Algorithmic Aspects of Telecommunication Networks
CS 6385.001: Project #1

Due on Wednesday October 9, 2019 at 11:59am

Instructor: Prof. Andras Farago

Shyam Patharla **(sxp178231)**

Contents

1 Introduction

- In this project, we try to implement the basic network design model
- Give an input number of nodes, traffic demands between different nodes and a parameter k, our program outputs a network topology i.e a directed graph with links and capacities assigned to these links
- We do this for range of values of k
- We also analyze how the cost and density of the output network vary with the value of k
- We discuss the possible reasons for the above characteristics.

2 Design Decisions

- We implement the solution in the **Java** programming language
- The program modules were run on a **Mac** operating system
- We use the **Floyd Warshall** algorithm for the shortest path algorithm, since it is a dynamic programming approach and the results can be stores in a 2D array. the algorithm also has a worst case time complexity of $O(n^3)$.

3 Solution Approach

3.1 Generating Input Examples

Module 1 consists of generating input parameters for simulating the network. These parameters are then passed on to second module which designs the network using the Floyd Warshall all pairs shortest paths algorithm. The parameters are generated as follows.

- For all examples, we set the number of nodes in the network to 20 i.e $N = 20$.
- The *demands* matrix i.e. [bij] matrix is generated once using the steps mentioned in the project requirement document.
  - 10-digit student ID: 2021405427
  - Repeat it twice: 20214054272021405427
  - Let $d_0, d_1, ..., d_{19}$ be the digits in above number
  - $b_{ij} = abs(di - dj)$, for all $0 \leq i, j \leq 19$

- The *costs* matrix i.e. the [aij] matrix is generated for each value of k separately. We do this for all k values in the inclusive range 3,4,...,14
  - For any given node i, we select k random indices $j_1, j_2, ..., j_k$, all different from each other and different from i
  - We set $a_{ij} = 1$ for all $j \in j_1, j_2, ...j_k$ and $a_{ij} = 100$ for all $j \notin j_1, j_2, ...j_k$
  - Repeat for all nodes i

3.2 Shortest Paths Algorithm

Module 2 receives input parameters from the first module (number of nodes, cost matrix and demand matrix).

