CS 6385

Algorithmic Aspects of Telecommunication Network

Spring 2017

Project 3
Dependence of Network Reliability on the reliability of individual links

Submitted by:

Ankita S. Patil

(**<u>Net ID</u>:** asp160730)

Instructor:

Andras Farago

Table of Contents

Introduction

In this project, an experimental study is conducted to test the dependence of reliability of whole network on the reliability of individual links in specific situations.

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). Thus, 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.

Hence, the goal is to develop an algorithm to find the overall network reliability

In the next section, the goal or objective of an experiment is discussed, then pseudo code for is provided followed by explanation of how implementation works. For better understanding experimental results are included along with the analysis. Appendix contains the source code.

Problem Description

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 inoperational (failed) with probability $1 - p_i$. (These probabilities are usually known.)
- The reliability *R* of the system is some function of the component reliabilities:

  $$R = f(p_1, p_2, \ldots, p_N)$$

  where *N* is the number of components.

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

Network topology is a complete undirected graph. We are assuming that the nodes are always up (operational), but links may fail. Hence, some links or edges may disappear but remaining links will keep the network either connected or disconnected. Hence, the network reliability only depends on individual link reliability.

The system will be operational if the network topology is connected.

Goal: Given a complete undirected graph with N nodes develop an algorithm to find the reliability of overall network using Exhaustive Enumeration Method.

Other requirements:

- ⇨ Run the program for different values of *p*. Let the parameter *p* run over the [0,1] interval, in steps of 0.05. Show graphically in a diagram how the obtained network reliability values depend on *p*.
- ⇨ 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. Show in a diagram, how the reliability of the system changes due to this alteration, by showing how the change depends on *k*, in the range $k = 0,1,2,3,\ldots,20$. During this

experiment keep the value of the parameter *p* fixed at *p* = 0.85. To reduce the effect of randomness, run several experiments and average them out, for each value of *k*.

Hence, our goal is to design an algorithm to evaluate the reliability of a given network configuration.

For achieving this goal, we rely on the PROBABILITY of up and down states of links in the network. We use these probabilities to determine the overall probability of the network.

If there exists an edge between given two nodes, then that link is said to be UP.

If an edge between given two nodes does not exist, then that link is said to be DOWN.

We start by assuming a link reliability for all the links in the network.

Then, we consider all the possible combinations in which the given network can be connected with the same number of links. For all these combinations, we calculate the overall reliability of the network.

This kind of approach is called exhaustive enumeration and is useful for small network topologies without any particular connectivity pattern like series or parallel or series-parallel.

We also experiment by flipping the states of some links in the network from up to down. Keeping the link reliabilities same, we study the reliability of the network under these network changes.

Algorithm

NETWORK TOPOLOGY

We simulate the network topology as an undirected graph with 5 nodes and 10 edges.

In this topology, every node is connected to every other node, and we avoid parallel and self loops, making the number of edges in the network to be 10.

This leads to 2^10 = 1024 possible combinations in which the network topology states.

LINK RELIABILITY

The link reliability or link probability is p. For the first part of the experiment, we vary the reliability p between [0,1] in steps of 0.05. For each of these values of p, we will find the network reliability. For the second part of the experiment, we flip the states of links from k randomly chosen network combinations out of the 1024 combinations. Here, k goes from 0 to 20 and for each of this k values, we find the network reliability.

Generalized Pseudocode

N: number of nodes

G: Graph

p: probability that link is up

r: reliability of a network state

R: Reliability of a whole network

<u>**INPUT**</u>**:** => Network topology: an undirected complete graph G with n nodes,

[total edges = (n * (n -1)) / 2]

=> Link probability p : probability that link is up

<u>**OUTPUT:**</u> Reliability of a network, R

```
• Generate all the states of the graph
• Set reliability = 1
• For each state that belong to all states (every combination)
        do if G is connected
        then find the reliability r for each state
• Calculate and return R by summing up all r (by adding the reliabilities of each network state)
```

Algorithm for Regular Experiment

(1) To determine the network reliability, we consider all possible 1024 combinations
All possible combinations are generated using 10-digit binary number

```
All combinations involved in the network can be represented by the numbers
from 0 to 1023 as follows
       0 - 0000000000
        1 - 0000000001 - one link is up
        2 - 0000000010 - a different link is up
        3 - 0000000011 - two different links are up
        ...
        1023 - 1111111111 - all links are up!
```

(2) An adjacency matrix and mapToLink matrix is maintained.
Adjacency matrix stores whether their exists an edge between given two nodes and mapToLink matrix stores edge information.
Process these matrices to find the combination which represent the connected graph
We perform depth first search to find whether the connected components go through all the nodes in the network to ensure the network is connected.

