

# 6385.0W1 Algorithmic Aspects of Telecommunication Networks

## Project 2

Name: Adarsh Raghupati

NetID: axh190002

## Contents

Problem statement: .....	1
Assumptions on the reliability model: .....	2
Algorithm: .....	2
Part A. Different values of $p$ .....	2
Part B. Fixed $p=0.9$ and $k$ random flipped states .....	3
Graphical representation and analysis of the result:.....	3
Relationship between network reliability and link reliability: .....	3
Relationship between network reliability and random flipped states: .....	4
ReadMe: .....	5
Technologies/Tools used: .....	5
How to run the program: .....	5
Appendix: .....	5
Source Code: .....	5
Reference.....	12

## Problem statement:

The aim of the project is to study experimentally how the network reliability depends on the individual link reliabilities, in the specific situation described below.

**Network topology:** A complete undirected graph on  $n = 5$  nodes. Complete means that every node is connected with every other one (parallel edges and self-loops are excluded in this graph). Therefore, this graph has  $m = 10$  edges, representing the links of the network.

**Components that may fail:** The links of the network may fail. The nodes are always up. The reliability of the links is computed as follows. Index the links with the numbers  $1, \dots, 10$  (in arbitrary order). Take a parameter  $p$ ,  $0 \leq p \leq 1$ , the same for every link. The parameter  $p$  will take different values in the experiments. For link  $i$ , ( $i = 1, 2, \dots, 10$ ), let its reliability be  $p_i$ , which is given by the following formula

$$p_i = p^{\lceil \frac{d_i}{3} \rceil}$$

where  $d_i$  is the  $i$ th digit in your 10-digit student ID, assuming the ID consists of the digits  $d_1, d_2, \dots, d_{10}$ . The  $\lceil \dots \rceil$  sign means upper integer part, which rounds up any non-integer number to the nearest larger integer. For example, if your ID is 0123456789, then link 5 has reliability  $p_5 = p^{\lceil \frac{d_5}{3} \rceil} = p^2$ , and link 1 has reliability  $p_1 = p^{\lceil \frac{d_1}{3} \rceil} = p^0 = 1$ .

**Reliability configuration:** The system is considered operational if the network is connected.

## Assumptions on the reliability model:

- Each component has two possible states: operational or failed.
- The failure of each component is an independent event.
- Component  $i$  is functioning (operational) with probability  $p_i$  and is in-operational (failed) with probability  $1 - p_i$ . (These probabilities are usually known.)
- The reliability  $R$  of the system is some function of the component reliabilities:  $R = f(p_1, p_2, \dots, p_N)$  where  $N$  is the number of components.

The function  $f(\dots)$  above depends on the configuration, which defines when the system is considered operational, given the states of the components.

Given that network topology is a complete graph and nodes are always up. Hence the network reliability depends on individual link reliability. System is said to be operational if the graph is connected.

## Algorithm:

The network topology has 5 nodes and 10 edges. If there is an edge between two nodes, then that link is said to be up. If there is no edge between two nodes, then that node is said to be down. As per exhaustive enumeration approach, the algorithm will find the network reliability for all the combinations of link states. Hence the total number of states  $2^{10} = 1024$ .

### Part A. Different values of $p$

- Generate all 1024 states of the link by converting the number to binary representation
- Number 0 corresponds to "0000000000"
- Number 1 corresponds to "0000000001"
- In the above representation 1 indicates that the link is up and 0 indicates that link is down
- For each link state check if the graph is connected by depth first search
- For each link state calculate the reliability as follows:

```

If linkState==UP
    reliability = reliability*p
else if linkState==DOWN
    reliability = reliability*(1-p)

```

where  $p$  is the probability that link is UP. Individual link probability is generated according to the given rules.

