**Algorithmic Aspects of Telecommunication Networks**

**Project 3 – Aparna Pavithran (axp161730)**

**Experimental study of how the network reliability depends on the individual link reliabilities**

**Index**

|  |  |  |
| --- | --- | --- |
| 1 | Objective | 3 |
| 2 | Network Topology | 3 |
| 3 | Link Reliability | 3 |
| 4 | System configuration | 3 |
| 5 | Algorithm | 4 |
| 6 | Pseudo Code | 4 |
| 7 | Flow Chart | 6 |
| 8 | Correctness of the Algorithm | 8 |
| 9 | How does Algorithm works and implementation | 9 |
| 10 | Experimental Evaluation – for different values of p | 11 |
| 11 | Experimental Evaluation – for different values of k | 12 |
| 12 | Graph – System reliability Vs p values | 13 |
| 13 | Graph – System reliability Vs k values | 13 |
| 14 | Observation and Conclusion | 14 |
| 15 | Appendix – Source Code | 14 |
| 16 | Readme | 20 |
| 17 | References | 20 |

1. **Objective**

Input: Number of nodes in the graph.

* Design an algorithm to evaluate the network reliability of a system of networks with 5 nodes.
* Nodes are always up and probability of links can be varying.
* Undirected graph with no self-loops and parallel edges.
* Reliability of the system is calculated by exhaustive enumeration
* Reliability of each state is taken only if the system is up that is system is connected.
* Also experiment by flipping the states of links in the network from up to down or down to up.

1. **Network Topology**

Undirected graph with 5 nodes and 10 edges as it does not contain self-loops and parallel edges. Every node is connected to every other node. Possible combinations of edges will be 2^10 = 1024. So 1024 adjacency matrix can come. Nodes are always up.

1. **Link Reliability**

Each link has a reliability of p. p varies from 0 to 1 in iteration 0.05. For each value of link reliability p calculate the reliability of entire system by calculating for all possible combinations. In the second part choose any random k combinations and flip the system state from up to down or down to up and calculate the reliability. The k varies from 0 to 20.

1. **System Configuration**

The program is implemented in Java in MAC OS and by using Eclipse IDE.

1. **Algorithm**
2. Consider all possible 1024 combinations of edges to get the network reliability. Ie 000000000 - 1111111111
3. For each combination of edges find out the system is connected or not. If the system is connected then find out the reliability.
4. Add all reliability combinations to get the network reliability.
5. Repeat these steps for different values of p vary from 0-1 in steps of 0.05.
6. Take random k combinations from 1024 combinations and flip the values. Repeat the steps 1-3 for a particular value of p=0.85.
7. Find out the network reliability. Repeat this for 10 times and take the average value.
8. Repeat the steps for different values of k vary from 0-20.
9. **Pseudo Code**

**Stage 1:**

1. nodes =5 //number of nodes
2. pLink = 0 //p value of each link
3. change = 0.05 // increment value of p
4. while pLink <= 1
5. number =0;
6. while number <1024
7. sysrel=0
8. calculate the binary value of number
9. append / padding 0s to match the value to 10 bits
10. change the binary string to an integer array of edges.
11. For the particular integer array of edges calculate the adjacency matrix
12. for(int p=0;p<nodes-1;p++)
13. for(int q=p+1;q<nodes;q++)
14. adjacency[p][q]=edges[i++]
15. adjacency[q][p]=adjacency[p][q]
16. end for
17. end for
18. if the graph is connected then
19. rel=1
20. for each value of edge if the value is 1 then rel=rel\*p else rel=rel\*(1-p)
21. return rel
22. add rel to the sysrel
23. number=number+1
24. end while
25. Print sysrel
26. pLink=pLink + change
27. end while