(3) For each combination, we calculate the reliability.

```
Set reliability 1
For each edge,
    if edge is up
        reliability = reliability * p
    if edge is down
        reliability = reliability *(1 – p)
```

where,
p : probability that link is up
(1 – p) : probability that link is down

(4) Calculate the total network reliability by summing up the reliability for all 1024 possible network combinations.

(5) Repeat the steps 2 to 5 for p values from [0, 1] in the steps of 0.05

Algorithm for Flip Experiment

(1) To determine the network reliability, we consider all possible 1024 combinations
All possible combinations are generated using 10-digit binary number

```
All combinations involved in the network can be represented by the numbers
from 0 to 1023 as follows
       0 - 0000000000
        1 - 0000000001 - one link is up
        2 - 0000000010 - a different link is up
        3 - 0000000011 - two different links are up
        ...
 1023 - 1111111111 - all links are up
```

(2) From the 1024 combinations obtained from step 1, pick up k combinations randomly and flip their system states (I,.e. If the system state was up previously make it down and vice versa)
Hence, in this step we change the state of the network topology

```
Set reliability 1
For each edge,
    if edge is up
        reliability = reliability * p
    if edge is down
        reliability = reliability *(1 – p)
```

where,
    p : probability that link is up
(1 – p) : probability that link is down

(3) For each connected combination, calculate the reliability
(4) Calculate the total network reliability by summing up the reliability for all 1024 possible network combinations.
(5) Repeat the steps 2 to 5 for p = 0.85 and k values from [0, 20] in the steps of 1

Implementation Details

<u>Technologies used:</u>

Programming Language	Java
Operating System	Windows 10
Development Environment	Eclipse Neon

<u>Implementation Details:</u>

The project consists of following package and classes.

Package	com.ankita.atnproject3
classes	Graph.java NetworkReliability.java

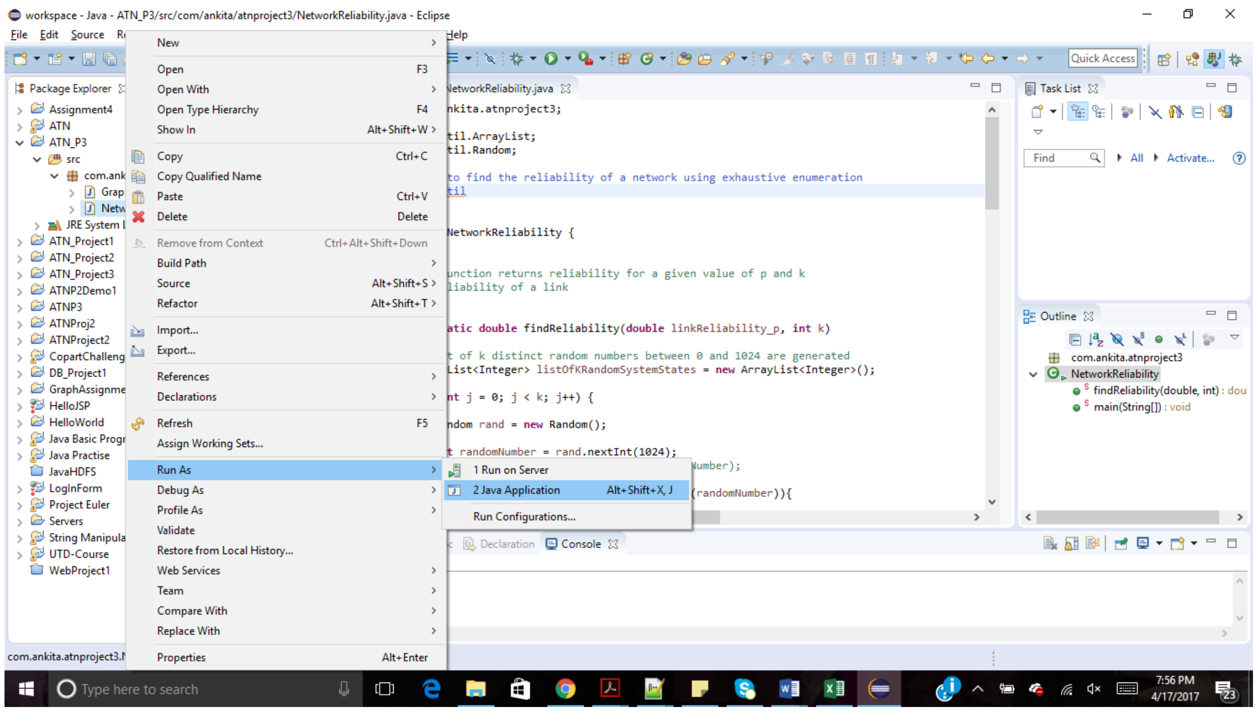
- NetworkReliability.java is a driver class and it invokes **findReliability()** function which takes a link reliability p and k as an argument. This function returns reliability for a given value of p and k for all 1024 network states.
- In the regular experiment, we are incrementing the values of p in the steps of 0.05
- In flip experiment, we are fixing the value of p to 0.85 and selecting 20 random states out of 1024 states and changing their system state.
- How algorithm works is very well explained in the previous sections.

