Project 3

Implementation of Network Reliability

CS 6385

Vyoma Trivedi

(vmt160030)

Table of Content

Objective:

The theme 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. This means, every node is connected with every other one (parallel edges and self-loops are excluded in this graph). As a result, 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 each link is p, the same for every link. The parameter p will take different values in the experiments.
- **Reliability configuration:** The system is considered operational, if the network is connected

Algorithm:

We can list down all possible states and assign "up" i.e. p=0 and "down" i.e. p=1 system condition to each state. Then the reliability can be obtained by summing the probability of the up states.

Even though the obtained expression may be simplified, this method is practical only for small systems, due to exponential growth of number of states. For N components there are 2^N possible states making exhaustive enumeration a non-scalable solution.

The algorithm can be described as below:

1) Declare and initialize variables. Input the number of nodes which is n=5 and take an adjacency matrix representing the graph having all values 0 which means that no components are down i.e. the system is up.

2) Increment the probability of each link by 0.05.

3) Based on the requirement that there are no parallel edges and self-loops, the exact number of distinct edges is found to be 10.

4) Using the number of edges, determine the $2^{(\text{edge combinations of selecting and deselecting the edges to be considered})}$.

5) Determine those sets of the links states that have the up condition for the network.

6) Iterate N times for N components and calculate the reliability values.

7) Calculate the resulting probabilities of those combinations which have an "up" condition for the network and sum it to get the reliability of the entire network.

A sample calculation for 5 nodes is as shown below:

Number of Component Failures	Event	System Condition
0	1. $ABCDE$	Up
1	2. $\overline{A}BCDE$	Up
	3. $A\overline{B}CDE$	Up
	4. $AB\overline{C}DE$	Up
	5. $ABC\overline{D}E$	Up
	6. $ABCD\overline{E}$	Up
2	7. $\overline{A}\overline{B}CDE$	Down
	8. $\overline{A}B\overline{C}DE$	Up
	9. $\overline{A}BC\overline{D}E$	Up
	10. $\overline{A}BCD\overline{E}$	Up
	11. $A\overline{B}\overline{C}DE$	Up
	12. $A\overline{B}C\overline{D}E$	Up
	13. $A\overline{B}CD\overline{E}$	Up
	14. $AB\overline{C}\overline{D}E$	Up
	15. $AB\overline{C}D\overline{E}$	Up
	16. $ABC\overline{D}\overline{E}$	Down
3	17. $A\overline{B}\overline{C}\overline{D}E$	Down
	18. $A\overline{B}C\overline{D}\overline{E}$	Down
	19. $A\overline{B}\overline{C}D\overline{E}$	Up
	20. $AB\overline{C}\overline{D}\overline{E}$	Down
	21. $\overline{A}BC\overline{D}\overline{E}$	Down
	22. $\overline{A}B\overline{C}D\overline{E}$	Down
	23. $\overline{A}B\overline{C}\overline{D}E$	Up
	24. $\overline{A}\overline{B}CD\overline{E}$	Down
	25. $\overline{A}\overline{B}C\overline{D}E$	Down
	26. $\overline{A}\overline{B}\overline{C}DE$	Down
4	27. $\overline{A}\overline{B}\overline{C}D\overline{E}$	Down
	28. $\overline{A}\overline{B}C\overline{D}\overline{E}$	Down
	29. $\overline{A}B\overline{C}\overline{D}\overline{E}$	Down
	30. $\overline{A}\overline{B}\overline{D}\overline{D}\overline{E}$	Down
	31. $\overline{A}\overline{B}\overline{C}\overline{D}E$	Down
5	32. $\overline{A}\overline{B}\overline{C}\overline{D}\overline{E}$	Down

