ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

CS 6385

PROJECT 1

"AN APPLICATION TO NETWORK DESIGN"

SUBMITTED BY

GODAVARTHI VENKATADATTA

vxg161330

Index:

Problem Statement:

- 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 (directed graph), with capacities assigned to the links (directed edges), according to the studied model, using the shortest path based fast solution method (see at the end of the referred lecture note). The program also computes the total cost of the designed network.
- Let the number of nodes be N = 20 in each example.
- For generating the bij values, take your 10-digit student ID, and repeat it 2 times, to obtain a 20-digit number. For example, if the ID is 0123456789, then after repetition it becomes 01234567890123456789. Let d1; d2; : : ; d20 denote the individual digits in this 20-digit number. Then the value of bij is computed by the formula bij = |di - dj |
- Run your program with k = 3; 4; 5 : : : ; 14. For each run generate new random aij parameters independently.

Shortest Path Algorithm:

Floyd Warshall Algorithm:

We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

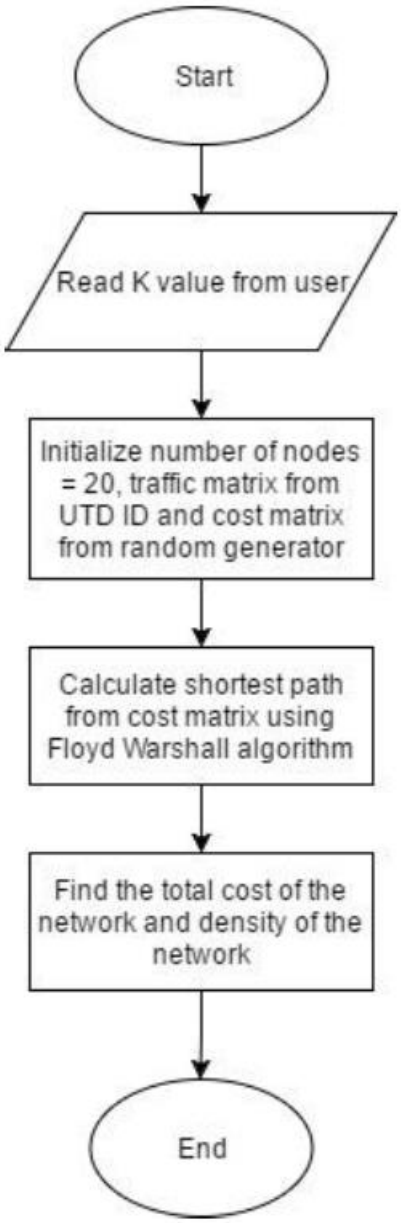
2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].

```
// Solves the all-pairs shortest path problem using Floyd Warshall algorithm
void floydWarshall (int graph[][V])
{
    /* dist[][] will be the output matrix that will finally have the shortest
      distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix same as input graph matrix. Or
       we can say the initial values of shortest distances are based
       on shortest paths considering no intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to the set of intermediate vertices.
      ---> Before start of a iteration, we have shortest distances between all
      pairs of vertices such that the shortest distances consider only the
      vertices in set {0, 1, 2, .. k-1} as intermediate vertices.
      ----> After the end of a iteration, vertex no. k is added to the set of
      intermediate vertices and the set becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++)
            {
                // If vertex k is on the shortest path from
                // i to j, then update the value of dist[i][j]
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(dist);
}
```

Algorithm:

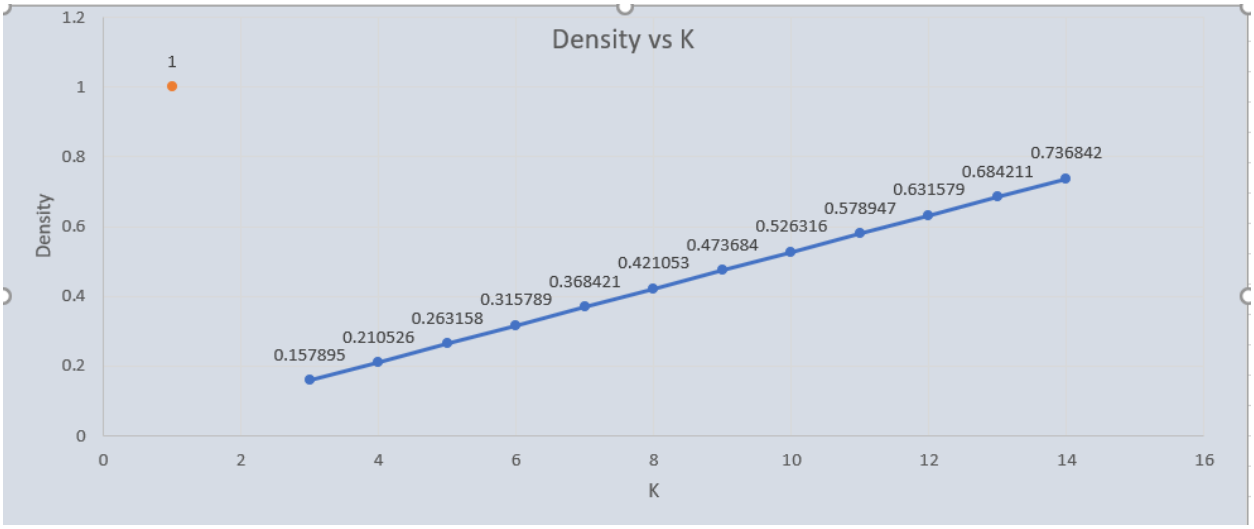
1. At the start of the program, user will be requested to enter the K value and based on this K value the cost matrix will be initialized i.e. For K random links independent of i but random value equal to j will be assigned unit cost Rest all links will be assigned cost of 100
2. Based on UTD ID (repeated two times to create a 20 digit value) the traffic matrix is initialized as per the given logic in problem statement
3. The cost matrix is given to the Floyd Warshall's algorithm for calculating the shortest path.
4. This output from the algorithm is multiplied with the traffic matrix.
5. Cost and density are calculated.
6. The graphs are generated from the website Graphonline.ru/en.