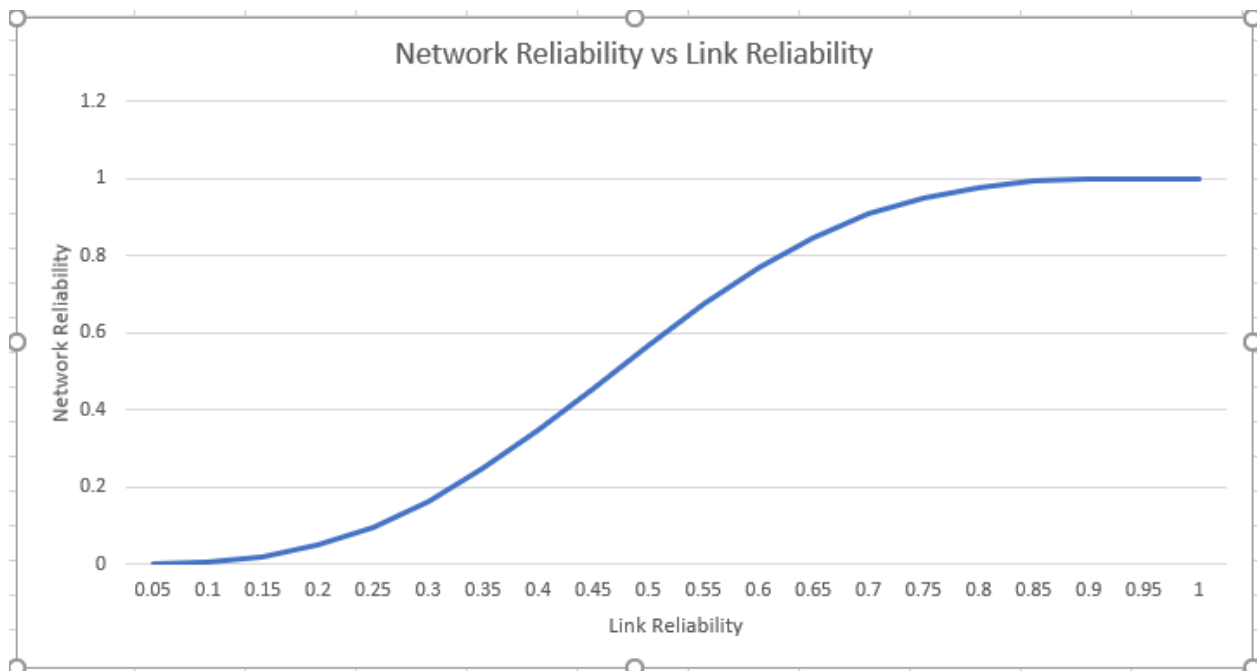
- Calculate the network reliability by adding the reliability of all 1024 combinations of network
- Repeat the above steps for  $p$  value ranging from [0.05,1] in steps of 0.05

### Part B. Fixed $p=0.9$ and $k$ random flipped states

- Randomly select  $k$  number of states from possible 1024 and flip the state by setting 1 to 0 and 0 to 1 in the binary representation.
- Repeat the steps from part 1 with fixed value 0.9 for  $p$
- Repeat the process for  $k$  values 0,1,2,...20
- To reduce the effect of randomness, calculate the reliability for each  $k$  value multiple times and find the average

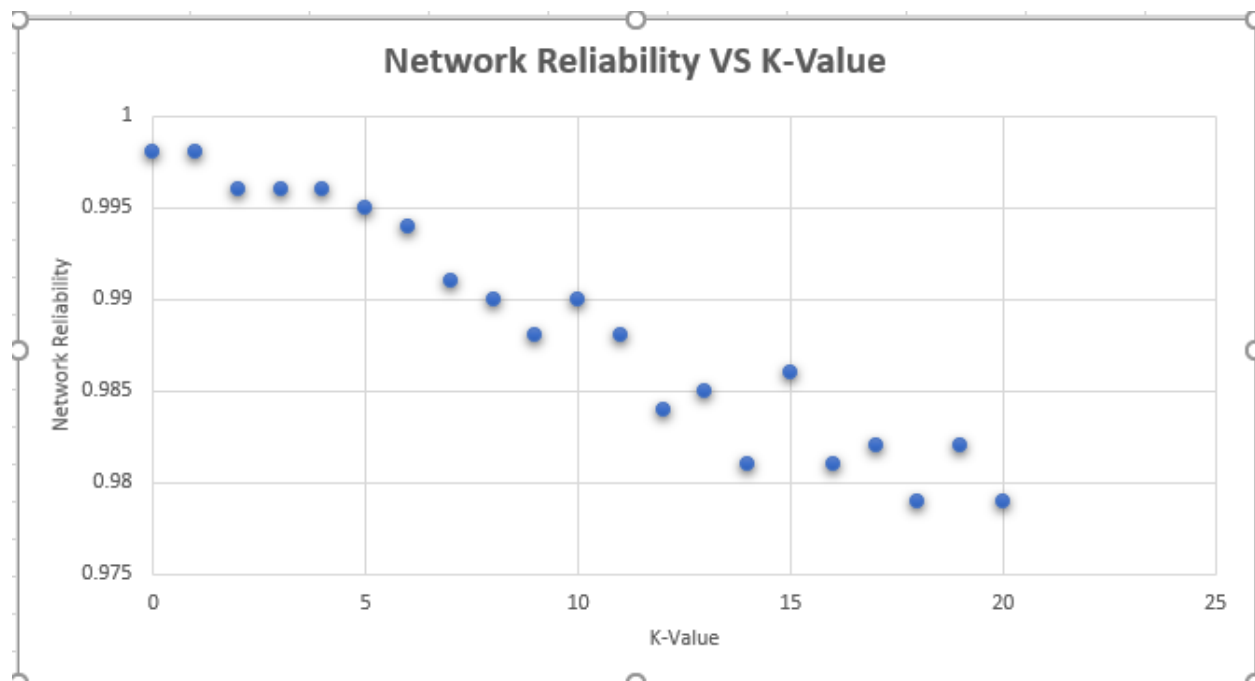
## Graphical representation and analysis of the result:

Relationship between network reliability and link reliability:



- From the above diagram it can be observed that there is a strong relationship between network reliability and link reliability.
- The chart follows logistic/biological curve, with slow increment initially followed by rapid growth and then steady growth after certain threshold.
- For the link reliability above 0.8, the network reliability is almost constant.
- Hence by maintaining link reliability higher than 0.8 we can obtain more reliable network

Relationship between network reliability and random flipped states:



- From the above figure, we can see that the variation in network reliability is very small with the range 0.998 to 0.979.
- Even though there is an overall downward trend, there are random spike for few values of k
- When the number of sample component states (k) increases, the network reliability is expected to decrease but flipping the states adds more randomness.

## ReadMe:

### Technologies/Tools used:

Programming language: Java

IDE: IntelliJ

Visualization tool: MS Excel

### How to run the program:

- Create an empty java project in IntelliJ
- Create a package called project2
- Create the java files Graph.java and NetworkReliabilityFinder.java
- Copy the code from the appendix section to respective java files
- Run the NetworkReliabilityFinder.java
- Output is generated at console and two csv files output1.csv, output2.csv are generated
- Open the output csv files and visualize using MS Excel

## Appendix:

### Source Code:

Graph.java

```
package project2;
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
public class Graph {
```

```
    public static int[] studentID = new int[]{2,0,2,1,4,8,6,1,1,5};
```

```
    //Data structures to store edge and node information
```

```
    public int adjMatrix[][];
```

```
    public int edgeToNodesMap[][];
```

```
    Map<String,Integer> nodesToEdgeMap;
```

```
    //Probability to generate link reliability values
```

```
    public double p;
```

```
    public Graph(){
```

```
        p = 0;
```

```
        adjMatrix = new int[5][5];
```

```
        edgeToNodesMap = new int[10][2];
```

```
        nodesToEdgeMap = new HashMap<>();
```

```
        int key = 0;
```

```
        for(int i = 0; i < 5; i++){
```

```

        for(int j = 0; j < 5; j++){
            if (i != j) {
                if(i > j) {
                    edgeToNodesMap[key][0]= i;
                    edgeToNodesMap[key][1]= j;
                    String str = ""+i+","+j;
                    nodesToEdgeMap.put(str,key);
                    key++;
                }
                adjMatrix[i][j]=1;
            }
            else
                adjMatrix[i][j]=0;
        }
    }
}

/**
 * Depth first search algorithm. Helper function to check the graph connectivity
 * @param i
 * @param visited
 */
public void dfs(int i, boolean visited[]) {

    visited[i] = true;
    for(int j = 0;j < 5; j++){
        if(adjMatrix[i][j] == 1){
            if(!visited[j]){
                dfs(j, visited);
            }
        }
    }
}

/**
 * Check the graph connectivity to decide whether the network state is UP or DOWN
 * @return
 */
public boolean isGraphConnected(){
    boolean seen[] = new boolean[5];
    boolean isConnected = true;
    for(int i = 0; i < 5; i++){
        seen[i] = false;
    }

    dfs(0, seen);

    for(int i = 0;i < 5; i++){
        if(!seen[i]){
            isConnected = false;
        }
    }
    return isConnected;
}

```

```

    /**
     * Calculates the reliability of the network topology by using individual link
    reliability
     * @return
     */
    public double calculateReliability() {
        double reliability = 1;
        for(int i = 0; i < 5; i++){
            for(int j = 0; j < 5; j++){
                if (i > j){

                    //if the link is up
                    if (adjMatrix[i][j] == 1) {
                        reliability = reliability * getReliability(i,j, p);
                    }
                    //if the link is down
                    else {
                        reliability = reliability * (1 - getReliability(i,j, p));
                    }

                }
            }
        }
        return reliability;
    }

    /**
     * Generates the link reliability based on the student ID
     * @param i indicates i'th node
     * @param j indicates j'th node
     * @param p given probability
     * @return
     */
    public double getReliability(int i,int j, double p){
        String str = ""+i+","+j;
        int value = (int) Math.ceil(studentID[nodesToEdgeMap.get(str)]/3.0);
        return Math.pow(p,value);
    }

}

