CS6385 Algorithmic Aspects of Telecommunication Networks

Project 1

An Application to Network Design

Student : Samuel Chen

Instructor : Andras Farago

Index

1. Introduction

Design a basic network topology by given the unit cost and traffic demand between every pairs of nodes. Implement the shortest path based fast solution method to find the optimal path that have the minimum cost.

Input:

- Number of Nodes
- Traffic demand values (b_{ij})
- Unit costs for potential links (a_{ij})

Output:

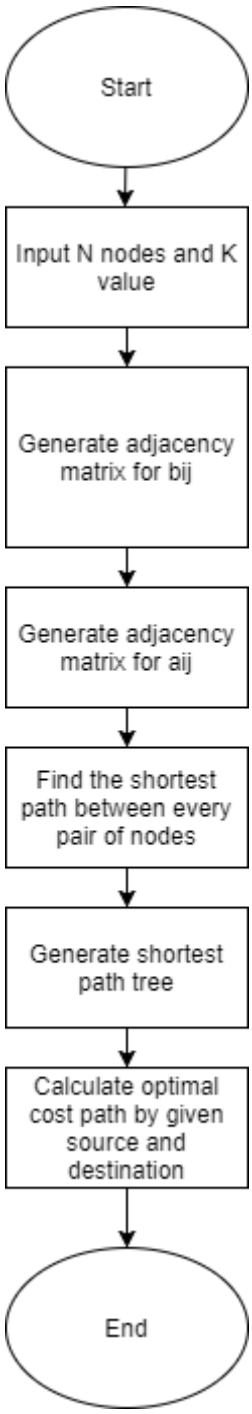
A network topology with minimum cost between every pair of node based on different K, where K = 3, 4, …, 15.

2. Implementation

- Unit Costs (a_{ij}) : For every node i, pick k nodes randomly set it as 1, all different from each other and I, for the other node set it as 250. This will ensure for every node I, there will be k low cost links going out of the node.
- Traffic demands (b_{ij}) : For every pairs b_{ij}, set its value randomly in the range [0; 1; 2; 3].
- Shortest path for the network topology : Implement Dijkstra algorithm for every node as source node to find its shortest path to the other nodes. Finally, the shortest path tree will be constructed.
- Shortest path based fast solution : Once the shortest path tree being constructed, we can obtain the minimum cost k -> I path:

$$Z_{opt} = \sum_{k,l}\left(b_{kl} \sum_{(i,j)\in E_{kl}} a_{ij}\right)$$

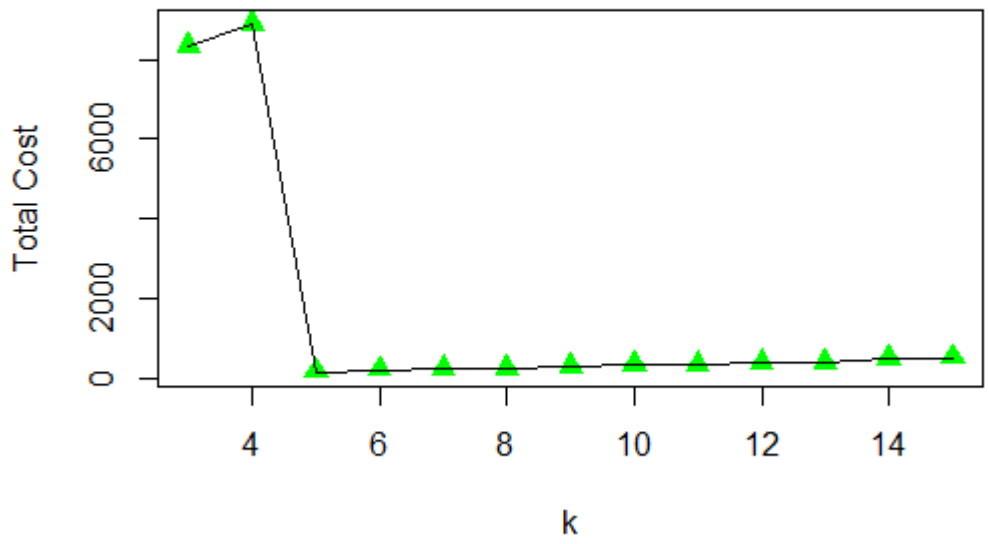
3. Flow Chart


Start
Input N nodes and K value
Generate adjacency matrix for bij
Generate adjacency matrix for aij
Find the shortest path between every pair of nodes
Generate shortest path tree
Calculate optimal cost path by given source and destination
End

