PROJECT

An Application to Network Design

Work done

by

Arun Prakash Themothy Prabu Vincnet

NET ID : AXT161330

SPRING'17 CS/TE6385

ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKING

CONTENTS

Chapter 1

Introduction

Network Planning and Design

Network planning and design is an iterative process, encompassing topological design, network-synthesis, and network-realization, and is aimed at ensuring that a new telecommunications network or service meets the needs of the subscriber and operator. The process can be tailored according to each new network or service.

A Network Planning Methodology :

A traditional network planning methodology in the context of business decisions involves five layers of planning, namely:

- need assessment and resource assessment
- short-term network planning
- IT resource sourcing
- long-term and medium-term network planning
- operations and maintenance.[1]

The network planning process begins with the acquisition of external information. This includes:

- forecasts of how the new network/service will operate;
- the economic information concerning costs; and
- the technical details of the network's capabilities

Network planning process involves three main steps:

- **Topological design**: This stage involves determining where to place the components and how to connect them. The (topological) optimisation methods that can be used in this stage come from an area of mathematics called Graph Theory. These methods involve determining the costs of transmission and the cost of switching, and thereby determining the optimum connection matrix and location of switches and concentrators.[1]
- **Network-synthesis**: This stage involves determining the size of the components used, subject to performance criteria such as the Grade of Service (GOS). The method used is known as "Nonlinear Optimisation", and involves determining the topology, required GoS, cost of transmission, etc., and using this information to calculate a routing plan, and the size of the components.[1]
- **Network realization**: This stage involves determining how to meet capacity requirements, and ensure reliability within the network. The method used is known as "Multicommodity Flow Optimisation", and involves determining all information relating to demand, costs and reliability, and then using this information to calculate an actual physical circuit plan.[1]

These steps are performed iteratively in parallel with one another

Chapter 2

Project Description

The objective of the project is to generate a network topology, with capacities assigned to the links, according to the studied model, using the shortest path based fast solution method i.e Dijkstra's shortest path algorithm. The program also computes the total cost of the designed network .

The project is an application to network design. In this project, the number of nodes is given as the input (N) along with the traffic demand values (b_{ij}) between pairs of nodes, and the unit cost values for the potential links (a_{ij}). . The agenda of this project is to design a network given the number of nodes and cost of the link between nodes.

The agenda of the project is to experiment with the network that has been designed randomly and calculate the optimal cost and network density and analyze how they vary with respect to random connectivity.

Once the network is formed, the shortest path algorithm is used to get the path from the source to destination. The Dijkstra's algorithm is used to calculate the shortest path from the source to destination. Once this path has been found the cost of the total network is found.

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

Where,

Z_{opt} = Optimal cost of the network.

b_{kl} = Traffic Demand.

