Project 1

Application to Network Design

CS 6385

Vyoma Trivedi (vmt160030)

Table of Content

Objective

Create software that is capable of doing the following:

• As input, it receives the number of nodes (N), the traffic demand values (bij) between pairs of nodes, and the unit cost values for the potential links (aij).

• As output, the program generates a network topology, with capacities assigned to the links, according to the studied model, using the shortest path based fast solution method. The program also computes the total cost of the designed network.

Algorithm used to Calculate Shortest Path

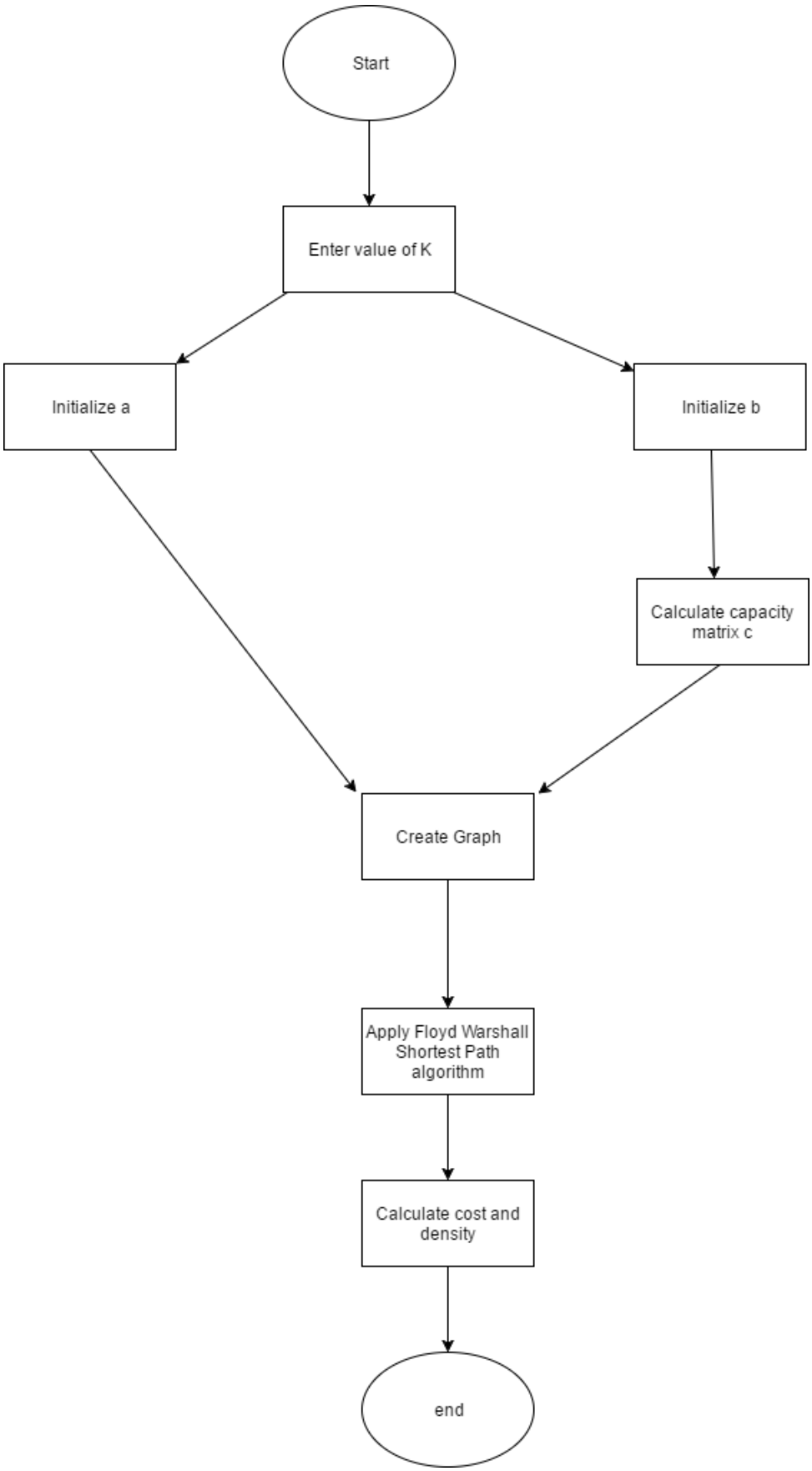
Consider a graph G with vertices V numbered 1 through N. Further consider a function shortestPath(i, j, k) that returns the shortest possible path from i to j using vertices only from the set {1,2,...,k} as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each i to each j using only vertices 1 to k + 1.