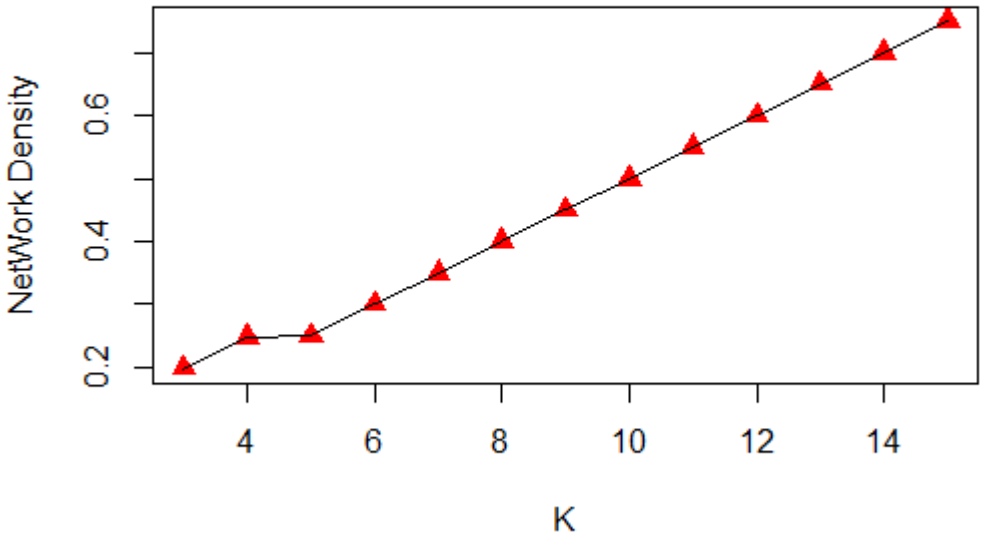
4. Graph

Used Path	Network Density	Total Cost
83	0.19761905	8338
104	0.24761905	8876
105	0.25	147
126	0.3	199
147	0.35	241
168	0.4	237
189	0.45	282
210	0.5	325
231	0.55	324
252	0.6	383
273	0.65	397
294	0.7	473
315	0.75	500

