Project 3

Implementation of Network Reliability

CS 6385

Rajan Jhaveri

(rjj160330)

Table of Content

Objective:

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

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

  - **Components which may fail:** We consider here that the links of network can fail but the nodes will always be up. The reliability of each link is p and it remains the same for every link. The parameter p will take different values in the experiments. The range will be from 0 to 1 in increments of 0.05.

  - **Reliability configuration:** The system is considered operational, if the network is complete.

- **Tasks**

  - To create an algorithm to compute the network reliability in the above described situation, using the method of exhaustive enumeration
  - Write a computer program that implements the algorithm.
  - Make sure, however, that your program is well structured to support finding potential errors (debugging), checking correctness or trying out algorithm changes. Explain how your program supports these goals. Include a section that tells how to run your program (this is usually called ReadMe file).
  - Now fix the p parameter at p = 0.85, and do the following experiment. Among the 2^{10} = 1024 possible combinations of component states pick k of the combinations randomly, and flip the corresponding system condition. That is, if the system was up, change it to down, if it was down, change it to up. This aims at modeling the situation when there is some random error in the system status evaluation.
  - Short explanation why the obtained diagrams look the way they look. In other words, try to argue that they exhibit a reasonable behavior that one could intuitively expect, so the program is likely to work correctly.

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 from range 0 to 1.

3) There are 10 edges in consideration at any given time.

4) Using the number of edges, determine the $2^{(\text{edge combinations to be considered})}$. Here it’s value is 10. Thus $2^{10} = 1024$.

5) If the graph is such that each node is connected by at least one up link, it is said to be in up condition or the up state.

6) Determine sets of the links states that are in the up condition for the network.

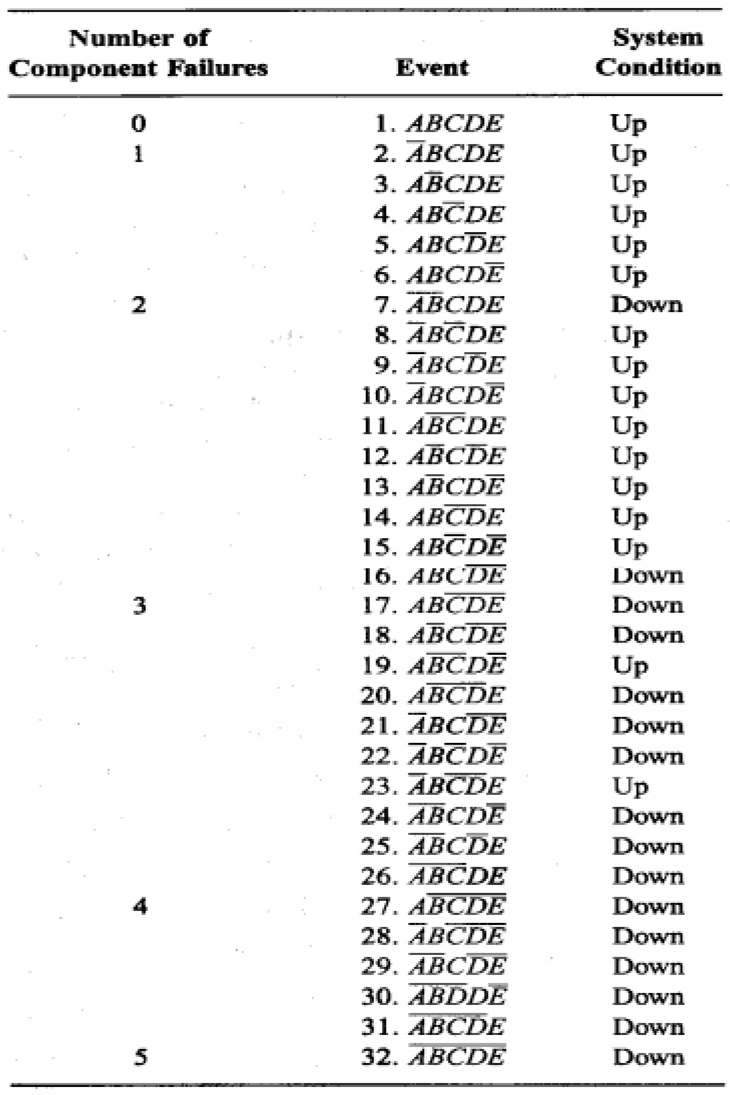
7) Iterate N times for all components and calculate the reliability values.

8) 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.

9) Also calculate the reliability values by considering the bit to be flipped.

