Algorithmic Aspects of Telecommunication Networks
Project 1

Siddartha Guthikonda
sxg122630@utdallas.edu

INDEX

1. Objective:

Implement basic network design module that generates a network topology with capacities assigned to links, according to the model, An Application to Network Design, using shortest path based fast solution. The implementation need to take the input of number of nodes and traffic demand values (bij) between pair of nodes and unit cost values of potential links (aij).

Input:

- Number of nodes
- Traffic demand values (bij)
- Unit cost values of potential links (aij) which are computed according to the problem statement

Output:

- Run through k = 3 to 15, the implementation generates a network topology according to the model cited in “An Application to Network Design”
- Also, the shortest cost of the shortest cost tree is computed

2. Introduction:

Dijkstraw's Shortest path algorithm

Dijkstra's algorithm is a graph search algorithm that solves the single source shortest path problem for a graph with negative edge path costs, producing a shortest path tree. It was developed by a Dutch scientist Edsger Dijkstra in 1956 and published in 1959.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm, once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

In this program the shortest path is found by running Dijkstraw's algorithm n times where n is the number of nodes. It is run n times because, we check the shortest path from all different possible source

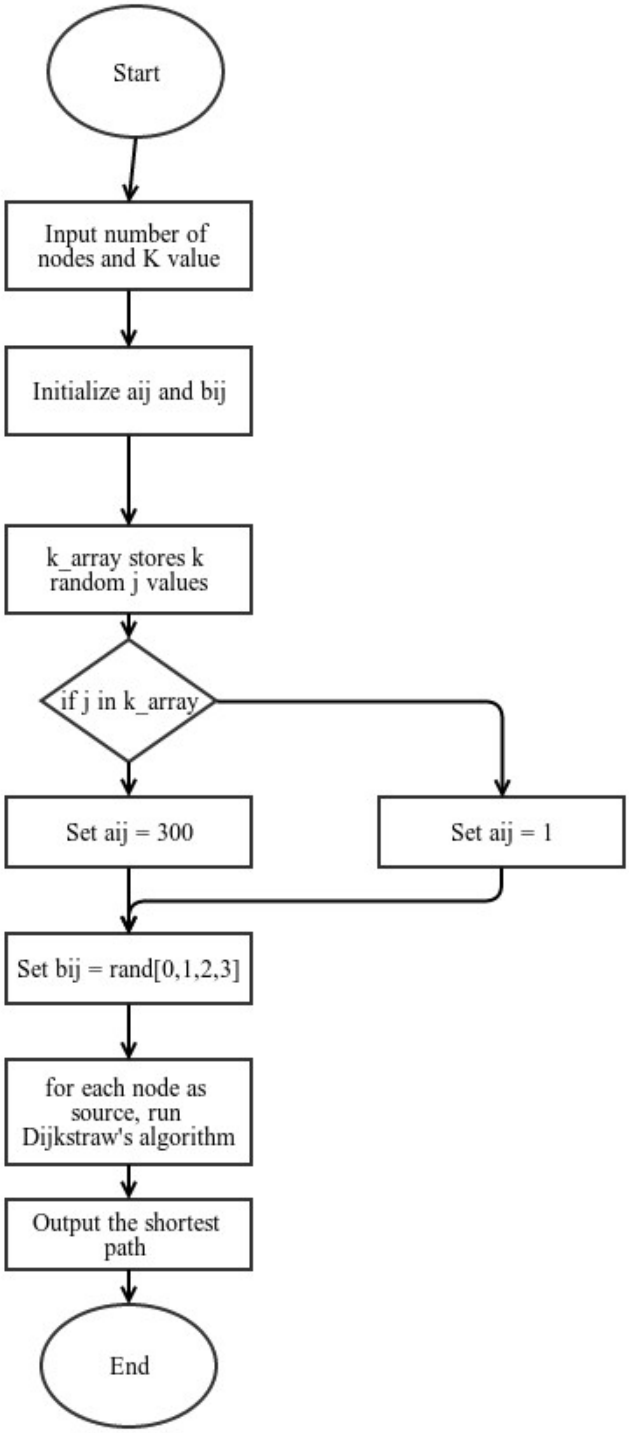
nodes.

3. Implementation:

We use the shortest path Dijkstraw's algorithm to conctruct the network topology. Below are the steps,

- User inputs the number of nodes and value of K
- All cost and traffic datastructures are inititlized
- An array of size k is taken to store k random values of j.
- Set aij as, if, aij = 300 if j != element in k array
- Set aij as, if, aij = 1 if j == element in k array
- set bij to a random variable in [0,1,2,3]
- Run Dijkstraw algorithm for each node in the available nodes as source node
- the shortest path tree is generated by taking the minimum value of the dijkstraw's algorithm.