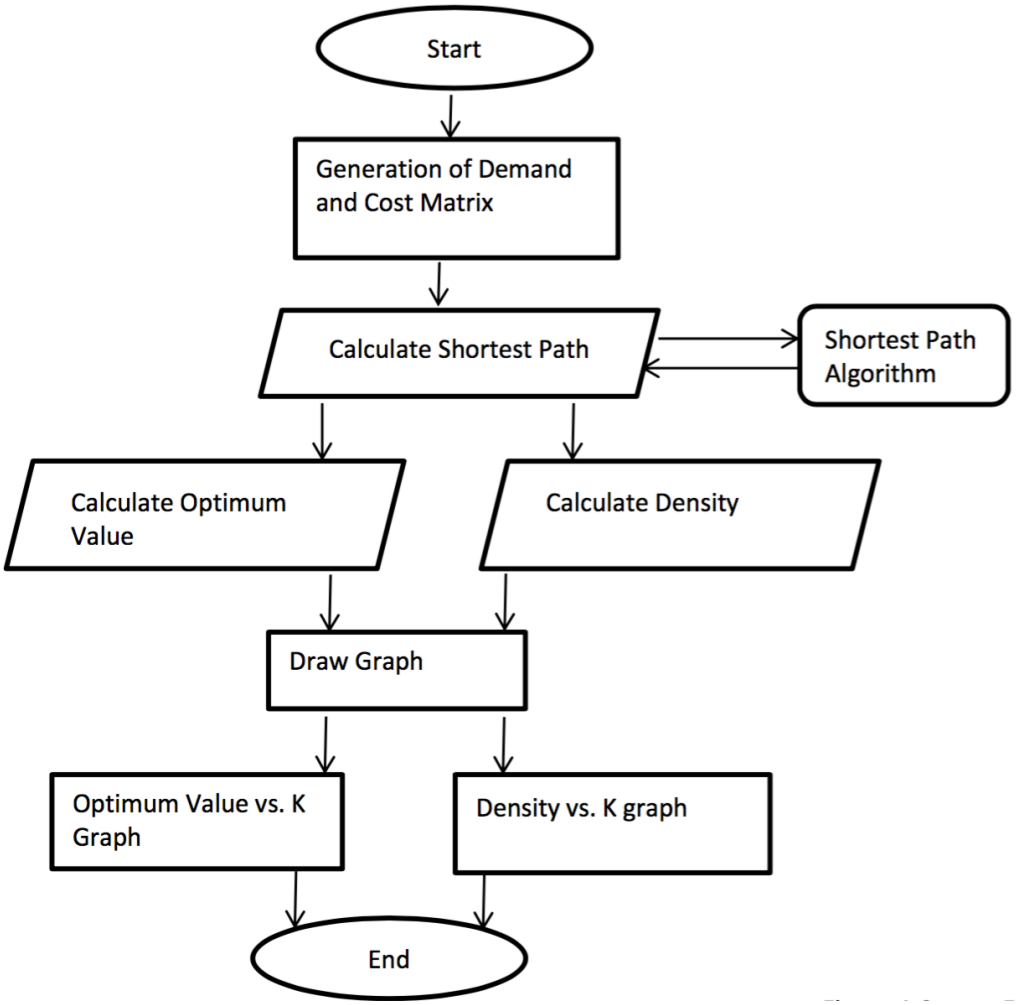
a_{ij}= Cost of the Link.

After the cost of the network is calculated the network density is calculated where the density of the network is the ratio of total number of non-zero connections to the total number of possible connections.

- The program runs for k = 3,4,5...,13. For each run new random a_{ij},b_{ij} parameters are generated independently. and for each value of k the changes are made in the cost and demand and a graph is plotted to check the variation of total cost and total network density with respect to 'k'.

Chapter 3

Flow Diagram



Chapter 4

Algorithm and Implementation:

4.1 Algorithm:

The algorithm used in this project to find the shortest path is the Djikstra's Algorithm. Once the cost matrix is generated, it is given as input to find the shortest path from one node to another node to generate the shortest path matrix.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,[2] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

4.2 Implementation:

This software is developed in JAVA, has following modules,

1. Traffic Matrix Determination
2. Cost Matrix Determination for different values of K
3. ShortestPath
4. Network Design
5. OptimumCost for different values of K
6. DenistyCalculation for different values of k
7. DrawGraph
   7.1 Cost vs K
   7.2 Density vs K

The Gephi Software tool is used to draw the Nework topology for k=3, k=8 and k=13.

