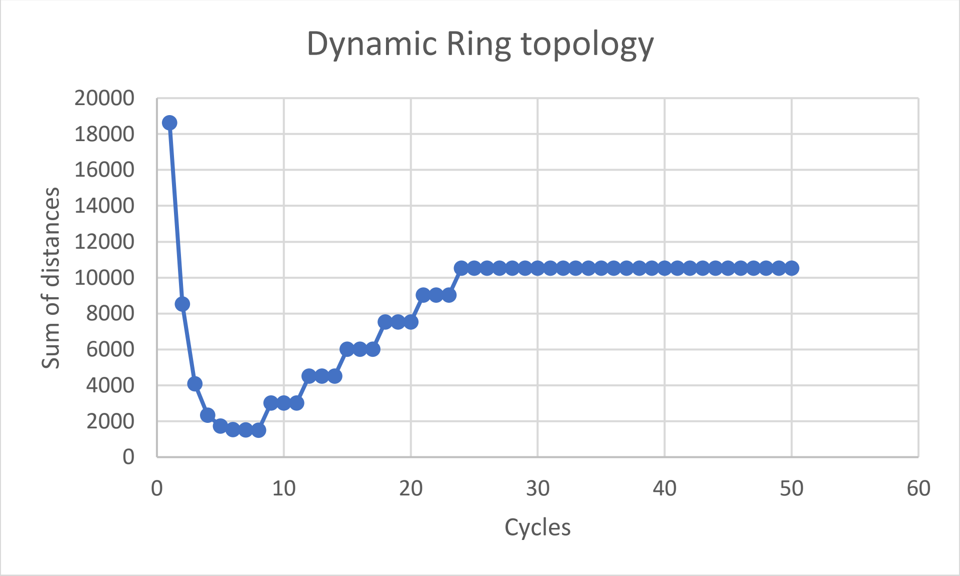
iv. Cycle 15

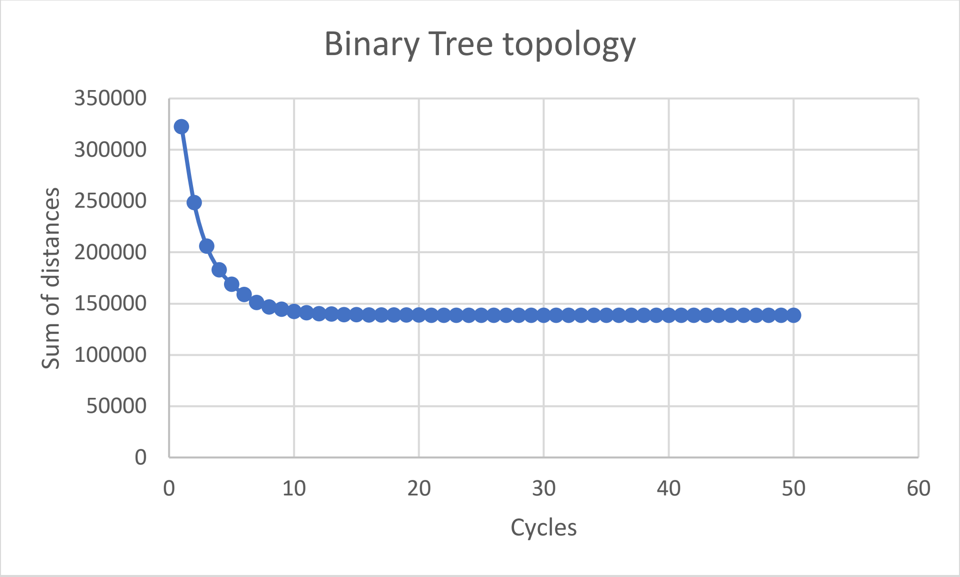


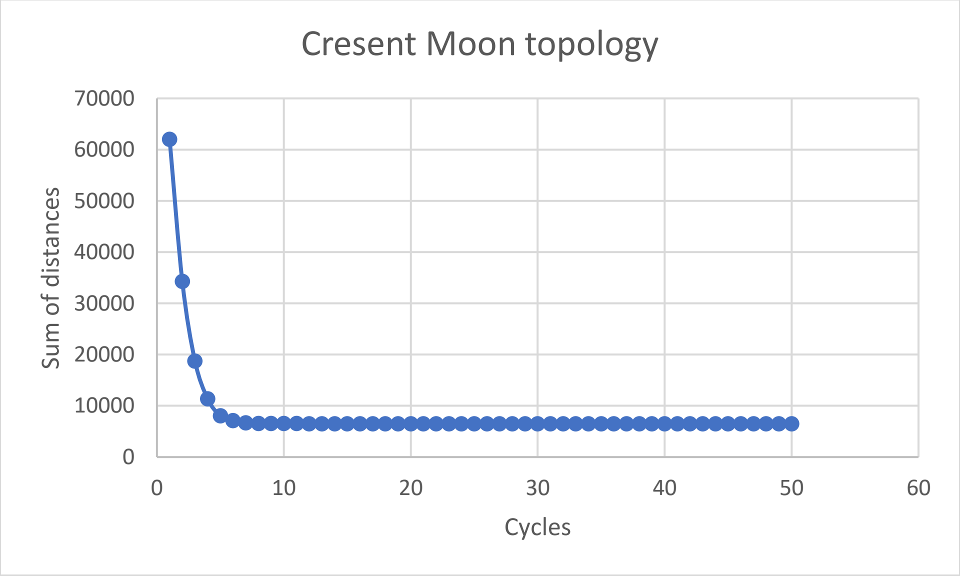
v. Cycle 50



2) XY plots of sum of distances against cycles for each of the three topologies:







3) **Crescent moon topology:** Crescent moon topology is a special case of dynamic ring topology where the nodes instead of being placed on a circle of radius 'r ' are rather placed on two intersecting circles of radii 'r_1' and 'r_2' appropriately.

For the purpose of this homework, the circles are chosen as follows:

x_1 = r_1*cos(angle), y_1 = r_1*sin(angle)

x_2 = r_2*cos(angle)-a, y_2 = r_2*sin(angle)

where C_1 , C_2 are two circles intersecting on y-axis. The radius of C_1 can be obtained by solving for the points of intersection of C_2 with y-axis. Then those points for which x_1, x_2 <=0 are chosen as the coordinates for respective nodes.

During the Network evolution phase, the distance formula is given by the Euclidian distance between two coordinates.

Distance formula: $((x_1-x_2)^2+(y_1-y_2)^2)^{1/2}$

4)
a. Storing the created nodes in an arraylist and iterating over the arraylist during Network initialization and Network evolution phase makes sure that there are no separated nodes in the Dynamic ring and Crescent moon topologies. Because, iterating over all the nodes where each node exchanges information with its neighbors ensures that there are no separates nodes after the iteration is complete.

b. Having duplicate neighbors defeats the purpose of the whole overlay topologies. Therefore, the neighbor list cannot have same node in multiple entries. A HashSet implementation has been used specifically for this purpose.

CODE

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Random;

import java.awt.Color;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileOutputStream;
import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.Writer;


import org.jfree.chart.ChartFactory;

import org.jfree.chart.JFreeChart;

import org.jfree.data.xy.XYDataItem;

import org.jfree.data.xy.XYSeries;

import org.jfree.chart.plot.PlotOrientation;

import org.jfree.data.xy.XYSeriesCollection;

import org.jfree.chart.ChartUtilities;



public class TMAN {


        public static void main(String args[]){


                //Assigning passed arguments from main


                int N = Integer.parseInt(args[0]);

                int k = Integer.parseInt(args[1]);

                String topology = args[2];

                ArrayList<Integer> radii = new ArrayList<Integer>();