A sample calculation for 5 nodes is as shown below:

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



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. $\overline{AB}CDE$	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. $A\overline{BC}DE$	Up
	12. $A\overline{B}C\overline{D}E$	Up
	13. $A\overline{B}CD\overline{E}$	Up
	14. $AB\overline{CD}E$	Up
	15. $AB\overline{C}D\overline{E}$	Up
	16. $ABC\overline{DE}$	Down
3	17. $AB\overline{CDE}$	Down
	18. $A\overline{B}C\overline{DE}$	Down
	19. $A\overline{BC}D\overline{E}$	Up
	20. $A\overline{BCD}E$	Down
	21. $\overline{A}BC\overline{DE}$	Down
	22. $\overline{A}B\overline{C}D\overline{E}$	Down
	23. $\overline{A}B\overline{CD}E$	Up
	24. $\overline{AB}CD\overline{E}$	Down
	25. $\overline{AB}C\overline{D}E$	Down
	26. $\overline{ABC}DE$	Down
4	27. $A\overline{BCDE}$	Down
	28. $\overline{A}B\overline{CDE}$	Down
	29. $\overline{AB}C\overline{DE}$	Down
	30. $\overline{ABD}D\overline{E}$	Down
	31. $\overline{ABCD}E$	Down
5	32. $\overline{ABCDE}$	Down

<u>Pseudo code:</u>

- Generate 2^{10}= 1024 possible states randomly using the Random function and then convert them to binary so that we have a 0 or 1 value for each link where 0 means system is UP and 1 means system is DOWN(find_Rel()).

- 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 isConnected() function).

- Add up the reliabilities of all those UP states.

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

Goals:

Changes: We have used object oriented style of programming so changes are easy to incorporate as we have a modular approach. One change in a function can impact the whole program without requiring to make multiple changes. This can be pretty helpful at industry level.

Debugging: It is possible by using print statements at various points. Though this is a naïve approach, we have used it. A better approach would be to open the code Integrated Development Environment and use debug mode with Step-over and Step-in functionalities. This helps in checking the values of various attributes at different steps. It might not be very necessary to try all this at this level but is very important when there are hundreds of classes and thousands of LOC(Lines of Code) so it is good practice to use debugging.

Correctness: The produced outputs show the probability and reliability values for the given graph with number of nodes as 5 and number of links as 10. The values are between 0 and 1 and that is what was desired. Also we see a trend that with the increase in value of probability, the value of reliability goes up as well. This is pretty similar to what we saw in class and the Professor has laid special emphasis on this topic which helped us in understanding the topic pretty clearly.

Results:

- **Link Probability and Reliability**

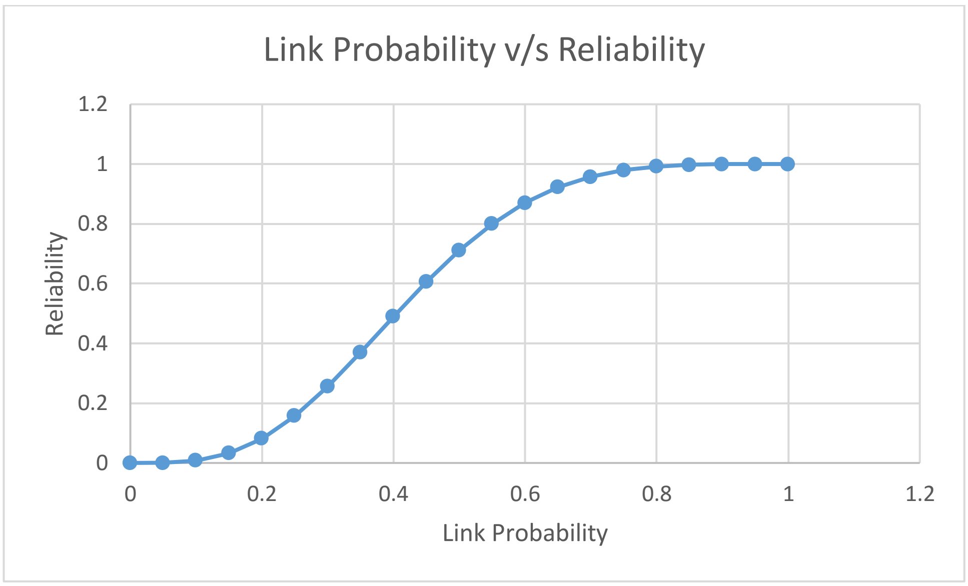
Link Probability	Reliability
0	0
0.05	0.00063
0.1	0.008097
0.15	0.0327
0.2	0.08194
0.25	0.15769
0.3	0.2562
0.35	0.37005
0.4	0.48965
0.45	0.6058
0.5	0.71093
0.55	0.7998