For each of these pairs of vertices, the true shortest path could be either (1) a path that only uses vertices in the set {1, ..., k} or (2) a path that goes from i to k + 1 and then from k + 1 to j. We know that the best path from i to j that only uses vertices 1 through k is defined by shortestPath(i, j, k), and it is clear that if there were a better path from i to k + 1 to j, then the length of this path would be the concatenation of the shortest path from i to k + 1 (using vertices in {1, ..., k}) and the shortest path from k + 1 to j (also using vertices in {1, ..., k}).

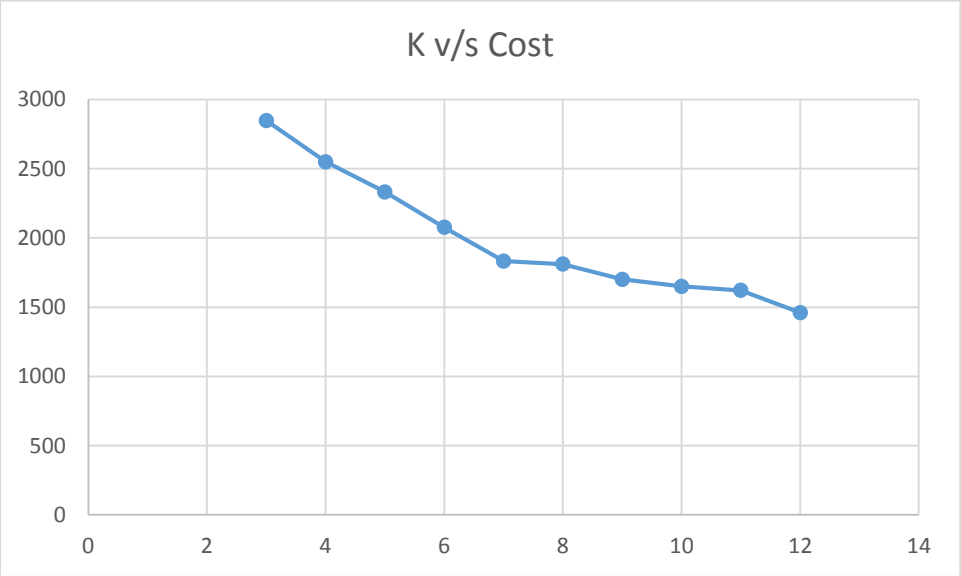
The formula and flow of the Floyd Warshall Algorithm is as shown below :

```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 for each vertex v
3    dist[v][v] ← 0
4 for each edge (u,v)
5    dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
6 for k from 1 to |V|
7    for i from 1 to |V|
8       for j from 1 to |V|
9          if dist[i][j] > dist[i][k] + dist[k][j]
10             dist[i][j] ← dist[i][k] + dist[k][j]
11         end if
```