Chapter 5

Results

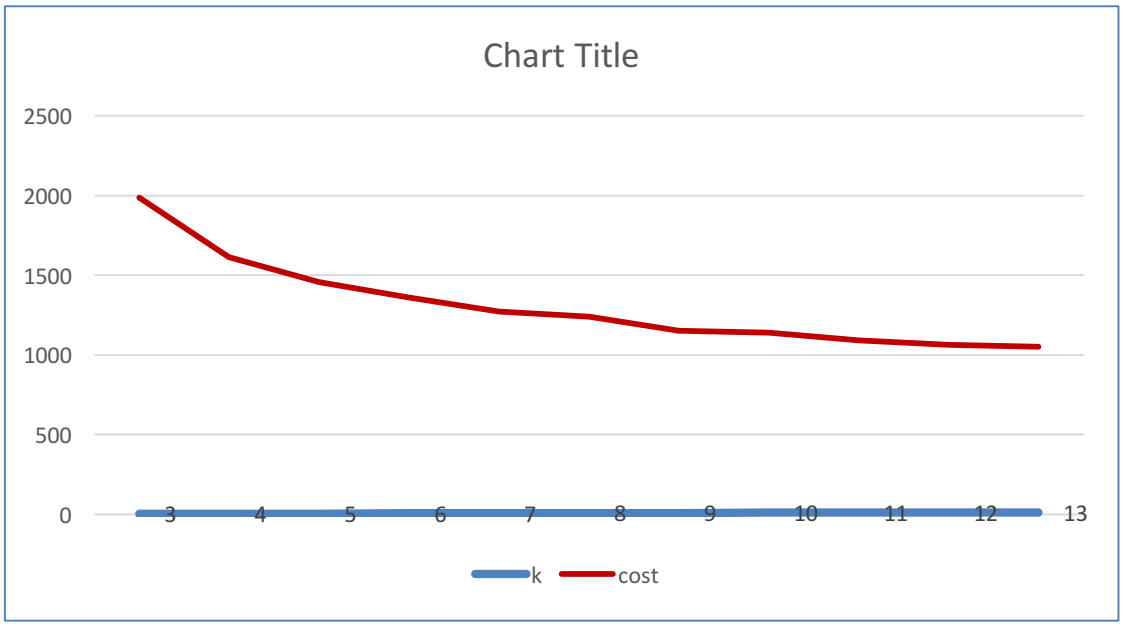
5.1 Program Output:

The software runs with given Number of nodes as input(N) . Nodes is taken as 20 , applications is executed for a range of connectivity factor ‘K’ from 3 to 13, during each run the cost matrix is generated randomly keeping the demand constant.

```
K = 3
Cost = 1860    Density = 1.0

K = 4
Cost = 6278    Density = 1.0

K = 5
Cost = 1413    Density = 1.0

K = 6
Cost = 1337    Density = 1.0

K = 7
Cost = 1282    Density = 1.0

K = 8
Cost = 1227    Density = 1.0

K = 9
Cost = 1176    Density = 1.0

K = 10
Cost = 1123    Density = 1.0

K = 11
Cost = 1143    Density = 1.0

K = 12
Cost = 1092    Density = 1.0

K = 13
Cost = 1060    Density = 1.0
```

5.2 Graphs :

A.Cost vs K

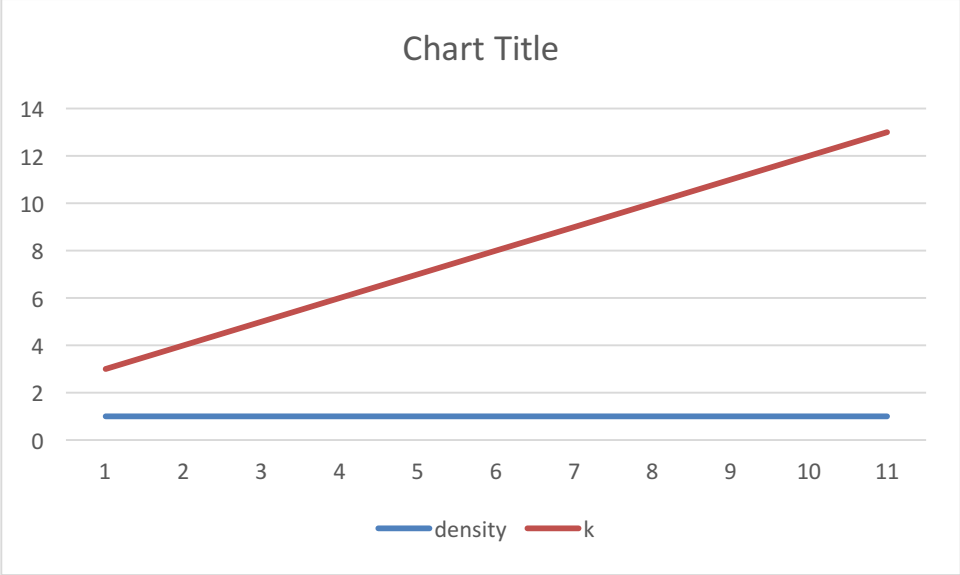


Analysis:

From the above graph we can conclude that the optimum cost is inversely proportional to K (Node connectivity factor). That's because with the increase in K each node will get many outgoing edges, which will result in new least cost shortest paths thus decreasing the overall cost of the network design.

