ALGORITHMIC ASPECTS OF TELECOMMUNICATION

NETWORKS

PROJECT – 2

“Dependence of Network Reliability on the reliability of Individual links “

**Submitted By –**

Navaneeth Venugopala Rao

2021223814

[nbv140130@utdallas.edu](mailto:nbv140130@utdallas.edu)

1. AbstractCONTENTS

2. Pseudo Code

3. Flow Chart

4. Exhaustive Enumeration Algorithm

5. Graphical Observation and Explanation

6. Output

7. Source Code

8. References

ABSTRACT

 This project is an experimental study to test reliability of a network based on individual link reliability. We have to develop an algorithm to find out the overall network reliability. In this case network reliability only depends on individual link reliability. Each of the links in the network may fail but nodes are always up. The system will be operational if the network topology is connected. We will develop our algorithm to find the overall network by using Exhaustive Enumeration method.

 Create an algorithm to compute the network reliability in the above described situation, using the method of exhaustive enumeration.

 To plot a graph between the obtained networks reliability with the corresponding increase of p in steps of 0.02.

 Flip the corresponding system condition. That is, if the system was up, change it to down, if it was down, change it to up

 To find the reliability of the system when the p value is kept fixed at 0.96 and for different values of k. This also needs to be plotted on a graph.

PSEUDO-CODE

**Reliability\_Network** (G,p)

**Input**: Complete Graph, G Link reliability, p

**Output**: Network Reliability, R Initialization;

State <- Generate all state of the graph, G; R <- 0;

**For** each statei ϵ State **do**

**if** G is connected **then**

Find reliability r of statei; Increase R by r;

**return** R;

**end**

**end**

**end** procedure Reliability\_Network

FLOWCHART

Start

Create network topology with n(number of nods) = 5 and m(number of links) = 10

If flag-flip

== 0

Find all the links i.e. Failed or Working

Find all the links i.e. Failed or Working

While incrementing the value of k in steps of 0.02, calculate the reliability of each component.

While keeping the value of p constant and varying k from

0 – 50, calculate the reliability of each of the component.

Plot a graph between network reliability and p

Plot a graph between network reliability and different values of k

Stop Stop

Exhaustive\_enumeration Algorithm

1. Network is generated.

2. Determine whether a link is UP or DOWN.

3. IF path exists between two nodes i.e. Source and Destination

Link is UP ELSE

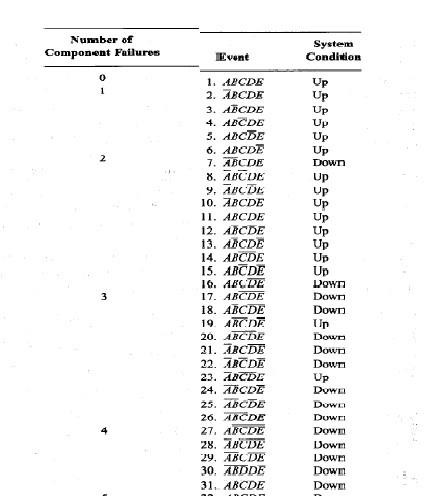
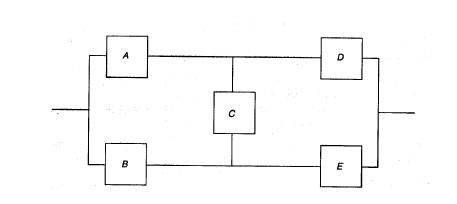
Link is DOWN

4. All combinations of state are generated (thus we would have 2^m-1 combinations for m links).

5. The reliability of the network is computed as the sum of these reliabilities whose states are considered to be UP.

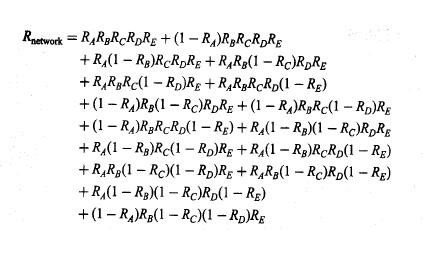
Illustration of Exhaustive Enumeration:

For a sample network such as the one shown below, the combination of states is enumerated exhaustively as shown the table.



The reliability of the above could be computed by summing all the reliabilities of

the “up states” combinations as follows:



 The states are chosen randomly by Math.Random class in all the possible combinations.