Total Cost



Network Density



5. Network Topology

K=3

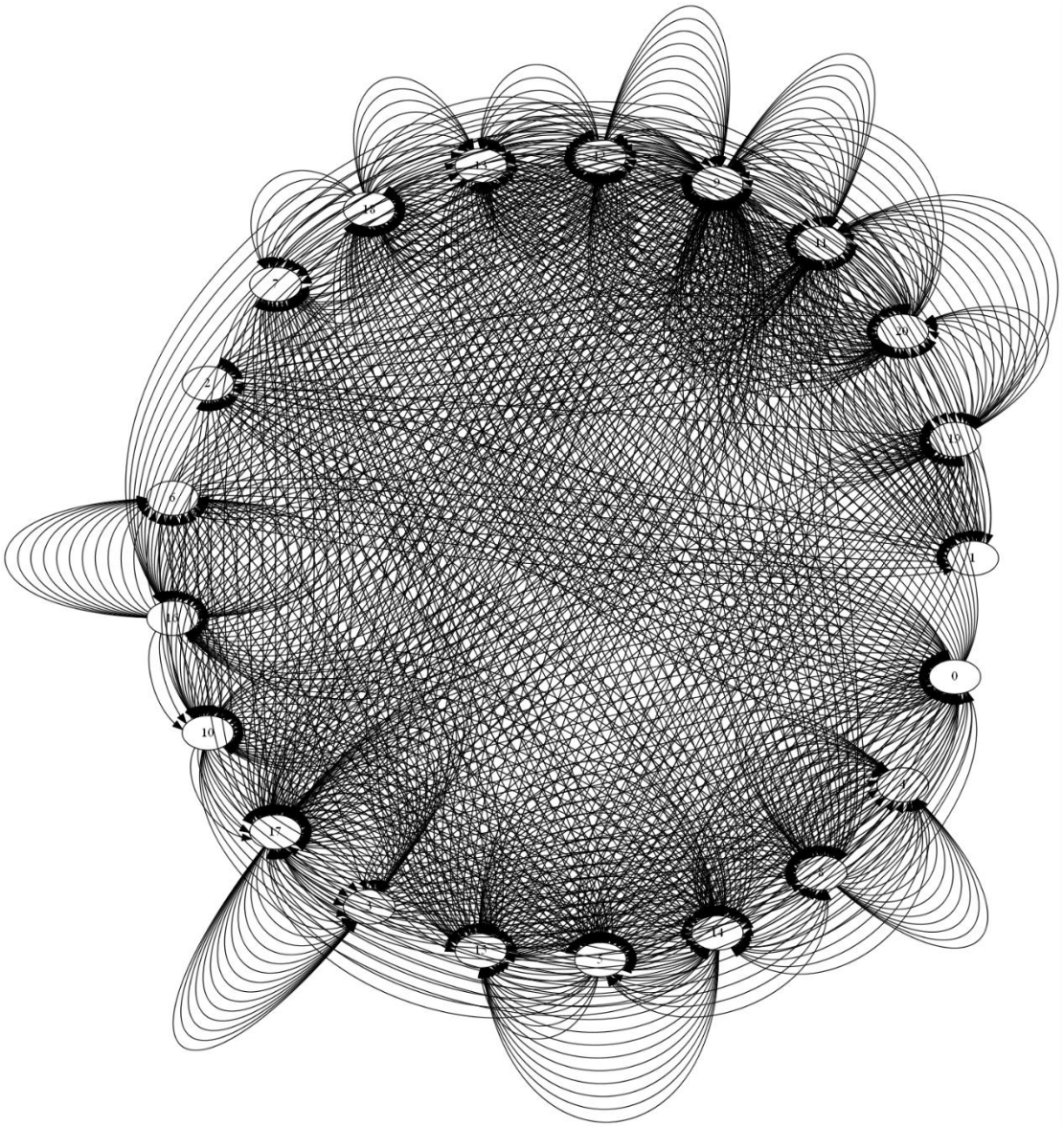


Figure 1 : N = 21, k= 3. This graph shows every node as source node to the others node shortest path.

K=8

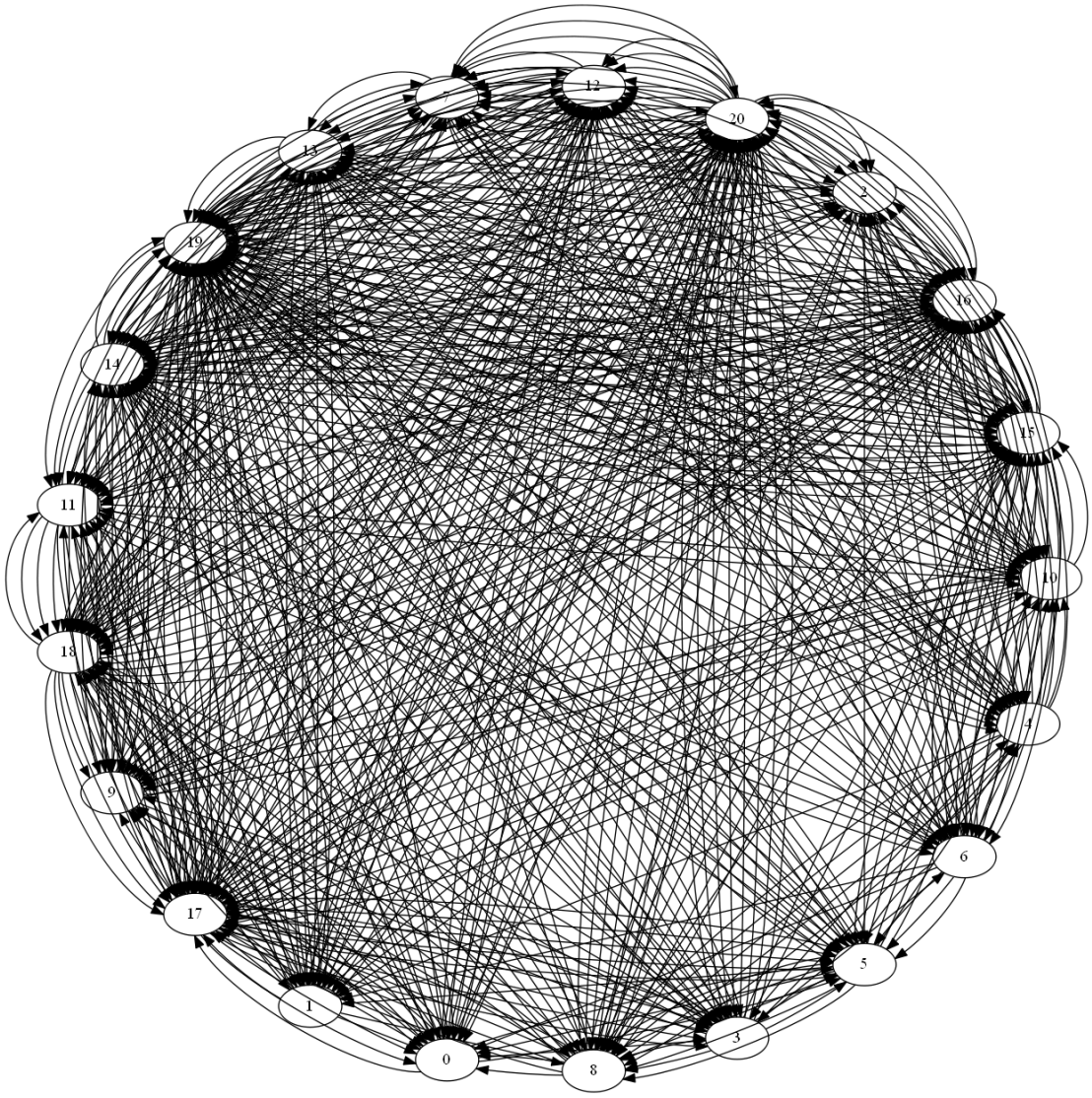


Figure 2 : N = 21, k= 8. This graph shows every node as source node to the others node shortest path.

