STUDY EXPERIMENTALLY HOW NETWORK RELIABILITY DEPENDS ON THE INDIVIDUAL LINK RELIABILITIES

Algorithmic Aspect of Telecommunication Networks

DECEMBER 1, 2014

RAHUL SINGHAL
(Rxs132730)

Table of Contents

1 Problem statement

The purpose of the project is to student the experiment results of how the network reliability depends on the individual link reliabilities for the following network topology:

Network topology: A complete undirected graph on n = 5 nodes. This means, every node is connected with every other one (excluding self-loops and parallel edges). Hence the 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, same for every link. The parameter p will take different values starting from 0 to 1 inclusive.

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

1.1 Exhaustive Enumeration

Consider the configuration shown in the figure below. This is neither series nor parallel and not even obtained as a combination of the basic configuration. We show how we can compute the network reliability using exhaustive enumeration approach.

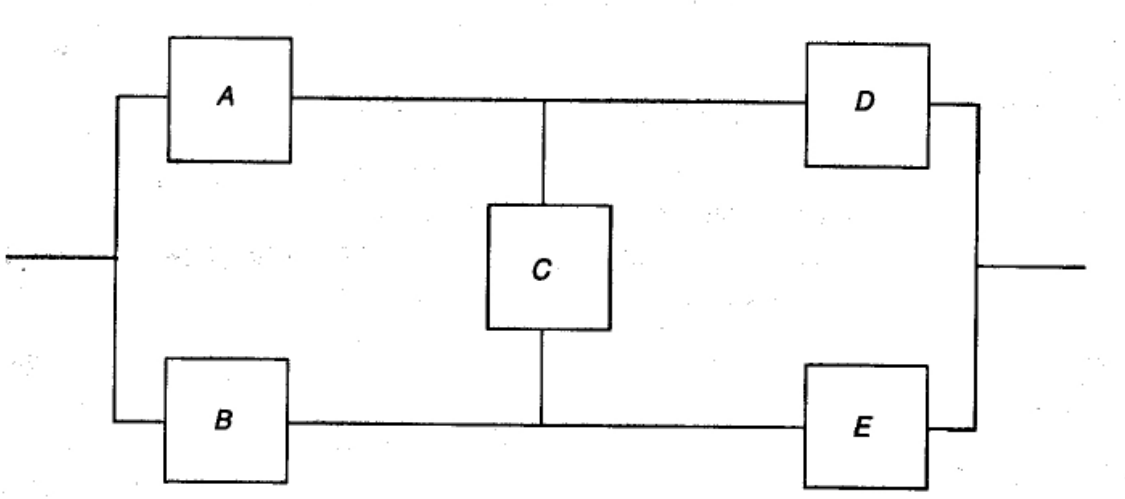


Figure 1 Network configuration

Exhaustive Enumeration

We can list all the possible states of the system (see the table below) 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.

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. $AB\overline{C}\overline{D}E$	Down
	18. $A\overline{B}C\overline{D}E$	Down
	19. $A\overline{B}CD\overline{E}$	Up
	20. $A\overline{B}\overline{C}\overline{D}E$	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. $A\overline{B}\overline{C}D\overline{E}$	Down
	28. $\overline{A}B\overline{C}\overline{D}E$	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. $\overline{A}\overline{B}\overline{C}\overline{D}E$	Down
5	32. $\overline{A}\overline{B}\overline{C}\overline{D}\overline{E}$	Down

Figure 2 All possible states for 5 links

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

Figure 3 Formula to calculate Network Reliability

2 Understanding

Input:

The input to the program is all the combination of the network 0 to 1023 where 0 – 0000000000 (one link is up), 1 – 0000000001 (one link is up), 2 – 0000000010 (one link is up), 3 – 0000000011 (two links are up),… , 1023 – 1111111111 (all links are up).

Output:

The output of the program computes the reliability of the network topology, based on the network connectivity depending on the entire state of the system (up or down).

The program should run in the following order:

1. Compute the reliability of the network based on different probability values from 0 to 1 increments of 0.02
    a. Loop for probabilities p = 0 to 1 increments of 0.02
    b. Generate all possible states from 0 to 1023
    c. Fill the down edges in the network
    d. Determine the down edges and update the adjacency matrix
    e. If the network is connected then compute the network reliability
2. Compute the reliability of the network for different values of k (determines the number of down states in the network) for a fixed probability p = 0.9 and average out the result for 100 trials each
    a. Loop k = 0 to 100

b. Loop for trials = 0 to 100
c. Generate k-sized nodes from the list of all the nodes
d. For all possible state combinations = 0 to 1023 fill the down edges
e. If the down edge belongs to the k-sized list then flip the bit for that particular edge
f. Determine the down edges and update the adjacency matrix
g. If the network is connected then compute the network reliability

3 Algorithm

We have used 'Depth-first search' (DFS) algorithm to compute the connectivity of the network for the given adjacency matrix.

DFS is an algorithm used for traversing or search a graph data structures. One starts at the root and explores as far as possible along each branch before backtracking.

3.1 Pseudo code

Exhaustive Enumeration

```
for p = 0 to 1.01, 0.02 increment
     for all possible states, i=0 to 1023
      fillDownEdges(i, numberOfNodes)
      create new Network Graph (numberOfNodes)
      fill the adjacency matrix in the graph created above by updating the
down edges
      if the Network Graph is connected
            compute the network reliability with the following formula
            goodEdges =( numberOfNodes * (numberOfNodes – 1) / 2 ) –
downEdges
            networkReliability += (1 – p)^downEdges * (p)^goodEdges
```

Flipping edges using exhaustive enumeration

```
Repeat experiment for k = 0 to 99
      For trials = 0 to max_trials (100)
            Generate a k-sized random set of edges
            For all possible states, i = 0 to 1023
                  If(k-seized set contains i)
                        fillDownEdges(~i, numberOfNodes)
                  else
                        fillDownEdges(i, numberOfNodes)
                  create new Network graph (numberOfNodes)
                  fill the adjacency matrix in the graph created above by
            updating the down edges
                  if the Network Graph is connected
                        compute the network reliability with the following
formula
                        goodEdges =(numberOfNodes * (numberOfNodes – 1) / 2 )
```

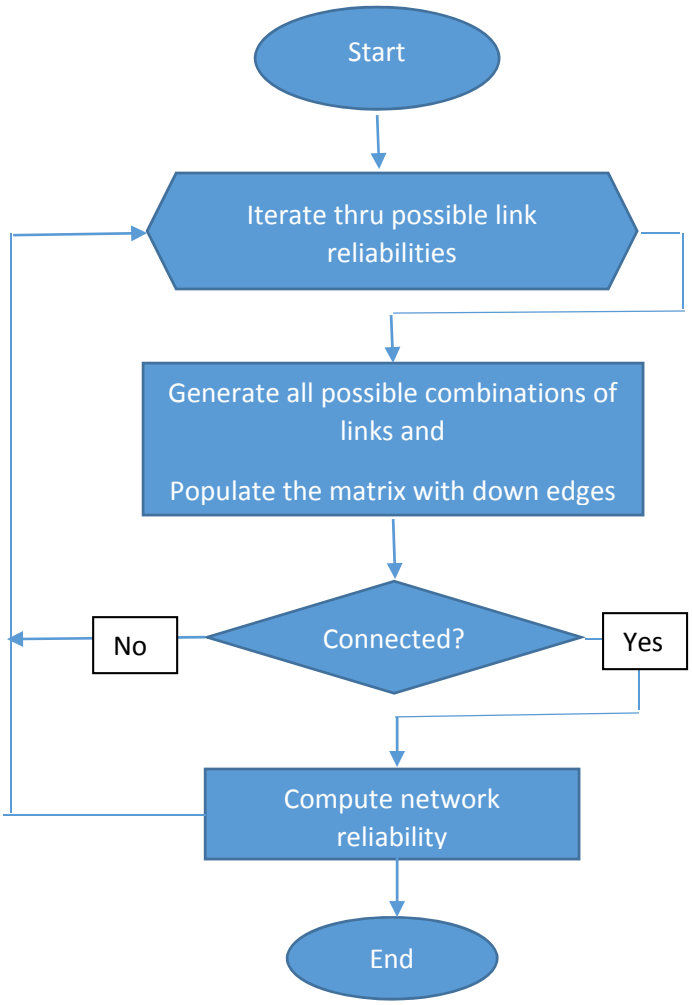
$-downEdges$

$$networkReliability \mathrel{+}= (1 - p)^{downEdges} * (p)^{goodEdges}$$

Iterative DFS

```
procedure DFS-iterative(G,v):
     let S be a stack
     S.push(v)
     while S is not empty
          v ← S.pop()
          if v is not labeled as discovered:
             label v as discovered
             for all edges from v to w in G.adjacentEdges(v) do
               S.push(w)
```

3.2 Flow chart



4 Briefing of the Program

Regular experiment using exhaustive enumeration

The program is implemented in Java programming language and windows 8 operating system.

For displaying the graph, Microsoft Excel's chart function is used. The input to the graph are the values of the network reliability of regular experiment and the flipped bit experiment.

For each of the computation, we generate all possible states for number of node, n = 5 and number of edges, m = 10 a total of 1024 states. We fill the downEdges for the possible state in each iteration and update the adjacency matrix.

The network is checked for connectivity and if the network is connected then the network reliability is computed.

For the flip bit experiment

For each value of k = 0 to 99, perform trials from 0 to 99 and For each of the computation, we generate all possible states for number of node, n = 5 and number of edges, m = 10 a total of 1024 states. Then for each value of k, we compute random, k-sized set of edges in the state and if the all combination state contains the k-sized edge then we fill downEdge(~i) else We fill the downEdges(i) for the possible state in each iteration and update the adjacency matrix.

The network is checked for connectivity and if the network is connected then the network reliability is computed by averaging the number of trials for each k.

5 Experimental Results and Graphs

5.1 Experiment 1:

Link Reliability (P) vs Network Reliability (R)

Reliability of each link (P)	Network Reliability (R)
0	0
0.02	1.84E-05
0.04	2.70E-04
0.06	0.001252
0.08	0.003625
0.1	0.008098
0.12	0.015351
0.14	0.025974
0.16	0.040431
0.18	0.059036
0.2	0.081945
0.22	0.109159
0.24	0.14053
0.26	0.17578
0.28	0.214517
0.3	0.25626
0.32	0.300459
0.34	0.346518
0.36	0.393816
0.38	0.441731
0.4	0.489654
0.42	0.537008
0.44	0.583259
0.46	0.62793
0.48	0.670606
0.5	0.710938
0.52	0.74865
0.54	0.78354
0.56	0.815472
0.58	0.84438
0.6	0.870257
0.62	0.893154
0.64	0.913171
0.66	0.93045
0.68	0.945163
0.7	0.957513

0.72	0.967717
0.74	0.976003
0.76	0.982604
0.78	0.987751
0.8	0.991665
0.82	0.994556
0.84	0.996618
0.86	0.998028
0.88	0.998942
0.9	0.999492
0.92	0.999793
0.94	0.999935
0.96	0.999987
0.98	0.999999
1	1

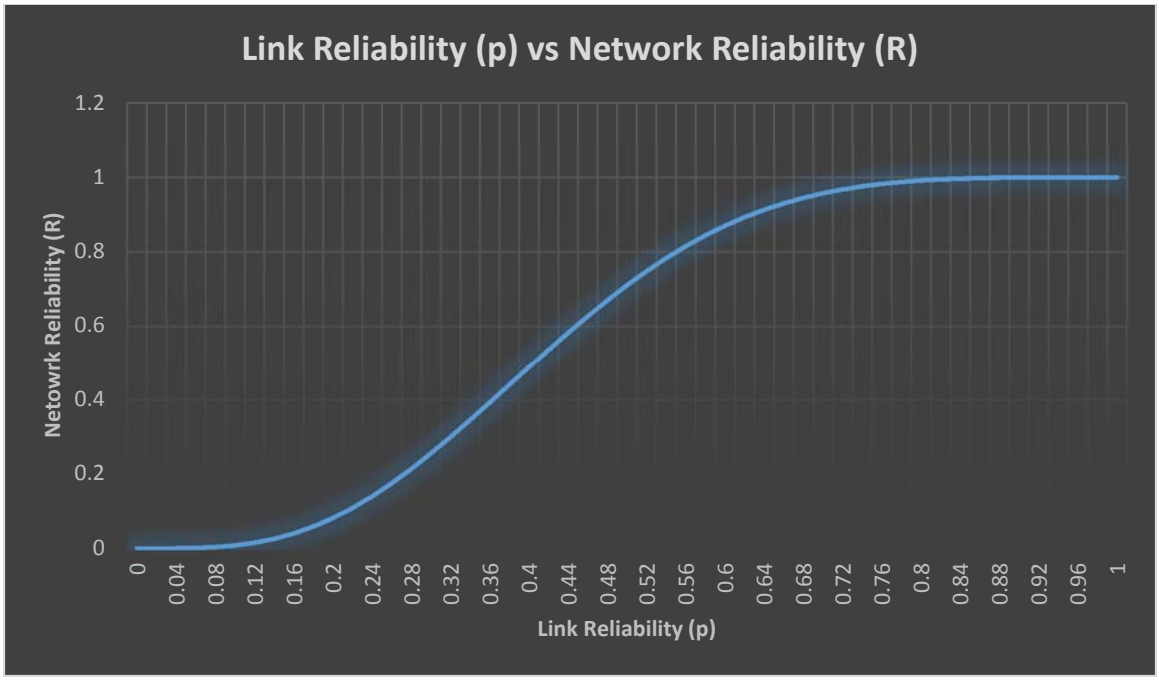


Figure 4 Link Reliability vs Network Reliability curve

5.2 Experiment 2:

K-flipped Link Reliability (k) vs Network Reliability (R)

K-flipped Link (k)	Network Reliability (R)
0	0.999492
1	0.996085
2	1.00295
3	1.000081
4	0.998035
5	1.006098
6	0.99764
7	1.000954
8	0.995297
9	0.994645
10	1.001672
11	0.999661
12	1.009919
13	1.000021
14	0.998916
15	1.008179
16	0.998574
17	0.99475
18	1.006231
19	1.00083
20	1.000871
21	1.002463
22	1.001713
23	1.0008
24	0.998195
25	0.993292
26	1.006442
27	0.995655
28	1.002484
29	0.982747
30	1.002494
31	0.998594
32	1.019744
33	0.987025
34	1.000487
35	1.000722
36	1.005288
37	0.991129

38	0.990273
39	0.989684
40	0.99365
41	0.988716
42	1.008051
43	0.988633
44	1.002078
45	1.004855
46	1.012752
47	0.990025
48	0.990339
49	0.994349
50	0.98484
51	0.994698
52	0.975876
53	1.00174
54	1.007155
55	1.006158
56	1.013293
57	0.995483
58	1.001829
59	0.99968
60	0.983731
61	1.009813
62	0.984632
63	1.014558
64	1.003665
65	0.993699
66	1.010177
67	0.966074
68	0.99696
69	1.001588
70	0.995484
71	0.99383
72	1.025003
73	0.992865
74	1.002105
75	0.990066
76	1.021072
77	1.013477
78	1.00077

79	0.982428
80	0.985032
81	0.9893
82	1.001647
83	1.003886
84	0.986198
85	0.99059
86	0.991982
87	1.021931
88	0.997387
89	1.021222
90	1.010725
91	1.000308
92	0.995892
93	1.009805
94	0.990912
95	0.999226
96	0.99946
97	1.005004
98	0.98092
99	1.005342

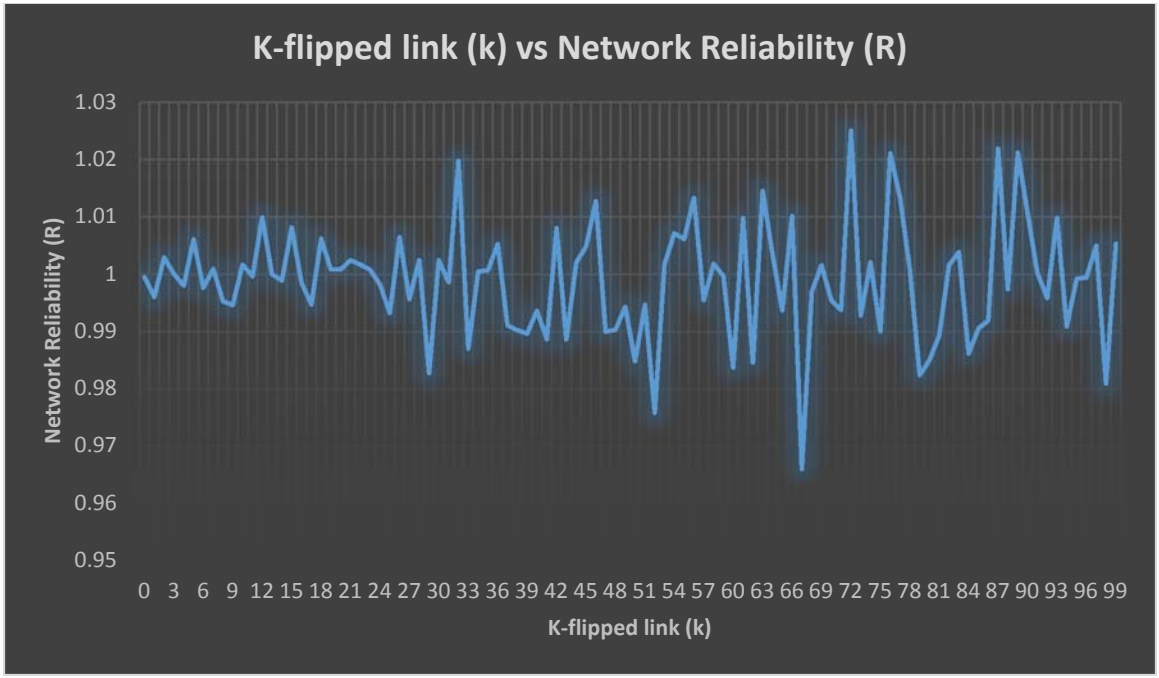


Figure 5 K-flipped link vs Network Reliability

The below graph is the same as the Figure 5 but with a different scale of representation.

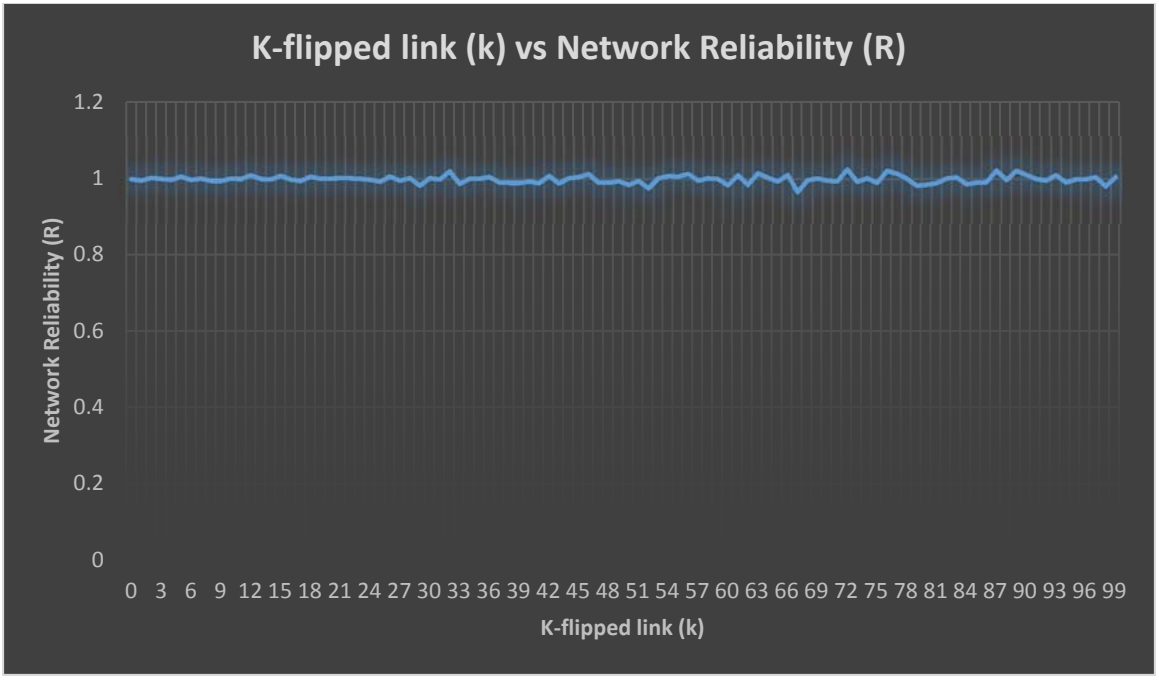


Figure 6 k vs R (0 to 1 scale)

The above graph is the same as the previous one, but has a different scale (0 to 1) which depicts the network reliability for p = 0.9, when the values of k is varied the entire network reliability is close above 0.9 and 1.0.

6 Conclusion

- We conclude from our experiments that the value of the network reliability (R) increases and become stable with increase in each link reliability (p).
- When the bits are flipped for randomly chosen k-links in the network, the network reliability (R) remains almost constant and the value varies between 0.9 and 1.0.
- The main goal of the project, to compute the network reliability is achieved.

7 Deduction from the above graph(s):

The above graph shows the variation of each link reliability (p) vs the total network reliability (R). We notice that curve increases with the reliability of each link in the network. When the each link reliability is low, the entire network reliability is low too and when the each link reliability increases and the entire network reliability become more reliable as each link in the network is reliable.

The above graph shows the variation of K-flipped links i.e., if the link is up, we set it to down and vice versa for the k-links in the network (k) vs the total network reliability (R). We notice that curve is more like a heartbeat. The reason for such a behavior is that the k links are chosen at random and for each randomly chosen set, if the flipped is among the combination that adds up for several runs sometimes makes the network more reliable with each link's reliability being equal to 0.9 which is much higher; this configuration may lead to an increased network reliability due to the flipping of the network.

8 References

- http://en.wikipedia.org/wiki/Depth-first_search

9 Source Code

NetworkReliability.java

```
/* Author : Rahul Singhal
 * Created Date: 11/15/2014
 * Description: Solution for the ATN project 3
 */

import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Random;

public class NetworkReliability {

	static ArrayList<Integer> downEdges = null;
	static Network network = null;

	public static void main(String[] args) {
		computeReliability();
		flippedBitReliability();
	}

	// computes the reliability of the network using various value of p
	private static void computeReliability() {

		/*
		 * All combination of the network can be represented by numbers 0
to
		 * 1023 0 - 0000000000 (one link is up) 1 - 0000000001 (one link
is up)
```

```
             * 2 - 0000000010 (one link is up) 3 - 0000000011 (two links are
up) ..
             * .. 1023 - 1111111111 (all links are up)
             */

            /*
             * The edges are encoded as follows: 0-1 : 0 0-2 : 1 0-3 : 2 0-4
: 3 1-2
             * : 4 1-3 : 5 1-4 : 6 2-3 : 7 2-4 : 8 3-4 : 9
             */

            // number of nodes
            int n = 5;
            // store link probability values
            ArrayList<Double> probabilityValues = new ArrayList<>();
            // store reliability values
            ArrayList<Double> reliabilityValues = new ArrayList<>();
            // store the edges that are down
            downEdges = new ArrayList<>();

            for (double p = 0.00; p <= 1.01; p = p + 0.02) {
                  // Reliability of the netowrk computed wrt p
                  double networkReliability = 0.00;

                  for (int i = 0; i < 1024; i++) {
                        downEdges.clear();
                        // a function that fills up all the down edges for
the given
                        // index i
                        fillDownEdges(downEdges, i, n);
                        // create a new graph with n nodes and set all the
links to 1
                        // (up)
                        network = new Network(n);
                        // get all the down edges and set its value (weight)
and update
                        // the matrix
                        for (Integer dE : downEdges) {
                              Edge e = getRowColumn(dE, n);
                              network.addUndirectedDownEdge(e);
                        }
                        // if the network is up add to the reliability values
                        if (network.isConnected()) {
                              int upEdge = n * (n - 1) / 2 -
downEdges.size();
                              int downEdge = downEdges.size();
                              networkReliability += Math.pow(1 - p, downEdge)
                                          * Math.pow(p, upEdge);
                        }
                  }
```

```
                // store the p and network reliability values
                probabilityValues.add(p);
                reliabilityValues.add(networkReliability);
            }
            System.out.println(reliabilityValues);
            // write the results to file
            writeToFile(probabilityValues, reliabilityValues, "PvsR.csv");
      }

      private static void flippedBitReliability() {
            /*
             * All combination of the network can be represented by numbers 0
to
             * 1023 0 - 0000000000 (one link is up) 1 - 0000000001 (one link
is up)
             * 2 - 0000000010 (one link is up) 3 - 0000000011 (two links are
up) ..
             * .. 1023 - 1111111111 (all links are up)
             */

            /*
             * The edges are encoded as follows: 0-1 : 0 0-2 : 1 0-3 : 2 0-4
: 3 1-2
             * : 4 1-3 : 5 1-4 : 6 2-3 : 7 2-4 : 8 3-4 : 9
             */

            // number of nodes
            int n = 5;
            // p (link reliability) constant for the experiment
            double p = 0.9;
            // k = 0,1,2..,99
            ArrayList<Double> kProbabilityValues = new ArrayList<>();
            // store network reliability values for corresponding k
            ArrayList<Double> kReliabilityValues = new ArrayList<>();
            // store k random edges for which the bits will be flipped
            ArrayList<Integer> KRandomConfig = new ArrayList<>();

            for (double k = 0; k < 100; k++) {
                // reliability of the network with the given p values
                double kNetworkReliability = 0.00;
                for (int trial = 0; trial < 100; trial++) {
                    KRandomConfig.clear();
                    // method to select k random edges from the 2^10
edges available
                    KRandomConfig = getKRandomConfigurations(k,
                            (int) Math.pow(2, n * (n - 1) / 2));
                    for (int i = 0; i < 1024; i++) {
                        downEdges.clear();
                        // check if the k random edges contains the
index i
```

```
                        if (KRandomConfig.contains(i))
                              // fill it by flip the edge value
                              fillDownEdges(downEdges, ~i, n);
                        else
                              fillDownEdges(downEdges, i, n);
                        network = new Network(n);
                        // for each of the down edge add / update the
appropriate
                        // adj matrix value
                        for (Integer dE : downEdges) {
                              Edge de = getRowColumn(dE, n);
                              network.addUndirectedDownEdge(de);
                        }

                        // if the network is up, compute and add the
network
                        // reliability
                        if (network.isConnected()) {
                              int upEdge = n * (n - 1) / 2 -
downEdges.size();
                              int downEdge = downEdges.size();
                              kNetworkReliability += Math.pow(1 - p,
downEdge)
                                          * Math.pow(p, upEdge);
                        }
                  }
            }
            // store the k and network reliability values by diving the
number
            // of trials
            kProbabilityValues.add(k);
            kReliabilityValues.add(kNetworkReliability / (double)
100.0);
      }
      System.out.println(kReliabilityValues);
      // write the result to a file
      writeToFile(kProbabilityValues, kReliabilityValues, "KvsR.csv");
}

// returns an array list of number from 0 to pow-1 (2^10 -1)
private static ArrayList<Integer> getKRandomConfigurations(double k,
int pow) {
      Random rand = new Random();
      ArrayList<Integer> randomSet = new ArrayList<Integer>();
      while (randomSet.size() < k) {
            int randomValue = Math.abs(rand.nextInt()) % pow;
            if (!randomSet.contains(randomValue))
                  randomSet.add(randomValue);
      }
      return randomSet;
```

```
	}

	// returns the edge of the down edge (dE), if condition is satisfied
then
	// return the new edge of weight 0, else return the 0 edge
	private static Edge getRowColumn(Integer dE, int n) {
		n = n - 1;
		int row = 0;
		do {
			if (dE < n)
				return new Edge(row, row + dE + 1, 0);
			row++;
			dE = dE - n;
			n = n - 1;
		} while (n >= 0 && dE >= 0);
		return new Edge(0, 0, 0);
	}

	// fill the down edges in the array list for each value of i; by
generating
	// all possible values
	private static void fillDownEdges(ArrayList<Integer> downEdges, int i,
int n) {
		int checker = 1;
		for (int k = 0; k < n * (n - 1) / 2; k++) {
			int result = checker & i;
			if (result == 0)
				downEdges.add(k);
			checker = checker << 1;
		}
	}

	// write the information in the list to a csv file
	private static void writeToFile(ArrayList<Double> probabilityValues,
			ArrayList<Double> reliabilityValues, String fname) {
		try {
			FileWriter fw = new FileWriter(fname);
			// code to write to files
			DecimalFormat df = new DecimalFormat("#.##");
			for (int i = 0; i < probabilityValues.size(); i++) {
				fw.write(df.format(probabilityValues.get(i)) + ","
						+ reliabilityValues.get(i) + "\n");
			}
			fw.close();
		} catch (IOException ex) {
			ex.printStackTrace();
		}
	}
}
```

Network.java

```
/*References:
 * http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/Graph/dfs.html
 *
 */

import java.util.ArrayList;

public class Network {

	/*
	 * Network is defined as an adjacencyMatrix of n*n n is number of
vertices
	 * of the graph
	 */

	int[][] adjacencyMatrix;
	int nodes;

	// when a new network graph is connected, fill it with 1's (up)
	public Network(int n) {
		nodes = n;
		adjacencyMatrix = new int[nodes][nodes];
		for (int i = 0; i < nodes; i++) {
			for (int j = 0; j < nodes; j++) {
				adjacencyMatrix[i][j] = 1;
			}
		}
	}

	// set the value of the graph to 0 (down) from the down edge list
	public void addUndirectedDownEdge(Edge e) {
		adjacencyMatrix[e.row][e.col] = e.weight;
		adjacencyMatrix[e.col][e.row] = e.weight;
	}

	// DFS algorithm to check the connectivity of the graph
	// returns true if the graph is connected, else false
	public boolean isConnected() {
		boolean[] visited = new boolean[nodes];
		int vertices = nodes;
		ArrayList<Integer> next = new ArrayList<>();
		next.add(0);
		while (next.size() != 0) {
			int i = next.get(0);
			visited[i] = true;
			vertices--;
```

```
                        next.remove(0);
                        for (int k = 0; k < nodes; k++) {
                              if (adjacencyMatrix[i][k] > 0 && !visited[k]
                                          && !next.contains(k)) {
                                    next.add(k);
                              }
                        }
                  }
                  if (vertices == 0) {
                        return true;
                  } else {
                        return false;
                  }
            }
}
```

10 Readme

Create java file with class name as 'NetworkReliability'.

(1). Language used: Java

(2). Compilation Command or the IDE you are using.
IDE used- Eclipse
For compilation using command prompt refer below section.

(3). Command for running.
For compilation using command prompt refer below section.
With Eclipse IDE, Press CTRL+F11 to run the program.
Please, follow the sequence of instructions given.
******* INSTRUCTIONS *******

For compiling and executing using command prompt:
1. Open a command prompt window and browse up to this extracted folder.
Navigate to the 'src' folder.

2. Compilation of all the file/s
Command to be entered: javac <filename>.java
e.g. javac Program.java

3. Execution of compiled files
Command to be entered: java <filename>
e.g. java Program