**Stage 2:**

1. nodes =5 //number of nodes
2. pLink = 0.85 //p value of each link
3. save sysrel\_p system reliability from the stage 1 for the p value 0.85
4. k = 0
5. while k <= 20
6. sysrel=sysrel\_p
7. pick a random number between 0 and 1023
8. calculate the binary value of number
9. append / padding 0s to match the value to 10 bits
10. change the binary string to an integer array of edges.
11. For the particular integer array of edges calculate the adjacency matrix
12. for(int p=0;p<nodes-1;p++)
13. for(int q=p+1;q<nodes;q++)
14. adjacency[p][q]=edges[i++]
15. adjacency[q][p]=adjacency[p][q]
16. end for
17. end for
18. if the graph is connected then
19. rel=1
20. for each value of edge if the value is 1 then rel=rel\*p else rel=rel\*(1-p)
21. return rel
22. sysrel= sysrel - rel
23. Reverse the values of edges of array ie flip the values and repeat steps 10 to 20 to get new adjacency matrix and reliability.
24. sysrel= sysrel + rel
25. Repeat steps 6 to 23 10 times and add the reliability values
26. Take average of reliability values and Print it.
27. k=k + 1
28. end while
29. **Flow Chart**

**Stage 1**

relCalc

altReliability

adjCalc

edgeCalc

Input //number of nodes

Main function

yes

bfs

**Stage 2 -**

relCalc

netkReliability

adjCalc

edgeCalc

Input //number of nodes

Main function

bfs

yes

revEdgeCalc

adjCalc

bfs

relCalc

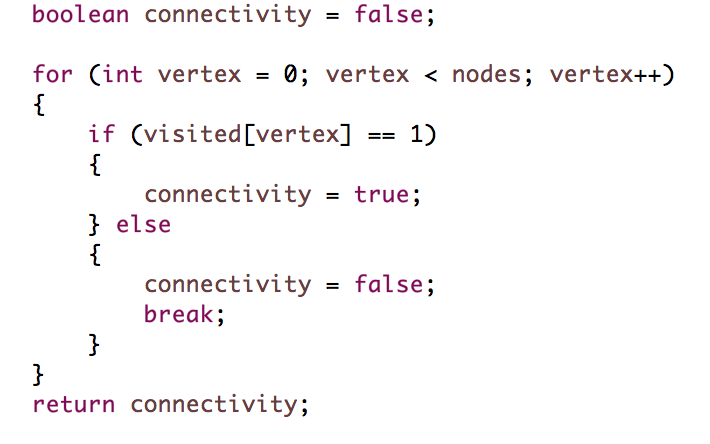
yes

1. **Correctness of the Algorithm**

As this is an exhaustive enumeration, each type of combination is considered. Ie the algorithm runs from 0-1023 and calculates the reliability for each network. Then adds up the reliability to get the system reliability.

Each time before calculating the reliability, algorithm checks whether the network is connected or not. If the network is connected then the reliability is calculated and adds up with the system reliability But if the network is disconnected, ie any node is not reachable then the reliability is taken as 0 for the particular combination.

Below figure shows that if each node is visited once in the bfs traversal then it is returned as true. Else it is returned as false. That means if each node is not visited then there is one node, which is not reachable, and so the system is unconnected. In that case reliability is not added up to the system reliability.



1. **How does Algorithm works and implementation**

**Driver.java**

**Main** **function**:-

Enter the number of nodes. Here in our case its 5. Since undirected graph with out self-loops and parallel edges, number of edges are calculated as node\*(node-1)/2.

Set p value of link as 0. And change as 0.05. For each value of p, call netkReliability function to calculate the reliability of network. Print the result. Increment p value by change (0.05) and repeat this step till p reaches 1.

Fix p value as 0.85. Iterate k value from 0-20. Call altReliability function to calculate the network reliability by flipping the values from up to down or down to up. Repeat this step 10 times for the same value of k and take the average value of network reliability. Print the result. Repeat the same steps till k reaches 20.