K=15

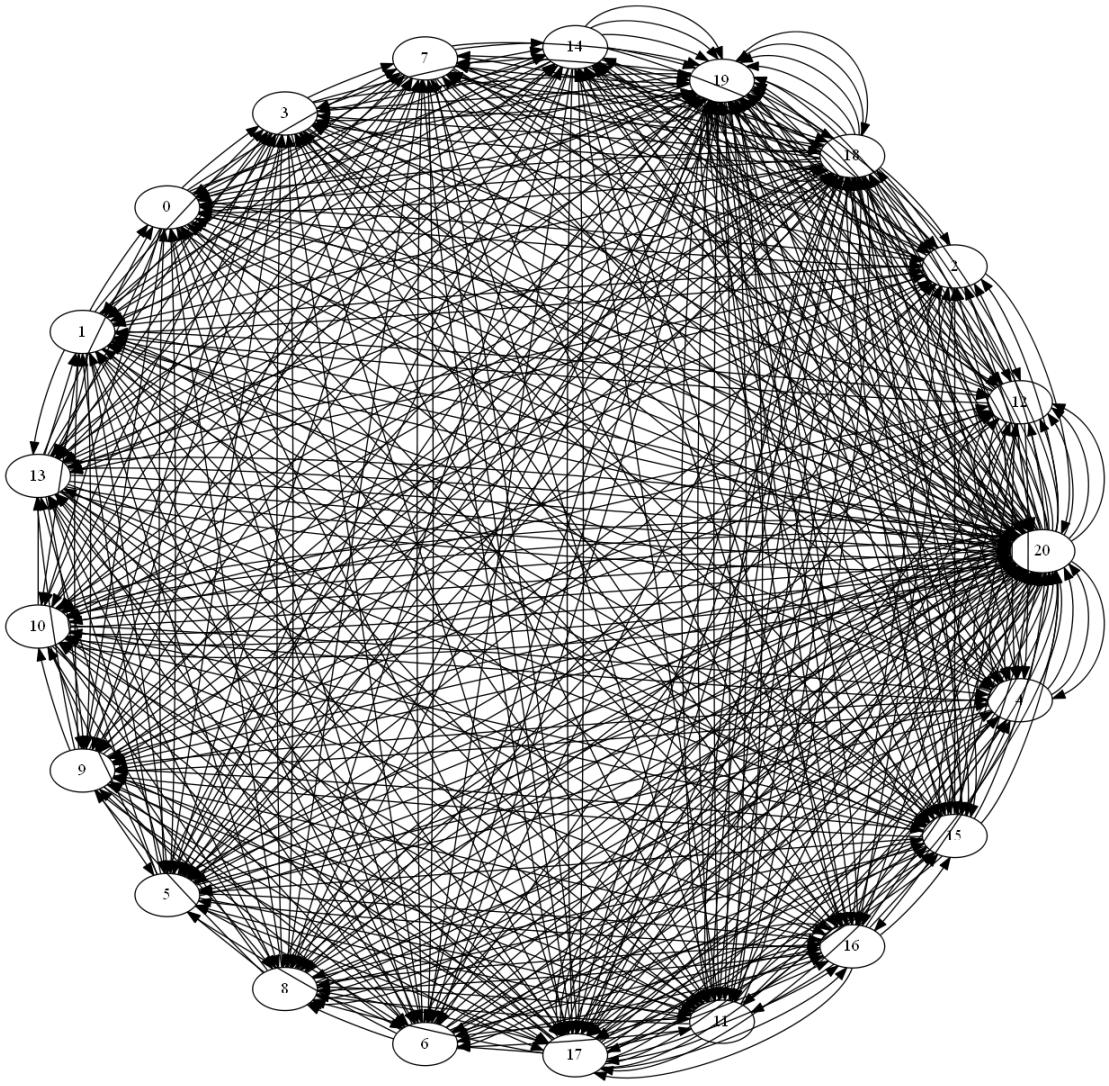


Figure 2 : N = 21, k= 15. This graph shows every node as source node to the others node shortest path.