4. Flowchart:


Start
Input number of nodes and K value
Initialize aij and bij
k_array stores k random j values
if j in k_array
Set aij = 300
Set aij = 1
Set bij = rand[0,1,2,3]
for each node as source, run Dijkstraw's algorithm
Output the shortest path
End

5. Results:

1. Nodes = 40, k= 3

Enter the number of nodes

40
Enter the value of K
3
10 -> 0 = 1

10 -> 1 = 300
10 -> 2 = 300
10 -> 3 = 300
10 -> 4 = 300
10 -> 5 = 300
10 -> 6 = 1
10 -> 7 = 300
10 -> 8 = 300
10 -> 9 = 300
10 -> 10 = 300
10 -> 11 = 300
10 -> 12 = 300
10 -> 13 = 300
10 -> 14 = 300
10 -> 15 = 300
10 -> 16 = 300
10 -> 17 = 1
10 -> 18 = 300
10 -> 19 = 300
10 -> 20 = 300
10 -> 21 = 300
10 -> 22 = 300
10 -> 23 = 1
10 -> 24 = 300
10 -> 25 = 300
10 -> 26 = 300
10 -> 27 = 300
10 -> 28 = 300
10 -> 29 = 300
10 -> 30 = 300
10 -> 31 = 300
10 -> 32 = 300
10 -> 33 = 300
10 -> 34 = 300

10 -> 35 = 300
10 -> 36 = 300
10 -> 37 = 300
10 -> 38 = 300
10 -> 39 = 300
Total cost of the shortest path tree is 12603