ReadMe Section
-Instructions on how to run a program

(1) Import the package com.ankita.atnproject3 in Eclipse IDE.

(2) The package consists of two java files.
   - Graph.java
   - NetworkReliability.java

(3) Run NetworkReliability.java from the package com.ankita.atnproject3

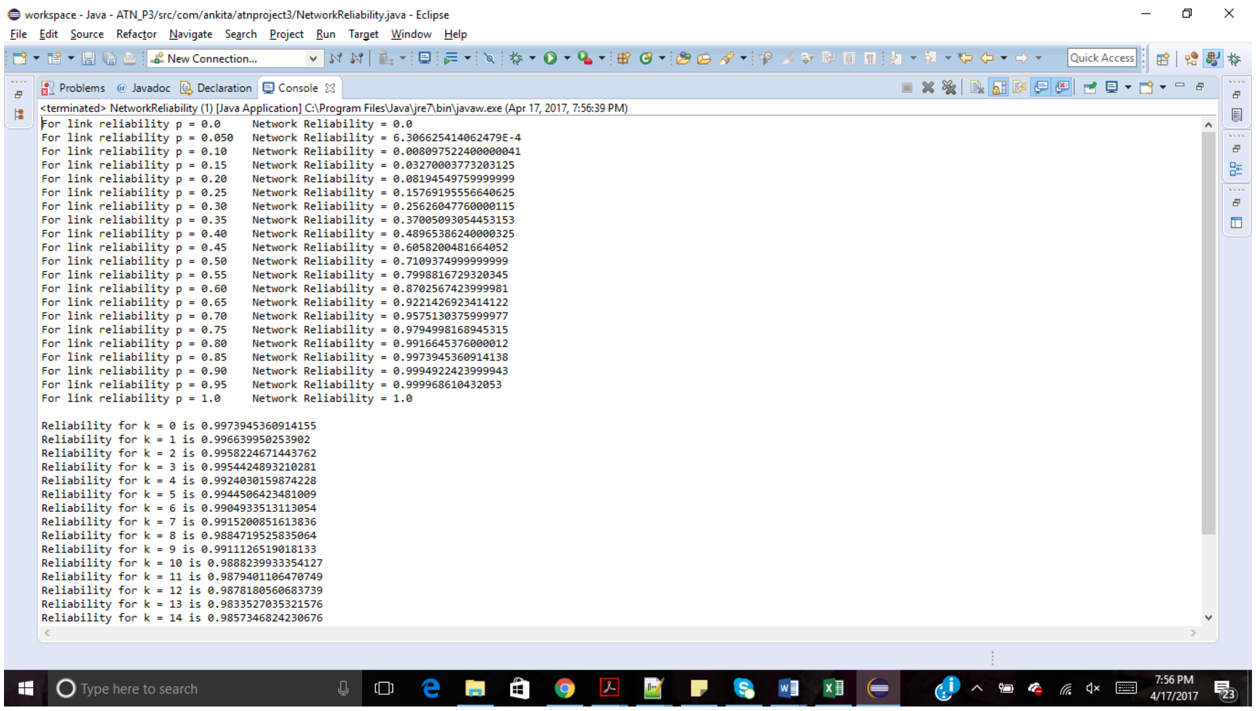


```
nkita.atnproject3;

til.ArrayList;
til.Random;

to find the reliability of a network using exhaustive enumeration
til


NetworkReliability {


unction returns reliability for a given value of p and k
liability of a link


atic double findReliability(double linkReliability_p, int k)

t of k distinct random numbers between 0 and 1024 are generated
List<Integer> listOfKRandomSystemStates = new ArrayList<Integer>();

nt j = 0; j < k; j++) {

ndom rand = new Random();

t randomNumber = rand.nextInt(1024);
                                   Number);

                                   (randomNumber)){
```

(4) Output can be seen from console.