Description : We can observed that when k is small, most of the path will pass through some particular node. This reflect our network design principle that when k is very small, every node will have very little outgoing path that is small (In our case, there is only k outgoing path is 1, the other will be 250). The Dijkstra Algorithm will try to avoid the high cost path, so path will gathering around particular node.
When the k is large, the topology will come to be having roughly the same outgoing path for every node.

The three graphs above show every node as source node to other node shortest path, to show whether another type of topology will follow the conclusion we get above, I make another three graph that have half of the topology's node as source node, which is 0, 2, … , 9.

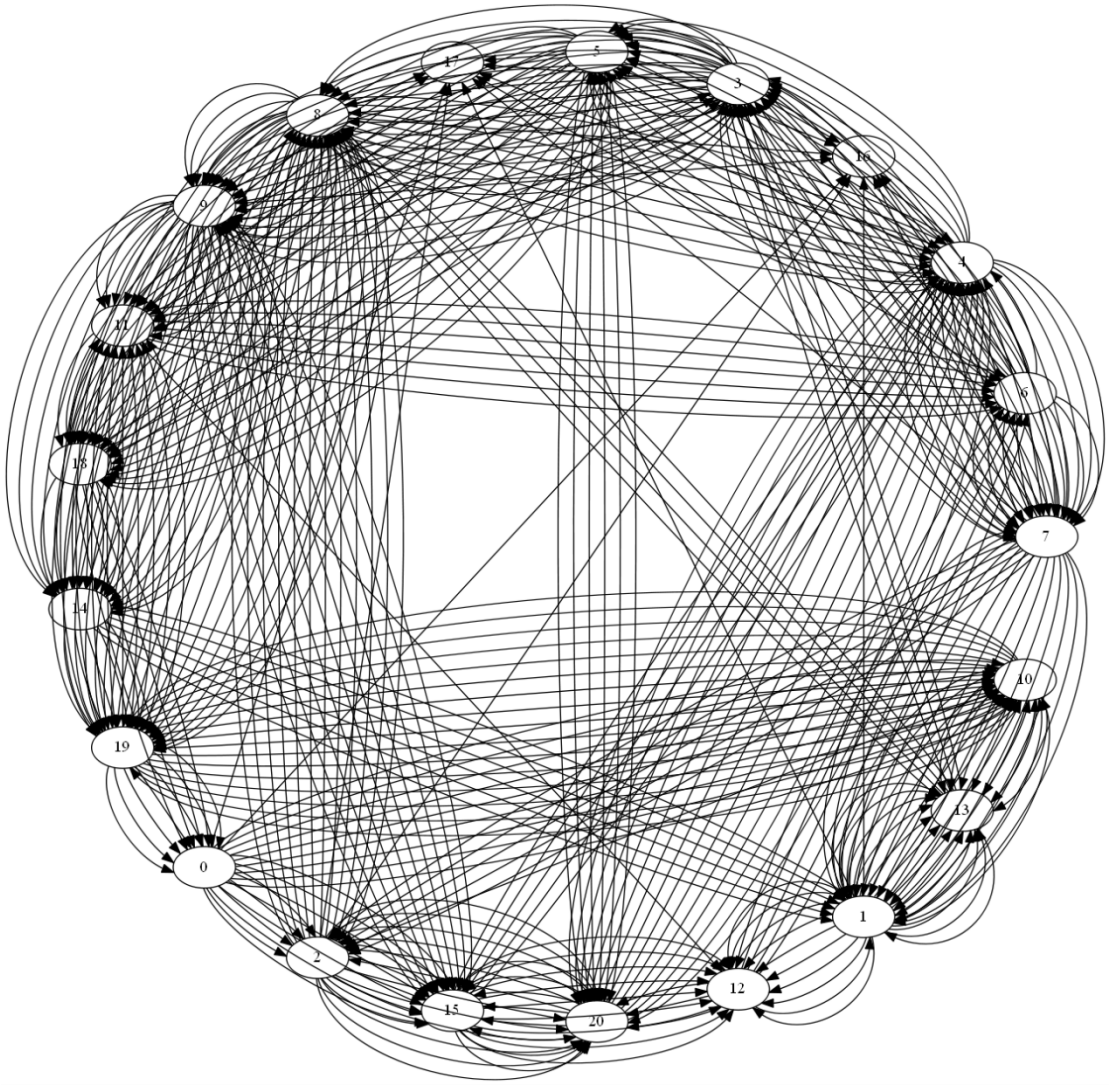


Figure 4 : N = 21, k= 3. This graph shows half of the topology's node as source node to the others node shortest path.