**Reliability.java**

**netkReliability** **function** :-

Input is number of nodes. Output/return value is network reliability. Starts with a number value as 0 and repeat until 1023. Calculate the binary value of the number. Call the function edgeCalc to store the binary value in an array of edges. Call the function adjCalc to calculate the adjacency matrix for the particular edge values. Call the bfs function to check if the system is connected or not. If the system is connected, call relCalc function to calculate the reliability. Repeat this for all 1024 combinations and add all the reliabilities to calculate the network reliability of the system.

**edgeCalc function** :-

Take input as number of edges and the binary number to convert to edges of array. Output is an array of edges with values 0 or 1 depending on the edge exists or not. The number is converted to binary value and appended 0s in left to pad the length as 10 bits. The binary number with 10 bits is converted to a n array of 0s and 1s and returned as edge array.

**adjCalc function** :-

Input is the number of nodes and the array of edges. Output is adjacency matrix of the particular combination. For each value of 0/1 of edge, 2 values are put in adjacency matrix, as this is an undirected graph. The adjacency matrix is returned back as the output.

**relCalc function** :-

Input is the array of edges for the particular combination and value of p. Calculate the reliability by multiplying by p if edge value is 1 and (1-p) if edge value is 0. Return the reliability calculated for the specific combination of edges.

**altReliability** **function** :-

Inputs are reliability value of original system with a specific p value, number of nodes, p value, k value. Choose random numbers between 0-1023. Repeat this for k times. For each random number generated, call edgeCalc function and find out the array of edges associated with the particular random number. Calculate the adjacency matrix by calling adjCalc function. If the graph is connected (by calling bfs function) then calculate the reliability value (by calling relCalc function) for the particular number edge combination. Store the value of reliability in a temp variable. Now from the old reliability value minus this temp value.

Now reverse the edge values by putting 1 if the value was 0 and vice versa. This is done by calling the function revEdgeCalc. This function returns the new edge array by reversing the situations. For the new edge array calculate the new adjacency matrix. If the graph is connected then calculate the reliability value for the new adjacency matrix graph. Add this value to the original reliability.

Repeat this steps for k times. Return the new reliability for the k flipped system.

**revEdgeCalc function** :-

Takes input as array of edges (integer array) and return output as array of edges (integer array). If the value in the array is 1 then change it to 0 and if the value in the array is 0 then change it to 1. Return the resultant array of edges.

**IsConnected.java**

**bfs function** :-

Takes adjacency matrix, number of nodes, and source vertex/node as input. Returns a Boolean value true or false depending on the graph is connected or not. Graph is connected if all the nodes are reachable from the source node.

1. **Experimental Evaluation – for different values of p**

|  |  |
| --- | --- |
| **p value** | **Reliability** |
| 0 | 0 |
| 0.05 | 6.31E-04 |
| 0.1 | 0.008097533 |
| 0.15 | 0.032599038 |
| 0.2 | 0.081945497 |
| 0.25 | 0.155698956 |
| 0.3 | 0.227214813 |
| 0.35 | 0.370050931 |
| 0.4 | 0.489653862 |
| 0.45 | 0.658790414 |
| 0.5 | 0.7109375 |
| 0.55 | 0.75 |
| 0.6 | 0.826130914 |
| 0.65 | 0.870256742 |
| 0.7 | 0.930724865 |
| 0.75 | 0.957513038 |
| 0.8 | 0.96151379 |
| 0.85 | 0.991664538 |
| 0.9 | 0.99950099 |
| 0.95 | 0.999968758 |
| 1 | 1 |

1. **Experimental Evaluation – for different values of k**

|  |  |
| --- | --- |
| **k value** | **Reliability** |
| 0 | 0.991664538 |
| 1 | 0.995500908 |
| 2 | 0.993312806 |
| 3 | 0.991354623 |
| 4 | 0.993622432 |
| 5 | 0.988111928 |
| 6 | 0.990471976 |
| 7 | 0.953554404 |
| 8 | 0.989354781 |
| 9 | 0.958419372 |
| 10 | 0.947409969 |
| 11 | 0.95083765 |
| 12 | 0.939828247 |
| 13 | 0.856659672 |
| 14 | 0.834352781 |
| 15 | 0.835401908 |
| 16 | 0.83121032 |
| 17 | 0.623142548 |
| 18 | 0.618277581 |
| 19 | 0.597958571 |
| 20 | 0.59903947 |