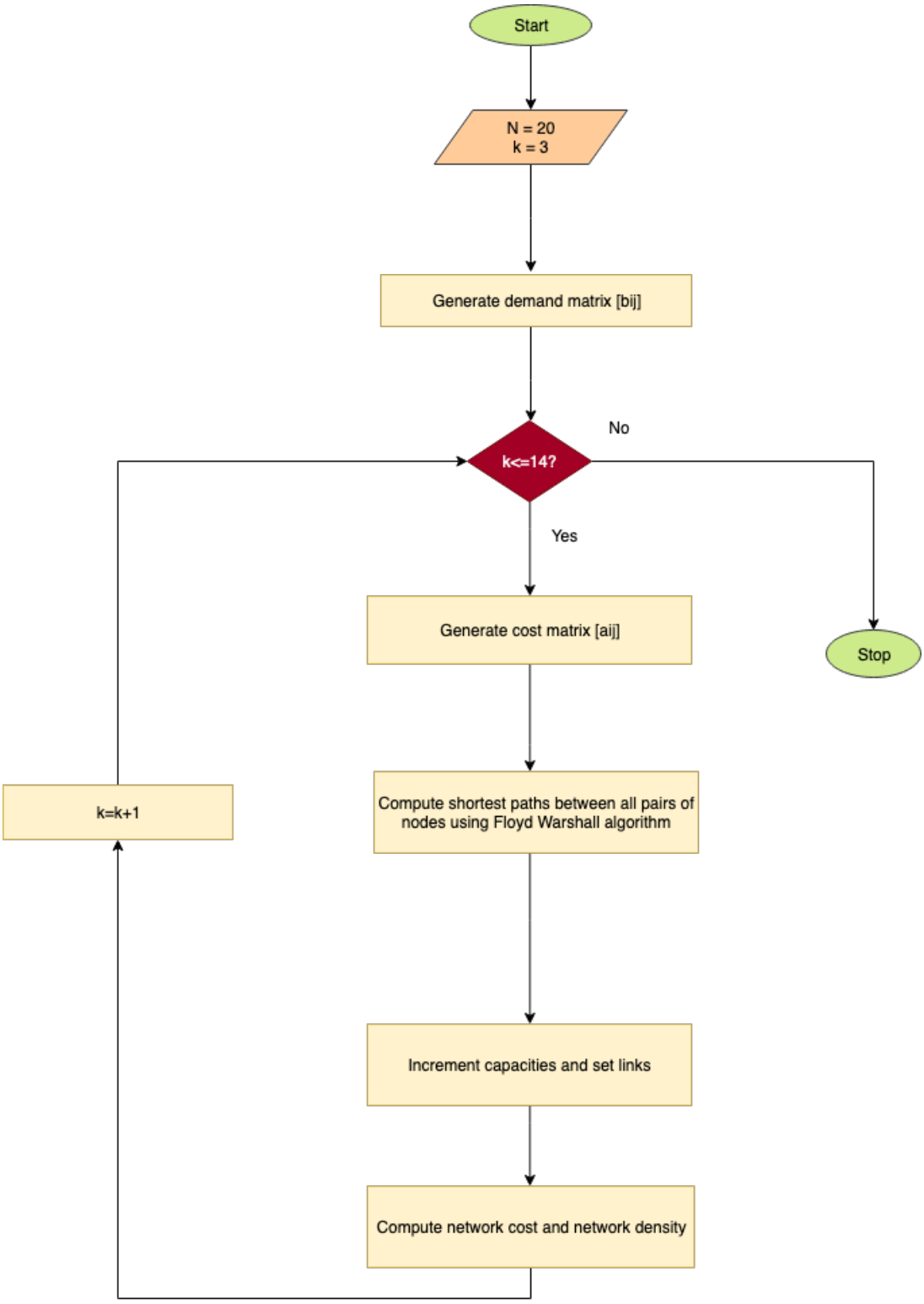
- All pairs shortest paths
  - We compute the shortest distances between all pairs of nodes using the **Floyd-Warshall algorithm** i.e. and store them in a matrix *dist*
  - While computing the above *dist* matrix, we also store the shortest paths between each pair of nodes (i,j) in the pred matrix. Each entry $\mathbf{pred_{ij}}$ gives the node with **maximum** number in the shortest path between nodes i and j.
  - **Special case:** If there is no node in the shortest path between i and j, $pred_{ij}$=-1
- Design Links and Capacities
  - The *capacities* matrix and *links* matrix are initiated with all values equal to **zero**
  - Let S be the shortest path between nodes k and l, where k≠l. For each edge (i,j) on S
    - If $\mathbf{link_{ij}}$ has not been set to 1, set it to 1
    - The capacity of $link_{ij}$ is incremented by $\mathbf{demand_{kl}}$
  - The above process is repeated for each pair of nodes (k,l) where k≠l.
- The *links* and *capacities* matrices are passed on to Module 3, which analyzes how the total cost of the network and densities change with respect to our k parameter.

3.3 Presentation of Results

- Module 3 takes the output parameters of *Module 2* i.e. the capacities and link matrices and computes
  - the **total cost** of the network

    - the **density** of the network i.e. the number of links with **non-zero** capacities divided by the maximum possible number of links which is N*(N-1)
  - The above computation is repeated for all k values
  - We then try to answer questions such as:
    - How does the total cost of the network depend on k?
    - How does the density of the obtained network depend on k?
  - We show some of the obtained network topologies graphically

We present the execution of our project using the flow chart below.


Start
N = 20
k = 3
Generate demand matrix [bij]
No
k<=14?
Yes
Generate cost matrix [aij]
Stop
Compute shortest paths between all pairs of nodes using Floyd Warshall algorithm
k=k+1
Increment capacities and set links
Compute network cost and network density

4 Floyd Warshall Algorithm - Explanation

- Suppose we have N vertices and the vertices are numbered from 1,2,...,N
- For every pair of vertices i and j we define a path $\pi(i, j, r)$ which is the shortest path from i to j that passes through vertices numbered atmost r.
- This problem has a simple recursive structure
- **Base Case**: The path $\pi(i, j, 0)$ cannot pass through any vertices, so it is the edge(if any) between i an j
- **Recursive Case**: The path $\pi(i, j, r)$ will have two cases
  - The path does not pass through the vertex labeled r, so it must pass through the vertices numbered atmost r-1 $\implies \pi(i, j, r) = \pi(i, j, r-1)$
  - The path passes through the vertex labeled r. It will consist of the path from i to r and the path from r to j, each of which passes through vertices numbered atmost r-1
- We first try our hands at a straightforward dynamic programming approach, which is shown below