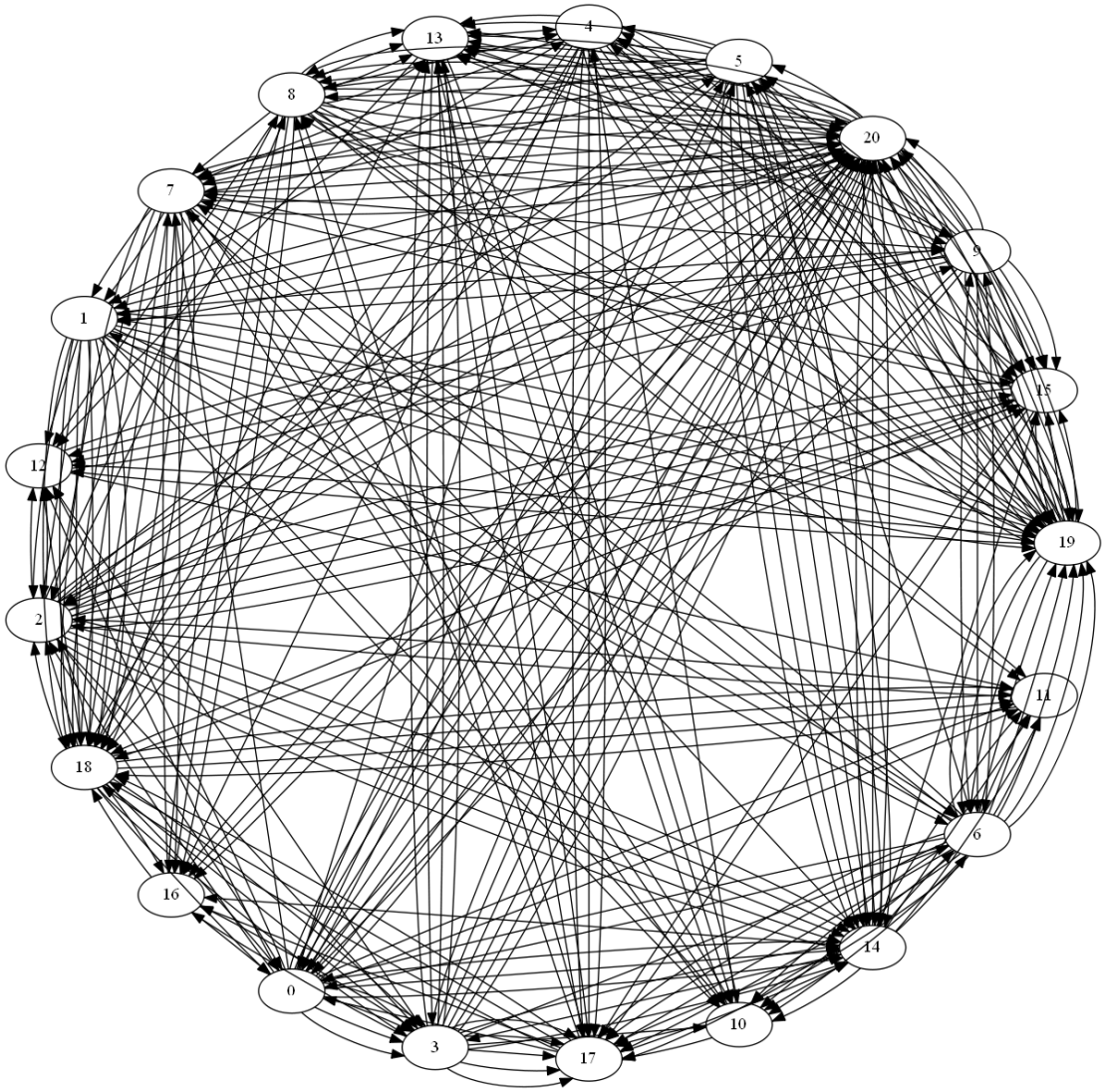


Figure 5 : N = 21, k= 8. This graph shows half of the topology's node as source node to the others node shortest path.