GRAPHS

1.2

Y axis - Network Reliability

X axis - Probability

1

0.8

0.6

0.4

0.2

0

0

0.04

0.08

0.12

0.16

0.2

0.24

0.28

0.32

0.36

0.4

0.44

0.48

0.52

0.56

0.6

0.64

0.68

0.72

0.76

0.8

0.84

0.88

0.92

0.96

1

Explanation –

The above graph shows as expected. X axis in the graph is Probability (p) i.e. reliability of each link. Now as the reliability of each link increases, the overall reliability of network should also increase theoretically, which is exactly what we see as a result in the above graph. When the value of p = 1, then value of Reliability is also = 1 which means when all the links are working then the overall reliability of the network will also be 1.

1.4

Reliability

1.2

1

0.8

0.6

0.4

0.2

0

0 20 40 60 80 100 120

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

The above graph is run from 0 – 100 to get a better understanding of reliability

Explanation –

For the second question 'k' is given a value between 0 and 50. 'k' denotes the links whose states have to be flipped. If the link state is 'down' then that link will be included in computing 'R'. A link will not be included in the computation for 'R' if the state is 'up'. As the value 'p' increases the value '1-p' decreases. If a link is 'up' we use 'p' and if the link is 'down' we use '1-p' in calculating 'R'. The flipping of states decreases the value of 'R' as 'p' increases.