                //Array of radii for Dynamic ring topology
```

```
if(args.length>3){
        String radii_input = args[4];
        String[] radii_comma = radii_input.split(",");
        int num_of_radii = Integer.parseInt(args[3]);
        for(int i=0;i<num_of_radii;i++){
                radii.add(Integer.parseInt(radii_comma[i]));
        }

}

//Running specific methods

switch(topology){

case "D" :
        ExecuteRing(N,k,radii,topology);
        break;

case "B" :
        ExecuteTree(N,k,topology);
        break;

case "C" :
        ExecuteCresent(N,k,topology);
        break;

}
```

```
	}

/*******************************************************End of
Main*************************************************************/

/***************************************************Dynamic Ring
topology******************************************************/

	public static void ExecuteRing(int N, int k, ArrayList<Integer> radii, String topology){

		//Nodes list
		ArrayList<Node> nodes_list = new ArrayList<Node>();

		int current_radius =radii.get(0);
		int num_of_nodes = N;
		int num_of_neighbors = k;
		double angle_diff = (double)360/num_of_nodes;
		double angle = 0;

		//Creating the nodes

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

			double x_value = Math.cos(angle)*current_radius;
			double y_value = Math.sin(angle)*current_radius;
			nodes_list.add(new Node(i, x_value, y_value));
			angle+=angle_diff;
```

```
}

//Network initialization

Iterator<Node> iter = nodes_list.iterator();
Random rand = new Random();
HashSet<Integer> set = new HashSet<Integer>();
while(iter.hasNext()){

        Node n = (Node) iter.next();

        //picking k random neighbors

        while(set.size()<num_of_neighbors){
                int r = rand.nextInt(num_of_nodes);
                set.add(r);
                if(r == n.get_id())
                        set.remove(r);
        }

        Iterator<Integer> it = set.iterator();
        while(it.hasNext()){
                n.add_neighbor(nodes_list.get((int) it.next()));
        }
        set.clear();
```

```
}

//Network Evolution

int target_radius = 0;
int radius_counter = 2; //random value



//Cycles
for(int i=0;i<50;i++){
        //Rereading the radius value
        if(i%5 == 0 && i/5<radii.size()){
                target_radius = radii.get(i/5);
        }

        if(i==8){
                radius_counter=0;
        }

        //Incrementing the radius by 1
        if(radius_counter%3 == 0 && current_radius<target_radius){
                current_radius++;
                //Updating coordinate values
                updateRingCoordinates(nodes_list,current_radius,angle_diff);
        }
```

```
            if(i>=8)
                radius_counter++;

            Iterator<Node> it = nodes_list.iterator();
            while(it.hasNext()){
                Node node = (Node) it.next();
                Node neighbor_node = node.pickNeighbor(num_of_neighbors);

    node.rearrange(neighbor_node.getList(),num_of_neighbors,topology);

    neighbor_node.rearrange(node.getList(),num_of_neighbors,topology);
            }

            WriteSumDistances(nodes_list,i,N,k,topology);

            if(i==0 || i==4 || i==9 || i==14 || i==49){
                new Export(nodes_list,topology,N,k,i+1);
                WriteNeigbors(nodes_list,i+1,N,k,topology);
            }

        }

    }

    public static void WriteNeigbors(ArrayList<Node> nodes_list, int i, int N, int k, String
topology) {
```

```
                    String merged ="";
                    Iterator itrt = nodes_list.iterator();
                    while(itrt.hasNext()){
                              Node n = (Node)itrt.next();
                              String n1 = Integer.toString(n.get_id());
                              Iterator ir = n.getList().iterator();
                              merged+= n1 + " --> ";
                              ArrayList<Integer> neighbors = new ArrayList<Integer>();
                              while(ir.hasNext()){
                                        Node nd = (Node)ir.next();
                                        neighbors.add(nd.get_id());
                                        Collections.sort(neighbors);
                              }
                              String n2="";
                              Iterator itrtr = neighbors.iterator();
                              while(itrtr.hasNext()){
                                        n2+=itrtr.next();
                                        n2+=",";
                              }
                              merged+=n2;
                              merged+="\r\n";
                    }