1. **Graph – Network reliability Vs p values**
2. **Graph – Network reliability Vs k values**
3. **Observation and Conclusion**

* As the p value increases network reliability increases slowly and reaches 1 when p value reaches 1.
* Reliability is always between 0 and 1 as it is the probability.
* For small values of p network reliability increases very slowly.
* K value and network reliability is varying as edges are randomly selected.
* For small values of k network reliability do not change much also sometimes it increases or decreases randomly.
* But for larger values of k consider 0 and 20, in that case network reliability reduced.

1. **Appendix – Source Code**

**Driver.java**

**import** java.util.Scanner;

**public** **class** Driver {

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

System.***out***.println("Enter the number of nodes:");

Scanner in=**new** Scanner(System.***in***);

**int** nodes=in.nextInt();

//System.out.println("Enter the link probability:");

**double** linkP=0.0;

//System.out.println("Enter the k value:");

//int k=in.nextInt();

**double** rel;

**double** change=0.05;

**while**(linkP<=1){

rel=Reliability.*netkReliability*(nodes,linkP);

System.***out***.println("Reliability : "+rel+" for p value : "+linkP);

linkP+=change;

}

linkP=1;

rel=Reliability.*netkReliability*(nodes,linkP);

System.***out***.println("Reliability : "+rel+" for p value : "+linkP);

linkP=0.85;

**int** k=0;

**double** rel\_p=Reliability.*netkReliability*(nodes,linkP);

System.***out***.println("Reliability without k: "+rel\_p+" for p value : "+linkP);

System.***out***.println("Reliabilities with k for p value : "+linkP);

**int** ran=10;

**while**(k<=20){

**double**[] temp=**new** **double**[ran];

rel=0;

**for**(**int** i=0;i<ran;i++){

temp[i]=Reliability.*altReliability*(rel\_p,nodes,linkP,k);

rel+=temp[i];

}

rel=rel/ran;

System.***out***.println("Reliability : "+rel+" for k value : "+k);

k++;

}

in.close();

}

}

**IsConnected.java**

**import java.util.LinkedList;**

**import java.util.Queue;**

**public class IsConnected {**

**private static Queue<Integer> q;**

**public static boolean bfs(int adjacency\_matrix[][], int source,int nodes)**

**{**

**int[] visited = new int[nodes];**

**int i, element;**

**q = new LinkedList<Integer>();**

**visited[source] = 1;**

**q.add(source);**

**while (!q.isEmpty())**

**{**

**element = q.remove(); //first element is removed**

**i = element; //saved and checks for all others connected with the node.**

**while (i < nodes)**

**{**

**if (adjacency\_matrix[element][i] == 1 && visited[i] == 0)**

**{**

**q.add(i);**

**visited[i] = 1; //if it is visited then its marked as 1**

**}**

**i++;**

**}**

**}**

**boolean connectivity = false;**

**for (int vertex = 0; vertex < nodes; vertex++)**

**{**

**if (visited[vertex] == 1)**

**{**

**connectivity = true;**

**} else**

**{**

**connectivity = false;**

**break;**

**}**

**}**

**return connectivity;**

**}**

**}**

**Reliability.java**

**import** java.util.Random;

**public** **class** Reliability {

**public** **static** **int**[] revEdgeCalc(**int**[] edges){

**for**(**int** i=0;i<edges.length;i++){

**if**(edges[i]==0){

edges[i]=1;

}

**else**{

edges[i]=0;

}

}

**return** edges;

}

**public** **static** **double** altReliability(**double** rel,**int** nodes,**double** linkP,**int** k){

**int** edge=(nodes\*(nodes-1))/2;

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

Random r=**new** Random();