Value of p is incremented with steps of 0.02

P Reliability

0.0 0.0

0.02 2.281387158886812E-4

0.04 0.0017325668028016071

0.06 0.005543576019831763

0.08 0.012441396393761756

0.1 0.02297787039999961

0.12 0.03749924944814784

0.14 0.05616956460508702

0.16 0.07899409532902762

0.18 0.10584252692356087

0.2 0.13647144959999974

0.22 0.17054590960780033

0.24 0.207659776005416

0.26 0.24735473544584144

0.28 0.28913777199149276

0.3 0.3324970295999962

0.32 0.37691599168519113

0.34 0.421885945204035

0.36 0.4669167261998558

0.38 0.5115457697928991

0.4 0.5553455104000035

0.42 0.597929197634731

0.44 0.6389552100359955

0.46 0.6781299626458639

0.48 0.7152095156544086

0.5 0.75

0.52 0.782356982105596

0.54 0.8121838939942234

0.56 0.8394296570088449

0.58 0.8640856274111806

0.6 0.8861819903999983

0.62 0.9057837257204939

0.64 0.922986263180525

0.66 0.9379109401968949

0.68 0.9507003661121796

0.7 0.9615137895999958

0.72 0.9705225561623312

0.74 0.9779057326646363

0.76 0.9838459652022361

0.78 0.988525625493533

0.8 0.9921232896000003

0.82 0.9948105812289852

0.84 0.996749400331296

0.86 0.9980895463101191

0.88 0.9989667340596922

0.9 0.9995009904000017

0.92 0.999795408415973

0.94 0.9999352278952418

0.96 0.9999872016358202

0.98 0.999999200012795

1.0 1.0

\*\*\*\*\*\*\*\* Constant Value of p \*\*\*\*\*\*\*\*\*

P is fixed at 0.96 and k is incremented from 0 – 50 with steps of 1

K Reliability

0 0.9999872016083609

1 0.9999872016083609

2 0.9999892019933991

3 1.0000352943168318

4 0.9999832008382843

5 0.9999872885816741

6 1.0011896889978986

7 1.00123586481571

8 1.0010955944182425

9 0.9988312340160861

10 0.9977311035213416

11 0.9988393155716838

12 0.999947451085516

13 1.0012018681538666

14 0.9988393257185757

15 0.9999957175956578

16 1.0276911689105475

17 1.0253326272453331

18 1.0228398163377663

19 1.0229820105837621

20 1.0229820975570754

21 1.026406443281388

22 1.056358159527098

23 1.0577468496066469

24 1.0045982922719323

25 1.0044940183415239

26 1.034694203148677

27 1.0081950674555624

28 1.0092710370946312

29 0.978353539611973

30 0.9795179401320329

31 0.9793779168837233

32 0.9781174844546087

33 0.94940624926151

34 0.9480557334596192

35 0.9758212054718025

36 0.9456209302124016

37 0.9491877067501089

38 0.9500454602907734

39 1.001875243413553

40 0.9745066126445281

41 0.33867599709909346

42 0.36431746216609984

43 0.39329530474759544

44 0.3644294482785967

45 0.33922907265518626

46 0.34134881414150126

47 0.34255306534867824

48 0.3414068564462387

49 1.0042154468313234

50 1.033834522135594

SOURCE CODE

**public class** Reliability\_Network {

**private static double** *sum* = 0;

**private static double** *Row\_Probabaility* = -1;

**private static int** *flip\_flag* = 0;

**private static final int *Count\_100*** = 100;

**private static final int *Count\_1000*** = 1000;

**private static final int *Count\_1024*** = 1024; // total possible combinations of the

components

**private static final int *Count\_10*** = 10;

**private static int**[] *random\_Generator* = **new int**[***Count\_100***];

**private static int**[] *array* = **new int**[***Count\_1000***];

**private static int**[][] *edge\_Matrix* = **new int**[***Count\_1024***][***Count\_10***];

**private static int**[][] *Matrix\_One* = **new int**[***Count\_1024***][***Count\_10***];

/\*

\* This method will be used to assign the edge or the connection between the two

\* nodes as up or down

\*/

**public static int**[][] SetLinks(**int**[][] link\_Mat) {

**for** (**int** i = 0; i < ***Count\_1024***; i++) {

String currentString = String.*format*("%010d", Integer.*parseInt*(Integer.*toBinaryString*(i)));

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

{

**if**(currentString.charAt(j) == '0')

{

}

**else**

{

}

}

}

link\_Mat[i][j] = 0;

link\_Mat[i][j] = 1;

**return** link\_Mat;

}

/\*

\* This function will round off upto n places

\*/

**public static double** round(**double** value, **int** places) {

**if** (places < 0)

{

**throw new** IllegalArgumentException();

}

**long** factor = (**long**) Math.*pow*(10, places);

value = value \* factor;

**long** tmp = Math.*round*(value);

**return** (**double**) tmp / factor;

}

/\*

\* This method will find if the nodes are up or down

\*/

**public static int**[] UP\_DOWN\_Search(**int**[][] matrix) {

**int**[] temp = **new int**[***Count\_1024***];

**int** counter = 0;

**for** (**int** i = 0; i < ***Count\_1024***; i++)

{

temp[i] = 0;

}

**for** (**int** i = 0; i < ***Count\_1024***; i++)

{

**int**[] matrix\_chk = **new int**[5];

**for** (**int** k = 0; k < 5; k++)

{

matrix\_chk[k] = -1;

}

**if** (matrix[i][0] == 1)

{

matrix\_chk[0] = 0;

matrix\_chk[1] = 1;

}

**if** (matrix[i][1] == 1)

{

matrix\_chk[0] = 0;

matrix\_chk[2] = 2;

}

**if** (matrix[i][2] == 1)

{

matrix\_chk[0] = 0;

matrix\_chk[3] = 3;

}

**if** (matrix[i][3] == 1)

{

matrix\_chk[0] = 0;

matrix\_chk[4] = 4;

}

**if** (matrix[i][4] == 1)

{

matrix\_chk[1] = 1;

matrix\_chk[2] = 2;

}

**if** (matrix[i][5] == 1)

{

matrix\_chk[1] = 1;

matrix\_chk[3] = 3;

}

**if** (matrix[i][6] == 1)

{

matrix\_chk[1] = 1;

matrix\_chk[4] = 4;

}

**if** (matrix[i][7] == 1)

{

matrix\_chk[2] = 2;

matrix\_chk[3] = 3;

}

**if** (matrix[i][8] == 1)

{

matrix\_chk[2] = 2;

matrix\_chk[4] = 4;

}

**if** (matrix[i][9] == 1)

{

matrix\_chk[3] = 3;

matrix\_chk[4] = 4;

}

**int** count = 0;

**for** (**int** j = 0; j < 5; j++)

{

count = count + matrix\_chk[j];

}

**if** (count == ***Count\_10***)

{

temp[counter] = i;

counter++;

}

}

**return** temp;

}

/\*

\*

\*/

**public static void** main(String[] args) {

**int** k, i;

**for** (i = 0; i < ***Count\_100***; i++)

{

*random\_Generator*[i] = -1;

}

*sum* = 0;

**float** q = (**float**) 1.0;

**float** p = (**float**) 0.0;

**if** (*flip\_flag* == 0)

{

*edge\_Matrix* = *SetLinks*(*edge\_Matrix*);