                    String filename = topology + "_N" + Integer.toString(N) + "_k" + Integer.toString(k)
+"_" + i;
```

```
        try(PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(filename+".txt", true)))) {
           out.println(merged);


        }catch (IOException e) {
           System.err.println(e);
        }



}


    public static void WriteSumDistances(ArrayList<Node> nodes_list, int i, int N, int k, String
topology) {


        double dist =0;
        Iterator it = nodes_list.iterator();
        while(it.hasNext()){
            Node n = (Node) it.next();
            Iterator ir = n.getList().iterator();
            while(ir.hasNext()){
                Node nd = (Node)ir.next();
                dist+= Node.getDistance(n,nd,topology);
            }
        }

        String filename = topology + "_N" + Integer.toString(N) + "_k" + Integer.toString(k);
```

```
                    try(PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(filename+".txt", true)))) {
                       out.println("cycle " + i + ": " + dist);
                       out.println();
                    }catch (IOException e) {
                       System.err.println(e);
                    }


}




          public static void updateRingCoordinates(ArrayList<Node> nodes_list, int current_radius,
double angle_diff) {
                    double angle =0;
                    Iterator irtr = nodes_list.iterator();
                    while(irtr.hasNext()){
                              Node node = (Node) irtr.next();
                              node.set_X(Math.cos(angle)*current_radius);
                              node.set_Y(Math.sin(angle)*current_radius);
                              angle+=angle_diff;
                    }


          }

/****************************************End of Dynamic ring
topology****************************************************/
```

```
/************************************************Binary Tree Topology****************************************************************/


	public static void ExecuteTree(int N, int k, String topology){


		ArrayList<Node> tree_nodes = new ArrayList<Node>();


		//Creating the tree nodes


		for(int i=1;i<=N;i++){
			if(i==1){
				Node t = new Node(i,0,10);
				tree_nodes.add(t);
			}
			else if(i%2==0){
				Node tn = new Node(i,tree_nodes.get(i/2-1).get_X()-0.1,10-Math.floor((Math.log10(i)/Math.log10(2))));
				tree_nodes.add(tn);
			}
			else{
				Node tn = new Node(i,tree_nodes.get(i/2-1).get_X()+0.1,10-Math.floor((Math.log10(i)/Math.log10(2))));
				tree_nodes.add(tn);
			}


		}


		//Network initialization
```

```
Iterator iter = tree_nodes.iterator();
Random rand = new Random();
HashSet<Integer> set = new HashSet<Integer>();
while(iter.hasNext()){

    Node n = (Node) iter.next();

    //picking k random neighbors

    while(set.size()<k){
        int r = rand.nextInt(N)+1;
        set.add(r);
        if(r == n.get_id())
            set.remove(r);
    }

    Iterator it = set.iterator();
    while(it.hasNext()){
        n.add_neighbor(tree_nodes.get((int) it.next()-1));
    }
    set.clear();
}

//Network evolution

//Cycles
```

```
        for(int i=0;i<50;i++){

            Iterator it = tree_nodes.iterator();

            while(it.hasNext()){

                Node node = (Node) it.next();

                Node neighbor_node = node.pickNeighbor(k);

                ArrayList<Node> my_neighbors = node.getList();

                ArrayList<Node> your_neighbors = neighbor_node.getList();

                node.rearrange(your_neighbors,k,topology);

                neighbor_node.rearrange(my_neighbors,k,topology);

            }


            WriteSumDistances(tree_nodes,i,N,k,topology);


            if(i==0 || i==4 || i==9 || i==14 || i==49){

                new Export(tree_nodes,topology,N,k,i+1);

                WriteNeigbors(tree_nodes,i+1,N,k,topology);

            }

        }


    }


/**********************************************End of Binary tree
topology*********************************************/