ZOptimum Varies Inversely as K

B. Density vs K



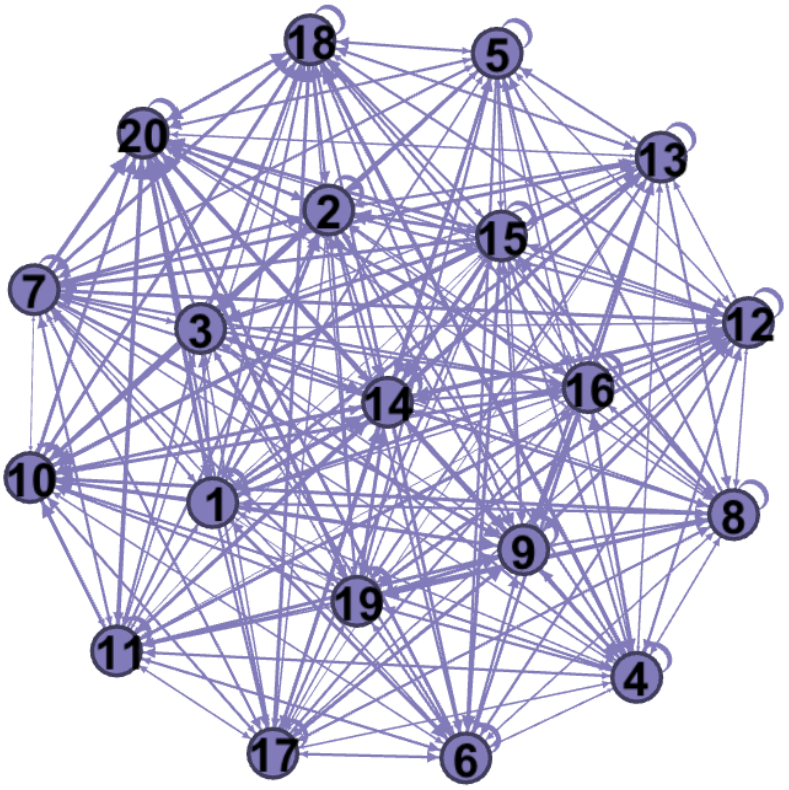
Analysis:

This graph shows impact of k on desity. As density is calculated using Zoptimal matrix in which we consider the non-zero edges. The density remains unaffected by the connectivity factor K. Thus we can see the graph as above. The K has very less impact on the value of Zoptimal and it remains a constant .