```

NetworkReliabilityFinder.java

```

package project2;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

```

```

public class NetworkReliabilityFinder {

    /**
     * Calculate network reliability for given p and k value
     * @param p is link reliability
     * @param k is number of random network states
     * @return
     */
    public static double findReliability(double p, int k)
    {
        Set<Integer> randomStateSet = new HashSet<>();
        for(int j = 0; j < k; j++) {
            Random rand = new Random();
            int randomNumber = rand.nextInt(1024);
            while(randomStateSet.contains(randomNumber)){
                randomNumber = rand.nextInt(1024);
            }
            randomStateSet.add(randomNumber);
        }

        double R = 0;
        for(int i = 0; i < 1024; i++) {
            Graph Graph= new Graph();
            Graph.p = p;
            String linkStates = String.format("%10s",
Integer.toBinaryString(i)).replace(" ", "0");
            for(int j = 0; j < 10; j++){
                if (linkStates.charAt(j) == '1'){

Graph.adjMatrix[Graph.edgeToNodesMap[j][0]][Graph.edgeToNodesMap[j][1]] = 0;
Graph.adjMatrix[Graph.edgeToNodesMap[j][1]][Graph.edgeToNodesMap[j][0]] = 0;
                }
            }

            if(randomStateSet.contains(i)){
                if(!Graph.isGraphConnected()){
                    R = R + Graph.calculateReliability();
                }
            }
            else{
                if(Graph.isGraphConnected()){
                    R = R + Graph.calculateReliability();
                }
            }
        }

        return R;
    }
}

```



```

public static void main(String[] args) throws IOException {

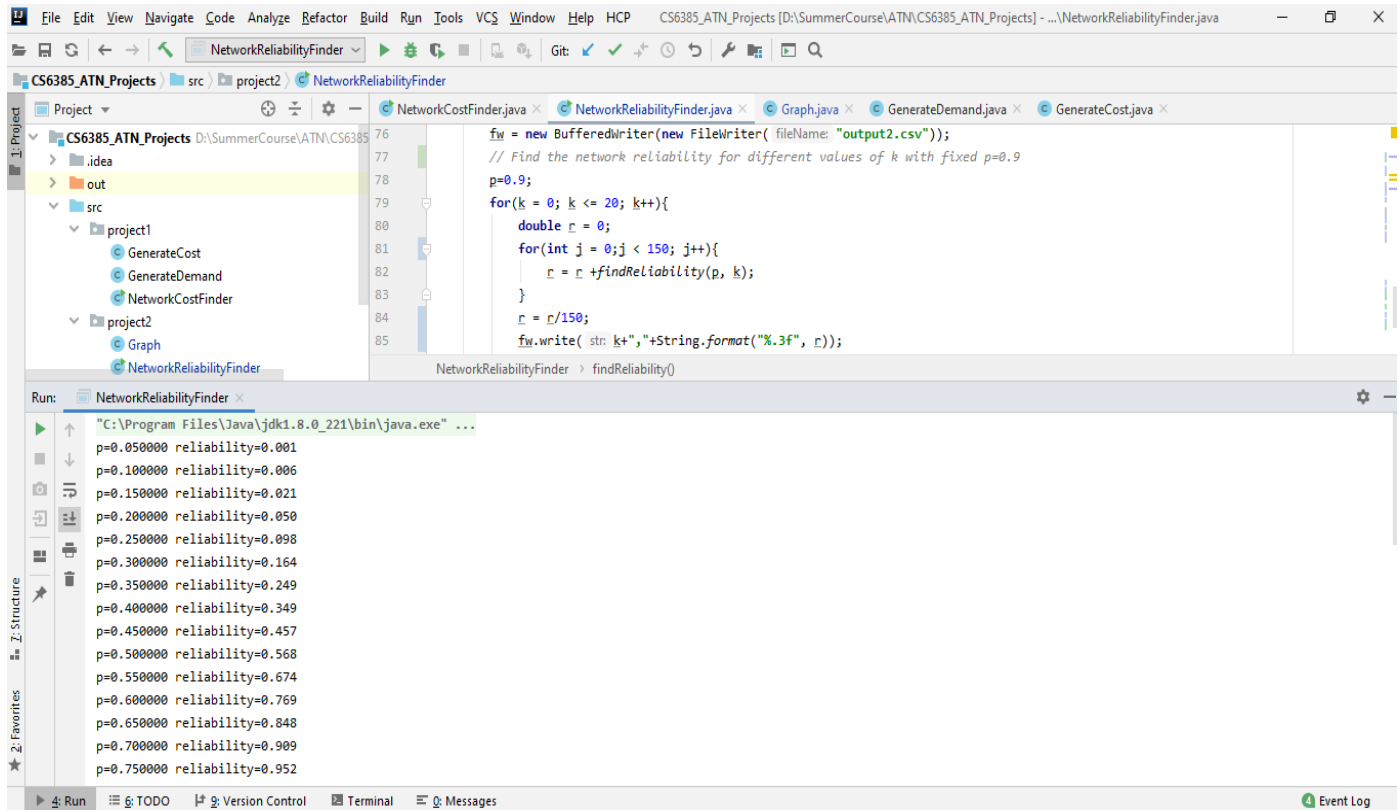
    int k=0;
    double p;
    BufferedWriter fw = new BufferedWriter(new FileWriter("output1.csv"));
    // Find the network reliability for different values of p
    for ( p = 0.05; p < 1.05; p += 0.05) {
        double r = findReliability(p,k);
        fw.write(p+","+String.format("%.3f", r));
        fw.newLine();
        System.out.println("p="+String.format("%.2f",p)+"
reliability="+String.format("%.3f", r));
    }

    fw.close();
    System.out.println();
    fw = new BufferedWriter(new FileWriter("output2.csv"));
    // Find the network reliability for different values of k with fixed p=0.9
    p=0.9;
    for(k = 0; k <= 20; k++){
        double r = 0;
        for(int j = 0; j < 150; j++){
            r = r +findReliability(p, k);
        }
        r = r/150;
        fw.write(k+","+String.format("%.3f", r));
        fw.newLine();
        System.out.println("k="+k+" reliability="+String.format("%.3f", r));
    }
    fw.close();
}
}