$$
\begin{aligned}
R_{\text{network}} = {} & R_A R_B R_C R_D R_E + (1 - R_A) R_B R_C R_D R_E \\
& + R_A (1 - R_B) R_C R_D R_E + R_A R_B (1 - R_C) R_D R_E \\
& + R_A R_B R_C (1 - R_D) R_E + R_A R_B R_C R_D (1 - R_E) \\
& + (1 - R_A) R_B (1 - R_C) R_D R_E + (1 - R_A) R_B R_C (1 - R_D) R_E \\
& + (1 - R_A) R_B R_C R_D (1 - R_E) + R_A (1 - R_B)(1 - R_C) R_D R_E \\
& + R_A (1 - R_B) R_C (1 - R_D) R_E + R_A (1 - R_B) R_C R_D (1 - R_E) \\
& + R_A R_B (1 - R_C)(1 - R_D) R_E + R_A R_B (1 - R_C) R_D (1 - R_E) \\
& + R_A (1 - R_B)(1 - R_C) R_D (1 - R_E) \\
& + (1 - R_A) R_B (1 - R_C)(1 - R_D) R_E
\end{aligned}
$$

Pseudo code:

- Generate 2^{10} possible states randomly using the Random function and then convert them to binary so that we have 0/1 value for each link where 0 means system is UP and 1 means system is DOWN(ExhaustiveEnumeration()).

- Based on the state of each link calculate the reliability for those states that leave the system in UP condition i.e. the system is connected (which can be checked using the CheckConnected() function).

- Add up the reliabilities of all those UP states.

- Then using the ExhaustiveFlipped() function we flip all the bits and calculate the reliability and see how K affects it.

Goals:

Debugging: It is possible by using print statements at various points. Moreover, null values and other exceptions have also been taken care of.

Correctness: Adjacency matrix is used for checking the correctness and from the produced outputs we can see that probability and reliability values are between 0 and 1 and that is what was desired.

Changes: They are easy to incorporate as we have used a modular approach so one change in a function can impact the whole program without requiring to make multiple changes.

Results:

- **Link Probability and Reliability**

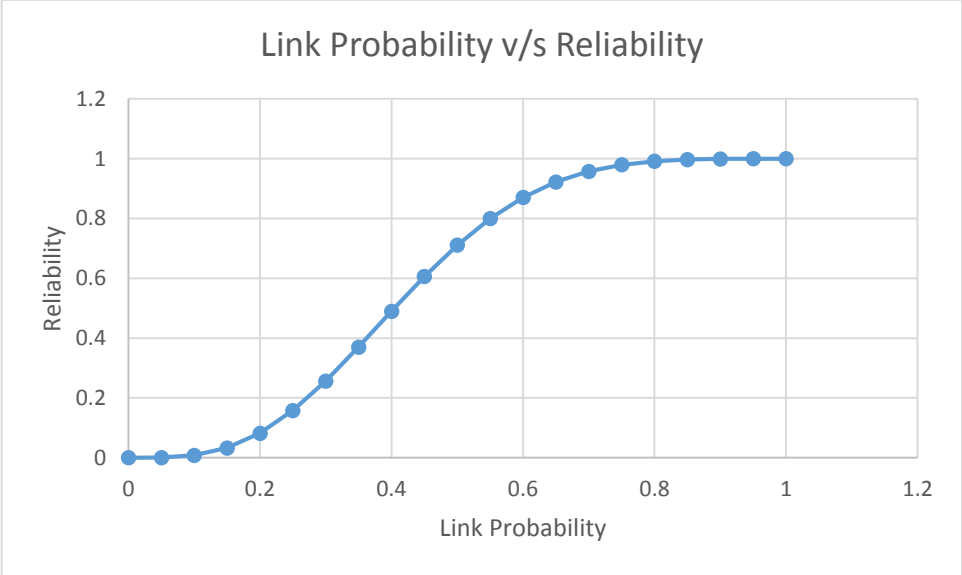
Link Probability	Reliability
0	0
0.05	0.00063
0.1	0.008097
0.15	0.0327
0.2	0.08194
0.25	0.15769
0.3	0.2562
0.35	0.37005
0.4	0.48965
0.45	0.6058
0.5	0.71093
0.55	0.7998
0.6	0.87025
0.65	0.9221
0.7	0.9575
0.75	0.9794
0.8	0.9916
0.85	0.9974
0.9	0.9994
0.95	0.9999
1	1