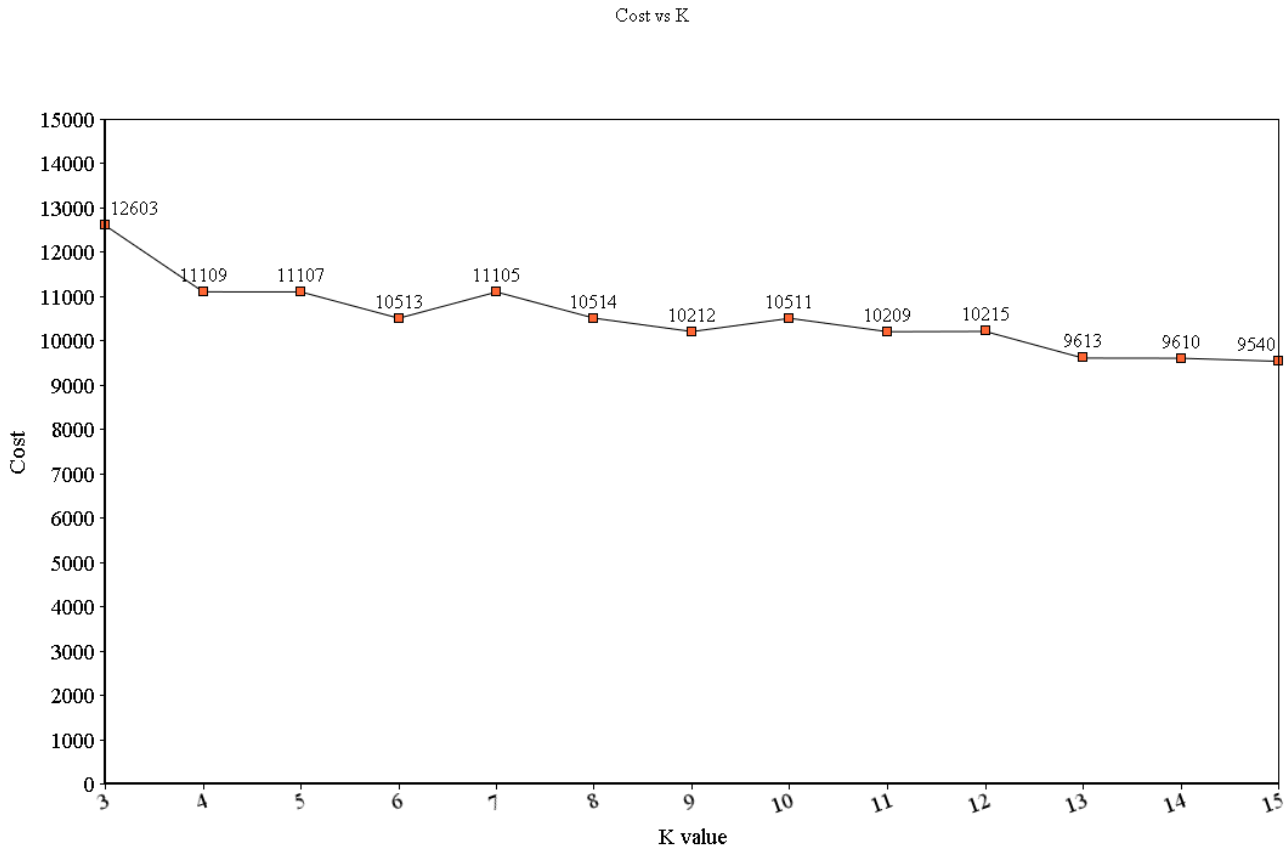
2. Nodes = 40, K=9

9 -> 0 = 300
9 -> 1 = 300
9 -> 2 = 300
9 -> 3 = 300
9 -> 4 = 300
9 -> 5 = 300
9 -> 6 = 1
9 -> 7 = 300
9 -> 8 = 300
9 -> 9 = 300
9 -> 10 = 300
9 -> 11 = 300
9 -> 12 = 1
9 -> 13 = 300
9 -> 14 = 300
9 -> 15 = 1
9 -> 16 = 300
9 -> 17 = 1
9 -> 18 = 300
9 -> 19 = 300
9 -> 20 = 300
9 -> 21 = 300
9 -> 22 = 300
9 -> 23 = 1
9 -> 24 = 300
9 -> 25 = 300
9 -> 26 = 1
9 -> 27 = 300
9 -> 28 = 300
9 -> 29 = 300
9 -> 30 = 300
9 -> 31 = 300
9 -> 32 = 300
9 -> 33 = 1
9 -> 34 = 300
9 -> 35 = 1
9 -> 36 = 300
9 -> 37 = 300

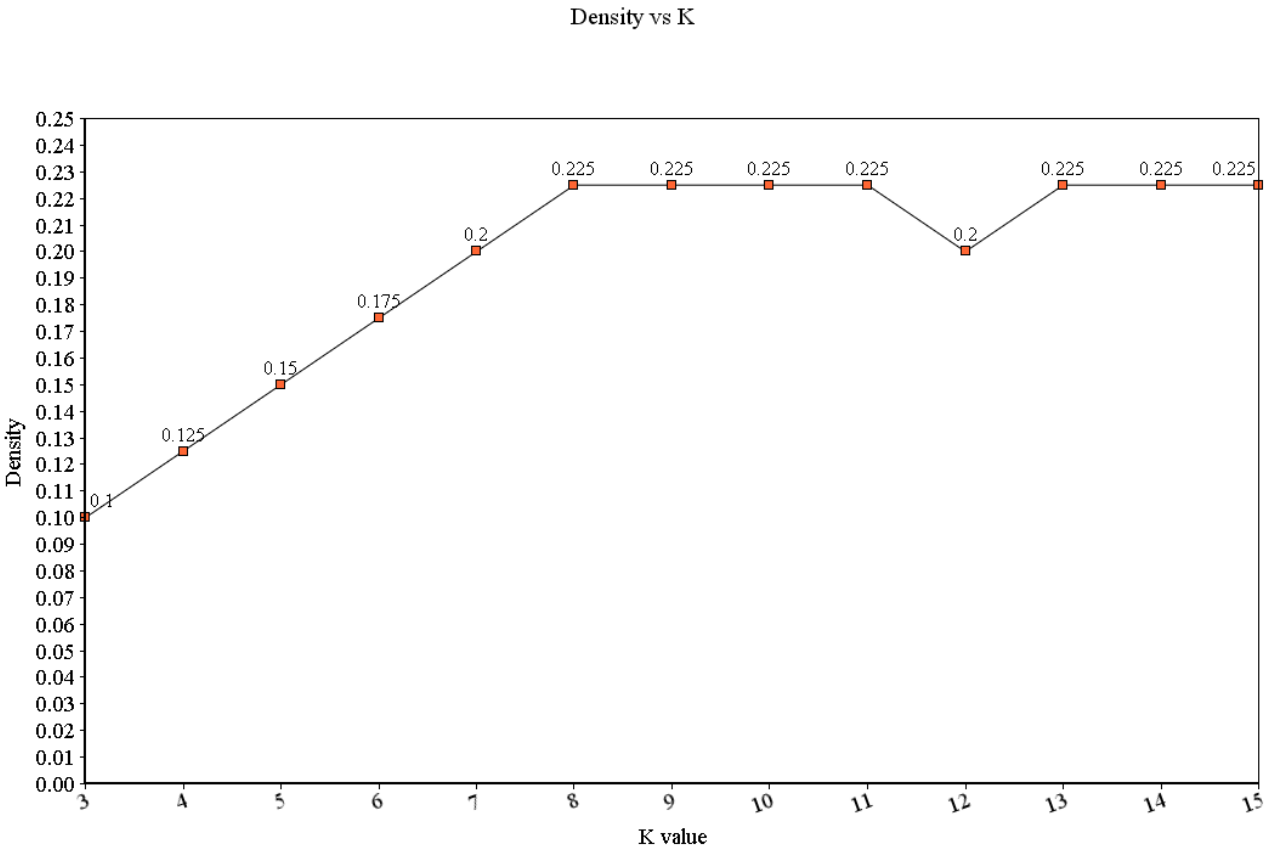
9 -> 38 = 300
9 -> 39 = 300
Total cost of the shortest path tree is 10514