Algorithm 1 FloydWarshallAlgorithm3D

```
1: procedure FloydWarshall3D(V, E, w)
2:     for all vertices i do
3:         for all vertices j do
4:             dist[i, j, 0] = w(i- > j)
5:         end for
6:     end for
7:     for r = 1 to V do
8:         for all vertices i do
9:             for all vertices j do
10:                if dist[i, j, r − 1] < dist[i, r, r − 1] + dist[r, j, r − 1] then        ▷ . ◁
11:                    dist[i, j, r] = dist[i, j, r − 1]
12:                else
13:                    dist[i, j, r] = dist[i, r, r − 1] + dist[r, j, r − 1]
14:                end if
15:            end for
16:        end for
17:    end for
18: end procedure
```

- We can simplify the algorithm by removing the third dimension. We can also modify the algorithm to support numbering the vertices arbitrarily

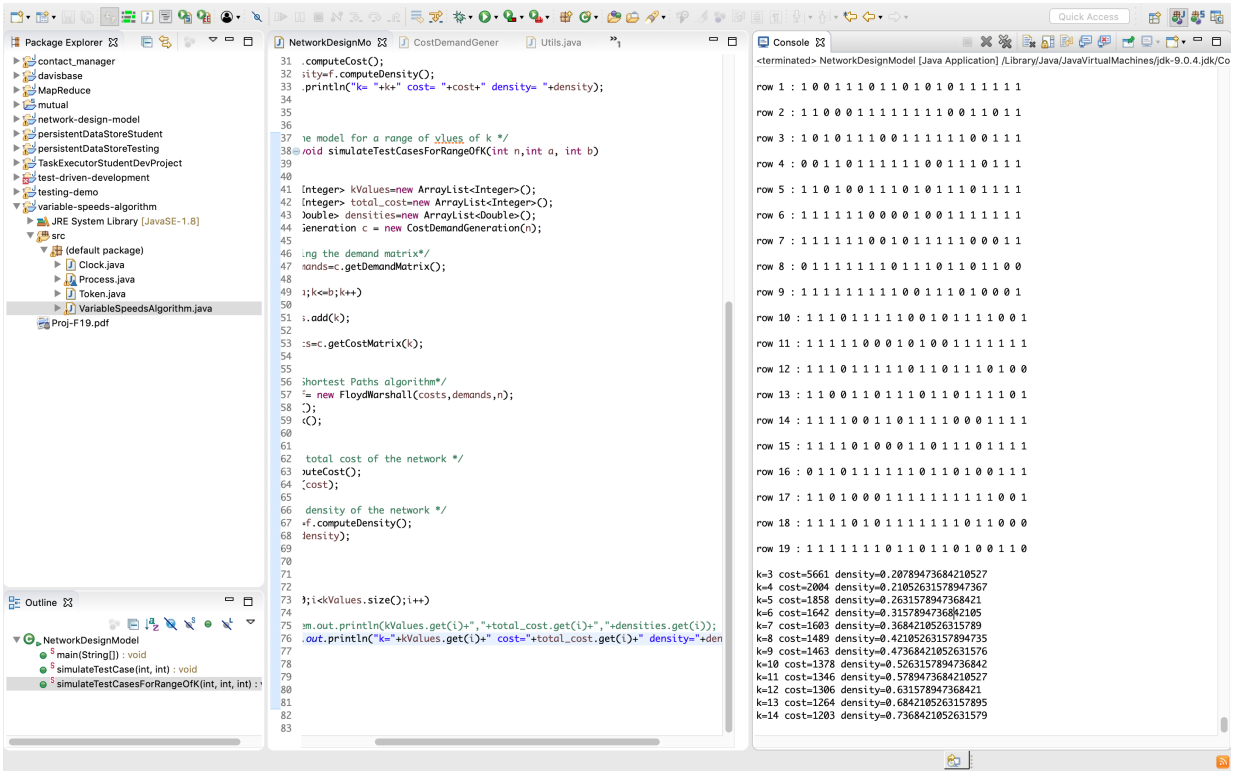
Algorithm 2 FloydWarshallAlgorithm

```
1: procedure FloydWarshall(V, E, w)
2:     for all vertices i do
3:         for all vertices j do
4:             dist[i, j] = w(i- > j)
5:             pred[i, j] = −1
6:         end for
7:     end for
8:     for all vertices r do
9:         for all vertices i do
10:            for all vertices j do
11:                if dist[i, r] + dist[r, j] < dist[i, j] then          ▷ . ◁
12:                    dist[i, j] = dist[i, r] + dist[r, j]
13:                    pred[i, j] = r
14:                end if
15:            end for
16:        end for
17:    end for
18: end procedure
```

