CS 6385

Algorithmic Aspects of Telecommunication Networks SUMMER 2017

Project 1

An Application to Network  
Design

Submitted By

**Aravind R**

(**Net ID: axr156530**)

Instructor: Andras Farago

# Objective

In the project, we simulate a basic network model. The network simulations models network topology based on a given number of low cost links for each node in the network. Each node is assigned a predefined (k) number of low unit data cost outgoing links. All the other outgoing links are assigned a high unit cost value. The simulations calculates the cost of sending a demand value (dynamically generated) by finding the shorted path between any two nodes and then multiplying the total unit cost of that path with the demand value. The Simulations has to determine the lowest cost possible for the given network topology. Finally the project evaluates the relation between the number of low cost links (k- value) of the nodes with the total network cost and Network density.

# Program Methodology

The Project requires us to determine the minimum cost of the given network topology and load. Each of the nodes has a defined number of low cost outgoing links and the network is a fully connected network. In order to find the minimum cost network of any topology, we have to find the minimum path tree between a node to the all the other nodes. For example we need to find the minimum path from node j = 1 to all the other nodes k=2,3,4,…,30. For finding the minimum path between any two nodes, we can use the Dijkstra’s shortest path algorithm. On generation of the shortest path between any two nodes the calculation of the network cost simply becomes

*Network Cost = ∑ bjk*(*ai*1*,i*2 + *...* + *air*−1*,ir*)

Where bjk – The demand load to send between the node j and node k.

aik,ir – the unit cost of the link between intermediary nodes ik and ir in the shortest path.

# Implementation

**Technologies used**

|  |  |
| --- | --- |
| Programming Language | Java |
| Operating System | Windows 10 |
| IDE | NetBeans |
| Visualization Tools | Gephi, MS Excel,MS Visio |

**Program Details**

|  |  |
| --- | --- |
| Main Program | ShortestPath.java |

* The shortestPath.java contains the entire program.
* The main program generates the network topology using the input values of N and K.
* The low cost links are determined randomly using java Random function.
* dijkstra(int graph[][], int src) contains the Dijkstra’s path algorithm that calculate the shortest path for a source node to all other nodes.
* We use the Greedy implementation of Dijkstra’s Algorithm [1].
* The link[][] matrix keep track of all the used links in the shortest paths of the network
* b[] array contains the load for each of the links and it is determined by the formula loadij =|b[i] - b[j]|
* The program runs the shortest path algorithm with all the nodes as the source.
* The total cost is calculated by using cost of each optimal path.
* The Network topology is generated for each run of the program.

# Algorithm

The simulation can be done as follows:

1. Receive the number of nodes and number of low cost links(k) for each nodes as inputs from the user.
2. Generate the network topology by randomly assigning the low cost value to k outgoing links of each node.
3. Initialize the load matrix by repeating 10 digits 3 times.
4. For each of the node i , generate the shortest path with node i as the source and the rest of the other nodes as the destination using Dijkstra’s algorithm.
5. Mark the links in the shortest path as 1 in the solution path matrix Link[j][k]=1.
6. Calculate the cost of the shortest path between node i and every other node j using formula *cost i,j = ∑aij*
7. Generate the total Network cost using the formula: NW cost+= bi,j\*costi,j
8. If i!= final node , goto step 4
9. Print the Network cost.
10. Determine used link count = ∑ Link[j][k]==1
11. Determine the total link count = N\*(N-1) and unused count = Total count –used count.

# Flow Chart

**1) Intialization**



Generates the random network topology with each nodes having k low-cost outgoing links.

**2) Generating the Statistics**



**3) Dijkstra’s Algorithim**



# Running the Program

1. Copy the ShortestPath.java program to the location to run.
2. To Compile Javac ShortestPath.java.
3. To run the program Java ShortestPath.