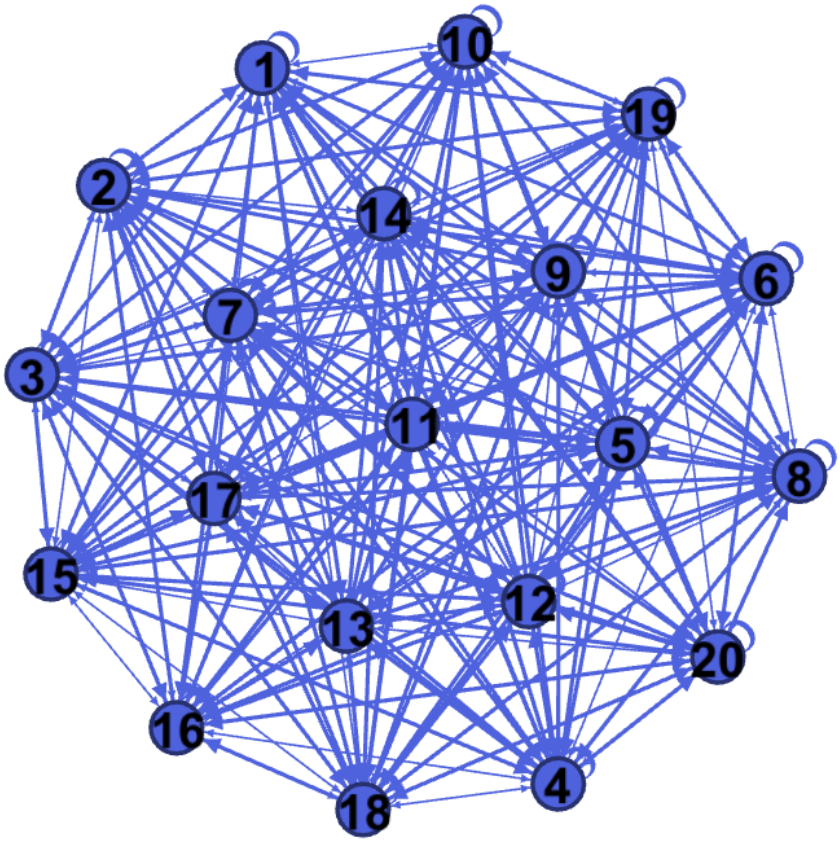
5.3 Network Topology Graph

The Gephi Software is used to draw network graphically for different values of k. Here three values of k are considered k=3 , k=8 and k=13 and the different graphical representation is as below. As the k value increases the number of outgoing edges from each node increases and it can be clearly seen in the graphical representations below.

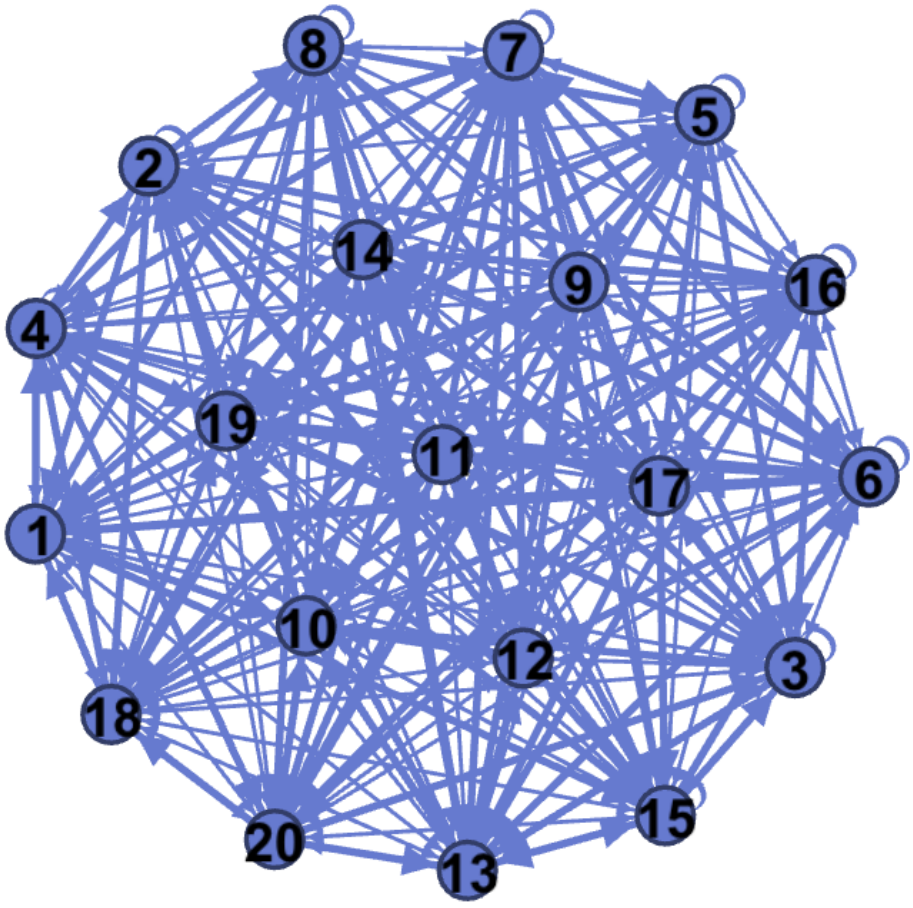
5.3.1Network Graph for k=3 :