- **K and Reliability**

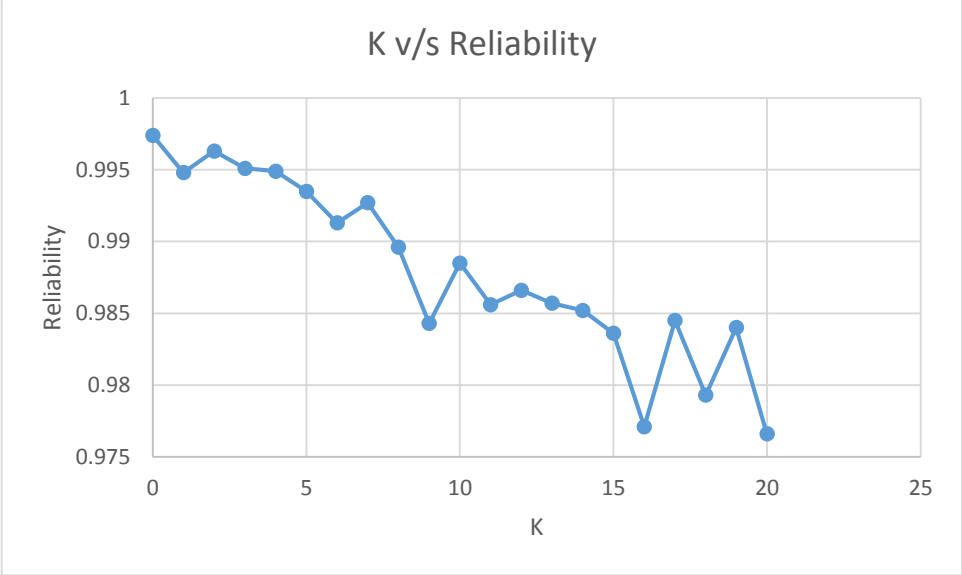
K	Reliability
0	0.9974
1	0.9948
2	0.9963
3	0.9951
4	0.9949
5	0.9935
6	0.9913
7	0.9927
8	0.9896
9	0.9843
10	0.9885
11	0.9856
12	0.9866

13	0.9857
14	0.9852
15	0.9836
16	0.9771
17	0.9845
18	0.9793
19	0.984
20	0.9766

Charts:



Like we can see from the graph above reliability increases with increase in link probability and after some time it starts approaching 1 and the graph starts becoming constant.



Like we can see from the graph above Reliability mostly decreases with increase in value of k but at some values it also increases. That is because more states are flipped from UP to DOWN.

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 atn_network_reliability;
import java.util.ArrayList;
import java.util.Random;

/**
 *
 * @author VYOMA
 */
class Graph {
        public int graph[][];
        public int src_des[][];
        public double link_prob;
```

```
public Graph(){

        link_prob = 0;
        graph= new int[5][5];
        src_des = new int[10][2];
        int node = 0;

        //adjacency matrix representation of a connected graph having 5 nodes
        for(int i = 0; i < 5; i++){
            for(int j = 0; j < 5; j++){
                if (i != j) {
                    if(i > j) {
                        src_des[node][0]= i;
                        src_des[node][1]= j;
                        node++;
                    }
                    graph[i][j]=1;
                }
                else
                    graph[i][j]=0;
            }
        }
}
```

//Function to check if the graph is connected to calculate the relaibility of only those components that leave

//the system in UP state.

