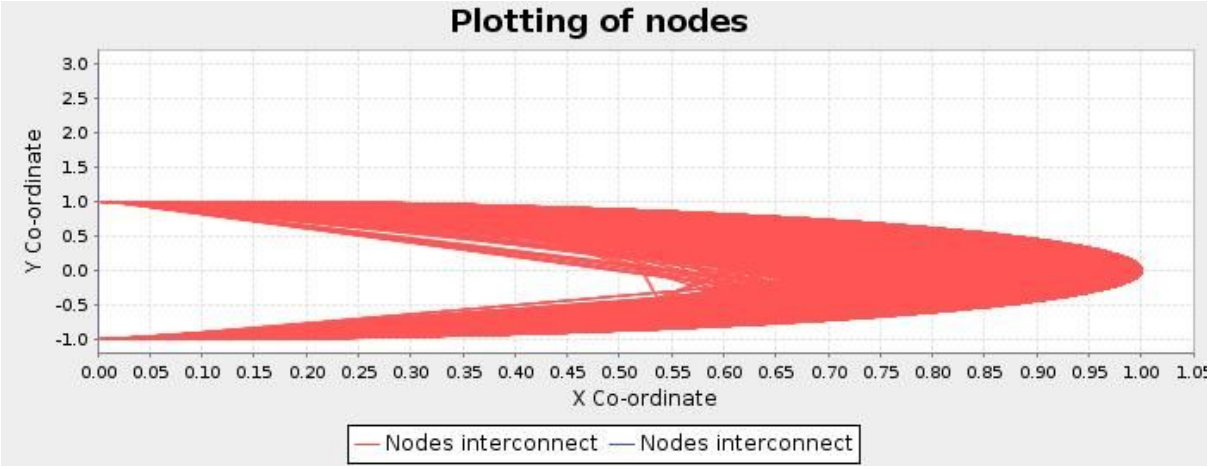
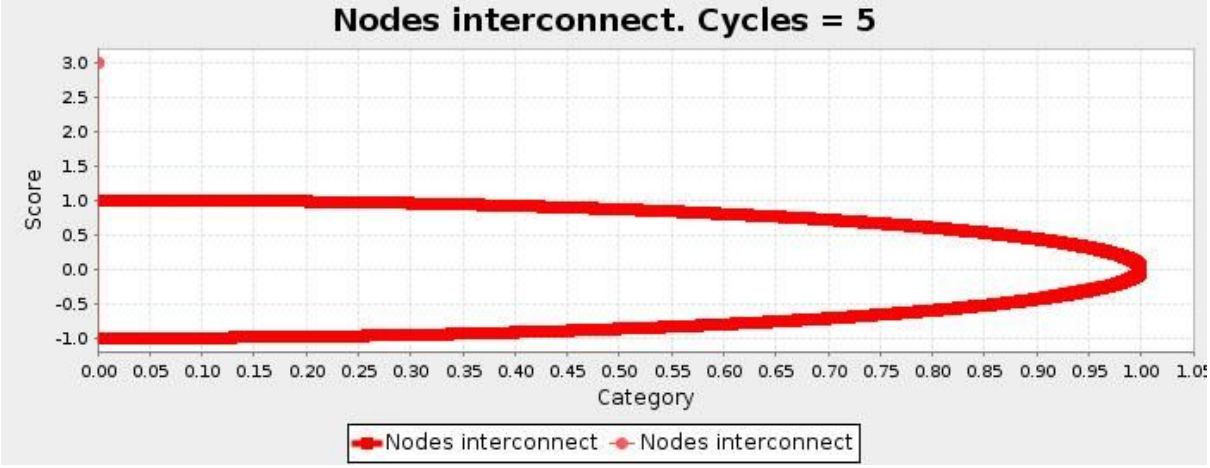
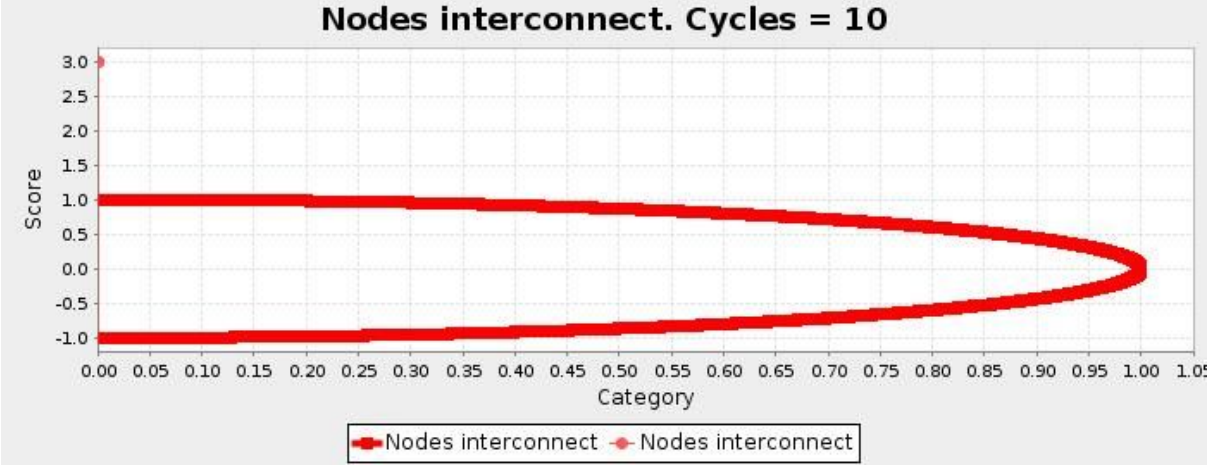
DISTANCE FORMULA FOR SPECTACLES TOPOLOGY:

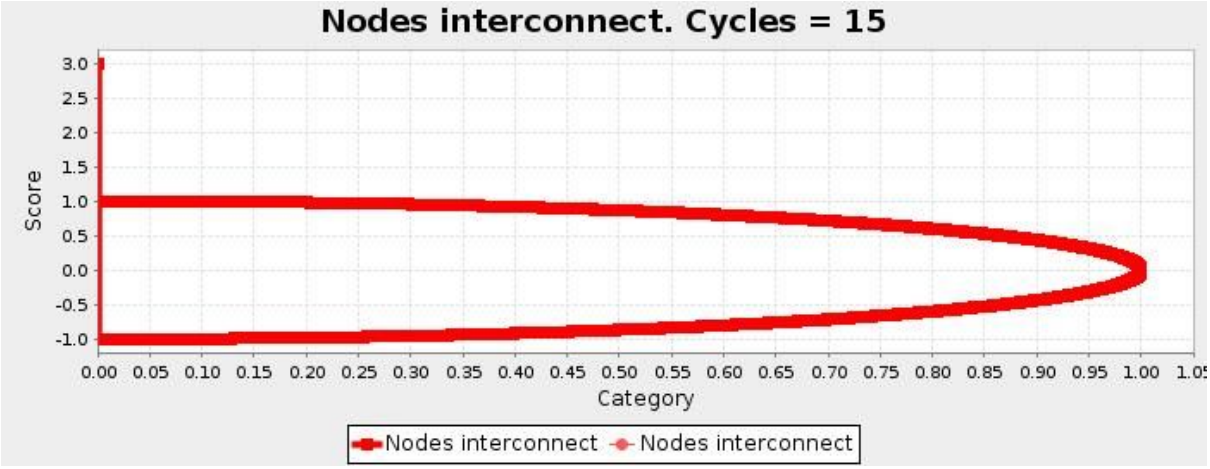
For the orientation of N nodes in a topology similar to the shape of a spectacle (2 full circles connected by a semi-circle in between), we distribute the N nodes evenly among these 2 circles and the semi-circle. Therefore, the first circle will have 400 nodes, the semi-circle in middle will have 200 nodes and the second circle will have 400 nodes. The formula for θ and the X and Y co-ordinate for the nodes is given by :

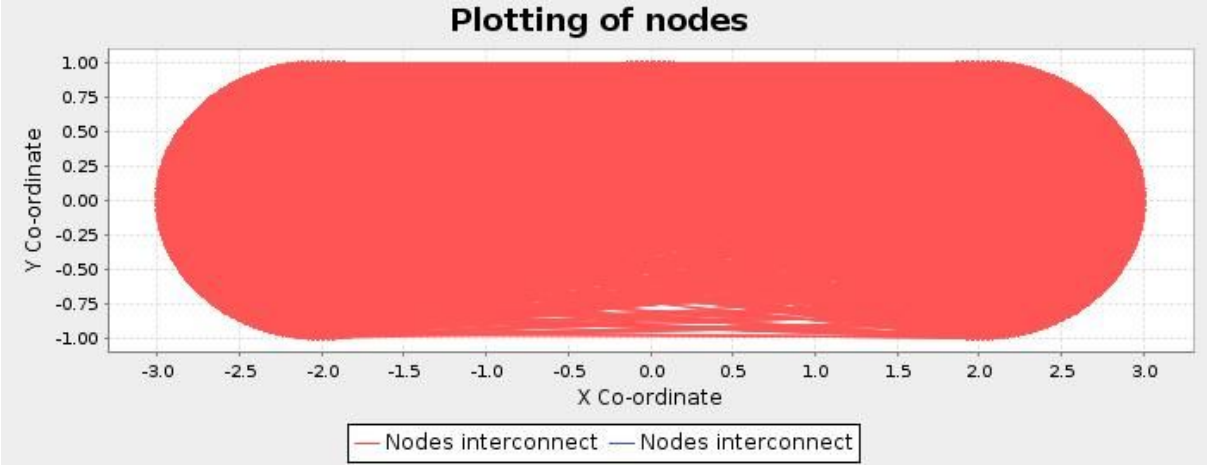
```
double e = Math.PI;
System.out.println("Topology = S");
for(int i =1; i<=400; i++){
        e = e - (Math.PI*2/400);