5.3.2 Network Graph for k=8:



5.3.3 Network Graph for k=13 :



Chapter 6

References

1. Dijkstra's Shortest path algorithm . http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

2. Network Planning Theory – https://en.wikipedia.org/wiki/Network_planning_and_design

Appendices

Appendix A – Source Code

```
package Atn;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class NetworkDesign {

    /* Function to Initialize TrafficDemandMatrix*/
    public static int[][] trafficDemand(int N){
      int d[] = new int[N];
      int difference;
      int b[][] = new int[N][N]; // Declaring the traffic Demand
matrix
      d[0] = 2; // Initializing D with the UTD Id
      d[1] = 0;
      d[2] = 2;
      d[3] = 1;
      d[4] = 3;
      d[5] = 1;
      d[6] = 3;
      d[7] = 2;
      d[8] = 8;
      d[9] = 2;
      for(int j=10,i=0;j<N;++j,++i){
      d[j] = d[i];
      }
      /*Initializing the trafficDemandMatrix*/
      for(int i=0;i<N;++i)
      for(int j=0;j<N;++j){
       difference = d[i]-d[j] ;
       b[i][j] = Math.abs(difference);
      }
      return b;

    }

    /* To initialize the Cose Matrix */
    public static int[][] adjacencyMatrix(int N , int k){
     int a[][] = new int[N][N]; //Declaring Cost Matrix
     List<Integer> indices = new ArrayList<Integer>();

     for(int i=0 ;i<N ;++i)
           indices.add(i);
     int index[] = new int[k] ;
```

```
 /*Intializing Cost Matrix */
 for(int i=0;i<N ;++i){

      Collections.shuffle(indices); // Randomizing the Indices

      /* Generating k Random Indices */
      for(int m=0;m<k;++m){
           if(indices.get(m) == i)
                index[m] = indices.get(N-k);
           else
                index[m] = indices.get(m);
      }

      for(int j=0; j<N;++j){
          if (i==j)
           a[i][j] = 0; //Initializing cost to 0 if i==j
          else
            for(int m=0 ;m<k ;++m){
           if(index[m] == j){
                a[i][j] = 1;
               break ;
           }
           else
                a[i][j] = 200;
            }

      }
 }
 return a;
}

static final int V=20;

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

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

    return min_index;
}


// Funtion that implements Dijkstra's single source shortest path
```

```
    // algorithm for a graph represented using adjacency matrix
    // representation
    public static int[] dijkstra(int graph[][], int src)
    {
        int dist[] = new int[V]; // The output array. dist[i] will
hold
                                 // the shortest distance from src to
i

        // sptSet[i] will true if vertex i is included in shortest
        // path tree or shortest distance from src to i is finalized
        Boolean sptSet[] = new Boolean[V];

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

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

        // Find shortest path for all vertices
        for (int count = 0; count < V-1; count++)
        {
            // Pick the minimum distance vertex from the set of
vertices
            // not yet processed. u is always equal to src in first
            // iteration.
            int u = minDistance(dist, sptSet);

            // Mark the picked vertex as processed
            sptSet[u] = true;

            // Update dist value of the adjacent vertices of the
            // picked vertex.
            for (int v = 0; v < V; v++)

                // Update dist[v] only if is not in sptSet, there is
an
                // edge from u to v, and total weight of path from src
to
                // v through u is smaller than current value of
dist[v]
                if (!sptSet[v] && graph[u][v]!=0 &&
                        dist[u] != Integer.MAX_VALUE &&
                        dist[u]+graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
        }
```

```
        return dist;
    }

    public static int costCalculation(int b[][], int g[][] ,int N){
     int cost= 0;
     for(int i=0; i<N; ++i)
          for(int j=0; j<N ;++j){
               cost = cost+( b[i][j]*g[i][j]) ;
          }
     return cost;
    }

    public static float densityCalculation(int[][] cost, int N){
     float density;
     float possibleConnection = N*(N-1) ;
     int actualConnection = 0;
     for(int i=0; i<N; ++i){
          for(int j=0; j<N ;++j){
               if(cost[i][j] != 0)
                    actualConnection++;
          }
     }
     density = (float)(actualConnection/possibleConnection) ;
     return density;
    }
    public static void main(String[] args) {


     int N = 20; //Number of Nodes
     for(int k=3; k<=13; ++k){
     int cost = 0;
     int B[][] = trafficDemand(N);
     int A[][] = adjacencyMatrix(N,k);
     int distance[] = new int[N];
     int graph[][] = new int[N][N];
        for(int source =0; source<N ;++source){
          distance = dijkstra(A, source);
          graph[source] = distance;
     }
     cost = costCalculation(B,graph,N);
     float density = densityCalculation(A ,N);

     System.out.println(" K = " + k + "\n Cost = " +cost + "
Density = " + density + "\n");
    }
  }

}
```