Flowchart:


Start
Read K value from user
Initialize number of nodes = 20, traffic matrix from UTD ID and cost matrix from random generator
Calculate shortest path from cost matrix using Floyd Warshall algorithm
Find the total cost of the network and density of the network
End

Outputs:

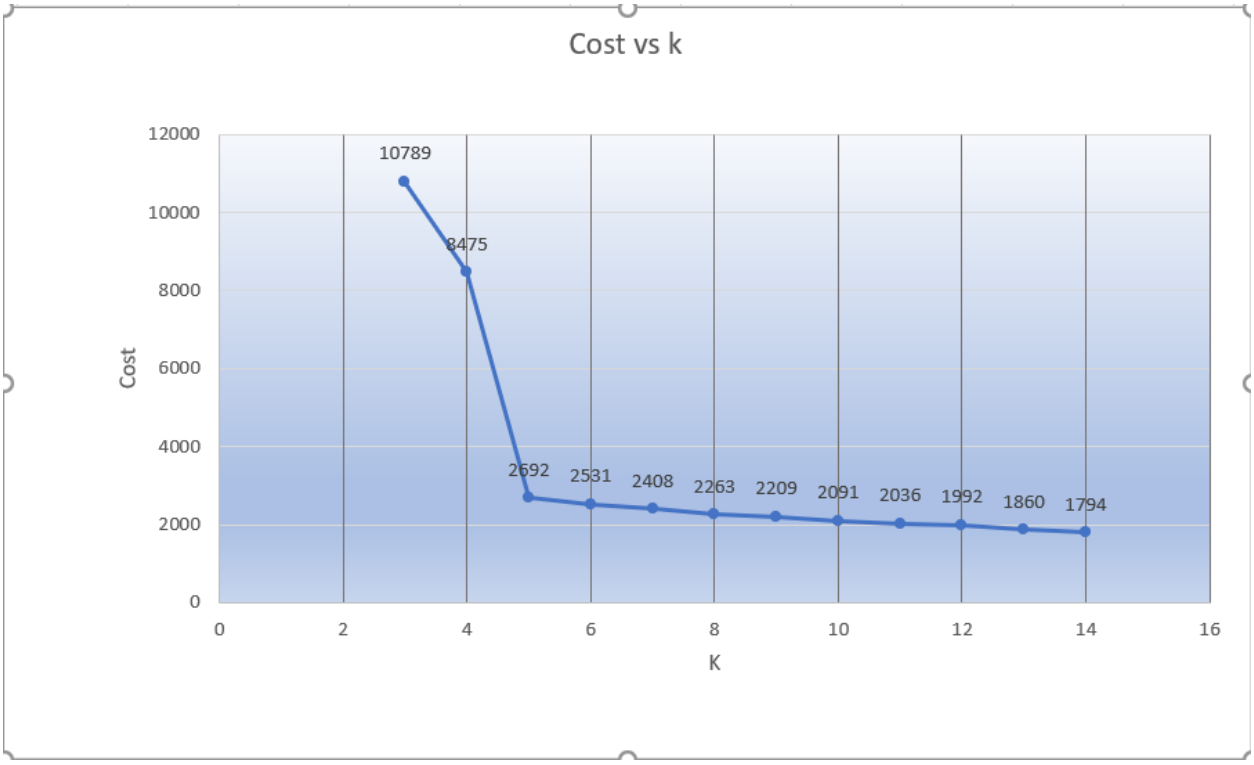
Density vs K:



For Density calculation, only the low-cost potential links are taken into counting which resulted in a varying density with K value.