        nodes[i].x_co = 2 + Math.cos(e);
        nodes[i].y_co = Math.sin(e);

}
e = 0;
for(int i =401; i<=600; i++){
        e = e + (Math.PI/200);
        nodes[i].x_co = Math.cos(e);
        nodes[i].y_co = Math.sin(e);
}
e = 0;
for(int i =601; i<=1000; i++){
        e = e - (Math.PI*2/400);
        nodes[i].x_co = -2 + Math.cos(e);
        nodes[i].y_co = Math.sin(e);
}
```

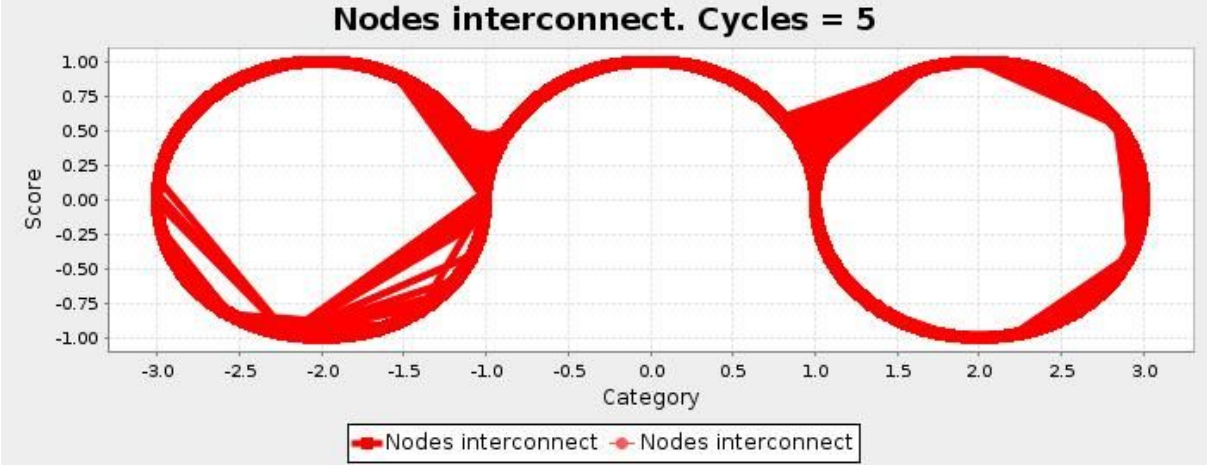

Plotting of nodes
Y Co-ordinate
X Co-ordinate
Nodes interconnect
Nodes interconnect

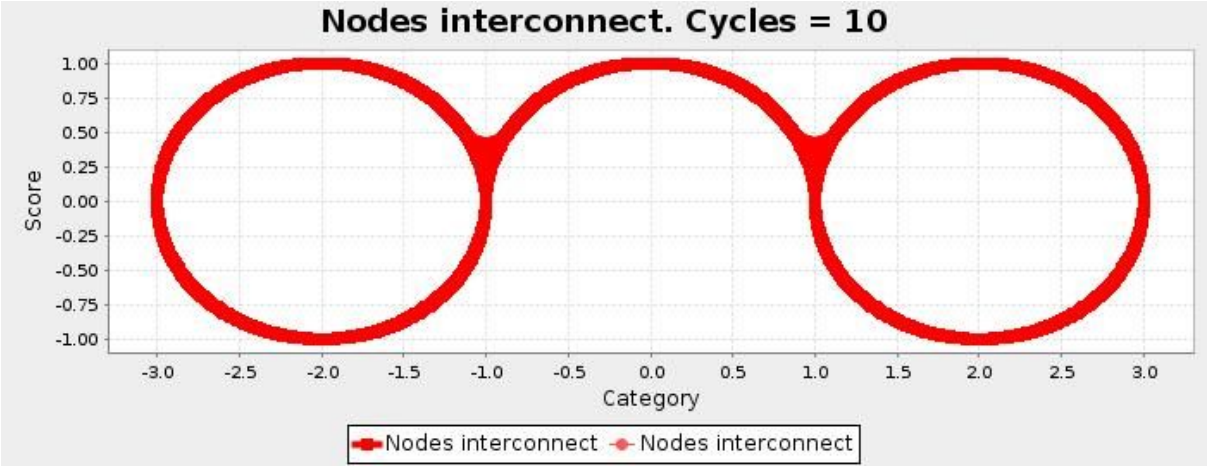

Nodes interconnect. Cycles = 5
Score
Category
Nodes interconnect
Nodes interconnect

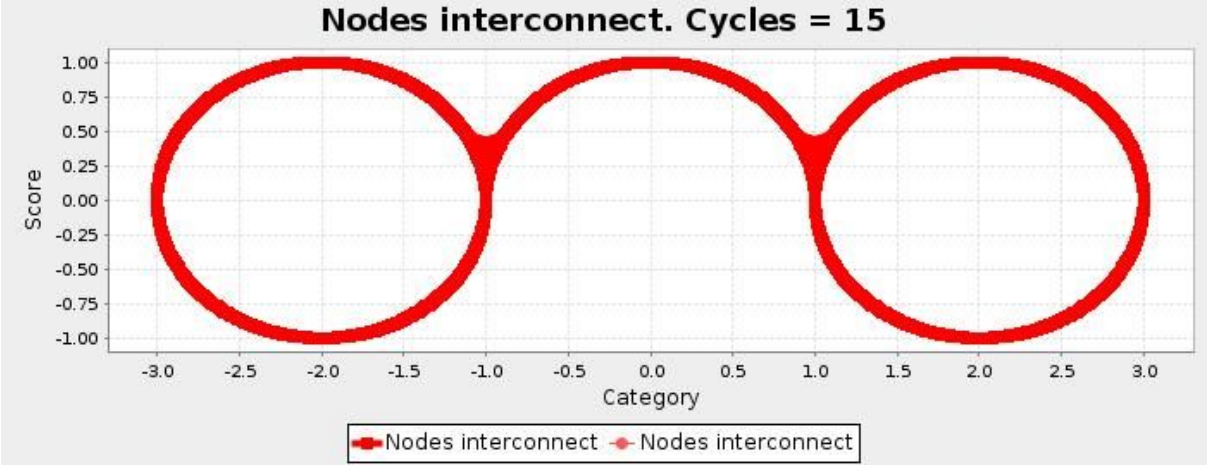

Nodes interconnect. Cycles = 10
Score
Category
Nodes interconnect
Nodes interconnect


Nodes interconnect. Cycles = 15
Score
Category