6. Graphs:

1. Total cost vs K graph

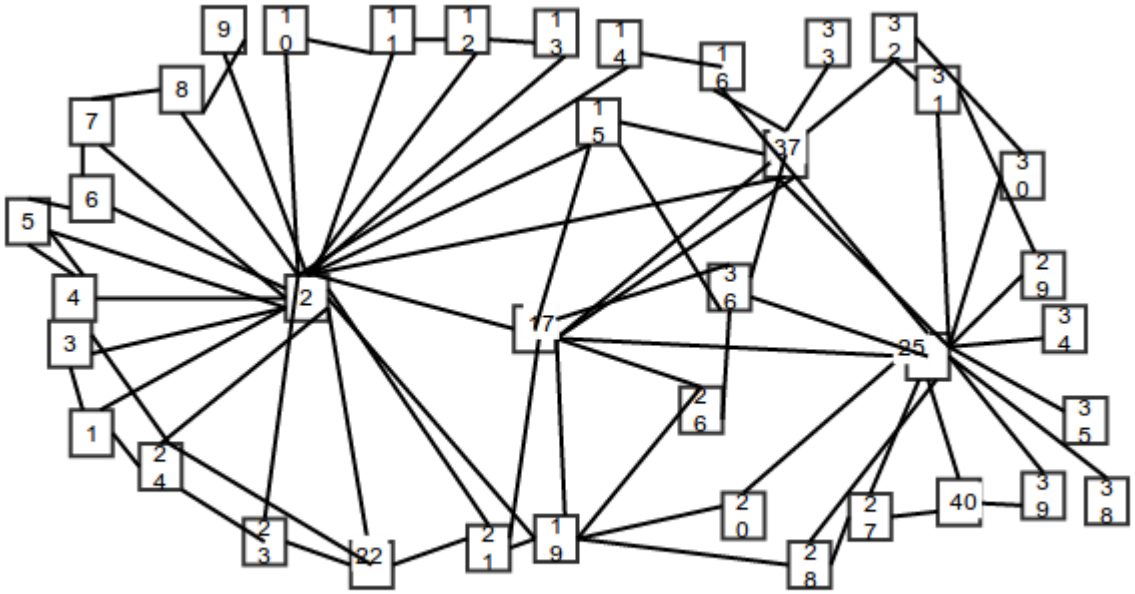


2. Density vs K graph



As k increases the number of edges increase, therefore the density also increases. And as the number of edges increase, the algorithm as more number of edges to choose from, therefore the cost decreases.