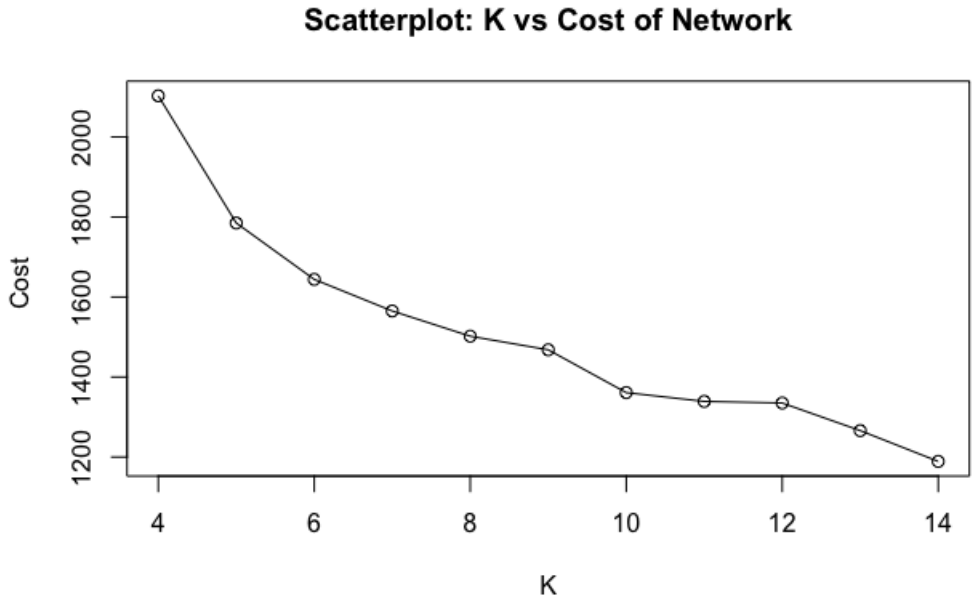
- The **dist** matrix gives the shortest path distances between all pairs of nodes.
- The **pred** matrix gives the node with maximum number in the shortest path between any two nodes.

5 Observations and Analysis

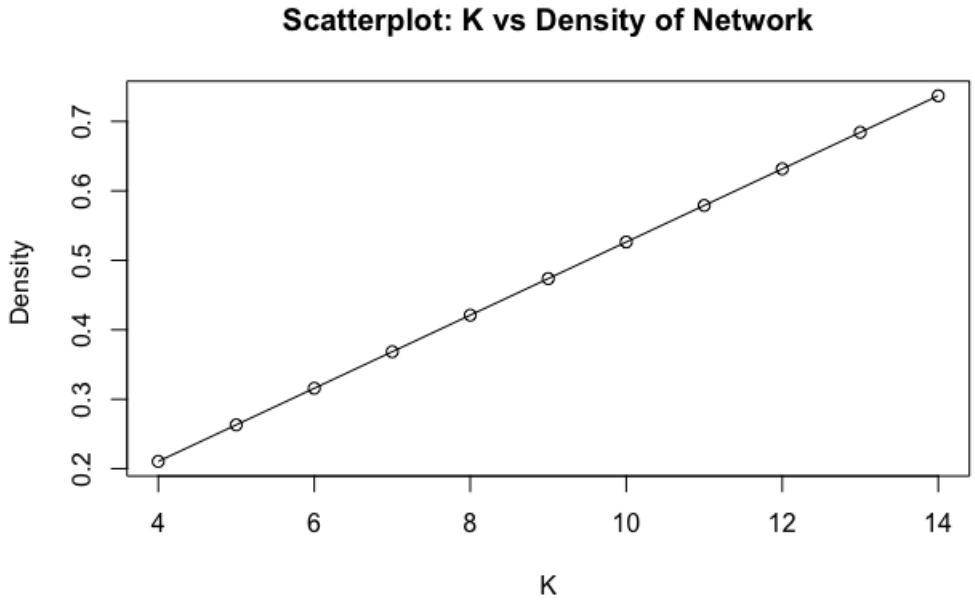
- The program produces an output as shown in the figure below.