Nodes interconnect
Nodes interconnect


Plotting of nodes
Y Co-ordinate
X Co-ordinate
Nodes interconnect
Nodes interconnect


Nodes interconnect. Cycles = 5
Score
Category
Nodes interconnect
Nodes interconnect





CODE -

```
//Author : Akash R Vasishta
//UFID : 53955080
import java.util.*;
import org.jfree.ui.RefineryUtilities;
import java.io.*;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
```

```
//TMAN class which implements the TMAN's algorithm.
public class TMAN {
	static int N;
	static int k;
	static Node nodes[];
	static int cycles = 40;
	static char topology;
	public static double[] dis = new double[cycles];

  public static void main(String[] args) {
	N=Integer.parseInt(args[0]);
	k=Integer.parseInt(args[1]);
	topology = args[2].charAt(0);
	System.out.println(topology);
	nodes = new Node[N+1];
//	nodes.received_list = new int[k];
//	Map<String, String> node = new HashMap<String, String>();/
//	node.put("dog", "type of animal");
//	System.out.println(node.get("dog"));
	networkInitialization();
//	System.out.println("The generated nodes are - " + nodes[0].node_id);
   System.out.println("The nodes are - " );
   System.out.print("node_id - x_co - y_co - theta");
	for(int i = 1; i<=N; i++){
		System.out.println("\n"+nodes[i].node_id + "  " + nodes[i].x_co + "  " +
nodes[i].y_co + "  " + nodes[i].theta);
		System.out.print("    neighbors = ");
		for(int j = 0; j < k; j++){
			System.out.print(nodes[i].neighbors[j] + " ");
		}
	}

	networkEvolution();

  }
//The Network initialization phase. Every node randomly selects k neighbors and places
them into its neighbor list.