7. Network Topology:



8. Code:

```
/*
 * Project_1.cpp
 *
 *  Created on: Jun 14, 2014
 *      Author: sidhhu
 */
#include<stdio.h>
#include<iostream>
//#include<conio.h>
#include<vector>
#include<algorithm>
```

```
#include<math.h>
#include<stdlib.h>
using namespace std;
int cost[500][500],a[500][500],b[500][500],dist[500][500];


//Dijkstra's algorithm
int djikstra(int n,int cost[][500],int s[],int d[],int source,int b[][500])
{

	int i,j,u,v,small,cost_shortest = 0;
	int previous[40];
	for(i=0;i<n-1;i++)
	{
		small=300;
		u=0;
		for(j=0;j<n;j++)
		{
			if(s[j]==0 && d[j]<small)
			{
				small=d[j];
				u=j;
				printf("tree trace prev: %d for source %d \n" , u, source);
			}
		}
		s[u]=1;
		printf("tree trace prev: %d for source %d \n" , u, source);
		for(v=0;v<n;v++)
		{
			if(d[u]+cost[u][v] < d[v] && s[v]==0){
				d[v]=d[u]+cost[u][v];
```

```
                        }


                }
        }
//      for(i=0;i<n;i++)
//                      printf("%d -> %d = %d\n",source,i,d[i]);

        //Calculating the total cost of the shortest path tree
        //for(int j=0;j<n;j++){
        for(i=0;i<n;i++)
        {
                cost_shortest = cost_shortest + (d[i] * b[source][i]);
        }

        return cost_shortest;
}

int generateKRandom(int nodes){
        int k_random = rand() % nodes;
        return k_random;
}

int searchArray(int k_array[], int kRandom){
        int flag = 0;
        for(int i = 0; i<sizeof(k_array);i++){
                if(kRandom == k_array[i]){
                        flag = 1;
                }
        }
```

```
	return flag;
}

int main()
{
	int i,j,source,s[500],d[500],K,x;
	int cost_shortest=0,sum = 0,nodes,p,q;

	FILE *file = fopen("unit_costs.xls","w");
	FILE *file1 = fopen("traffic_demand.xls","w");
	printf("Enter the number of nodes\n");
	scanf("%d",&nodes);

	printf("Enter the value of K\n");
	scanf("%d",&K);
	int k_array[K];
//	printf("Enter the source node\n");
//	scanf("%d",&source);

	for(i=0;i<nodes;i++)
	{
		for(j=0;j<nodes;j++)//initialising the arrays
		{
			a[i][j]=0;
			dist[i][j]=0;
			b[i][j]=0;
			cost[i][j]=0;
		}
	}

	/* Assigning values to a[i][j] */
```

```
for(int i=0; i<K; i++){

        int kRandom = generateKRandom(nodes);
        while(searchArray(k_array, kRandom) == 1){
                kRandom = generateKRandom(nodes);
        }

        k_array[i] = kRandom;
        printf("the k_random array = %d", k_array[i]);

}

for(int i=0;i<nodes;i++){
                for(int j=0;j<nodes;j++){
                        if(searchArray(k_array, j) == 1){
                                a[i][j] = 1;
                        }
                        else
                                a[i][j] = 300;
                        cost[i][j] = a[i][j];
                }
}


/* Assigning random cost values to pairs of nodes
for (i=0;i<nodes;i++)
{
        for (j=0;j<nodes;j++)
        {
                if ((cost[i][j] != 200) && (i!=j))
                {
```

```
                        x = rand() % 9;
                        if (x != 0)
                        {
                                cost[i][j] = x;
                                a[i][j] = x;
                                if (cost[j][i] != 200)
                                {
                                        cost[j][i] = cost[i][j];
                                        a[j][i] = a[i][j];
                                }
                        }
                }
        }
}
*/
/* Generating random traffic demand values edit--> pick bij from [0,1,2,3]*/
for (i=0;i<nodes;i++)
{
        for (j=0;j<nodes;j++)
        {
                        b[i][j] = rand() % 4;
        }
}

/*displaying the link capacities */
fprintf(file,"\n The unit costs of the links are \n");
for (i=0;i<nodes;i++)
{
        for (j=0;j<nodes;j++)
        {
                fprintf(file,"\n %d -> %d : %d",i,j,a[i][j]);
```

```
        }
}

//Executing the Dijkstra's algorithm to find the shortest path in the network topology
printf("The shortest path tree constructed using Djikstra's algorithm is\n");
vector<int> shortVec;
vector<int>::iterator pos;
for(int j=0;j<nodes;j++){
for(i=0;i<nodes;i++)
{
        s[i]=0;
        d[i]=cost[j][i];
}

s[j]=1;
shortVec.push_back(djikstra(nodes,cost,s,d,j,b));
}
pos = min_element(shortVec.begin(), shortVec.end());

int min_index = pos - shortVec.begin();
for(i=0;i<nodes;i++)
                printf("%d -> %d = %d\n",min_index,i,d[i]);
printf("Total cost of the shortest path tree is %d\n\n",*pos);
/* Printing traffic demand values from source */
fprintf(file1,"\n\nThe traffic demand values from source are\n\n");
for(i=0;i<nodes;i++)
{
        fprintf(file1,"%d -> %d : %d \n",source,i,b[source][i]);
}

fclose(file);
```

```
        fclose(file1);
        //getch();
        return 0;
}
```

9. References:

[1] http://en.wikipedia.org/wiki/Dijkstra's_algorithm

[2] http://sourcecode4u.com/categories/data-structures/94-c-program-to-implement-dijkstra-algorithm

[3] http://www.c-program-example.com/2011/10/c-program-to-solve-dijkstras-algorithm.html

[4] Lecture Notes - Application to Network Design