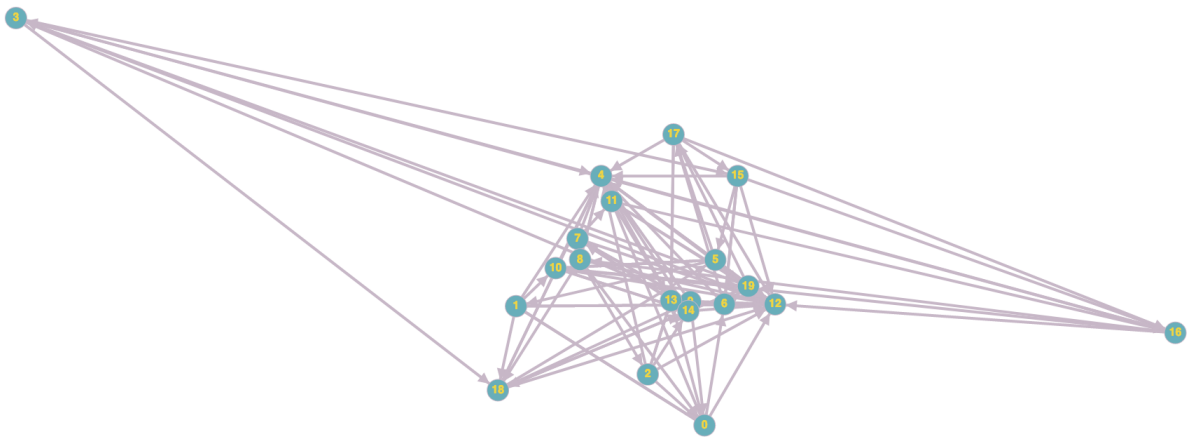
- The output results are stored in a *csv* file. The graphs are generated in **R**.
- We plot the graph of the parameter **k** vs. **cost of the network**. We can clearly see that the cost of the network **decreases** with increase in the value of k.



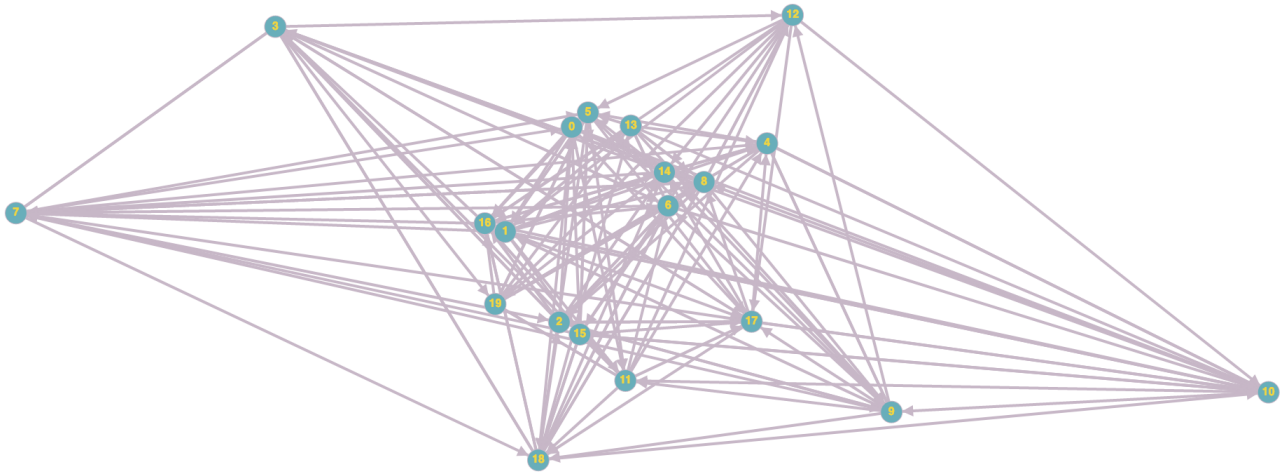
- We plot the graph of the parameter **k** against the **density of the network**. We can clearly see that the density of the network **increases** with increase in the value of k.