As per the logic given in the problem statement, every link is assigned a cost except the principle diagonal elements (i.e. cost to the same node itself).

Cost vs K



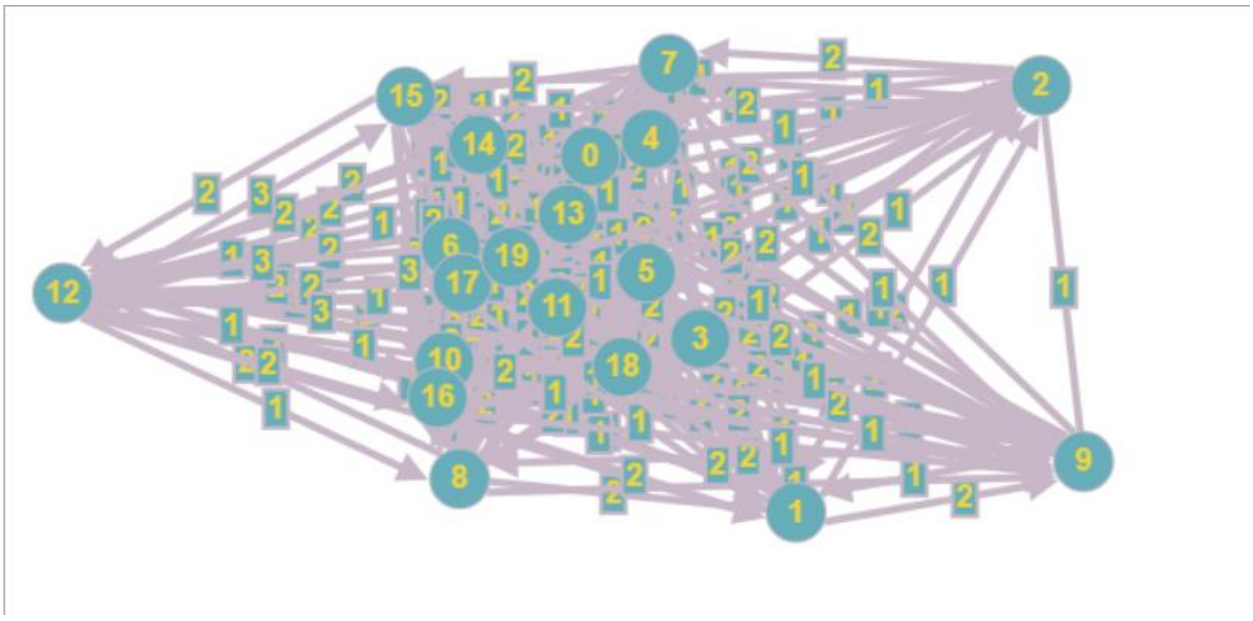
As the K value increases the cost of the network decreases and becomes relatively constant.

The cost kept decreasing because, as the K value increase more number of low-cost link (cost=1) links are available for the algorithm to find the shortest path.

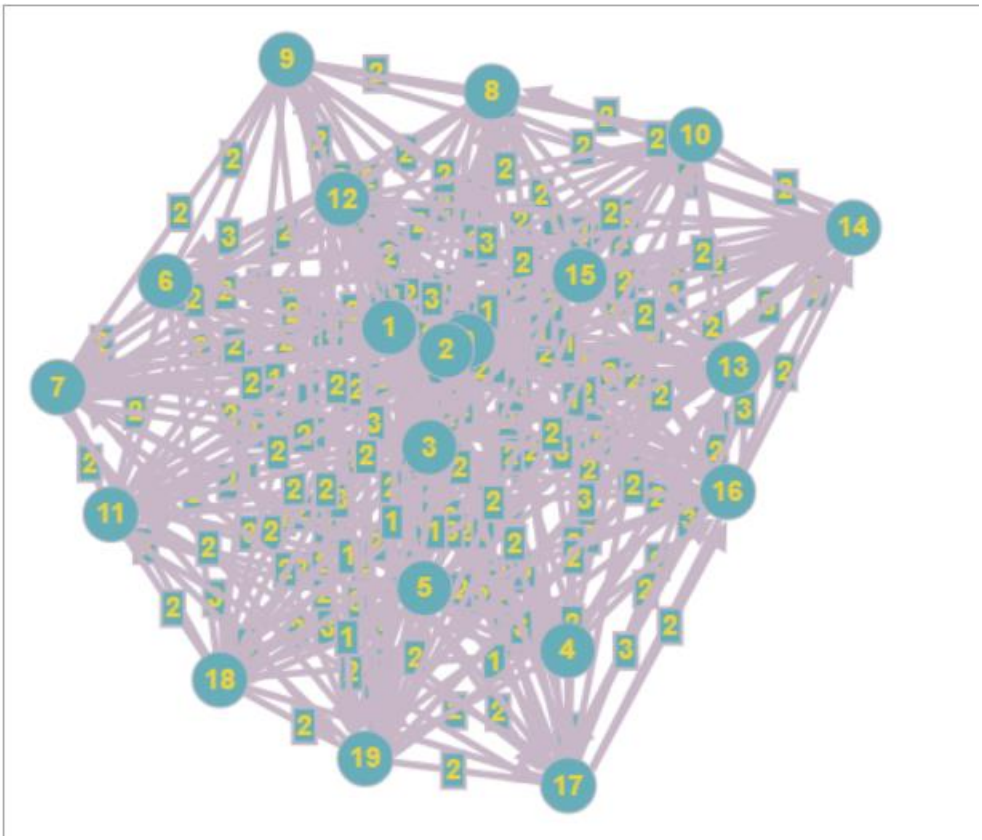
Network Graphs:

K=3:

K=8:



K=14:



ReadMe:

Copy the code from the appendix onto a new file in the C++ console like Dev Cpp.

Compile and Run the program.

The console should display the Shortest Path Cost Matrix, the Edge Matrix, the Density and the Cost.

Programming Language: C++;

IDE: Dev Cpp

UTD ID: 2021321999

Appendix:

```
#include <iostream>
#include <algorithm>
using namespace std;

int list[19];
int n=20;
int cost_matrix[20][20];
int traffic_matrix[20][20];


//This is a random number generator
void Random_Number(int z)
{
	int x=0;
	for(int i = 0; i <n; i++)
	{
		if(i==z)	i++;

		list[x]=i;
		x++;
	}
random_shuffle(list, list+19);
}
```

```
float Cost_Matrix(int k)
{
	int count=0;
	for(int i=0;i<n;i++)
	{
		Random_Number(i);
		for(int j=0;j<n;j++)
		{
			cost_matrix[i][j]=100;
			for(int p=0;p<k;p++)
			{
				if(list[p]==j) cost_matrix[i][j]=1;

				if(i==j) cost_matrix[i][j]=0;
			}
		}
	}
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
		{
			if(cost_matrix[i][j]==1) count=count+1;
		}
	}
float density = (float)count/(20*19);
```

```
        return density;
        }


void Traffic_Matrix()
{
        int d[]={2,0,2,1,3,2,1,9,9,9,2,0,2,1,3,2,1,9,9,9};
        int temp;
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<n;j++)
                {
                        temp = (d[i]-d[j]);
                        if(temp>0)
                                traffic_matrix[i][j]=temp;
                        else
                                traffic_matrix[i][j]= -temp;
                }
        }

}

//FloydWarshall Algorithm for shortest path

int FloydWarshallAlgorithm()
{
```

```
int edge_matrix[n][n];
for(int i=0;i<n;i++)
{
	for(int j=0;j<n;j++)
	{
		edge_matrix[i][j]=0;
	}
}
for (int i=0;i<n;i++)
{
	for (int j=0;j<n;j++)
	{
		for (int l=0;l<n;l++)
		{
			if ((cost_matrix[j][i]+cost_matrix[i][l]) < cost_matrix[j][l])
			{

cost_matrix[j][l]=cost_matrix[j][i]+cost_matrix[i][l];
				edge_matrix[j][i]=1;
				edge_matrix[i][l]=1;
			}
			if(j==l)	cost_matrix[j][l]=0;
		}
	}
}
cout<<"Shortest Path Cost Matrix: ";
```

```
	for(int i=0;i<n;i++)
	{
		cout<<endl;
		for(int j=0;j<n;j++)
		{
			cout<<cost_matrix[i][j]<<" ";
		}
	}
	cout<<"\n\nEdge Matrix : ";
	for(int i=0;i<n;i++)
	{
		cout<<endl;
		for(int j=0;j<n;j++)
		{
			cout<<edge_matrix[i][j]<<" ";
		}
	}
}
int Cost()
{
	int cost=0;
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
		{
			cost_matrix[i][j]=cost_matrix[i][j]*traffic_matrix[i][j];
```

```
                cost += cost_matrix[i][j];
            }
        }
        return cost;
}
main()
{
        int k,cost=0;
        float density=0;
        for(int i=3; i<=14;i++){
            k=i;
            cout<<"\n\nFor k=" <<k;
            cout<<"\n\n";
            Traffic_Matrix();
            density=Cost_Matrix(k);
            FloydWarshallAlgorithm();
            cout<<"\n\nThe Density is : "<<density;
            cout<<"\nThe Cost is : "<<Cost();
            cout<<"\n";
        }
}
```

References:

1. Floyd Warshall Algorithm: https://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/

2. The Graphs are generated using http://graphonline.ru/en/