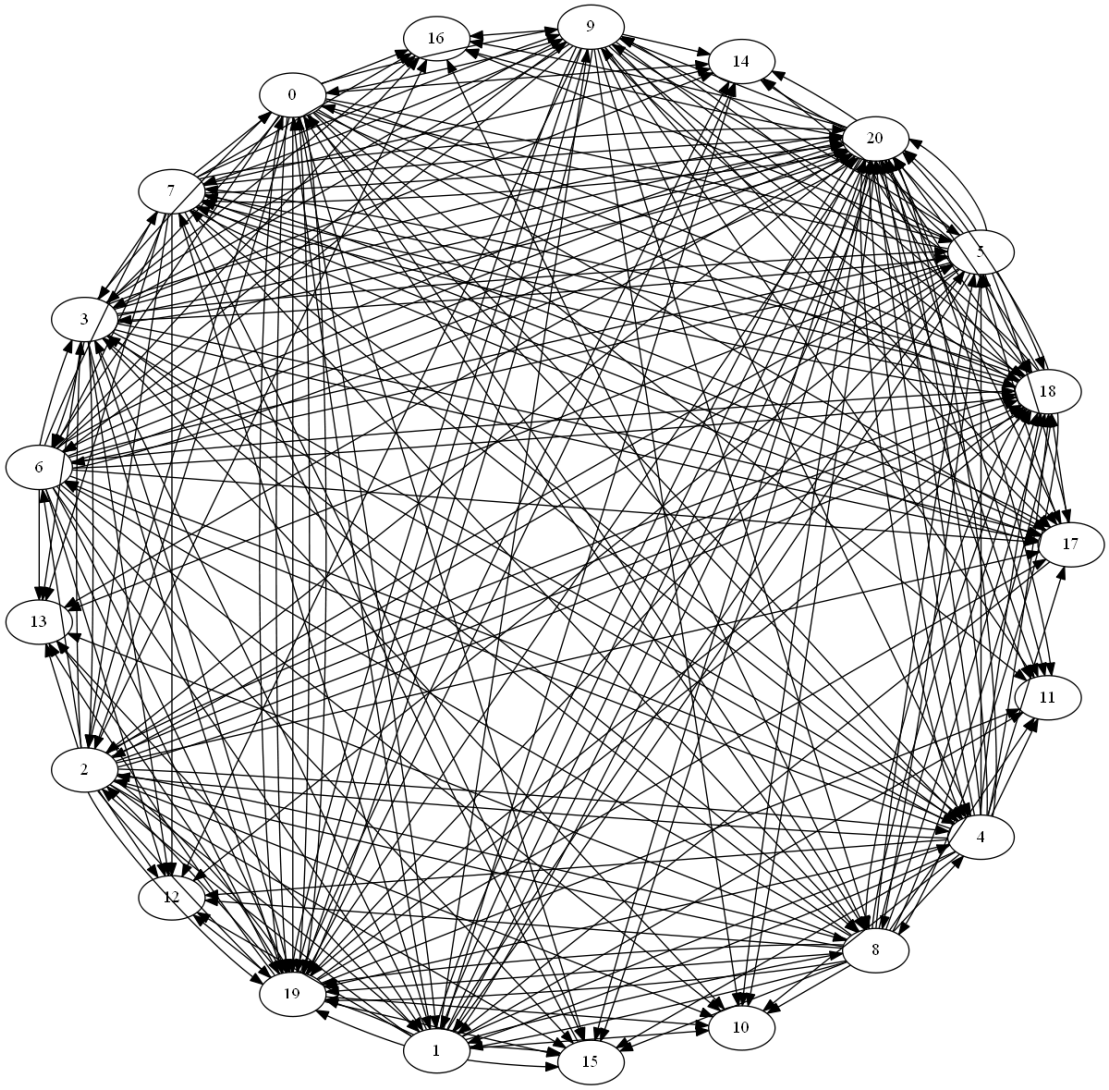


Figure 6 : N = 21, k= 15. This graph shows half of the topology's node as source node to the others node shortest path.

6. Source Code

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class Graph {

	public int cost[][];
	public int traffic[][];
	public int totalcost = 0;
	public int totalpath = 0;
	public String shortestpath[][];
	public int used[][];

	public int number;
	public int k;

	public Graph(int number, int k) {
		this.number = number;
		this.k = k;
	}

	//Generate aij of undirected graph
	public void GenerateCost() {

		Random random = new Random();

		for(int i = 0; i < number; i++)
			for(int j = 0; j < number; j++) {
				int temp = random.nextInt(4);
```

```
                traffic[i][j] = temp;
            }

    }

    //Generate bij of undirected graph
    public void GenerateTraffic() {

        int pick[] = new int[number];

        for(int i = 0; i < number; i++)
            for(int j = 0; j < number; j++)
                cost[i][j] = 250;

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

            List list = new ArrayList();

            int count = 0;
            for(int j = 0; j < number; j ++)
                list.add(j);

            while(count < k) {
                int random = (int) (Math.random() *list.size());
                int j = (int)list.get(random);
                if(i != j) {
                    cost[i][j] = 1;
                    count++;
                    list.remove(random);
                }
            }

        }

        /*
        //For every node i, pick k random number and assign the edge as 1
        for(int i = 0; i < number; i++) {
```

