# PROJECT 3

## Network Reliability

## Work done

## by


## Arun Prakash Themothy Prabu Vincent

## NET ID : AXT161330


## SPRING'17 CS/TE6385

## ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKING

# CONTENTS

# Chapter 1

## Introduction

The communication networks are made up of nodes and links. The nodes are connected by hardware and software components through links which allows the networks to communicate with each other.
The Network Reliability is known as the reliability of the network to provide sustained communication between networks even though a components or a combination of components fail. The reliability of the network depends on sustainability of both the hardware and software components. A network failure may last ranging from a few seconds to a few days depending on the extent of failure. These failures generally arise from malfunctions of hardware components that will result in the downtime of a network element.

### Series Configuration

In the series configuration the system is operational if and only if all components are functioning. The system is considered operational if there is an operational path between the two endpoints, that is, all components are functioning: The reliability of the series configuration is computed simply as the product of the component reliabilities:

$R_{series} = p1p2 \cdots pN$

### Parallel Configuration

The parallel configuration is defined operational if at least one of the components are functioning

The reliability can be computed as follows. The probability that component i fails is $1 - p_i$. The probability that all components fail is $(1 - p_1)(1 - p_2) \ldots (1 - p_N)$. The complement of this is that not all component fails, that is, at least one of them works:

$R_{parallel} = 1-(1-p_1)(1-p_2)...(1-p_N) = 1- \Pi(1-p_i)$ for( i= 1,2,…N)

# Chapter 2

# Project Description

In this project, the main objective is to find an algorithm to calculate the reliability of a network topology using exhaustive enumeration method. The possible states of each link and the up/down state system condition for the entire network is determined and converted into the reliability of the system. This reliability is based on the probability of each link associated with the network topology.

The other part of the objective is to choose 'k' states out of the $2^{10} = 1024$ possible combinations of component states 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. To reduce the effect of randomness, the program is run for several(1024 times) and averaged them out, for each value of k.
The project is developed for the following situation

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.

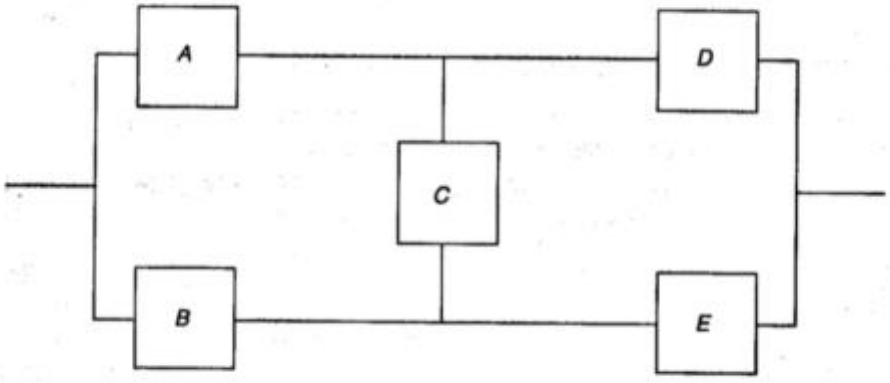
Reliability configuration: The system is considered operational, if the network is connected.

# Chapter 3

# Description of Exhaustive enumeration Method

The exhaustive enumeration method is used for the computation of reliability of the network. It is straight forward but a brute force approach to the problem. For a subsystem having five elements whose reliability has to be calculated, we need to compute the number of states associated with the network. In this case, we need to compute $2^{10} = 1024$ possible states of the network. The up and down states of all the possible states of the network has to be assigned after setting up all those states of the network. The reliability of the network is then calculated by all the probabilities of the up states. Due to the exponential growth of the system, this method is practical only for a small subsystem or a system with lesser number of states. For example, if there are N components, there are $2^N$ possible states, which, makes this exhaustive enumeration practically difficult for larger number of components.

**Example**



We can list all possible states of the system (see table on the next page) and assign ``up" and ``down" system condition to each state. Then the reliability can be obtained by summing the probability of the ``up" states. This usually yields a long expression, see next page.

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

| 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. $A\overline{B}\overline{C}D\overline{E}$ | Down |
| | 19. $A\overline{B}C\overline{D}\overline{E}$ | 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. $A\overline{B}\overline{C}\overline{D}\overline{E}$ | Down |
| | 28. $\overline{A}B\overline{C}\overline{D}\overline{E}$ | Down |
| | 29. $\overline{A}\overline{B}C\overline{D}\overline{E}$ | Down |
| | 30. $\overline{A}\overline{B}\overline{D}\overline{D}\overline{E}$ | Down |
| | 31. $\overline{A}\overline{B}\overline{C}D\overline{E}$ | Down |
| 5 | 32. $\overline{A}\overline{B}\overline{C}\overline{D}\overline{E}$ | Down |

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

# Chapter 4

# Algorithm Implementation Logic

## 4.1 Pseudo Code

Parameters :

- Number of Nodes (N=5)
- Reliability of the link p
- Number of links failed L
- Reliability of the system R
- Reliability of the state $R_s$
- Number of randomly chosen combination K

**Probability vs Reliability :**

1.Form a Network Graph : The program establishes a graph where every node is connected to every other (No loops and no parallel edges) .
2. Set p→0.005
3. Vary L from(0 to 1024)
    a) Determine all possible states for given L using linkFailureCombination() function.
    b) For every state,determine if the graph is connected using the function checkStateCondition()
    c) If the graph is connected → set the condition for state to be up
    d) If the graph is not connected → set the condition for the state to be down
    e) If the condition is up for a given state set R→ $R+R_s$

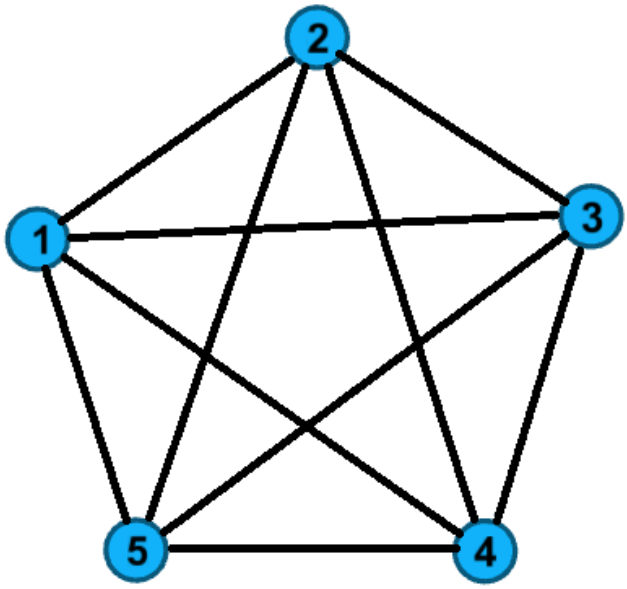
4.Set p→p+0.05
5.Repeat steps 3&4 while p<=1.
6.Plot graph for p vs R

**K vs Reliability**

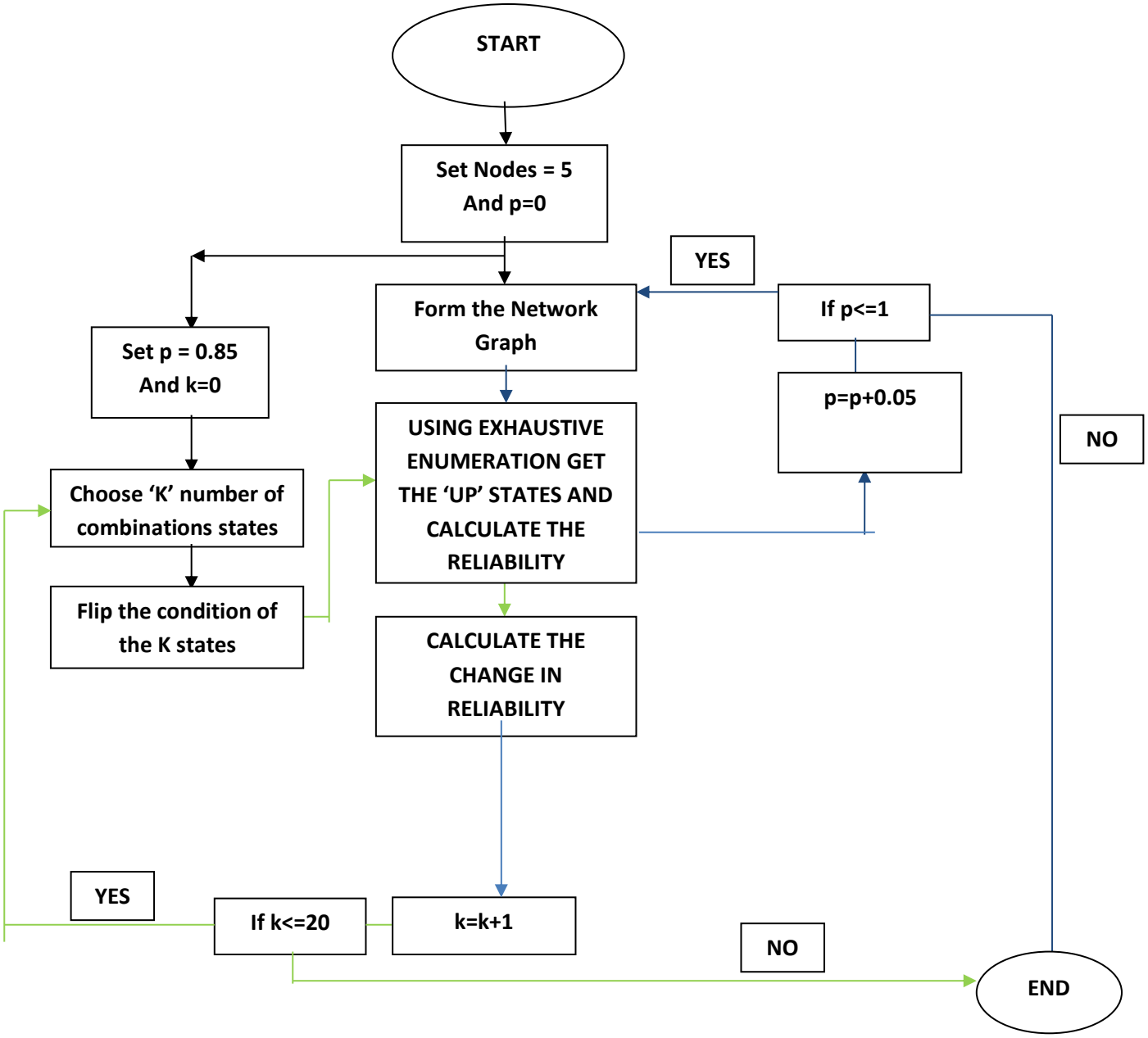
1. Set p→ 0.85
2. Set K=0
3. Randomly choose K states out of $2^{10} = 1024$
4. if the condition of the state is up →set the condition of the state to down
5. if the condition of the state is down →set the condition of the state to up.

6. for all states, if the state condition is up set $R \rightarrow R+R_s$.
7. Increment K by 1
8. Repeat steps 4 to 7 until K<=20
9. Repeat steps 3 to 8 for 1024 times.
10. Calculate the average reliability of the system for each value of K.
11. Calculate the change in reliability $R_c \rightarrow R_{\text{before appling k}} - R_{\text{after applying k}}$
12. Plot a graph between K vs change in reliability
13. Plot a graph between K vs Reliability.

## NETWORK DIAGRAM:



## 4.2 FLOW CHART:



# Chapter 5

# Results and Analysis

## 5.1 Program Output:

The software runs with given Number of nodes and edges as inputs. Number of Nodes is taken as 5 and the probability is varied from 0 to 1 by step of 0.05.

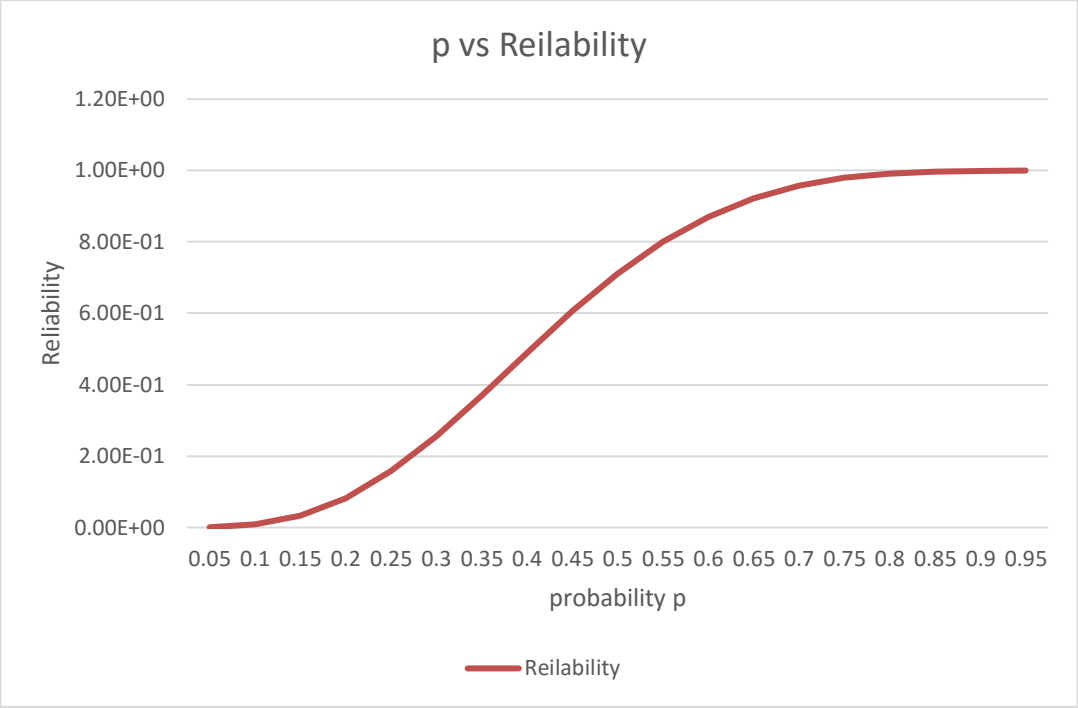
```
p : 0.05  Reliability : 6.306625414062497E-4
p : 0.1  Reliability : 0.008097522400000027
p : 0.15  Reliability : 0.03270003773203141
p : 0.2  Reliability : 0.08194549760000001
p : 0.25  Reliability : 0.15769195556640625
p : 0.3  Reliability : 0.2562604776000005
p : 0.35  Reliability : 0.3700509305445309
p : 0.4  Reliability : 0.48965386240000336
p : 0.45  Reliability : 0.6058200481664
p : 0.5  Reliability : 0.7109374999999999
p : 0.55  Reliability : 0.7998816729320468
p : 0.6  Reliability : 0.8702567423999946
p : 0.65  Reliability : 0.9221426923414128
p : 0.7  Reliability : 0.957513037599997
p : 0.75  Reliability : 0.9794998168945316
p : 0.8  Reliability : 0.9916645376000025
p : 0.85  Reliability : 0.9973945360914138
p : 0.9  Reliability : 0.9994922423999945
p : 0.95  Reliability : 0.999968610432053
```

The K is varied from 0 to 20 by the step of 1. To reduce the effect of randomness, the program is run for several(1024 times) to calculate reliability and averaged them out, for each value of k.

```
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9973945360914063    Change in Reliability = 0.0
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9967333410012067    Change in Reliability = 6.611950901995733E-4
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9959904927450909    Change in Reliability = 0.0014040433463153867
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.994211799250402     Change in Reliability = 0.0031827368410042 3
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9934784154010705    Change in Reliability = 0.003916120690335756
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9923535025955674    Change in Reliability = 0.0050410334958388425
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9920955392512899    Change in Reliability = 0.005298996840116366
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9913322173361467    Change in Reliability = 0.0060623187552595326
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9896396836562475    Change in Reliability = 0.007754852435158721
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9887614474592031    Change in Reliability = 0.008633088632203134
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9874741177425722    Change in Reliability = 0.009992041834883406
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9865150522058161    Change in Reliability = 0.010879483885590124
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9850451120618273    Change in Reliability = 0.012349424029578948
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9843717399901866    Change in Reliability = 0.013022796101219702
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9842480698492271    Change in Reliability = 0.013146466242179189
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9837294766748577    Change in Reliability = 0.013665059416548608
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9821506185444203    Change in Reliability = 0.015243917546985952
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9820415569060259    Change in Reliability = 0.015352979185380367
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9808068732861472    Change in Reliability = 0.01658766280525903
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9784254648663941    Change in Reliability = 0.018969071225012124
Before Applying K-Reliability = 0.9973945360914063    After Applying K-Reliability = 0.9784381578656591    Change in Reliability = 0.01895637822574714
```

## 5.2 Graphs :

### A. Probability p vs Reliability

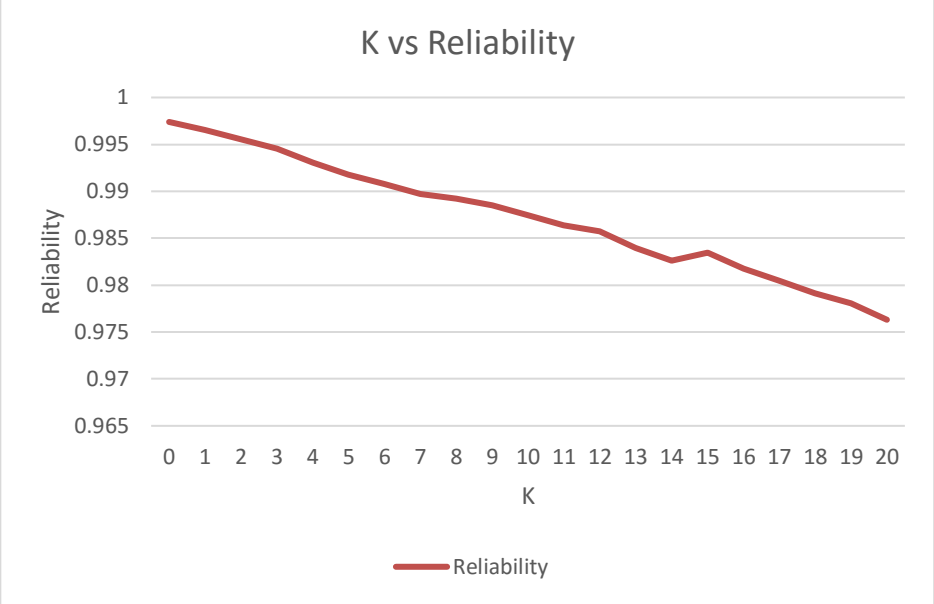



**Analysis:**

From the above graph we can conclude that the Reliability of the system increases with increase in probability.

As we increase the reliability of the individual link (i.e p ) the Reliability of the system increases because the reliability of the state is the function of the product of the reliability of individual probability of the links, thereby the reliability of the state increases. Since the reliability of the system is the sum of reliability of all the up states, it also increases.

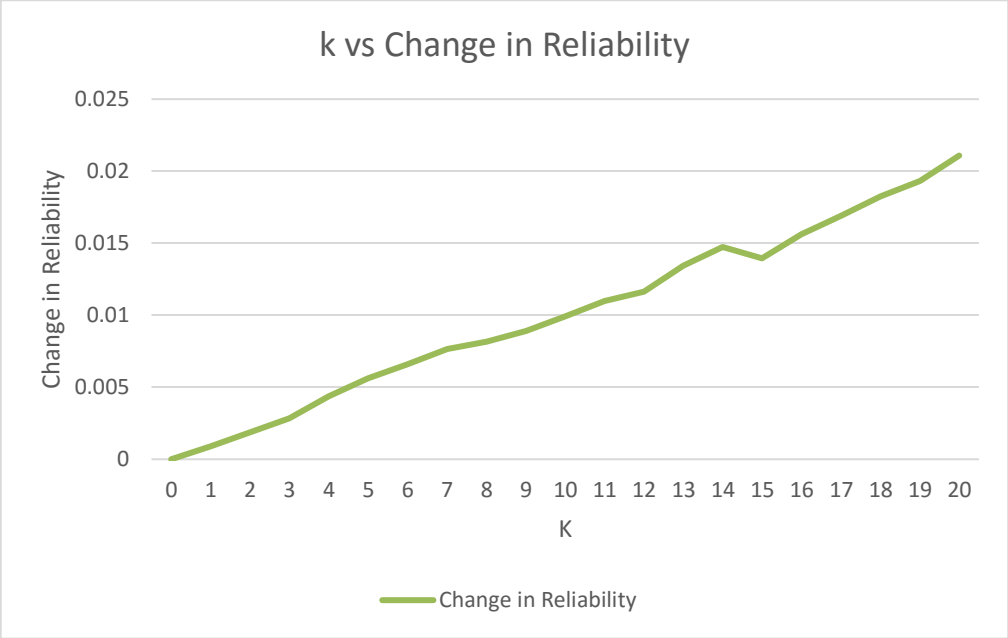
## B. k vs Reliability




Analysis :

As K increases the reliability of the system decreases. This is because as we increase the value of k, the number states that get flipped increases, therefore the number of states that were initially up may be changed to down, since the reliability of the system is the sum of reliability of all the Up states, it decreases.At some points the reliability increases, this is because the number of states that were initially down may be changed to up, since the reliability of the system is the sum of reliability of all the Up states, it may increase at some points.

## C. k vs Change in Reliability




Analysis:

As K increases the reliability of the system decreases. This is because as we increase the value of k, the number states that get flipped increases, therefore the number of states that were initially up may be changed to down, since the reliability of the system is the sum of reliability of all the Up states, it decreases.At some points the reliability increases, this is because the number of states that were initially down may be changed to up, since the reliability of the system is the sum of reliability of all the Up states, it may increase at some points.

Since the reliability of the system after flipping k states decreases with increase in k ,the change in reliability, which is the difference of reliability before flipping k states and reliability after flipping k states increases with increase in k. At some points the change in reliability decreases due to the increase in the reliability of the system after flipping k states as discussed above.

# Appendices

## Appendix A – Source Code

```
package Atn;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;

public class NetworkReliability {

	public static Map<Integer,Integer> indexMap= new HashMap<Integer,Integer>() ;// Hashmap of
links present
	public static int counter = 0;
	public static double reliability = 0; //Reliability of the system
	public static double p ;// reliability of each link
   public static boolean[] stateCondition = new boolean[1024]; //condition of all possible 1024 states
   public static double[] statesProbability = new double[1024]; // Reliability of all possible 1024 states
   public static int statesCounter = 0;

   /*Function to generate the graph of the network*/
   public static int[][] graphGenerator(int nodes) {

	int[][] graph = new int[nodes][nodes];
	for(int i=0; i<nodes; ++i)
	  for(int j=0; j<nodes; ++j){
		if(i==j)
			graph[i][j]=0; // to exclude self-loop
		else
			graph[i][j]=1;
	  }
	return graph;
   }

   /* Function to calculate the total links present */
   public static int calcEdges(int[][] graph, int nodes){

	int edgeCount =0;
	for(int i=0; i<nodes; ++i)
	  for(int j=0; j<nodes; ++j){
		if(graph[i][j] != 0)
```

```
                        edgeCount++;
        }
    return edgeCount;
}

/* Function to generate all possible link */
public static void createMap(int[][] graph, int nodes){

     int key = 1;
     int value =0;
     for(int i=1; i<nodes+1; ++i)
        for(int j=i+1; j<nodes+1; ++j){
                value = (10*i) + j;
                indexMap.put(key, value);// The possible links are stored in a HaspMap with a key
                ++key;
        }
}


/* Function to find the path between the source and every other nodes in the network */
static int[] checkStateCondition(int[][] graph, int nodes ,int src){

   int dist[] = new int[nodes]; // The output array. dist[i] will hold
                    // the shortest distance from src to i

   // sptSet[i] will true if vertex i is included in shortest
   // path tree or shortest distance from src to i is finalized
   Boolean sptSet[] = new Boolean[nodes];

   // Initialize all distances as INFINITE and stpSet[] as false
   for (int i = 0; i < nodes; i++)
   {
      dist[i] = 1000;
      sptSet[i] = false;
   }

   // Distance of source vertex from itself is always 0
   dist[src] = 0;

   // Find shortest path for all vertices
   for (int count = 0; count < nodes-1; count++)
   {
      // Pick the minimum distance vertex from the set of vertices
      // not yet processed. u is always equal to src in first
      // iteration.
      int u = minDistance(dist, sptSet);

      // Mark the picked vertex as processed
      sptSet[u] = true;

      // Update dist value of the adjacent vertices of the
      // picked vertex.
```

```
        for (int v = 0; v < nodes; v++)

            // Update dist[v] only if is not in sptSet, there is an
            // edge from u to v, and total weight of path from src to
            // v through u is smaller than current value of dist[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];
      }

      return dist;
   }

   public static int minDistance(int dist[], Boolean sptSet[])
   {
      // Initialize min value
      int min = Integer.MAX_VALUE, min_index=-1;

      for (int v = 0; v < 5; v++)
         if (sptSet[v] == false && dist[v] <= min)
         {
            min = dist[v];
            min_index = v;
         }

      return min_index;
   }

   /* Function to determine all possible combination of links failure states given number of links failed */
   static void linkFailureCombination(int arr[], int data[], int start,int end, int index, int r ,int[][] graph , int
nodes ,int edgeCount)
   {
       if (index == r) { // r is the number of link failure
         ++counter;
         int[][] tempGraph = new int[nodes][nodes];// initialize temp graph to original graph matrix
         for(int i=0; i<nodes; ++i)
            for(int j=0; j<nodes; ++j)
                     tempGraph[i][j] = graph[i][j];

          for (int j=0; j<r; j++){
          int value = indexMap.get(data[j]);
          int index_j = (value % 10);
          int index_i = (value-index_j)/10;

            tempGraph[index_i-1][index_j-1] = 0; // setting the link to fail
            tempGraph[index_j-1][index_i-1] = 0; // setting the link to fail

          }
          int[][] matrix = new int[nodes][nodes];
          /*Finding the distance between every node to every other node*/
          for(int src =0; src<nodes; ++src){
```

```
            matrix[src] = checkStateCondition(tempGraph, nodes, src);
        }

        boolean state = true;// set the state condition initally up

        for(int i=0; i<nodes; ++i){
                for(int j=0; j<nodes; ++j){
          if(i==j){
                  continue;
          }
          else if(matrix[i][j] == 0 || matrix[i][j] >= 1000 ) //if there is no path between any two nodes
                  state = false;// set the state condition to down
                }
        }
        stateCondition[statesCounter] = state;
        statesProbability[statesCounter++] = (Math.pow((1-p),r)*Math.pow(p,(edgeCount-r)));


        if(state){ // if the state condition is up calculate reliability for the state
                reliability += (Math.pow((1-p),r)*Math.pow(p,(edgeCount-r)));
          }
        return;
    }

    /*possible combination of r down links*/
    for (int i=start; i<=end && end-i+1 >= r-index; i++)
    {
       data[index] = arr[i];
       linkFailureCombination(arr, data, i+1, end, index+1, r, graph,nodes ,edgeCount);
    }
 }


static void linkCombination(int arr[], int n, int r, int[][] graph ,int nodes ,int edgeCount)
{
   int data[]=new int[r];
   linkFailureCombination(arr, data, 0, n-1, 0, r ,graph ,nodes ,edgeCount);
}

public static void main(String[] args) {

        int nodes = 5; // number of nodes
        int edgeCount ; // number of links
        p = 0.05;  // initailaizing probability
        DecimalFormat f = new DecimalFormat("#.##"); // Reliability output formatter

        while(p<=1){ // for p from 0.005 t0 1

        int[][] graph = new int[nodes][nodes];
        graph = graphGenerator(nodes); // function call to generate network graph
        edgeCount = calcEdges(graph,nodes) / 2; // function call to generate the links present
        createMap(graph,nodes); // generate hashmap of possible links
```

```
int[] indexArray = new int[edgeCount];
for(int i=0; i<edgeCount; ++i){
        indexArray[i] = i+1;
}
/*number of link failure varied from 0 to 10*/
for(int i=0; i<=edgeCount; ++i){
        linkCombination(indexArray,edgeCount, i,graph ,nodes, edgeCount);
}

System.out.println("p : " + f.format(p) + "  Reliability : " + reliability);

p += 0.05;
reliability = 0;
statesCounter = 0;
}

/* Flipping K system condition*/

p = 0.85;
int k=0;

ArrayList<Integer> stateNum = new ArrayList();
/*Array containg state numbers for random selection*/
for(int i=0; i<1024; ++i){
  stateNum.add(i);
}

double[] BefReliability = new double[21]; //Reliability of the system before flipping k states
double[] AfterReliability = new double[21];//Reliability of the system after flipping k states

int counter = 0;

/* Taking Average of 1024 trials for each value of k to reduce the effect of randomness*/

 while(counter < 1024){
 k=0;
while(k<=20)
{
        int[][] graph = new int[nodes][nodes];
        graph = graphGenerator(nodes);
        edgeCount = calcEdges(graph,nodes) / 2;
        createMap(graph,nodes);
        int[] indexArray = new int[edgeCount];
        for(int i=0; i<edgeCount; ++i){
                indexArray[i] = i+1;
        }

        /*number of link failure varied from 0 to 10*/
        for(int i=0; i<=edgeCount; ++i){
                linkCombination(indexArray,edgeCount, i,graph ,nodes, edgeCount);
        }
```

```
                Collections.shuffle(stateNum);// randomizing the states

                int[] kArray = new int[k];
                /*selecting the first k states of from random states list for flipping their condition*/
                for(int i=0; i<k;++i){
                        kArray[i] = stateNum.get(i);
                }

                /*Calculating reliability of the system before flipping k states*/
                double sum=0;
            reliability = 0;
                for(int i=0; i<1024;++i){
                        sum+=statesProbability[i];
                        if(stateCondition[i]) // if system condition is up calculate reliability
                                reliability += statesProbability[i];
                }

            BefReliability[k] += reliability;

            /*Fliping the condition of k states*/
            for(int i=0; i<k; ++i){

                boolean state = stateCondition[kArray[i]];
                stateCondition[kArray[i]] = (!state) ; // change the state condition
            }

                reliability = 0;
                sum=0;

                /*Calculating reliability of the system before flipping k states*/
                for(int i=0; i<1024;++i){
                        sum+=statesProbability[i];
                        if(stateCondition[i]) // if the system condition is up calculate reliability
                                reliability += statesProbability[i];
                }
                AfterReliability[k] += reliability;
                reliability = 0;
                statesCounter = 0;
                ++k; // k is varied from 0 to 20
        }
            counter++;// update counter
         }

        k=0;
        while(k<=20){
           System.out.print("\n Before Applying K-Reliability = " + BefReliability[k]/1024);
           System.out.print("     After Applying K-Reliability = " + (AfterReliability[k]/1024));
           System.out.print("     Change in Reliability = " + ((BefReliability[k] -
AfterReliability[k])/1024));
            k++;
        }
```

```
    }
}
```