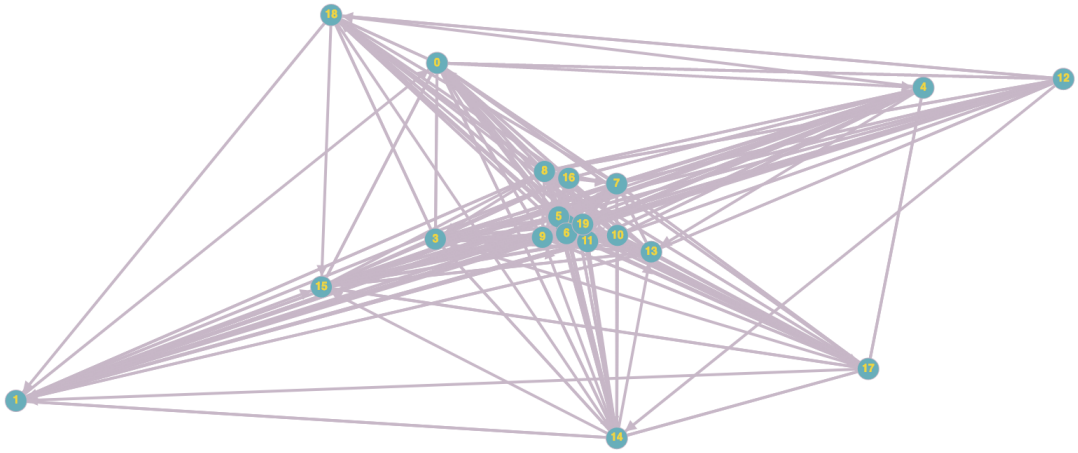
- Network topology for k=3



- Network topology for k=8



- Network topology for k=14



- We can clearly see that the **density** of the resulting network **increases** with increasing value of k.

6 Discussion

- Consider a single run of our program with any value of k (say k=5). For each node i, there will be 5 low cost links going out of i (cost=1). When the shortest path algorithm executes, it will try to **avoid** the high cost links (cost=100) going out of node i.
- Thus by **limiting k**, we limit the number of links that go out of any node i, and hence limit the **density** of the network
- This is the reason we observe that as k **increases**, the density of our resulting network also **increases**

- Consider any pair of nodes (s,t). Let i be any node i on the shortest path from s to t (t cannot be included in this set). As k increases we have more options at all such nodes i to reach t and we could take different paths.
- The number of edges which repeatedly fall in the shortest path between any 2 pair of nodes **decreases** and this is why the cost of the network **decreases**.
- The network has more of different edges of various edges of different costs rather than few edges some of which may have high costs

7 ReadMe File

This section shows how to run the project files.

- Downloads the project files and store them in a folder
- Open the project folder in Eclipse
- Open the file **NetworkDesignModel.java**
- Right Click $->$ Run as $->$ Java Application
- Alternatively,navigate to the folder in **terminal** and run the following commands
  - javac NetworkDesignModel.java
  - java NetworkDesignModel

8 Code

Module 1: CostDemandGeneration.java

```
1  import java.lang.Math;
2  import java.util.ArrayList;
3
4  import java.util.List;
5
6  /* Module 1 : Generating examples */
7  public class CostDemandGeneration {
8      private int range;
9      private int low;
10     private int high;
11     private int n;
12
13     CostDemandGeneration(int n) {
14         this.n = n;
15     }
```

```
16
17
18    public List < Integer > kRandomIndices(int k, int row) {
19        System.out.print("generating " + k + " random indices for node " + row
      + " : ");
20        List < Integer > list = new ArrayList < Integer > ();
21        int t;
22        while (list.size() != k) {
23
24            t = (int)(Math.random() * (range)) + low;
25            if (!list.contains(t) && t != row) {
26                list.add(t);
27
28                System.out.print(t + " ");
29            }
30        }
31
32        System.out.println("\n");
33
34        return list;
35
36    }
37
38    public int[][] getDemandMatrix() {
39        int[] arr = new int[] {2,0,2,1,4,0,5,4,2,7};
40
41        int demands[][] = new int[n][n];
42        for (int i = 0; i < n; i++) {
43            for (int j = 0; j < n; j++) {
44                demands[i][j] = Math.abs(arr[i % 10] - arr[j % 10]);
45            }
46        }
47
48        /* print the demand matrix*/
49        Utils.printMatrix(demands, n, "Demand matrix");
50        return demands;
51    }
52
53    public int[][] getCostMatrix(int k) {
54        /* set range for generating random indices for cost matrix*/
55        this.low = 0;
56        this.high = n - 1;
57        this.range = high - low + 1;
58
59   /* Generating the cost matrix*/
60        System.out.println("Generating cost matrix for k= " + k + " ...");
61        int[][] costs = new int[n][n];
62        List < Integer > list = new ArrayList < Integer > ();
63        for (int i = 0; i < n; i++) {
```

```
64              list = kRandomIndices(k, i);
65              for (int j = 0; j < n; j++) {
66                  if (i == j) {
67                      costs[i][j] = 0;
68                  } else if (list.contains(j)) {
69                      costs[i][j] = 1;
70                  } else {
71                      costs[i][j] = 100;
72                  }
73
74              }
75
76          }
77          /* print the cost matrix */
78          Utils.printMatrix(costs, n, "Cost matrix:");
79          return costs;
80      }
81  }
```