//The nodes are placed at a location given by the overlay network topology.
	public static void networkInitialization(){
		Random random = new Random();
		int next;
		double theta;
		System.out.println("Initializing the network");
		nodes[0] = new Node();
		System.out.println(" nodes.length = " + nodes.length);
		for(int i = 1; i<N+1; i++){
```

```
                        nodes[i] = new Node();
                        nodes[i].neighbors = new int[k];
                        HashSet<Integer> used = new HashSet<Integer>();
//Nodeid
                        nodes[i].node_id = i;
//                  System.out.println(nodes[i].node_id);
//Generating 'k' random neighbors list
                        for(int j = 0; j<k; j++){
                                next = random.nextInt(N-1) + 1;
                        while (used.contains(next) || next == i) { //while we have already used
the number
                        next = random.nextInt(N-1) + 1; //generate a new one because it's
already used
//                                      System.out.println("@" + next);
                        }
                        if(i == N && topology == 'B'){
                                used.add(next);
                                if(j==0)
                                        nodes[i].neighbors[j] = 1;
                                else if (j==1)
                                        nodes[i].neighbors[j] = 999;
                                else
                                                nodes[i].neighbors[j] = next;

                        } else{
                                        used.add(next);
                                        nodes[i].neighbors[j] = next;

                        }
                        }
//Calculating XCo and YCo
                        if(topology == 'B'){
                                theta = (Math.PI/2)-((i-1)*Math.PI/(N-2));
                                if (i == 1000) {
                                        nodes[i].x_co = 0;
                                        nodes[i].y_co = 3;
                                } else {
                                        nodes[i].x_co = Math.cos(theta);
                                        nodes[i].y_co = Math.sin(-theta);
                                }
                                nodes[i].theta = theta;
//                  System.out.println(nodes[0].x_co);
                        }
                        else{
                                System.out.println("Topology = S");
```

```
                        }
                }

                if (topology == 'S'){
                        double e = Math.PI;
                        System.out.println("Topology = S");
                        for(int i =1; i<=400; i++){
//                              e = (Math.PI/2)-((i-1)*(Math.PI/(10-2)));
//                              System.out.println("e =" + (random.nextInt(10) + 1));
                                e = e - (Math.PI*2/400);
//                              System.out.println(i +" "+(2+Math.cos(e)) + " " +
Math.sin(e));
                                nodes[i].x_co = 2 + Math.cos(e);
                                nodes[i].y_co = Math.sin(e);

                        }
                        e = 0;
                        for(int i =401; i<=600; i++){
//                              e = (Math.PI/2)-((i-1)*(Math.PI/(10-2)));
//                              System.out.println("e =" + (random.nextInt(10) + 1));
                                e = e + (Math.PI/200);
//                              System.out.println("\n" + i +" "+(Math.cos(e)) + " " +
Math.sin(e));
                                nodes[i].x_co = Math.cos(e);
                                nodes[i].y_co = Math.sin(e);
                        }
                        e = 0;
                        for(int i =601; i<=1000; i++){
//                              e = (Math.PI/2)-((i-1)*(Math.PI/(10-2)));
//                              System.out.println("e =" + (random.nextInt(10) + 1));
                                e = e - (Math.PI*2/400);
//                              System.out.println(i +" "+(-2 + Math.cos(e)) + " " +
Math.sin(e));
                                nodes[i].x_co = -2 + Math.cos(e);
                                nodes[i].y_co = Math.sin(e);
                        }

                }
//              nodes[1000] = new Node()
/*      for(int i = 0; i<N; i++){
                System.out.println(nodes[i].node_id);
        }
*/
        }
//Network Evolution phase.
//In every cycle of the iterative algorithm, every node randomly selects one of its neighbors,
```

```
//and then sends a list consisting of the identifiers of its neighbors and of itself to that neighbor. The selected
//neighbor also sends its neighbors list back to the node which initiated the action. Upon receiving the new
//neighbor list, the nodes select the nearest k nodes from both the new and old lists as their neighbors and
//discards all the others.