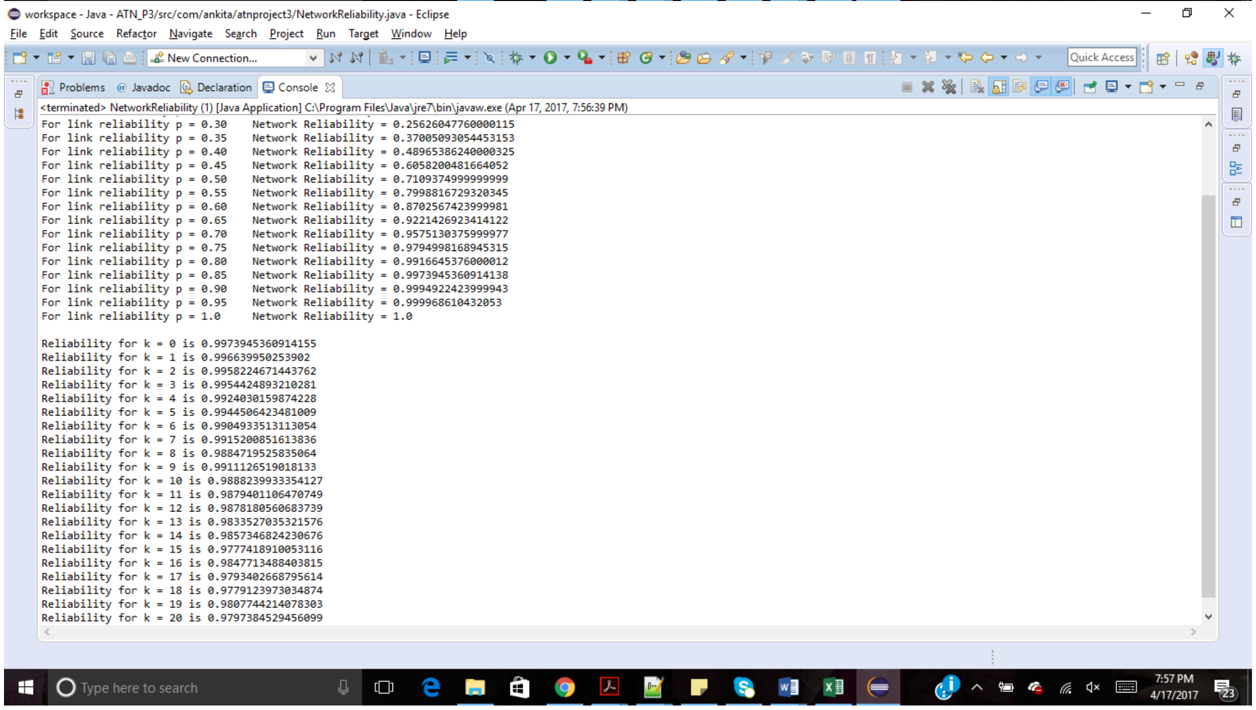
Sample Output

(Only few Screenshots are attached)