/************************************************Cresent moon
topology*************************************************/


    public static void ExecuteCresent(int N, int k, String topology){
```

```
//Nodes list
        ArrayList<Node> moon_nodes = new ArrayList<Node>();

        double r1 =2*Math.sqrt(3);
        double r2 = 4;
        int num_of_nodes = N;
        int num_of_neighbors = k;
        double angle_diff = (double)360/num_of_nodes;
        double angle = 0;

        //Creating the nodes

        int j=0;
        while(moon_nodes.size()<N){

            double x1 = Math.cos(angle)*r1;
            double y1 = Math.sin(angle)*r1;
            double x2 = Math.cos(angle)*r2-2;
            double y2 = Math.sin(angle)*r2;
            if(x1<=0){
                moon_nodes.add(new Node(j, x1, y1));
                j++;
            }
            if(x2<=0){
                moon_nodes.add(new Node(j, x2, y2));
                j++;
```

```
    }

    angle+=angle_diff;

}

//Network initialization

Iterator<Node> iter = moon_nodes.iterator();
Random rand = new Random();
HashSet<Integer> set = new HashSet<Integer>();
while(iter.hasNext()){

    Node n = (Node) iter.next();

    //picking k random neighbors

    while(set.size()<k){
        int r = rand.nextInt(N)+1;
        set.add(r);
        if(r == n.get_id())
            set.remove(r);
    }

    Iterator<Integer> it = set.iterator();
    while(it.hasNext()){
        n.add_neighbor(moon_nodes.get((int) it.next()-1));
```

```
                    }
                    set.clear();
                }

                //Network evolution

                //Cycles
                for(int i=0;i<50;i++){
                    Iterator it = moon_nodes.iterator();
                    while(it.hasNext()){
                        Node node = (Node) it.next();
                        Node neighbor_node = node.pickNeighbor(k);
                        ArrayList<Node> my_neighbors = node.getList();
                        ArrayList<Node>          your_neighbors          =
neighbor_node.getList();
                        node.rearrange(your_neighbors,k,topology);

    neighbor_node.rearrange(my_neighbors,k,topology);
                    }

                    WriteSumDistances(moon_nodes,i,N,k,topology);

                    if(i==0 || i==4 || i==9 || i==14 || i==49){
                        new Export(moon_nodes,topology,N,k,i+1);
                        WriteNeigbors(moon_nodes,i+1,N,k,topology);
                    }

                }
```

```
          }



/***********************************************End of cresent moon
topology*******************************************/

}

/********************************************End of TMAN
class*********************************************************/



import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Random;

public class Node {

          private int node_id;
          private double x_coordinate;

          private double y_coordinate;

          ArrayList<Node> k_neighbors = new ArrayList<Node>();


          public Node(int id, double x, double y){

                      this.node_id = id;

                      this.x_coordinate = x;

                      this.y_coordinate = y;

          }


          public int get_id(){

                      return node_id;

          }


          public double get_X(){

                      return x_coordinate;

          }


          public void set_X(double x){

                      this.x_coordinate = x;

          }


          public double get_Y(){

                      return y_coordinate;

          }


          public void set_Y(double y){
                  this.y_coordinate = y;
              }

              public void add_neighbor(Node n){
                  k_neighbors.add(n);
              }

              public Node pickNeighbor(int k){
                  Random ran = new Random();
                  int n;
                  n = ran.nextInt(k);
                  return k_neighbors.get(n);
              }

              public ArrayList<Node> getList(){
                  return k_neighbors;
              }

              public void rearrange(ArrayList<Node> List, int k, String topology){
                  ArrayList<Node> merged_list = new ArrayList<Node>();
                  merged_list.addAll(List);
                  merged_list.addAll(k_neighbors);

                  Iterator iter = merged_list.iterator();
                  HashMap<Node,Double> map = new HashMap<Node,Double>();
                  while(iter.hasNext()){
                      Node nd = (Node) iter.next();
                  if(this!=nd){
                  double dist = getDistance(this,nd,topology);
                  map.put(nd, dist);
                  }
              }