```
public boolean CheckConnected(){
        boolean flag = true;
```

```
        //to keep a track of visited nodes
        boolean processed[] = new boolean[5];

        for(int i = 0; i < 5; i++){
            processed[i] = false;
        }

        DFS(0, processed);

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

    //Helper function to chech the Connectivity of the graph

public void DFS(int node, boolean processed[]) {

        //To keep a track of visited nodes
        processed[node] = true;

        //System.out.println("Visited "+i);

        for(int i = 0;i < 5; i++){
            if(graph[node][i] == 1){
                if(!processed[i]){
                    DFS(i, processed);
```

```
                }
            }
        }

    }
```

//Function to calculate the reliability of a particular state based upon the state of the link(UP/DOWN)

```
    public double Reliability() {
        double reliability = 1;
        for(int i = 0; i < 5; i++){
            for(int j = 0; j < 5; j++){
                if (i > j){

                    //Link is UP
                    if (graph[i][j] == 1) {
                        reliability *=link_prob;
                    }
                    //Link is DOWN
                    else {
                        reliability *=(1 - link_prob);
                    }

                }
            }
        }
        return reliability;
    }
```

```
}

public class ATN_Network_Reliability {

   //Algorithm that generates 2^10 random states and assigns 0/1 to each link by converting the randomly generated numbers to binary

        public static double ExhaustiveEnumeration(double link_prob, int k)
        {
                //k distinct random numbers between 0 and 1024

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

                for(int j = 0; j < k; j++) {

                  Random rand = new Random();

                  int randomNumber = rand.nextInt(1024);


                  while(states.contains(randomNumber)){
                          randomNumber = rand.nextInt(1024);
                  }
                  states.add(randomNumber);
                }


                double rel =0;
```

```
for(int i = 0; i < 1024; i++) {


        Graph g= new Graph();

        g.link_prob = link_prob;

    //Converting the randomly generated numbers to binary

        String state = String.format("%10s",
Integer.toBinaryString(i)).replace(" ", "0");


        for(int j = 0; j < 10; j++){

            if (state.charAt(j) =='1'){

                g.graph[g.src_des[j][0]][g.src_des[j][1]]= 0;
                g.graph[g.src_des[j][1]][g.src_des[j][0]]= 0;

            }
        }


        if(states.contains(i)){
            if(!g.CheckConnected()){

                rel += g.Reliability();
            }
        }
        else{
```

```
                        if(g.CheckConnected()){

                            rel +=g.Reliability();
                        }
                    }
                }

                return rel;
        }

    // Function that calculates reliability given k between 0 to 20 by reversing the state of the system

    public static double ExhaustiveFlipped(double link_prob, int k)
        {

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

            for(int j = 0; j < k; j++) {

              Random rand = new Random();

              int random = rand.nextInt(1024);


              while(states.contains(random)){
                    random = rand.nextInt(1024);
              }
              states.add(random);
            }
```

```
double Rflipped =0;
for(int i = 0; i < 1024; i++) {

    Graph g1= new Graph();

    g1.link_prob = link_prob;

    String state = String.format("%10s",
Integer.toBinaryString(i)).replace(" ", "0");
    state = state.replaceAll("0", "x");
    state = state.replaceAll("1", "0");
    state = state.replaceAll("x", "1");

    for(int j = 0; j < 10; j++){

        if (state.charAt(j) =='1'){

            g1.graph[g1.src_des[j][0]][g1.src_des[j][1]]= 0;
            g1.graph[g1.src_des[j][1]][g1.src_des[j][0]]= 0;

        }
    }

    if(states.contains(i)){
        if(!g1.CheckConnected()){

            Rflipped = Rflipped + g1.Reliability();
        }
```

```
                }
                else{
                        if(g1.CheckConnected()){

                                Rflipped = Rflipped + g1.Reliability();
                        }
                }
        }



        return Rflipped;
    }

    public static void main(String[] args) {
        int k=0;

       System.out.println("Link Probability              Reliability");
        //finding variation of probability with p with k =0
        for ( double i = 0; i <1.05; i += 0.05) {

        System.out.println(Math.round(i*100.00)/100.00+"
"+ExhaustiveEnumeration(i, 0));
        }
        System.out.println();

        System.out.println("k              Reliability");
        //Fix the link reliability to 0.85
        for(k = 0; k <= 20; k++){
                double rel = 0;
                for(int j = 0;j < 100; j++){
```

```
                    rel = rel +ExhaustiveFlipped(0.85, k);
                }
                rel = rel/100;
                System.out.println(k+"          "+rel);

            }

        }
}
```

References:

1)Lecture Notes