<terminated> NetworkReliability (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 17, 2017, 7:56:39 PM)

```
For link reliability p = 0.0     Network Reliability = 0.0
For link reliability p = 0.050   Network Reliability = 6.306625414062479E-4
For link reliability p = 0.10    Network Reliability = 0.008097522400000041
For link reliability p = 0.15    Network Reliability = 0.03270003773203125
For link reliability p = 0.20    Network Reliability = 0.08194549759999999
For link reliability p = 0.25    Network Reliability = 0.15769195556640625
For link reliability p = 0.30    Network Reliability = 0.25626047760000115
For link reliability p = 0.35    Network Reliability = 0.37005093054453153
For link reliability p = 0.40    Network Reliability = 0.48965386240000325
For link reliability p = 0.45    Network Reliability = 0.6058200481664052
For link reliability p = 0.50    Network Reliability = 0.7109374999999999
For link reliability p = 0.55    Network Reliability = 0.7998816729320345
For link reliability p = 0.60    Network Reliability = 0.8702567423999981
For link reliability p = 0.65    Network Reliability = 0.9221426923414122
For link reliability p = 0.70    Network Reliability = 0.9575130375999977
For link reliability p = 0.75    Network Reliability = 0.9794998168945315
For link reliability p = 0.80    Network Reliability = 0.9916645376000012
For link reliability p = 0.85    Network Reliability = 0.9973945360914138
For link reliability p = 0.90    Network Reliability = 0.9994922423999943
For link reliability p = 0.95    Network Reliability = 0.999968610432053
For link reliability p = 1.0     Network Reliability = 1.0

Reliability for k = 0 is 0.9973945360914155
Reliability for k = 1 is 0.996639950253902
Reliability for k = 2 is 0.9958224671443762
Reliability for k = 3 is 0.9954424893210281
Reliability for k = 4 is 0.9924030159874228
Reliability for k = 5 is 0.9944506423481009
Reliability for k = 6 is 0.9904933513113054
Reliability for k = 7 is 0.9915200851613836
Reliability for k = 8 is 0.9884719525835064
Reliability for k = 9 is 0.9911126519018133
Reliability for k = 10 is 0.9888239933354127
Reliability for k = 11 is 0.9879401106470749
Reliability for k = 12 is 0.9878180560683739
Reliability for k = 13 is 0.9833527035321576
Reliability for k = 14 is 0.9857346824230676
```



<terminated> NetworkReliability (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 17, 2017, 7:56:39 PM)

```
For link reliability p = 0.30    Network Reliability = 0.25626047760000115
For link reliability p = 0.35    Network Reliability = 0.37005093054453153
For link reliability p = 0.40    Network Reliability = 0.48965386240000325
For link reliability p = 0.45    Network Reliability = 0.6058200481664052
For link reliability p = 0.50    Network Reliability = 0.7109374999999999
For link reliability p = 0.55    Network Reliability = 0.7998816729320345
For link reliability p = 0.60    Network Reliability = 0.8702567423999981
For link reliability p = 0.65    Network Reliability = 0.9221426923414122
For link reliability p = 0.70    Network Reliability = 0.9575130375999977
For link reliability p = 0.75    Network Reliability = 0.9794998168945315
For link reliability p = 0.80    Network Reliability = 0.9916645376000012
For link reliability p = 0.85    Network Reliability = 0.9973945360914138
For link reliability p = 0.90    Network Reliability = 0.9994922423999943
For link reliability p = 0.95    Network Reliability = 0.999968610432053
For link reliability p = 1.0     Network Reliability = 1.0

Reliability for k = 0 is 0.9973945360914155
Reliability for k = 1 is 0.996639950253902
Reliability for k = 2 is 0.9958224671443762
Reliability for k = 3 is 0.9954424893210281
Reliability for k = 4 is 0.9924030159874228
Reliability for k = 5 is 0.9944506423481009
Reliability for k = 6 is 0.9904933513113054
Reliability for k = 7 is 0.9915200851613836
Reliability for k = 8 is 0.9884719525835064
Reliability for k = 9 is 0.9911126519018133
Reliability for k = 10 is 0.9888239933354127
Reliability for k = 11 is 0.9879401106470749
Reliability for k = 12 is 0.9878180560683739
Reliability for k = 13 is 0.9833527035321576
Reliability for k = 14 is 0.9857346824230676
Reliability for k = 15 is 0.9777418910053116
Reliability for k = 16 is 0.9847713488403815
Reliability for k = 17 is 0.9793402668795614
Reliability for k = 18 is 0.9779123973034874
Reliability for k = 19 is 0.9807744214078303
Reliability for k = 20 is 0.9797384529456099
```