              Map<Node, Double> sorted_map = sortByValue(map);
              Iterator<Node> itr = sorted_map.keySet().iterator();
              int count = 1;
              ArrayList<Node> new_neighbors = new ArrayList<Node>();
              while(itr.hasNext() && count<=k){
                  new_neighbors.add((Node) itr.next());
                  count++;
              }

              this.updateNeighbors(new_neighbors);

          }

          public void updateNeighbors(ArrayList<Node> nlist) {
              this.k_neighbors.clear();
              k_neighbors.addAll(nlist);

          }

          //Method to sort map by values
```

```
    public static <K, V extends Comparable<? super V>> Map<K, V>
sortByValue( Map<K, V> map )
    {
      List<Map.Entry<K, V>> list =
        new LinkedList<Map.Entry<K, V>>( map.entrySet() );
      Collections.sort( list, new Comparator<Map.Entry<K, V>>()
      {
        public int compare( Map.Entry<K, V> o1, Map.Entry<K, V> o2 )
        {
          return (o1.getValue()).compareTo( o2.getValue() );
        }
      } );

      Map<K, V> result = new LinkedHashMap<K, V>();
      for (Map.Entry<K, V> entry : list)
      {
        result.put( entry.getKey(), entry.getValue() );
      }
      return result;
    }

    //Method to calculate distance between two nodes

    public static double getDistance(Node n1, Node n2, String topology) {

           double result = 0;
           switch(topology){
```

```
case "D" :
case "C" :
        double x1 = n1.get_X();
        double y1 = n1.get_Y();
        double x2 = n2.get_X();
        double y2 = n2.get_Y();

        result = Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
        break;

case "B" :
        int a = n1.get_id();
        int b = n2.get_id();
        int bits = 10;
        int alevel=bits;
        int blevel=bits;
        int commonprefix=0;
        int mask = 1 << bits-1;

        // find the level of node a
        while( (mask & a) == 0 )
        {
                a <<= 1;
                alevel--;
        }
```

```
            // find the level of node b
            while( (mask & b) == 0 )
            {
                b <<= 1;
                blevel--;
            }

            int length = Math.min(alevel,blevel);
            while( (mask & ~(a ^ b)) != 0 && length>0)
            {
                b <<= 1;
                a <<= 1;
                commonprefix++;
                length--;
            }
            result = alevel - commonprefix + blevel - commonprefix;

        }

        return result;
    }


}


import java.io.File;
```

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYDataItem;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

public class Export {

    Export(ArrayList<Node> list, String topology, int N, int k, int cycle){
        Iterator<Node> it =list.iterator();
        XYSeriesCollection dataset = new XYSeriesCollection();

        ArrayList<XYSeries> series_list = new ArrayList<XYSeries>();
        while(it.hasNext()){
            Node n = (Node)it.next();
            XYDataItem myXY = new XYDataItem(n.get_X(),n.get_Y());

            Iterator<Node> i = n.getList().iterator();
            while(i.hasNext()){
                Node nd = (Node)i.next();
                XYSeries series = new XYSeries("",false);
                series.add(myXY);
                series.add(new XYDataItem(nd.get_X(),nd.get_Y()));
                series_list.add(series);
            }
        }

        //System.out.println(series_list.size());
        Iterator<XYSeries> irt = series_list.iterator();
        while(irt.hasNext()){
            dataset.addSeries((XYSeries) irt.next());
        }

        JFreeChart xylineChart = ChartFactory.createXYLineChart(
           topology,
           "X",
           "Y",
           dataset,
           PlotOrientation.VERTICAL,
           false, true, false);

          int width = 640; /* Width of the image */
          int height = 480; /* Height of the image */
          String filename = topology + "_N" + Integer.toString(N) + "_k" +
Integer.toString(k) +"_" + Integer.toString(cycle);
          File XYChart = new File( filename + ".jpeg" );
          try {
                ChartUtilities.saveChartAsJPEG( XYChart, xylineChart, width,
height);
```

```
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }

}
```