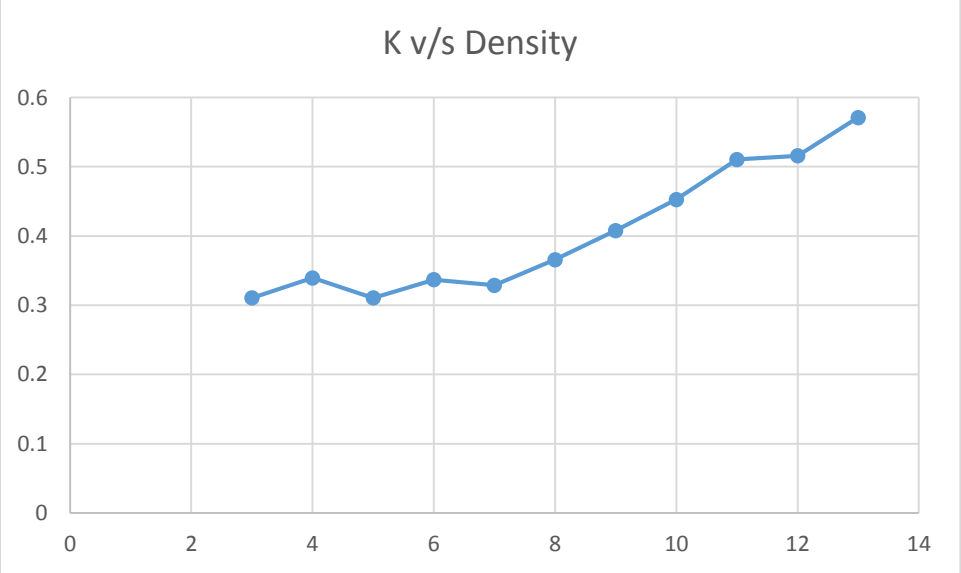
Flow of the program



Charts



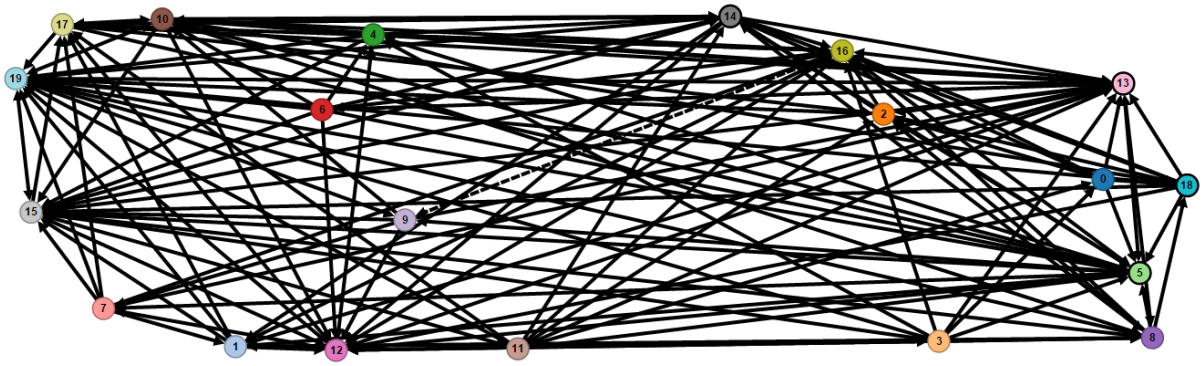
As we can see from the graph the cost decreases with increase in value of K as more edges get assigned a value of 1 instead of 250 so the cost will decrease as many low cost paths will become available to be selected as shortest.



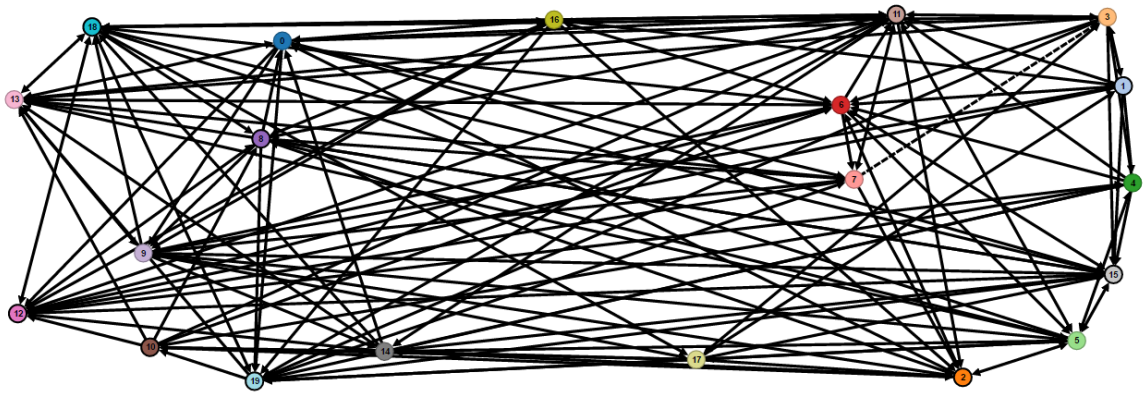
Since traffic and K are random the density fluctuates. So nothing can be said in specific about the relation between K and density but as can be seen from the graph the density increases with K.

Graphs

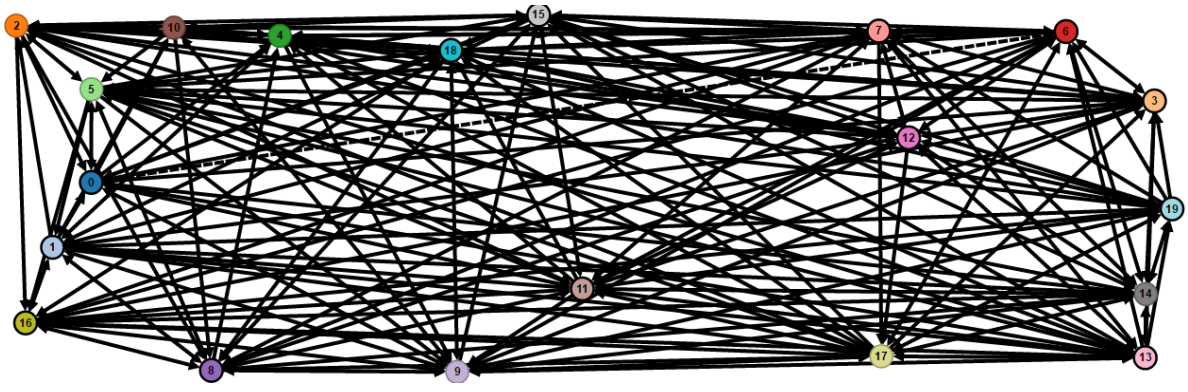
K=3



K=8



K=13



Like we can see from the graphs above , as the value of K increases the density increases as the ‘a’ matrix of more edges becomes 1 and so more of them get included in the shortest path thus making the graph dense.