# Topology generated

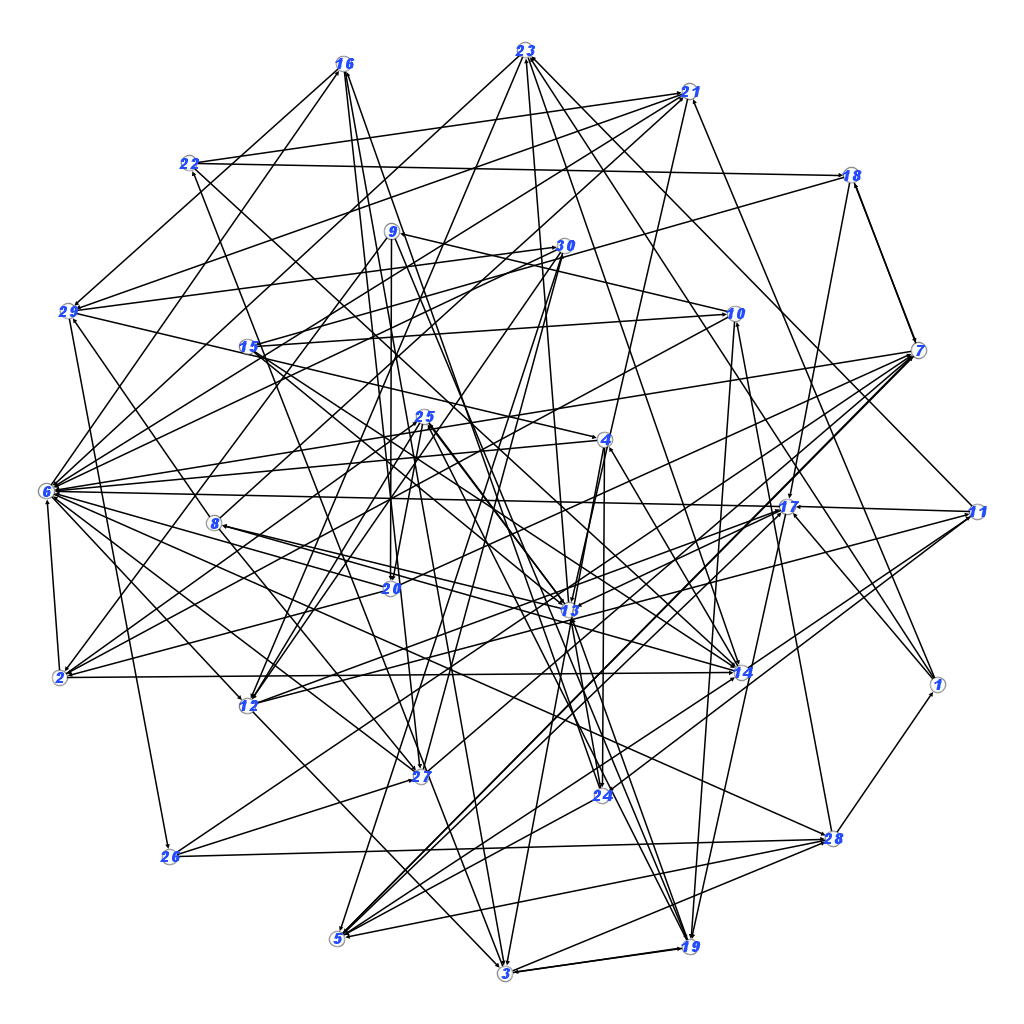
# **K=3**

**Used Link :** 90

**Unused Link :** 780

**Cost:** 6979

**Network Density :** 0.103448276



# **K=9**

# 

**Used Link :** 270

**Unused Link :** 600

**Cost:** 4078

**Network Density :** 0.310344828

# **K=15**

# 

**Used Link :** 450

**Unused Link :** 420

**Cost:** 3581

**Network Density :** 0.517241379

# Program

# package shortestpath;

# import java.util.\*;

# import java.util.Scanner;

# class ShortestPath

# {

# int V;

# int sum;

# int link[][];

# public ShortestPath(int n)

# {

# link=new int[n][n];

# }

# void set\_v(int x)

# {

# V=x;

# sum=0;

# }

# void printSolution(int dist[], int n,int src)

# {

# int b[]= {2,0,2,1,2,7,7,2,5,6,2,0,2,1,2,7,7,2,5,6,2,0,2,1,2,7,7,2,5,6};

# System.out.println("Vertex Distance from Source :"+src);

# for (int i = 0; i < V; i++)

# {

# int s=dist[i]\*Math.abs(b[src]-b[i]);

# sum+=s;

# System.out.println(i+" \t\t "+dist[i]+"\t\t"+s);

# }

# if((src+1)==V)

# {

# System.out.println("Total cost of Network : "+sum+"\nlinks: "+"\n");

# int cnt=0;

# for(int k=0;k<V;k++)

# {

# for(int j=0;j<V;j++)

# {

# if(link[k][j]==1&& k!=j)

# {

# System.out.println((k+1)+","+(j+1));

# cnt++;

# }

# }

# }

# System.out.println("Used link count:"+cnt+"\nUnused links:"+(870-cnt));

# }

# }

# 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;

# }

# void dijkstra(int graph[][], int src)

# {

# int dist[] = new int[V];

# Boolean sptSet[] = new Boolean[V];

# for (int i = 0; i < V; i++)

# {

# dist[i] = Integer.MAX\_VALUE;

# sptSet[i] = false;

# }

# dist[src] = 0;

# for (int count = 0; count < V-1; count++)

# {

# int x=0;

# int u = minDistance(dist, sptSet);

# sptSet[u] = true;

# for (int v = 0; v < V; 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];

# //System.out.println("link : "+u+" - "+v+" : "+dist[v]);

# x=v;

# if(graph[u][v]==1)

# {

# link[u][v]=1;

# }

# }

# printSolution(dist, V,src);

# }

# public static void main (String[] args)

# {

# System.out.println("Enter the nmber of nodes:");

# Scanner in = new Scanner(System.in);

# int no\_of\_nodes = in.nextInt();

# System.out.println("Enter the value of k:");

# int k = in.nextInt();

# Random r=new Random();

# int graph[][] = new int[no\_of\_nodes][no\_of\_nodes];

# for(int i=0;i<no\_of\_nodes;i++)

# {

# for(int j=0;j<no\_of\_nodes;j++)

# {

# if(i!=j)

# graph[i][j]=300;

# }

# }

# int cnt[]=new int[no\_of\_nodes];

# for(int i=0;i<no\_of\_nodes;i++)

# {

# while(cnt[i]<k)

# {

# int ind=(r.nextInt(10000)%no\_of\_nodes);

# if(graph[i][ind]!=1 && ind!=i)

# {

# graph[i][ind]=1;

# cnt[i]++;

# }

# }

# }

# for(int i=0;i<no\_of\_nodes;i++)

# {

# System.out.print("graph["+i+"]:\t");

# for(int j=0;j<no\_of\_nodes;j++)

# {

# System.out.print("\t" +graph[i][j]);

# }

# System.out.println("\n");

# }

# ShortestPath t = new ShortestPath(no\_of\_nodes);

# t.set\_v(no\_of\_nodes);

# for(int i=0;i<no\_of\_nodes;i++)

# {

# t.dijkstra(graph, i);

# }

# 

# }

# }

# Analysis

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **K** | **Total links** | **Unused Links** | **Used Links** | **Desnity** | **Cost** |