Experimental values

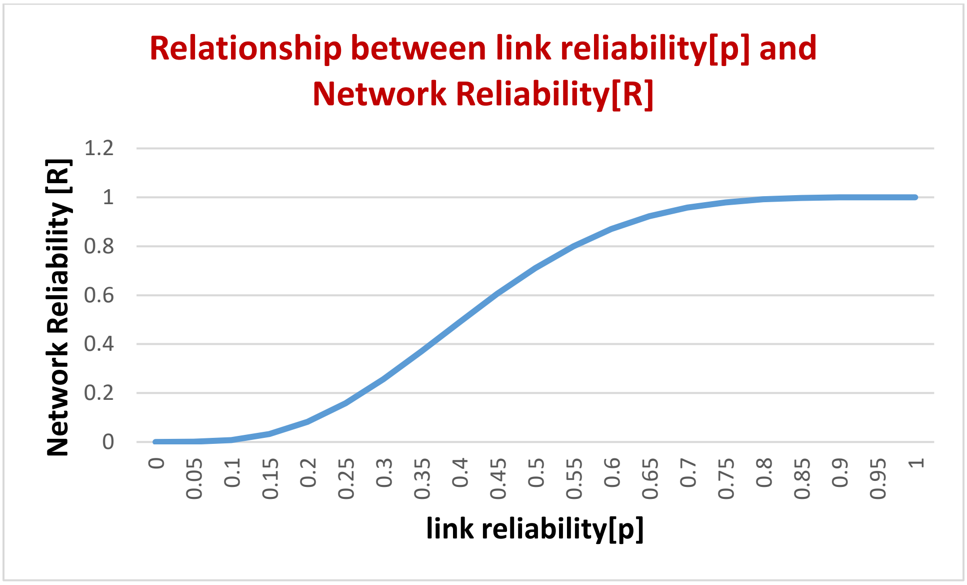
Link reliability[p]	Network Reliability[R]
0	0
0.05	6.31E-04
0.1	0.008097522
0.15	0.032700038
0.2	0.081945498
0.25	0.157691956
0.3	0.256260478
0.35	0.370050931
0.4	0.489653862
0.45	0.605820048
0.5	0.7109375
0.55	0.799881673
0.6	0.870256742
0.65	0.922142692
0.7	0.957513038
0.75	0.979499817
0.8	0.991664538
0.85	0.997394536
0.9	0.999492242
0.95	0.99996861
1	1

k	R (Run 1)	R (Run 2)	R (Run 3)	R (Run 4)	Average Network Reliability
0	0.9973945	0.997395	0.997395	0.997395	0.997394536
1	0.9941892	0.996674	0.99698	0.996897	0.99618497
2	0.9954786	0.995678	0.996392	0.994455	0.995500908
3	0.9900612	0.994149	0.995945	0.993097	0.993312806
4	0.992523	0.993822	0.993799	0.994346	0.993622432
5	0.9935963	0.990376	0.993163	0.989589	0.991680954
6	0.9855847	0.992628	0.989971	0.993704	0.990471976
7	0.991662	0.992148	0.988281	0.989583	0.99041839
8	0.9876783	0.991231	0.98965	0.98886	0.989354781
9	0.9853461	0.986124	0.991473	0.989099	0.988010571
10	0.9862004	0.986241	0.982469	0.988708	0.985904651
11	0.9828213	0.987548	0.985741	0.988975	0.986271307
12	0.9884511	0.985988	0.989321	0.984869	0.987157116
13	0.9840544	0.988145	0.981259	0.981596	0.983763435
14	0.9796264	0.984519	0.985807	0.983627	0.98339488
15	0.9826386	0.983818	0.982059	0.987928	0.984110833
16	0.9824937	0.980868	0.979597	0.984109	0.981766714
17	0.9838767	0.981805	0.977872	0.984392	0.981986257
18	0.9794887	0.982726	0.980271	0.978427	0.98022804
19	0.9738709	0.981007	0.981894	0.97507	0.977960354
20	0.9754203	0.975787	0.980905	0.980087	0.978049697

k	Average Network Reliability
0	0.997395
1	0.996687
2	0.995449
3	0.994118
4	0.993622
5	0.991477
6	0.9904719
7	0.99041839
8	0.9893547
9	0.9880105
10	0.985904
11	0.9862713
12	0.987157
13	0.983763
14	0.98339488
15	0.984110833
16	0.9817667
17	0.981986
18	0.980228
19	0.97796
20	0.9780496

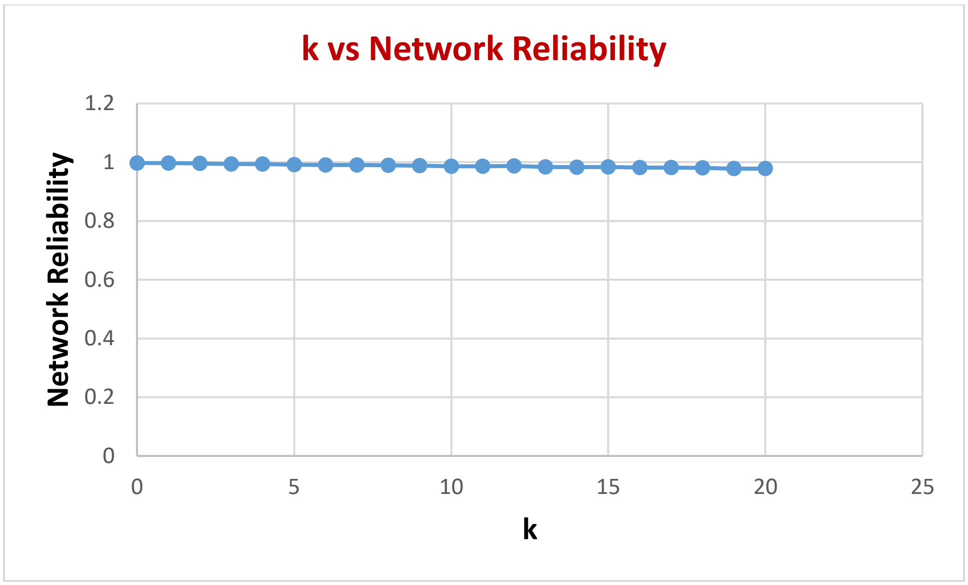
Observations and Analysis

(A)Determining the relationship between link reliability(p) and network reliability(R)



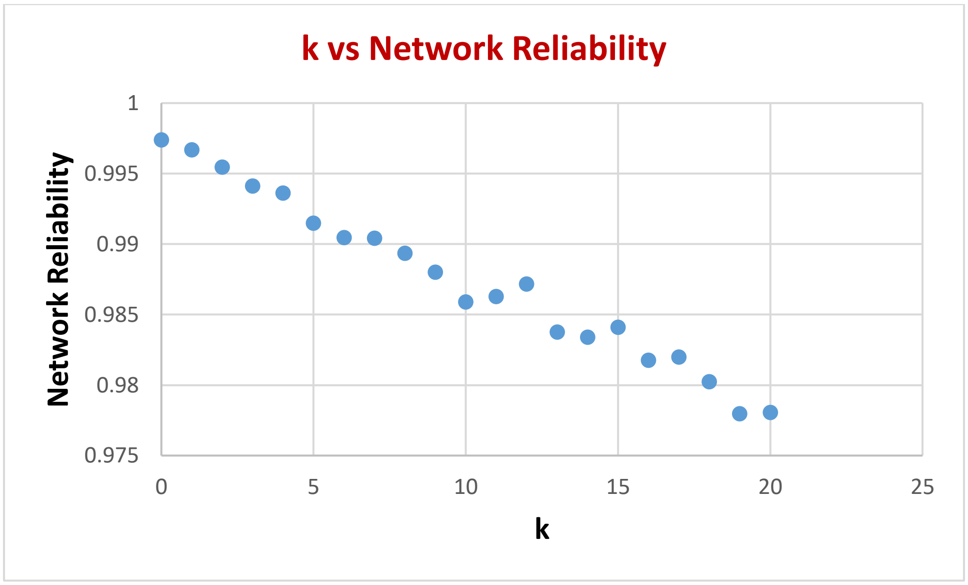
- From the diagram, we observe that there is strong relationship between link reliability and network reliability.
- For smaller values of p, the network reliability is incrementing slowly.
- After some point let's call it cut off, there is improvement in the reliability value. We observe that there is steady increase in the reliability for higher values of p.
- We also observe that for higher values of p from 0.8, the reliability becomes almost constant.
- From all the above observed data, we can infer that, higher the probability of up individual links or operational links, better is the reliability of a network.
- In order to achieve high reliable network, we should make sure that individual links are configured such that their link probability is at least 0.8

(B)Determining the relationship between k and Network Reliability where p = 0.85



In the above graph, p is kept fixed at 0.85 and for k = 0 to 20, network reliability is calculated. Since the value of k is chosen random, the program is run for several times and average value of network reliability for each k is taken.

From above graph, it is hard to find the relationship between network reliability and k. Let's observe from another plot of values where we have scaled the network reliability from 0.975 to 1 because since the value of p is kept fixed, we are getting the values in the range between 0.97 and 0.99



In this experiment, we keep the value of p stable at 0.85 Since, it is a very good link reliability value, we can see that the average network reliability is quite high, it falls in the range of 0.975 to 1.00.

From this, we can see that for some values of k the network reliability is increasing and for some values it is decreasing. So, the change in the values is random. Hence the relationship between k and network reliability is random.

Conclusion

In this project, we conducted an experimental study to test the dependence of reliability of whole network on the reliability of individual links in specific situations.

We observe that the overall network reliability can be measured by using a probabilistic model for the up/down states of a link. The better the probability to stay up, the better is the reliability.

References

(1) Lecture notes by Professor Andras Farago
(2) www.wikipedia.com
(3) L. E. Miller, J. J. Kelleher, and L. Wong, "Assessment of Network Reliability Calculation Methods," J. S. Lee Associates, Inc. report JC-2097-FF under contract DAAL02-92-C-0045, January 1993.

Appendix

```
---------------------------Source Code-----------------------------------------
```

Graph.java

```
package com.ankita.atnproject3;

public class Graph {

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

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

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

	public double linkReliability_p;

	public Graph(){

		linkReliability_p = 0;

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

		/*
		 * Total number of edges in undirected complete graph are [n * (n-1)]/2
		 * Hence initializing the following matrix with 10 edges and 2 nodes
		 * each row entry has the entry of two nodes having an edge and row
number indicates the edge number
		 */

		mapToLink = new int[10][2];

		int key = 0;
//adjacency matrix and map is generated for a 5 node network with all nodes connected
             for(int i = 0; i < 5; i++){
                    for(int j = 0; j < 5; j++){
                           if (i != j) {
                                  if(i > j) {
                                         mapToLink[key][0]= i;
                                         mapToLink[key][1]= j;
                                         key++;
                                  }
                                  adjacencyMatrix[i][j]=1;
                           }
                           else
                                  adjacencyMatrix[i][j]=0;
                    }
             }
      }

      /*
       * Output of mapToLink matrix:
       * The 10 edges of the network  can be encoded as follows:

             0--1 : 0
             0--2 : 1
             1--2 : 2
             0--3 : 3
             1--3 : 4
             2--3 : 5
             0--4 : 6
             1--4 : 7
             2--4 : 8
             3--4 : 9

       */

      /*
       * Following function implements depth first search starting from node i
       */

      public void depthFirstSearch(int i, boolean visited[]) {

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

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

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

      }
```

```
	/*
	 * Following function checks if the graph or network is connected using depth
first search
	 */

	public boolean isConnected(){
		boolean flag = true;

		//to keep a track of visited nodes
		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;
	}



	/*
	 * Following function prints the adjacency matrix of a network or graph
	 */
	public void print(){
		for(int i = 0; i < 5; i++){
			for(int j = 0; j < 5; j++){
				System.out.print(adjacencyMatrix[i][j]+" ");
			}
			System.out.println();
		}
	}
```

```
        /*
         * Following function returns the reliability of a given network or graph
         */

        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 (adjacencyMatrix[i][j] == 1) {
                                                reliability = reliability *linkReliability_p;
                                        }
                                        //if the link is down
                                        else {
                                        reliability = reliability * (1 - linkReliability_p);
                                        }

                                }
                        }
                }
                return reliability;
        }


}
```

NetworkReliability.java

```
package com.ankita.atnproject3;

import java.util.ArrayList;
import java.util.Random;
/**
 * A program to find the reliability of a network using exhaustive enumeration
 * @author Ankita Patil
 *
 */
public class NetworkReliability {

	/*
	 * This function returns reliability for a given value of p and k
	 * p : Reliability of a link
	 */

	public static double findReliability(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 R =0;

		 /*
	  All combinations involved in the network can be represented by the numbers
	 from 0 to 1023 as follows
	  0 - 0000000000
	  1 - 0000000001 - one link is up
	  2 - 0000000010 - a different link is up
	  3 - 0000000011 - two different links are up
	  ...
	  1023 - 1111111111 - all links are up!

	  */

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


			Graph Graph= new Graph();
			Graph.linkReliability_p = linkReliability_p;
		//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.adjacencyMatrix[Graph.mapToLink[j][0]][Graph.mapToLink[j][1]]= 0;
					
	Graph.adjacencyMatrix[Graph.mapToLink[j][1]][Graph.mapToLink[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
					R = R + Graph.calculateReliability();
				}
			}
			else{
				if(Graph.isConnected()){
					
					R = R + Graph.calculateReliability();
				}
			}
		}
		
		return R;
	}


	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 linkReliability_p;
```

```
                //finding variation of probability with p with k =0
    for ( linkReliability_p = 0; linkReliability_p <1.05; linkReliability_p += 0.05) {

                        System.out.println("For link reliability p =
            "+String.format("%.2g", linkReliability_p)+"\t Network Reliability =
            "+findReliability(linkReliability_p, k));
                }
                System.out.println();


                //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 +findReliability(0.85, k);
                        }
                        rel = rel/100;
                        System.out.println("Reliability for k = "+k+" is "+rel);

                }

        }
}
```