Module 2: FloydWarshall.java

```
public class FloydWarshall {
    private int n;
    private int[][] costs;
    private int[][] demands;

    private int[][] dist;
    private int[][] pred;

    private int[][] links;
    private int[][] capacities;

    /* Module 2: Shortest Paths Algorithm
     *
     * Step 1: Use the Floyd Warshall algorithm to get the shortest distances and paths
     * shortest distances are stored in dist matrix
     * shortest paths are stored in pred matrix
     * pred[i][j] is the node having the maximum number in the shortest path between i and j.
     *  If there is no such node, pred[i][j]=-1
     *
     *  Step 2: Increment capacities of links on the shortest path between i and j with demand[i][j], Repeat for all (i,j)
     *
     *  Step 3: Compute cost and density of the obtained network */


    FloydWarshall(int[][] costs, int demands[][], int n) {
        this.costs = costs;
        this.demands = demands;
        this.n = n;
        this.pred = new int[n][n];
        this.links = new int[n][n];
        this.capacities = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                pred[i][j] = -1;
                capacities[i][j] = 0;
                links[i][j] = 0;
                this.dist = costs;
            }
        }
    }

    /* run the shortest paths algorithm */
    public void runAlgorithm() {
        for (int r = 0; r < n; r++) {
            for (int i = 0; i < n; i++) {
```

```
46                 for (int j = 0; j < n; j++) {
47                     if (dist[i][r] + dist[r][j] < dist[i][j]) {
48                         dist[i][j] = dist[i][r] + dist[r][j];
49                         pred[i][j] = r;
50                     }
51                 }
52             }
53         }
54 
55         /* print the dist matrix */
56         Utils.printMatrix(dist, n, "All pairs shortest paths distances");
57 
58         /* print the pred matrix */
59         Utils.printMatrix(pred, n, "Predecessor matrix");
60     }
61 
62     /* designing the network using the shortest path distances */
63     public void designNetwork() {
64         for (int i = 0; i < n; i++) {
65             for (int j = 0; j < n; j++) {
66                 if (i != j) {
67                     adjustCapacities(i, j, demands[i][j]);
68                 }
69             }
70         }
71 
72         /* print the capacities matrix */
73         Utils.printMatrix(capacities, n, "capacities");
74 
75         /* print the links matrix */
76         Utils.printMatrix(links, n, "Links ");
77 
78     }
79 
80     /* compute the cost of the network */
81     public int computeCost() {
82         int total = 0;
83         for (int i = 0; i < n; i++) {
84             for (int j = 0; j < n; j++)
85 
86             {
87                 total += capacities[i][j] * costs[i][j];
88             }
89         }
90 
91         return total;
92     }
93 
94     /* compute the density of the network */
```

```
95      public double computeDensity() {
96          double density = 0;
97          for (int i = 0; i < n; i++) {
98              for (int j = 0; j < n; j++) {
99                  density += links[i][j];
100             }
101         }
102         return density / (n * (n - 1));
103     }
104
105     /* adjust capacities on links in a shortest path */
106     public void adjustCapacities(int src, int dest, int val) {
107         if (pred[src][dest] == -1) {
108             capacities[src][dest] += val;
109             links[src][dest] = 1;
110             return;
111         }
112         adjustCapacities(src, pred[src][dest], val);
113         adjustCapacities(pred[src][dest], dest, val);
114     }
115
116     /* print the shortest path between nodes src and dest */
117     public void printPath(int src, int dest) {
118         System.out.print("Path from " + src + " to " + dest + " : ");
119         getPath(src, dest);
120         System.out.println("\n");
121     }
122
123     /* print nodes in the shortest path between nodes src and dest */
124     public void getPath(int src, int dest) {
125
126
127         if (pred[src][dest] == -1) {
128             System.out.print(" ");
129             return;
130         }
131         getPath(src, pred[src][dest]);
132         System.out.print(" " + pred[src][dest]);
133         getPath(pred[src][dest], dest);
134
135
136     }
137
138
139
140
141 }
```