	public static void networkEvolution(){
		int neighbor_id;
		int[] neighborList = new int[k];
		String fileName;
//		int[] receivedNList = new int[k];
//		ArrayList<Integer> mergedList = new ArrayList<Integer>();
		int r;
		double d;
		System.out.println("\n*****************************************************\nIn the networkEvolution phase\n*****************************************************");
		for(int j=0; j<cycles; j++){
//			dis = 0;

System.out.println("\n#######################################\ncycle = " + j + "\n#######################################\n");
			for(int i = 1; i < N+1; i++){
//				System.out.println("\nNode = " + i);
				neighbor_id = selectPeer(i);
				neighborList = nodes[i].neighbors.clone();
/*				System.out.print("Neighbor List = [");
				for(int h=0; h<k; h++){
					System.out.print(" " + neighborList[h]);
				}
				System.out.println(" ]");
				System.out.print("Random Neighbor = " + neighbor_id);
*/
				Arrays.sort(neighborList);
//				System.out.print(" Sorted. To be ex NL = [");
//				for(int h=0; h<k; h++){
//					System.out.print(" " + neighborList[h]);
//				}
//				System.out.println(" ]");

				r = Arrays.binarySearch(neighborList, neighbor_id);
//				System.out.println(" index = " + r);
//				r = neighborList.indexOf(neighbor_id);
				neighborList[r] = i;
//				System.out.print("To be ex NL = [");
//				for(int h=0; h<k; h++){
```

```
//                                      System.out.print(" " + neighborList[h]);
//                                  }
//                                  System.out.println(" ]");
//                                  neighborList = ArrayUtils.removeElement(neighborList,
neighbor_id);
//For received_list hashmap, uncomment below line
//                                  nodes[neighbor_id].received_list.put(i, neighborList);
                                    nodes[neighbor_id].received_list = neighborList.clone();
//f                                 receivedNList = nodes[neighbor_id].neighbors.clone();
//For received_list hashmap, uncomment below line
//                                  nodes[i].received_list.put(neighbor_id, receivedNList);
                                    nodes[i].received_list = nodes[neighbor_id].neighbors.clone();
/*                                  System.out.print("\nReceived NeighborList = [");
                                    for(int h=0; h<k; h++){
                                        System.out.print(" " + nodes[i].received_list[h]);
                                    }
                                    System.out.println(" ]");
*/
                                    updateNL(i);
                                    updateNL(neighbor_id);
//                                  System.out.print("Final my Neighbors = [");
                                    for(int h=0; h<k; h++){
//                                      System.out.print(" " + nodes[i].neighbors[h]);
                                        d = distance(i, nodes[i].neighbors[h]);
                                        if(d != 10000)
                                            dis[j] = dis[j] + d;
                                    }
/*                                  System.out.println(" ]");
                                    System.out.print("Final neighbors Neighbors = [");
                                    for(int h=0; h<k; h++){
                                        System.out.print(" " + nodes[neighbor_id].neighbors[h]);
                                    }
                                    System.out.println(" ]");
*/
                            }
                            if(j == 0){
                 JFreeChart xylineChart = ChartFactory.createXYLineChart(
                "Plotting of nodes ",
              "X Co-ordinate" ,
             "Y Co-ordinate" ,
             createDataset() ,
             PlotOrientation.VERTICAL ,
             true , true , false);

                                    XYLineChart_AWT chart1 = new
XYLineChart_AWT("Topology","Nodes interconnect. Cycles = 1");
```

```
                        chart1.pack( );
                        RefineryUtilities.centerFrameOnScreen( chart1 );
                        chart1.setVisible( true );

                        try{
                                fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".txt";
                                PrintWriter writer = new PrintWriter(fileName, "UTF-8");
                                for(int m=1;m<=N;m++){
                                        writer.print("Node " + String.valueOf(m) + " neighbors =
");
                                        for(int n=0;n<k;n++){
                                                writer.print(" " +
String.valueOf(nodes[m].neighbors[n]));
                                        }
                                writer.println("");
                                }
                                        fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".jpg";
                                ChartUtilities.saveChartAsJPEG(new File(fileName),
xylineChart, 700, 270);
                                writer.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                                }

                        } else if(j == 4){
                                XYLineChart_AWT chart2 = new
XYLineChart_AWT("Topology","Nodes interconnect. Cycles = 5");
                        chart2.pack( );
                        RefineryUtilities.centerFrameOnScreen( chart2 );
                        chart2.setVisible( true );
                        try{
                                fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".txt";
                                PrintWriter writer = new PrintWriter(fileName, "UTF-8");
                                for(int m=1;m<=N;m++){
                                        writer.print("Node " + String.valueOf(m) + " neighbors =
");
                                        for(int n=0;n<k;n++){
                                                writer.print(" " +
String.valueOf(nodes[m].neighbors[n]));
                                        }
                                writer.println("");
                                }
```

```
                                        fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".jpg";
                                        ChartUtilities.saveChartAsJPEG(new File(fileName),
chart2.xylineChart, 700, 270);
                                        writer.close();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                        }

                                } else if(j == 9){