**int** low=0;

**int** high=(**int**) Math.*pow*(2,edge)-1;

**int** ran=r.nextInt(high-low)+low;

**int**[] edges=*edgeCalc*(edge,ran);

**int**[][] adjacency=*adjCalc*(nodes,edges);

**double** temp=0;

**if**(IsConnected.*bfs*(adjacency,0,nodes)){

temp=*relCalc*(edges,linkP);

//temp=relCalc(adjacency,linkP,nodes);

}

rel-=temp;

edges=*revEdgeCalc*(edges);

adjacency=*adjCalc*(nodes,edges);

**if**(IsConnected.*bfs*(adjacency,0,nodes)){

temp=*relCalc*(edges,linkP);

//temp=relCalc(adjacency,linkP,nodes);

}

rel+=temp;

}

**return** rel;

}

**public** **static** **double** netkReliability(**int** nodes, **double** linkP){

**double** netrel=0;

**int** edge=(nodes\*(nodes-1))/2;//undirected graph

**int** bin=(**int**) Math.*pow*(2,edge);

**int** i=0;

**for**(**int** number=0;number<bin;number++){

**int**[] edges=*edgeCalc*(edge,number);

**int**[][] adjacency=*adjCalc*(nodes,edges);

**if**(IsConnected.*bfs*(adjacency,0,nodes)){

netrel+=*relCalc*(edges,linkP);

//netrel+=relCalc(adjacency,linkP,nodes);

}

}

//int rel=2;

**return** netrel;

}

**public** **static** **double** relCalc(**int**[] edges, **double** linkP){

**double** p=1;

**for**(**int** i=0;i<edges.length;i++){

**if**(edges[i]==1){

p=p\*linkP;

}

**else**{

**double** t=1-linkP;

p=p\*t;

}

}

**return** p;

}

**public** **static** **int**[] edgeCalc(**int** edge,**int** number){

**int**[] edges=**new** **int**[edge];

**int** bin=(**int**) Math.*pow*(2,edge);

**int** i=0;

//int number=101;

//for(int number=0;number<bin;number++){

String binaryString = Integer.*toBinaryString*(number);

//System.out.println("binary "+binaryString);

//System.out.println(binaryString.length());

**int** j=binaryString.length()-1;

**for**(i=edge-1;i>=0 && j>=0;i--){

**if**(binaryString.charAt(j--)=='1')

edges[i]=1;

**else**

edges[i]=0;

//System.out.println(edges[i]+" "+i);

}

**while**(i>=0){

edges[i]=0;

//System.out.println(edges[i]+" "+i);

i--;

}

//}

**return** edges;

}

**public** **static** **int**[][] adjCalc(**int** nodes,**int**[] edges){

**int** i=0;

**int**[][] adjacency=**new** **int**[nodes][nodes];

**for**(**int** p=0;p<nodes;p++){

**for**(**int** q=0;q<nodes;q++){

adjacency[p][q]=0;

}

}

**for**(**int** p=0;p<nodes-1;p++){

**for**(**int** q=p+1;q<nodes;q++){

adjacency[p][q]=edges[i++];

adjacency[q][p]=adjacency[p][q];

}

}

/\*for(int p=0;p<nodes;p++){

for(int q=0;q<nodes;q++){

System.out.print(adjacency[p][q]+" ");

}

System.out.println();

}\*/

**return** adjacency;

}

**public** **static** **int** permutation(**int** n){

**int** perm=1;

**return** perm;

}

}

1. **Readme**

**Step 1:** Keep the programs in different files as Driver.java, Reliability.java and IsConnected.java

**Step 2:** Compile all the files. Javac \*.java

**Step 3:** Run the driver.java file.

**Step 4:** It will prompt for inputs. Give number of nodes as input.

**Step 5:** Program will output reliability for p values varying from 0 to 1 and reliability for p value = 0.85 and k values varying from 0 to 20.

1. **References**

* Lecture notes Professor Andras Farago