Module 3: NetworkDesignModel.java

```
import java.util.ArrayList;
public class NetworkDesignModel {

    /* Module 3: ANalysis and Presentation of Results */
    public static void main(String[] args) {

        /* Number of nodes */
        int n = 20;

        //simulateTestCase(n,3);
        //simulateTestCasesForRangeOfK(n,3,14);
        simulateTestCase(n, 8);
        simulateTestCase(n, 14);

    }

    /* Simulate the model for a given value of k */
    public static void simulateTestCase(int n, int k) {
        CostDemandGeneration c = new CostDemandGeneration(n);

        /* Generating the demand matrix*/
        int[][] demands = c.getDemandMatrix();

        int[][] costs = c.getCostMatrix(k);


        /* Module 2 : Shortest Paths*/
        FloydWarshall f = new FloydWarshall(costs, demands, n);
        f.runAlgorithm();
        f.designNetwork();
        int cost = f.computeCost();
        double density = f.computeDensity();
        System.out.println("k= " + k + " cost= " + cost + " density= " +
   density);

    }

    /* Simulate the model for a range [a,b] of values of k  */
    public static void simulateTestCasesForRangeOfK(int n, int a, int b) {

        ArrayList < Integer > kValues = new ArrayList < Integer > ();
        ArrayList < Integer > total_cost = new ArrayList < Integer > ();
        ArrayList < Double > densities = new ArrayList < Double > ();
        CostDemandGeneration c = new CostDemandGeneration(n);

        /* Generating the demand matrix*/
        int[][] demands = c.getDemandMatrix();

```

```
48          for (int k = a; k <= b; k++) {
49              kValues.add(k);
50
51              int[][] costs = c.getCostMatrix(k);
52
53
54              /* Module 2 : Shortest Paths algorithm*/
55              FloydWarshall f = new FloydWarshall(costs, demands, n);
56              f.runAlgorithm();
57              f.designNetwork();
58
59
60              /* compute the total cost of the network */
61              int cost = f.computeCost();
62              total_cost.add(cost);
63
64              /* compute the density of the network */
65              double density = f.computeDensity();
66              densities.add(density);
67
68
69          }
70
71          for (int i = 0; i < kValues.size(); i++) {
72              //System.out.println(kValues.get(i)+","+total_cost.get(i)+","+
    densities.get(i));
73              System.out.println("k=" + kValues.get(i) + " cost=" + total_cost.
    get(i) + " density=" + densities.get(i));
74
75
76          }
77
78      }
79  }
```

Utils.java

```
1
2  public class Utils {
3    public static void printMatrix(int[][] arr, int n, String s)
4    {
5      System.out.println(s);
6
7      for(int i=0;i<n;i++)
8      {
9        System.out.print("row "+i+" : ");
10       for(int j=0;j<n;j++)
11       {
12         System.out.print(arr[i][j]+" ");
13       }
14       System.out.println("\n");
15     }
16   }
17
18
19
20 }
```

Visualization.R

```
1  data <- read.csv(file="/users/psprao/eclipse-workspace/network-design-model/
       output1.csv")
2
3  # Getting K, cost and density  data
4  k<-data[,1]
5  cost<-data[,2]
6  density<-data[,3]
7
8  # Scatterplot of K vs Cost of Network
9  plot(k,cost,xlab="K",ylab="Cost ",main="Scatterplot: K vs Cost of Network")
10 lines(k,cost)
11
12 # Scatterplot of K vs Density of Network
13 plot(k,density,xlab="K",ylab="Density ",main="Scatterplot: K vs Density of
       Network")
14 lines(k,density)
```

9 References

- The Floyd Warshall algorithm was referred to from the online notes of Professor Jeff Eriksen at UIUC, which can be found here
- The specific algorithm can be found in the document All Pairs Shortest Paths on the webpage,specificaly in pages 317-318.

- The graph topologies for different k were visualized online on the website Graph Online using the link matrices obtained as the output
- Lecture Notes - An Application to Network Design