                                        XYLineChart_AWT chart3 = new
XYLineChart_AWT("Topology","Nodes interconnect. Cycles = 10");
                                chart3.pack( );
                                RefineryUtilities.centerFrameOnScreen( chart3 );
                                chart3.setVisible( true );
                                try{
                                        fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".txt";
                                        PrintWriter writer = new PrintWriter(fileName, "UTF-8");
                                        for(int m=1;m<=N;m++){
                                                writer.print("Node " + String.valueOf(m) + " neighbors =
");
                                                for(int n=0;n<k;n++){
                                                        writer.print(" " +
String.valueOf(nodes[m].neighbors[n]));
                                                }
                                        writer.println("");
                                        }
                                                fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".jpg";
                                        ChartUtilities.saveChartAsJPEG(new File(fileName),
chart3.xylineChart, 700, 270);
                                        writer.close();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                        }

                                } else if(j == 14){
                                        XYLineChart_AWT chart4 = new
XYLineChart_AWT("Topology","Nodes interconnect. Cycles = 15");
                                chart4.pack( );
                                RefineryUtilities.centerFrameOnScreen( chart4 );
                                chart4.setVisible( true );
                                try{
                                        fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".txt";
```

```
                                PrintWriter writer = new PrintWriter(fileName, "UTF-8");
                                for(int m=1;m<=N;m++){
                                        writer.print("Node " + String.valueOf(m) + " neighbors =
");
                                        for(int n=0;n<k;n++){
                                                writer.print(" " +
String.valueOf(nodes[m].neighbors[n]));
                                        }
                                writer.println("");
                                }
                                        fileName = topology + "_N" + String.valueOf(N) + "_k" +
String.valueOf(k) + "_"+ String.valueOf(j+1) + ".jpg";
                                ChartUtilities.saveChartAsJPEG(new File(fileName),
chart4.xylineChart, 700, 270);
                                writer.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                                }

                        }


                }
                System.out.println("The aggregate distance of all nodes for all cycle = {");
                for(int h=0; h<cycles; h++)
                        System.out.print(", " + dis[h]);
                System.out.println(" }");

                LineChart_AWT chart = new LineChart_AWT(
                   "Distance vs Cycles" ,
                "Sum of distances vs cycles");

                chart.pack( );
                RefineryUtilities.centerFrameOnScreen( chart );
                chart.setVisible( true );


        }
//Function to randomly select a node from its neighbor list
        public static int selectPeer(int i){
                int neighbor_id;
                int neid_index;
                Random random = new Random();
                neid_index = random.nextInt(k);
                neighbor_id = nodes[i].neighbors[neid_index];
                return neighbor_id;
```

```
	}

//Function to update the nodes neighbor list with the k nearest elements. The final nieghbor list consists of k nearest elements
//from its own neighbor list and the received neighbor list combined.
	public static void updateNL(int id){
		Set<Integer> mergedSet = new HashSet<>();
	List<Integer> aList = new ArrayList<Integer>();
	int n[] = new int[k];
	int j = 0;
	int u=0;
		for (int index = 0; index < nodes[id].neighbors.length; index++)
		{
		aList.add(nodes[id].neighbors[index]);
		}
		if(nodes[id].received_list.length != 0){
		List<Integer> bList = new ArrayList<Integer>();
			for (int index = 0; index < nodes[id].received_list.length; index++)
			{
			bList.add(nodes[id].received_list[index]);
			}
			mergedSet.addAll(bList);
		}
		mergedSet.addAll(aList);
		aList.clear();
		n =	distanceFunc(id, mergedSet);
		nodes[id].neighbors = n.clone();
//		for(j=0; j<k; j++){
//			nodes[id].neighbors[j] = n[];
//		}
//		Iterator<Integer> it = mergedSet.iterator();
//	while(it.hasNext()){
//	System.out.println(it.next());
//	}
	}

//Distance function
//Description : Selects the k nearest elements from the mergedSet based on the distance between the nodes.
//Parameters : id1 - Node id.
//			mergedSet - Combined set of id1 neighborlist and id1 received list.
//Returns : id1's neighbor list
	public static int[] distanceFunc(int id1, Set<Integer> mergedSet){
	List<Integer> newNeighborList = new ArrayList<Integer>();
		int id2;
		double d[] = new double[mergedSet.size()];
```

```
                int sortedArray[] = new int[mergedSet.size()];
                int i = 0;
                int u =0;
                int neighborArray[] = new int[k];
                Iterator<Integer> it = mergedSet.iterator();
//              System.out.print("mergedSet = ");
        while(it.hasNext()){
//      System.out.println(it.next());
                id2 = it.next();
                d[i] = distance(id1, id2);
                i++;
//              System.out.print(" " + id2);
        }
//      System.out.println("");
        sortedArray = sort(d, mergedSet);
//              System.out.print("Sorted based on distance merged Array = ");
//      for(i=0; i<sortedArray.length; i++){
//              System.out.print(" " + sortedArray[i]);
//      }
//      System.out.println("");
        for(i=0; i<k; i++){
                while(sortedArray[u] == id1)
                                u++;
//                      if(id1 == nodes[N].node_id)
//                              if(sortedArray[u] == nodes[N-1].node_id)
//                                              neighborArray[i++] = nodes[1].node_id;
                neighborArray[i] = sortedArray[u++];
        }
        return neighborArray;
        }
//Distance
//Description: Returns the distance between 2 nodes based on its x y coordinates.