Appendix

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package network_design1;

import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author VYOMA
 */
public class Network_Design1 {

    /**
     * @param args the command line arguments
     */
    public static void ShortestPath(int a[][],int b[][],int c[][],int k)
    {
        int shortest[][]=new int[20][20];
        int adjacency_matrix[][]=new int[20][20];
        int cost=0;
```

```
int total_edges=380;
String output="";
for(int i=0;i<20;i++)
{
   for(int j=0;j<20;j++)
   {
      shortest[i][j]=-1;
      adjacency_matrix[i][j]=0;
   }
}
for(int m=0;m<20;m++)
{
   for(int i=0;i<20;i++)
   {
      for(int j=0;j<20;j++)
      {
         if(a[i][m]+a[m][j]<a[i][j])
         {
            a[i][j] =a[i][m] + a[m][j];
            shortest[i][j]=m;

         }
      }
   }
}
for(int i=0;i<20;i++)
{
   for(int j=0;j<20;j++)
   {
      int node=shortest[i][j];
```

```
        int src=i;
        while(node !=-1)
        {
            c[src][node]=c[src][node]+b[src][node];
            adjacency_matrix[src][node]=1;
            src=node;
            node=shortest[src][j];
        }
        c[src][j]+=b[i][j];
        adjacency_matrix[src][j]=1;
    }
}
for(int i=0;i<20;i++)
{
    for(int j=0;j<20;j++)
    {
        cost=cost+a[i][j]*c[i][j];

    }
}
System.out.println("cost : "+cost);

int nonZeroEdges=0;
for(int i=0;i<20;i++)
{
    for(int j=0;j<20;j++)
    {
        if(c[i][j]!=0)
        {
            nonZeroEdges++;
```

```
            }

        }
    }
    float density=(float)nonZeroEdges/total_edges;
    System.out.println("Density : "+density);

    for(int i=0;i<20;i++)
    {
        for(int j=0;j<20;j++)
        {
            if(adjacency_matrix[i][j]!=0)
            {
                output+=" "+i+"->"+j+",";
            }

        }

    }
    System.out.println("Output Graph"+output);
}
public static void main(String[] args) {
    // TODO code application logic here
    int adjacency_matrix[][];
    int number_of_vertices;
    int source = 0;
    int temp[] = {2,0,2,1,3,1,9,1,7,0,2,0,2,1,3,1,9,1,7,0};
    int b[][]=new int[20][20];
    int a[][]=new int[20][20];
    int k;
```

```
int c[][]=new int[20][20];

for(int i=0;i<20;i++)
{
   for(int j=0;j<20;j++)
   {
      c[i][j]=0;
   }
}

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

Scanner scan = new Scanner(System.in);
number_of_vertices=temp.length;



   for(int i=0;i<number_of_vertices;i++)
   {
      for(int j=0;j<number_of_vertices;j++)
      {
         b[i][j]=Math.abs(temp[i]-temp[j]);
      }
   }
   for(int i=0;i<20;i++)
   {
      sel.add(i);
   }

   Random r = new Random();
```

```
 System.out.println("Enter the value of K: ");
k=scan.nextInt();
for(int i=0;i<number_of_vertices;i++)
{
   ArrayList<Integer> sel1 = new ArrayList<>();
   while(sel1.size()!=k)
                {
                   int selected=sel.get(r.nextInt(sel.size()));
                   if(!sel1.contains(selected))
                   {
                      sel1.add(selected);
                   }
                }
   for(int j=0;j<number_of_vertices;j++)
   {

      if(i==j)
         a[i][j]=0;
      else if(sel1.contains(j))
         a[i][j]=1;
      else
         a[i][j]=200;

   }
}


//          System.out.println("Enter the source ");
//          source = scan.nextInt();
```

```
        Network_Design1 nd = new Network_Design1();
        nd.ShortestPath(a, b, c, k);
    }

}
```

References

1) http://yiboyang.com/graphrel/
2) https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm