Algorithmic Aspects of Telecommunication Networks

CS 6385.001- Fall 2017

PROJECT 3

Network Reliability for Undirected Graphs

Submitted by:

Ponmalar Silambaram Chandrabose (pxs162030)

CONTENTS:

Network Reliability:

It is defined as the process by which the network offers the same services even during the event of a failure.

Objective:

To study experimentally how the network reliability depends on the individual link reliabilities where the conditions are:

Network Topology – A complete undirected graph with n=5 nodes. This means every node is connected to every other node.

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 very different values in the experiments.

Reliability configuration – The system should be operational, if the network is connected.

The Overview:

The main purpose of this system is to study experimentally how the network reliability depends on the individual link reliabilities. The main functionalities are as follows:

1. To create an algorithm the computes the network reliability using exhaustive enumeration.
2. Write a computer program that implements the algorithm.
3. Run the program for different values of p. Let the parameter p run over the interval [0,1] in steps of 0.04.
4. Show graphical representations of the obtained reliability to that of p.
5. Now, fix the parameter p as 0.85 and pick k of the combinations randomly from the possible combinations that will be generated. Show a representation of how the reliability of the system changes due to alteration of the condition of the system.

Solution to the problem:

1. An exhaustive enumeration algorithm involves listing all the possible states of the system in an up and down system condition to each state.

By summing up the probability of the up states in the system we can determine the reliability of the system.

2. For every link we determine the network reliability of the system for varying values of p and graphically plot them to determine how each reliability varies with the change in the value of p.

Exhaustive Enumeration:

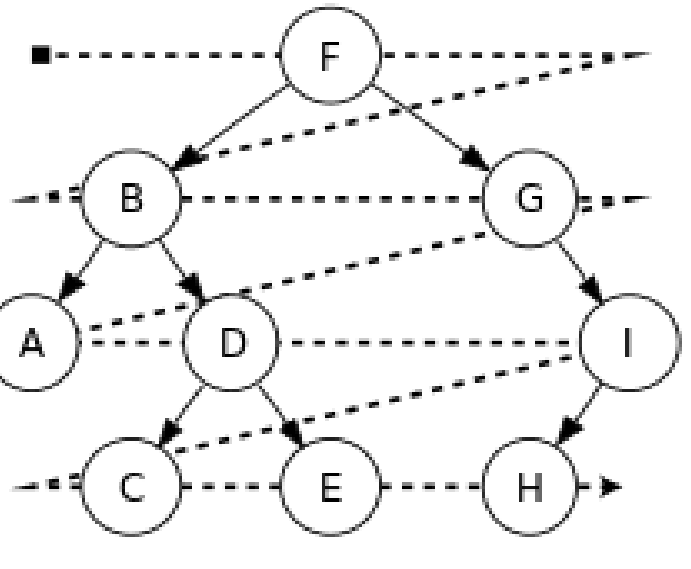
The algorithms that are used are as follows:

- Breadth First search
- Combination List generator
- Generating graphs and their reliability

Breadth First Search:

The BFS algorithm is used to traverse the tree structure or the network level by level to cover all the vertices that are connected to the root node.

BFS, at all times maintains a wave-front, that separates the states already visited from those yet to be encountered.



The above network is the accurate representation of how the BFS traverses through a network.

Pseudocode of BFS Algorithm:

```
unmark all vertices
    choose some starting vertex x
    mark x
    list L = x
    tree T = x
    while L nonempty
    choose some vertex v from front of list
    visit v
    for each unmarked neighbor w
        mark w
        add it to end of list
        add edge vw to T
```

Combination List Generator:

We create a temporary array which stores the outputs one by one. The two combinations that recur are:

1. The element is included in the current combination (Include value in the array and increment index by 1)
2. The element is excluded in the current combination (The element is not included in the array and the index is not incremented)

The logic is used recursively for a large set of numbers to obtain the respective combination list.

Graph Generator with reliabilities:

Each system state is analysed and all the possible states of the system are listed out as either UP/DOWN. This is

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

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

Output values:

When the value of p changes:

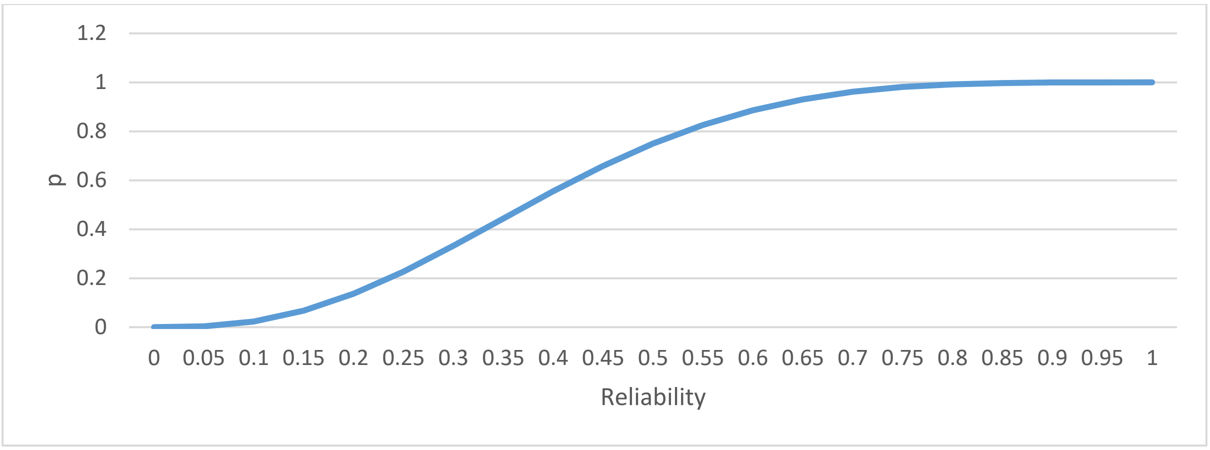
```
For p= 0.00 reliability =0.0
For p= 0.05 reliability = 0.003295370644921898
For p= 0.10 reliability = 0.02297787039999961
For p= 0.15 reliability = 0.06706778734336008
For p= 0.20 reliability = 0.13647144959999974
For p= 0.25 reliability = 0.22721481323242188
For p= 0.30 reliability = 0.3324970295999962
For p= 0.35 reliability = 0.44442338406210746
For p= 0.40 reliability = 0.5553455104000035
For p= 0.45 reliability = 0.6587904140824209s
For p= 0.50 reliability = 0.75
For p= 0.55 reliability = 0.8261309140433618
For p= 0.60 reliability = 0.8861819903999983
For p= 0.65 reliability = 0.9307248651949256
For p= 0.70 reliability = 0.9615137895999958
For p= 0.75 reliability = 0.9810447692871094
For p= 0.80 reliability = 0.9921232896000003
For p= 0.85 reliability = 0.9974854745699209
For p= 0.90 reliability = 0.9995009904000017
For p= 0.95 reliability = 0.9999687577933587
For p= 1.0 reliability = 1.0
```

When the value of k changes p remains same:

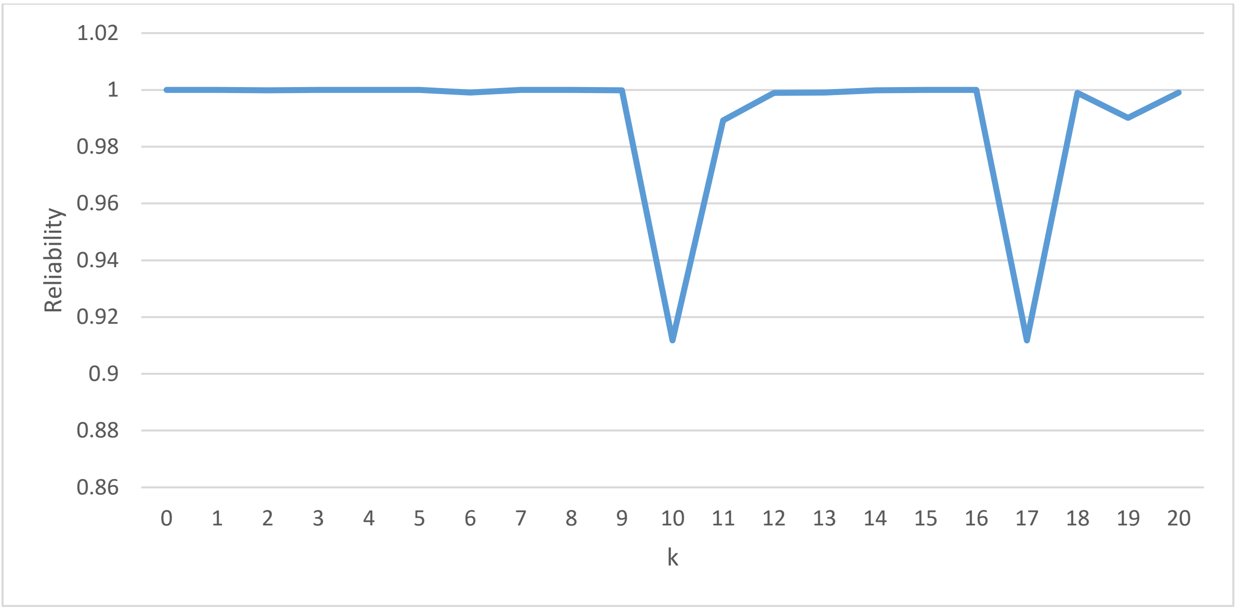
```
For p= 0.85 and k = 0 reliability =0.9999900000000002
For p= 0.85 and k = 1 reliability =0.9999900000000002
For p= 0.85 and k = 2 reliability =0.9999810000000002
For p= 0.85 and k = 3 reliability =0.9999900000000002
For p= 0.85 and k = 4 reliability =0.9999900000000002
For p= 0.85 and k = 5 reliability =0.9999900000000002
For p= 0.85 and k = 6 reliability =0.9990990000000002
For p= 0.85 and k = 7 reliability =0.9999900000000002
For p= 0.85 and k = 8 reliability =0.9999900000000002
For p= 0.85 and k = 9 reliability =0.9998910000000001
For p= 0.85 and k = 10 reliability =0.9117720000000002
For p= 0.85 and k = 11 reliability =0.9891990000000002
For p= 0.85 and k = 12 reliability =0.999
For p= 0.85 and k = 13 reliability =0.9990990000000002
For p= 0.85 and k = 14 reliability =0.9998910000000002
For p= 0.85 and k = 15 reliability =0.9999900000000002
For p= 0.85 and k = 16 reliability =0.9999900000000002
For p= 0.85 and k = 17 reliability =0.9117810000000002
For p= 0.85 and k = 18 reliability =0.999
For p= 0.85 and k = 19 reliability =0.9901890000000002
For p= 0.85 and k = 20 reliability =0.9990990000000002
```

Graphical Representation:

P v/s Reliability:



K v/s reliability



Appendix:

```
import java.util.Scanner;
import java.io.*;
import java.util.*;

class Graph
{
   public int N;   // Nodes in the graph
   public LinkedList<Integer> adj[]; //Adjacency Lists

   // Constructor
   public Graph(int n)
   {
      N = n;
      adj = new LinkedList[n];
      for (int i=0; i<n; ++i)
         adj[i] = new LinkedList();
   }

   public Graph(Graph g){
      this.N=g.N;
      adj=new LinkedList[g.adj.length];
      for(int i=0;i<g.adj.length;i++){
         adj[i]=new LinkedList();
         this.adj[i].addAll(g.adj[i]);
      }
   }

   // Function to add an edge into the graph
   void addEdge(int v,int w)
   {
      adj[v].add(w);
   }
   // Function to remove an edge from the graph
   void remove(int v, int w){
      for(int i=0;i<adj[v].size();i++){
         if(adj[v].get(i)==w){
            adj[v].remove(i);
         }
      }
```

```
    for(int i=0;i<adj[w].size();i++){
        if(adj[w].get(i)==v){
            adj[w].remove(i);
        }
    }
}

// using BFS traversal from a given source s to know UP/DOWN states
boolean BFS(int s)
{
    // Mark all the vertices as not visited(By default set as false)
    boolean visited[] = new boolean[N];

    // Create a queue for BFS
    LinkedList<Integer> queue = new LinkedList<Integer>();

    // Mark the current node as visited and enqueue it
    visited[s]=true;
    queue.add(s);

    while (queue.size() != 0)
    {
        // Dequeue a vertex from queue and print it
        s = queue.poll();
        Iterator<Integer> i = adj[s].listIterator();
        while (i.hasNext())
        {
            int n = i.next();
            if (!visited[n])
            {
                visited[n] = true;
                queue.add(n);
            }
        }
    }
// Counting the visited nodes
    int countVisited=0;
    for(int i=0;i<N;i++){
        if(visited[i]==true){
            countVisited++;
        }
        }
    // Verification and returning the decision
        if(countVisited==N){
            return true;
        }else{
            return false;
        }
    }
}

public class Project2 {
    public ArrayList<int[]> combinationList=new ArrayList<>();

    // Combination Utility recursive function
     void combinationUtil(int arr[], int data[], int start, int end, int index, int r)
     {
        if (index == r)
        {
            int temp[]=new int[r];
            for (int j=0; j<r; j++){
                temp[j]=data[j];
            }
            this.combinationList.add(temp);
            return;
        }

        for (int i=start; i<=end && end-i+1 >= r-index; i++)
        {
            data[index] = arr[i];
            combinationUtil(arr, data, i+1, end, index+1, r);
        }
     }
    // Parent function that calls the combination utility function
    void printCombination(int arr[], int n, int r)
    {
        int data[]=new int[r];
        combinationUtil(arr, data, 0, n-1, 0, r);

    }
    // Graph generator function with reliability calculations
    public double generateGraphs(double[] p,int E, int factor){
```

```
    double totalReliability = 0;
// Creating a base graph with 10 edges
    Graph g = new Graph(5);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(0, 3);
    g.addEdge(0, 4);

    g.addEdge(1, 0);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);

    g.addEdge(2, 0);
    g.addEdge(2, 1);
    g.addEdge(2, 3);
    g.addEdge(2, 4);

    g.addEdge(3, 0);
    g.addEdge(3, 2);
    g.addEdge(3, 1);
    g.addEdge(3, 4);

    g.addEdge(4, 0);
    g.addEdge(4, 2);
    g.addEdge(4, 3);
    g.addEdge(4, 1);

    //Temporary graph class object with same edges as the base graph
    Graph defGraph = new Graph(g);

    int arr[]={0,1,2,3,4,5,6,7,8,9};
    int mediator[][]={{0,1},{0,2},{0,3},{0,4},{1,2},{1,3},{1,4},{2,3},{2,4},{3,4}};

    for(int i=1;i<=E;i++)
      this.printCombination(arr, E, i);

    double reliable[]=new double[1024];
    double state[]=new double[1024];
    double test=1;
```

```
        for(int i=0;i<p.length;i++){
           test=test*p[i];
        }
        for(int i=0;i<this.combinationList.size();i++){
           double reliability=1;
           int arrEdge[]=this.combinationList.get(i);

           for(int j=0;j<arrEdge.length;j++){
              g.remove(mediator[arrEdge[j]][0], mediator[arrEdge[j]][1]);
           }
     // Reliability calculation for each combination
           for(int ko=0;ko<10;ko++){
              reliability=reliability*p[ko];

           }
           for(int ko=0;ko<arrEdge.length;ko++){
              reliability=(reliability/p[arrEdge[ko]])*(1-p[arrEdge[ko]]);
           }
     // BFS function called to find UP/DOWN based on that assigning the respective reliabilities
  and respective state
           if(g.BFS(0)){
              reliable[i]=reliability;
              state[i]=1;
           }else{
              state[i]=0;
              reliable[i]=reliability;
           }
           g=new Graph(defGraph);
        }
        double testReliable = 0;
     // Using random number generator for K values (K – factor variable)
        Random ran = new Random();

        for(;factor!=0;factor--){
           int x = ran.nextInt(1024);
           if(state[x]==0){
              state[x]=1;
           }else{
              state[x]=0;
           }
        }
```

```
    // Calculating final reliability and returning the results
        for(int boo=0;boo<1024;boo++){
            if(state[boo]==1){
                testReliable=testReliable+reliable[boo];
            }
        }
        return testReliable;
    }

    public static void main(String args[]){
        System.out.println("Enter UTD ID: ");
        Scanner s=new Scanner(System.in);
        String d=s.nextLine();
        double p[][]=new double[20][10];
    // Calculating link probability or p value
        for(int i=0;i<20;i++){
            for(int j=0;j<10;j++){
                p[i][j]=Math.pow((0.05+(i*0.05)),(Math.ceil(Integer.parseInt(d.charAt(j)+"")/3)));
            }
        }
        int reliable[]=new int[1024];
    // Calling generateGraph method to get Total reliability for [0.05,1]
        for(int i=0;i<=20;i++){
            Project2 proj=new Project2();
            System.out.println("For p= "+String.format("%.2f",(0.05*i))+" reliability
="+proj.generateGraphs(p[19-i],10,0));
        }

    // Calling generateGraph method to get Total reliability for p=0.9
        for(int factor=0;factor<=20;factor++){
            System.out.print("For p= 0.9 and k = "+factor+" reliability =");
            double sum=0;
            for(int rot=0;rot<10;rot++)
            {
                Project2 proj=new Project2();
                sum=sum+proj.generateGraphs(p[1],10,factor);
            }
            System.out.println(sum/10);

        }
```

```
   }
}
```

Read Me:

Save the program that is provided in the above appendix as Project2.java

Compile and run the above program and generate the required results.

References:

1. http://evolution.gs.washington.edu/sisg/2014/2014_SISG_12_2.pdf
2. https://pdfs.semanticscholar.org/6503/970408760fb822afef82fe8c87ecce00f46e.pdf