}

*array* = *UP\_DOWN\_Search*(*edge\_Matrix*);

steps

/\*

\* This case will handle variable p value which will be incremented in

\* of 0.02

\*/

**if** (*flip\_flag* != 0)

{

p = (**float**) 0.9;

q = (**float**) 0.9;

}

System.***out***.println("P\t\tReliability");

**for** (**double** id = p; id <= q;)

{

**for** (**int** j = 0; j < ***Count\_1000***; j++) { **if** (*array*[j] != 0)

{

**for** (**int** ik = 0; ik < ***Count\_10***; ik++)

{

**if** (*edge\_Matrix*[*array*[j]][ik] == 1)

{

**if** (*Row\_Probabaility* == -1)

{

*Row\_Probabaility* = id;

}

}

**else**

{

**else**

{

}

*Row\_Probabaility* = *Row\_Probabaility* \* id;

**if** (*Row\_Probabaility* == -1)

{

}

**else**

{

}

}

}

*Row\_Probabaility* = (1 - id);

*Row\_Probabaility* = *Row\_Probabaility* \* (1- id);

*sum* = *sum* + *Row\_Probabaility*;

*Row\_Probabaility* = -1;

}

}

System.***out***.println(id + "\t\t" + *sum*);

id = *round*(id + 0.02, 2);

*sum* = 0;

}

*Matrix\_One* = *edge\_Matrix*;

/\*

\* This case will handle constant k value

\*/

*flip\_flag* = 1;

System.***out***.println("\n\*\*\*\*\*\*\*\* Constant Value of p \*\*\*\*\*\*\*\*\*\n");

System.***out***.println("K\t\tReliability");

**for** (k = 1; k <= ***Count\_100***; k++)

{

**for** (i = 0; i < k; i++)

*random\_Generator*[i] = (**int**) (Math.*random*() \* ***Count\_1024***);

**int** j = 0;

**for** (i = 0; i < ***Count\_100***; i++) ;

**for** (i = 0; i < ***Count\_100***; i++)

{

**if** (*random\_Generator*[i] != -1)

{

**for** (j = 0; j < ***Count\_10***; j++)

{

**if** (*edge\_Matrix*[*random\_Generator*[i]][j] == 0)

{

}

**else**

{

}

}

}

}

*edge\_Matrix*[*random\_Generator*[i]][j] = 1;

*edge\_Matrix*[*random\_Generator*[i]][j] = 0;

*sum* = 0;

q = (**float**) 1.0 \* 1;

p = (**float**) 0.0 \* 1;

**if** (*flip\_flag* == 0)

{

*edge\_Matrix* = *SetLinks*(*edge\_Matrix*);

}

*array* = *UP\_DOWN\_Search*(*edge\_Matrix*);

**if** (*flip\_flag* != 0)

{

p = (**float**) 0.9 \* 1;

q = (**float**) 0.9 \* 1;

}

**for** (**double** di = p; di <= q;)

{

**for** (j = 0; j < ***Count\_1000***; j++)

{

**if** (*array*[j] != 0)

{

**for** (**int** ki = 0; ki < ***Count\_10***; ki++)

{

**if** (*edge\_Matrix*[*array*[j]][ki] == 1)

{

**if** (*Row\_Probabaility* == -1)

{

\* di;

}

**else**

{

}

**else**

{

}

*Row\_Probabaility* = di;

*Row\_Probabaility* = *Row\_Probabaility*

**if** (*Row\_Probabaility* == -1)

{

\* (1 - di);

}

**else**

{

}

}

}

*Row\_Probabaility* = (1 - di);

*Row\_Probabaility* = *Row\_Probabaility*

*sum* = *sum* + *Row\_Probabaility*;

*Row\_Probabaility* = -1;

}

}

System.***out***.println(k + "\t\t" + *sum*);

di = *round*(di + 0.02, 2);

*sum* = 0;

}

*edge\_Matrix* = *Matrix\_One*;

**for** (i = 0; i < k; i++)

*random\_Generator*[i] = -1;

}

}

}

(Note – Please do not execute this code directly. Because of the formatting, we may encounter compilation errors. Use the code from Source Code folder to execute and test the project)

REFERENCES

 The formulae for computation of reliabilities was borrowed from material posted by Dr Andras Farago in his lecture notes

 Definition of terms was borrowed from [www.wikipedia.org](http://www.wikipedia.org)