```
            List list = new ArrayList();
            for(int j = 0; j < number; j ++)
                list.add(j);

            while(pick[i] < k && list.size() > 0) {

                int random = (int) (Math.random() *list.size());
                int j = (int)list.get(random);

                if(cost[i][j] == 250 && cost[j][i] == 250 && i != j && pick[j] < k) {
                    cost[i][j] = 1;
                    cost[j][i] = 1;
                    pick[j]++;
                    pick[i]++;
                }
                list.remove(random);
            }
        }
        */
}

public void dijkstra(int graph[][], int source) {

        int dist[] = new int[number];
        int path[] = new int[number];
        Boolean sptSet[] = new Boolean[number];

        // Initialize all distances as INFINITE and stpSet[] as false
        for (int i = 0; i < number; i++)
        {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
            path[i] = -1;
        }

        // Distance of source vertex from itself is always 0
        dist[source] = 0;
```

```
        // Find shortest path for all vertices
        for (int count = 0; count < number-1; count++){

            int u = minDistance(dist, sptSet);
            // Mark the picked vertex as processed
            sptSet[u] = true;

            for (int v = 0; v < number; v++) {
                if (!sptSet[v] && graph[u][v]!=0 && dist[u] != Integer.MAX_VALUE
&& dist[u]+graph[u][v] < dist[v]) {
                    path[v] = u;
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }
        traverse(dist, path, source);
    }

    public int minDistance(int dist[], Boolean sptSet[]) {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index=-1;

        for (int v = 0; v < number; v++)
            if (sptSet[v] == false && dist[v] <= min)
            {
                min = dist[v];
                min_index = v;
            }
        return min_index;
    }

    // A utility function to print the constructed distance array
    public void traverse(int dist[], int path[], int source) {

        for (int i = 0; i < number; i++) {
            path(source, i, path, i);
        }
    }
```

```
	//Record every node along the shortest path
	public void path(int source, int destination, int path[], int i) {

		if(path[i] == -1)
			return;
		path(source, destination, path, path[i]);
		shortestpath[source][destination] = shortestpath[source][destination] +" "+
i;
	}
	//Initialize different aij and bij depend on different k
	public void initialize() {

		traffic = new int[number][number];
		cost = new int[number][number];
		shortestpath = new String[number][number];
		used = new int[number][number];
		for(int i = 0; i < number; i++) {
			for(int j = 0; j < number; j++) {
				cost[i][j] = 0;
				traffic[i][j] = 0;
				shortestpath[i][j] = String.valueOf(i);
				used[i][j] = 0;
			}
		}
		totalcost = 0;
	}

	public void WriteToFile(int source) throws IOException {

		String filename = "File"+k+".dot";
		File file = new File(filename);
		BufferedWriter writer = new BufferedWriter(new FileWriter(filename,
true));
		if(source == 0) {
			writer.write("digraph demo"+k+"{");
			writer.newLine();
		}
```

```
        for(int i= 0; i < number; i++) {
            String temp = "";
            temp = shortestpath[source][i].replaceAll(" ", "->");
            writer.write(temp+";");
            writer.newLine();
        }

        if(source == number-1) {
            writer.write("}");
        }
        writer.close();
    }

    public void calculate() {
        for (int i = 0; i < number; i++) {
            for(int j = 0; j < number; j++) {
                String[] temp = shortestpath[i][j].split(" ");
                int k = 0;
                while(k < temp.length-1) {
                    int current = Integer.parseInt(temp[k]);
                    int next = Integer.parseInt(temp[k+1]);
                    used[current][next] = 1;
                    k++;
                }
            }
        }

        for (int i = 0; i < number; i++) {
            for(int j = 0; j < number; j++) {
                if(used[i][j] == 1) {
                    totalcost += traffic[i][j]*cost[i][j];
                    totalpath++;
                }
            }
        }
    }
```

```
    public void NetworkAnalysis() throws IOException {

        float result;
        File file = new File("Density.txt");
        BufferedWriter writer = new BufferedWriter(new FileWriter(file, true));
        result = (float)totalpath /(float)420;
        writer.write(String.valueOf(result)+" ");
        writer.write(String.valueOf(totalpath)+" ");
        writer.write(String.valueOf(totalcost));
        writer.newLine();
        writer.close();
    }


    public static void main(String args[]) throws IOException {

        Graph[] a = new Graph[13];
        for(int count = 0; count < 13; count++) {
            int k = count+3;
            a[count] = new Graph(21, k);
            a[count].initialize();
            a[count].GenerateCost();
            a[count].GenerateTraffic();
            for(int i = 0; i < 21; i++) {
                a[count].dijkstra(a[count].cost, i);
                a[count].WriteToFile(i);
            }
            a[count].calculate();
            a[count].NetworkAnalysis();

            System.out.println(a[count].totalcost);
        }

    }
}
```

7. Reference

1. http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
2. http://www.graphviz.org/pdf/dotguide.pdf