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

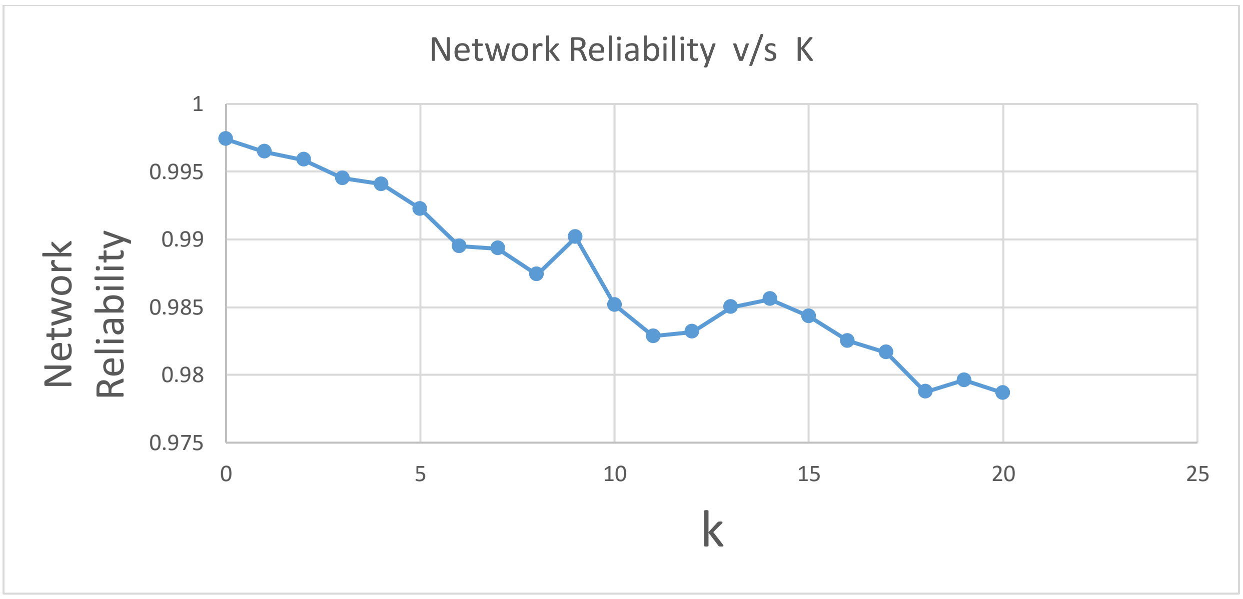
- **K and Reliability**

k	Reliability
0	0.997394536
1	0.996466539
2	0.995876252
3	0.994527637
4	0.994072337
5	0.992257688
6	0.989502391
7	0.98932844
8	0.987415735
9	0.990151583
10	0.985174511
11	0.982844867
12	0.983172193
13	0.984994638
14	0.985593639
15	0.984321166
16	0.982498567
17	0.981649086
18	0.978750969
19	0.979612181
20	0.97867144

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. This is a proof that as the value of probability increases, the value of reliability increases. It is pretty obvious that it behaves this way and can be seen from the class notes as well.



- 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 and vice-versa.

Appendix:

```
import java.util.ArrayList;
import java.util.Random;

public class Net_Reliability {

        public static double find_rel(double link_rel, int k)
        {
                //list of k distinct random numbers between 0 and 1024 are generated
                ArrayList<Integer> randomStates = new ArrayList<Integer>();

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

                  Random rnd = new Random();

                  int rndNumber = rnd.nextInt(1024);
                  //System.out.println("Random no" +randomNumber);

                  while(randomStates.contains(rndNumber)){
                          rndNumber = rnd.nextInt(1024);
                  }
                  randomStates.add(rndNumber);
                }


                double R =0;


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


                        Graph Graph= new Graph();

                        Graph.link_rel = link_rel;
                        //All possible combinations are generated using all 10 digit binary
numbers
                        String systemState = String.format("%10s",
Integer.toBinaryString(i)).replace(" ", "0");
                        //System.out.println(systemState);

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

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

Graph.adjMatrix[Graph.source_destination[j][0]][Graph.source_destination[j][1]]= 0;
Graph.adjMatrix[Graph.source_destination[j][1]][Graph.source_destination[j][0]]= 0;

                        }
                    }

                    //if i (current system state) is in the list of randomly chosen system
states then,
                    //flip or reverse the system condition(or state)
                    //else do not reverse state
                    if(randomStates.contains(i)){
                        if(!Graph.isConnected()){
                            //compute reliability by adding reliability of different
states

                            R = R + Graph.calc_Rel();
                        }
                    }
                    else{
                        if(Graph.isConnected()){

                            R = R + Graph.calc_Rel();
                        }
                    }
                }

                return R;
            }

            public static double Flipped_k_Reliability(double linkReliability_p, int k)
            {
                //list of k distinct random numbers between 0 and 1024 are generated
                ArrayList<Integer> listOfKRandomSystemStates = new ArrayList<Integer>();

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

                  Random rand = new Random();

                  int randomNumber = rand.nextInt(1024);
                  //System.out.println("Random no" +randomNumber);

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


                double Rfor1 =0;
```