```

## Program Output:

Console output:



The screenshot shows an IDE window titled "CS6385\_ATN\_Projects [D:\SummerCourse\ATN\CS6385\_ATN\_Projects] - ...NetworkReliabilityFinder.java". The project structure on the left includes "CS6385\_ATN\_Projects" with subfolders ".idea", "out", and "src". The "src" folder contains "project1" and "project2", each with several Java files. The "NetworkReliabilityFinder.java" file is open in the editor, showing a method "findReliability()" that calculates network reliability for different values of k (0 to 20) with a fixed p=0.9. The code uses a nested loop to calculate the reliability for each k value and writes the results to a file named "output2.csv".

```
76 fw = new BufferedWriter(new FileWriter( fileName: "output2.csv"));
77 // Find the network reliability for different values of k with fixed p=0.9
78 p=0.9;
79 for(k = 0; k <= 20; k++){
80     double r = 0;
81     for(int j = 0; j < 150; j++){
82         r = r + findReliability(p, k);
83     }
84     r = r/150;
85     fw.write( str: k+", "+String.format("%.3f", r));
```

The console output shows the results of the program execution, displaying the reliability values for each k value from 0 to 20. The output is as follows:

```
Run: NetworkReliabilityFinder
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
p=0.050000 reliability=0.001
p=0.100000 reliability=0.006
p=0.150000 reliability=0.021
p=0.200000 reliability=0.050
p=0.250000 reliability=0.098
p=0.300000 reliability=0.164
p=0.350000 reliability=0.249
p=0.400000 reliability=0.349
p=0.450000 reliability=0.457
p=0.500000 reliability=0.568
p=0.550000 reliability=0.674
p=0.600000 reliability=0.769
p=0.650000 reliability=0.848
p=0.700000 reliability=0.909
p=0.750000 reliability=0.952
```

Output CSV file contents:

Output1.csv has network reliability vs link reliability values

Link Reliability	Network Reliability
0.05	0.001
0.1	0.006
0.15	0.021
0.2	0.05
0.25	0.098
0.3	0.164
0.35	0.249
0.4	0.349
0.45	0.457
0.5	0.568
0.55	0.674
0.6	0.769
0.65	0.848
0.7	0.909
0.75	0.952
0.8	0.979
0.85	0.993
0.9	0.998
0.95	1
1	1

Output2.csv has network reliability vs k data

K	Network Reliability
0	0.998
1	0.998
2	0.997
3	0.996
4	0.995
5	0.993
6	0.994
7	0.989
8	0.99
9	0.992
10	0.988
11	0.987
12	0.989
13	0.987
14	0.98
15	0.984
16	0.984
17	0.979
18	0.982

## Reference

1. <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>
2. Lecture notes