| 30 | 3 | 870 | 780 | 90 | 0.103448276 | 6979 |
| 30 | 4 | 870 | 750 | 120 | 0.137931034 | 5831 |
| 30 | 5 | 870 | 720 | 150 | 0.172413793 | 5213 |
| 30 | 6 | 870 | 690 | 180 | 0.206896552 | 4837 |
| 30 | 7 | 870 | 660 | 210 | 0.24137931 | 4584 |
| 30 | 8 | 870 | 630 | 240 | 0.275862069 | 4243 |
| 30 | 9 | 870 | 600 | 270 | 0.310344828 | 4078 |
| 30 | 10 | 870 | 570 | 300 | 0.344827586 | 4037 |
| 30 | 11 | 870 | 540 | 330 | 0.379310345 | 3918 |
| 30 | 12 | 870 | 510 | 360 | 0.413793103 | 3833 |
| 30 | 13 | 870 | 480 | 390 | 0.448275862 | 3771 |
| 30 | 14 | 870 | 450 | 420 | 0.482758621 | 3660 |
| 30 | 15 | 870 | 420 | 450 | 0.517241379 | 3581 |

# As the K value increases the Network density also Increases. This is because the number of available paths for the shortest path also increases. This results in a denser Network

# Cost decreases as the K value increases.

# Used links count increase as k value increases.

# Conclusions

# **1) Total cost of the network depends on *k***

|  |  |
| --- | --- |
| K | Cost |
| 3 | 6979 |
| 4 | 5831 |
| 5 | 5213 |
| 6 | 4837 |
| 7 | 4584 |
| 8 | 4243 |
| 9 | 4078 |
| 10 | 4037 |
| 11 | 3918 |
| 12 | 3833 |
| 13 | 3771 |
| 14 | 3660 |
| 15 | 3581 |

# The values obtained by inputting various values k , we see that the cost of the network reduces as the k value increases. This is because as the number of the low cost links per node increases then the nodes reachable though low cost nodes increase. This indirectly results in lower overall cost. Dijkstra’s algorithm always selects the lowest cost path. As increase in K value results in more number of nodes reachable from the current considered node through lower cost links, the network cost reduces.

# **2)** **Density of the obtained network depends on k**

|  |  |
| --- | --- |
| K | Density |
| 3 | 0.103448 |
| 4 | 0.137931 |
| 5 | 0.172414 |
| 6 | 0.206897 |
| 7 | 0.241379 |
| 8 | 0.275862 |
| 9 | 0.310345 |
| 10 | 0.344828 |
| 11 | 0.37931 |
| 12 | 0.413793 |
| 13 | 0.448276 |
| 14 | 0.482759 |
| 15 | 0.517241 |

# The network density is given by the formula

# Network Density = Used Links/ Total links in NW

# Where

# Total Links = N \* (N-1)

# From the above graph plot, we can see that as the k value increases, the density of the network also increases. The is mainly because the increase in low cost links per node will result in the increase in the number of nodes reachable through them . Dijkstra’s Algorithim always select the lower cost path for reaching the destination. As a result almost all of the low cost links are selected by the algorithim. That is the reason why the Network density is linearly proportional to the K value of the network.

# Output

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Reference

# Class notes

# Dijkstra’s Algorithim <http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/>

# [www.gephi.org](http://www.gephi.org)

# Greedy Algorithm : <https://en.wikipedia.org/wiki/Greedy_algorithm>

# <http://www.sanfoundry.com/java-program-represent-graph-adjacency-matrix/>