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


			Graph Graph= new Graph();

			Graph.link_rel = linkReliability_p;
			//All possible combinations are generated using all 10 digit binary 
numbers
			String systemState = String.format("%10s", 
Integer.toBinaryString(i)).replace(" ", "0");
			systemState = systemState.replaceAll("1", "c");
			systemState = systemState.replaceAll("0", "1");
			systemState = systemState.replaceAll("c", "0");
			//System.out.println(systemState);

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

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

Graph.adjMatrix[Graph.source_destination[j][0]][Graph.source_destination[j][1]]= 0;
Graph.adjMatrix[Graph.source_destination[j][1]][Graph.source_destination[j][0]]= 0;

				}
			}

			//if i (current system state) is in the list of randomly chosen system 
states then,
			//flip or reverse the system condition(or state)
			//else do not reverse state
			if(listOfKRandomSystemStates.contains(i)){
				if(!Graph.isConnected()){
					//compute reliability by adding reliability of different 
states
					Rfor1 = Rfor1 + Graph.calc_Rel();
				}
			}
			else{
				if(Graph.isConnected()){

					Rfor1 = Rfor1 + Graph.calc_Rel();
				}
			}
		}

/*Keep a track of R for flipped 1
System.out.println("R value for flipped 1 : "+Rfor1);
*/
```

```
                return Rfor1;
        }


public static void main(String[] args) {

        /*
        * k is the number of system states with reversed or flipped  system conditions
        */
        int k=0;

        /*
        * link reliability
        */
        double link_rel;

        //finding variation of probability with p with k =0
        for ( link_rel = 0; link_rel <= 1.01; link_rel += 0.05) {

System.out.println("p = "+String.format("%.2g", link_rel)+"   Reliability = "+find_rel(link_rel,
k));
        }
        System.out.println();


        //Fix the link reliability to 0.85  - 0000000000000000
        for(k = 0; k <= 20; k++){
                double rel = 0;
                for(int j = 0;j < 100; j++){
                        rel = rel +find_rel(0.85, k);
                }

        }

        //Flipped --------------
        for(k = 0; k <= 20; k++){
                double rel = 0;
                for(int j = 0;j < 100; j++){
                        rel = rel +Flipped_k_Reliability(0.85, k);
                }
                rel = rel/100;
                System.out.println("Reliability of Flipped k = "+k+" is "+rel);

                }

        }
}
```

```
class Graph {

/*
 * adjacency matrix of a graph
 */
public int adjMatrix[][];

/*
 * map i and j values of adjacency matrix to the link numbers
 */
public int source_destination[][];

/*
 * p is the reliability of a link
 */

public double link_rel;


        public boolean isConnected(){
                boolean flag = true;

        /*for keeping a track of visited nodes
        by calling DFS(depth first Search)
        */
                boolean visited[] = new boolean[5];

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

                depthFirstSearch(0, visited);

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


public void depthFirstSearch(int i, boolean visited[]) {
        /*for keeping a track of visited nodes
 by calling DFS(depth first Search)
 */
        visited[i] = true;

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

        }

public void display(){
        for(int i = 0; i < 5; i++){
                for(int j = 0; j < 5; j++){
                        System.out.print(adjMatrix[i][j]+" ");
                }
                System.out.println();
        }
}

public Graph(){

        link_rel = 0;

        //Number of nodes in the network are 5, therefore initializing the adjacency matrix
with 5 nodes
        adjMatrix= new int[5][5];

        source_destination = new int[10][2];

        int val = 0;

        //adj_matrix and source_destination is generated for a 5 node network with all
nodes connected
        //This is to show a complete graph where all the nodes are connected
        for(int i = 0; i < 5; i++){
                for(int j = 0; j < 5; j++){
                        if (i != j) {
                                if(i > j) {
                                        source_destination[val][0]= i;
                                        source_destination[val][1]= j;
                                        val++;
```

```
                                                }
                                                adjMatrix[i][j]=1;
                                        }
                                        else
                                                adjMatrix[i][j]=0;
                                }
                        }
                }


                public double calc_Rel() {
                        double final_rel = 1;
                        for(int i = 0; i < 5; i++){
                                for(int j = 0; j < 5; j++){
                                        if (i > j){

/*for the state when
 *the link is up
*/
                                                if (adjMatrix[i][j] == 1) {
                                                        final_rel = final_rel * link_rel;
                                                }

/*for the state when
 *the link is down
*/
                                                        else {
                                                                final_rel = final_rel * (1 - link_rel);
                                                        }

                                                }
                                        }
                                }
                                return final_rel;
                        }

                }
```

References:

1)Lecture Notes