//For the Nth node, it returns a minimum distance for 1st and N-1th node and maximum
distance for all other nodes.
//Parameters: id1 - Node id
//                      id2 - Node id
//Returns : Distance
        public static double distance(int id1, int id2){
                double d;
                if(id2 == nodes[N].node_id && topology == 'B'){
//              if(id1 == id2){
                        if(id1 == nodes[1].node_id || id1 == nodes[N-1].node_id){
//
System.out.println("^^^^^^^^^^^^^^^^^Here^^^^^^^^^^^^^^^^^");
                                return 0.000001;
                        } else{
```

```
//                                      System.out.println("--------------------Here----------------");
                                        return 10000;
                              }
                    }
                    else if(id1 == nodes[N].node_id && topology == 'B'){
//                  if(id1 == id2){
                              if(id2 == nodes[1].node_id || id2 == nodes[N-1].node_id){
//
System.out.println("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^Here^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^");
//                                      System.exit(0);
                                        return 0.000001;
                              } else{
//                                      System.out.println("--------------------Here----------------");
                                        return 10000;
                              }
                    }

                    else{
//
System.out.println("$$$$$$$$$$$$$$$$$$$$$$Here$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$");
                              d = Math.hypot(nodes[id1].x_co - nodes[id2].x_co, nodes[id1].y_co -
nodes[id2].y_co);
                              return d;
                    }
          }

//Sort
//Discription : Given an array of distances and a mergedSet. Sort function will sort the
mergedSet based on the
//distance array arranged in an ascending order.
//Parameters : arr[] - The distance array.
//                                mergedSet - The set that needs to be sorted based on distance
array
//Returns : Sorted mergedSet
   public static int[] sort(double arr[], Set<Integer> mergedSet)
   {
      int n = arr.length;
                    Iterator<Integer> it = mergedSet.iterator();
                    int sortedArray[] = new int[mergedSet.size()];
                    for(int g=0; g<mergedSet.size();g++){
                              sortedArray[g] = it.next();
                    }
                    Iterator<Integer> it1 = mergedSet.iterator();
      // One by one move boundary of unsorted subarray
      for (int i = 0; i < n-1; i++)
```

```
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            // element
            double temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
            int temp1 = sortedArray[min_idx];
            sortedArray[min_idx] = sortedArray[i];
            sortedArray[i] = temp1;

        }

        return sortedArray;
    }


    public static XYDataset createDataset( ) {
        final XYSeries firefox = new XYSeries( "Nodes interconnect", false, true );
        final XYSeries chrome = new XYSeries( "Nodes interconnect", false, true );
        int ne;
        for(int i =1; i<=N; i++){
            for(int j =0; j<k; j++){
                ne = nodes[i].neighbors[j];
                if(i==N && topology == 'B'){
                    System.out.println("done");
//                    System.exit(0);
                    if (j==0){
                    chrome.add( nodes[i].x_co , nodes[i].y_co);
                    chrome.add( nodes[ne].x_co , nodes[ne].y_co);
                    } else if(j==1){
                    chrome.add( nodes[i].x_co , nodes[i].y_co);
                    chrome.add( nodes[ne].x_co , nodes[ne].y_co);
                    break;
                    }
                } else {
                    firefox.add( nodes[i].x_co , nodes[i].y_co);
                    firefox.add( nodes[ne].x_co , nodes[ne].y_co);
                }
```

```
        }
      }

      final XYSeriesCollection dataset = new XYSeriesCollection( );
      dataset.addSeries( firefox );
      dataset.addSeries( chrome );
//      dataset.addSeries( iexplorer );
      return dataset;
   }

}

//Class : Node
class Node{
            int node_id;
            double x_co;
            double y_co;
            double theta;
//          int k = 30;
            int neighbors[];
            int received_list[];

//          Map<Integer, int[]> received_list = new HashMap<Integer, int[]>();
           int N;
           public  int getNodeId(){
                    return node_id;
           }
           public  double getXCo(){
                    return x_co;
           }

           public  double getYCo(){
                    return y_co;
           }

           public  double getTheta(){
                    return theta;
           }

           public  int[] getNeighbors(){
                    return neighbors;
           }

           public  int[] getReceivedList(){
                    return received_list;
           }
```

```
	public void setNodeId(int node_id){
		this.node_id = node_id;
	}

	public void setXCo(int x_co){
		this.x_co = x_co;
	}

	public void setYCo(int y_co){
		this.y_co = y_co;
	}

	public void setTheta(double theta){
		this.theta = theta;
	}

	public void setNeighbors(int neighbors[]){
		this.neighbors = neighbors;
	}


}
```