CS6385.0W1 - Algorithmic Aspects of Telecommunication Networks

Project – 2 Report

Submitted by

Chandan Raju Vysyaraju (cxv200012)

CONTENTS

INTRODUCTION

Telecommunications plays a vital role in every aspect of life. It is the means of exchange of information from one individual to another. Due to rapid development in the field of telecommunications the means of obtaining information have become much easier. Not only the transmission of information but the reliability and the availability of information is very important. One of the ways in which the transmission is processed is by networks. The reliability of a network is dependent on the length of time that the infrastructure can operate without disruption. Reliability can be calculated using different methods but one of those is calculating the mean time between failures, that is the operating time of the network in between the outage. It is calculated by diving the total operating time with number of failures. Another way is to identify the failure rate which gives the average time between the failures. The failure rate is calculated by dividing the total failures by total operating time. And the reliability is calculated by subtracting the failure rate from 100%.

ALGORITHM

In this project the network reliability is computed using exhaustive enumeration method. Here, all the possible configurations of both working and failed nodes should be considered. This indicates all the possible binary states of the nodes, where 1 represents a working node and 0 represents a failed node. For a graph with n vertices there are 2^n possible ways to connect a network. So, for a network with 3 vertices there are 8 possible ways to generate the binary states.

Algorithm:

1. Take node up probability and network topology graph as input to function.
2. Calculate the total number of nodes in the graph and compute the total number of states.
3. For each state in 2^n states iterate the below steps 4 to 7.
4. Assign the up/down condition to node based on the binary value.
5. Check if the node is connected or not using the depth first search method.
6. If the state of the node is up, then multiply the reliability value with its node up probability.
7. If the node state is down, then multiply the reliability value with 1 – node up probability that is (relValue * 1-p).
8. Now Compute the total network reliability by summing up all the reliability values of binary states of all nodes.

Input:

1. p : value of probability that a node is up
2. ntkGraph : a graph with n nodes in the form of matrix.

Output:

1. The total reliability value of the network.

IMPLEMENTATION

In this project, the code is implemented using Python programming by constructing the function to calculate the value of network reliability. The function to calculate the network reliability takes probability that node up and graph.

Pseudo code:

1. n = total number of nodes in the graph
2. Initialize total reliability to 0.
3. For i in range(2^n):
    a. Current node state = generate the state from binary state i
    b. Assign Node reliability to 1
    c. For j in range(n):
        i. If the node state is 1 then multiply node reliability with p

```
if current_state[j] == 1:
        connProb *= p
```

        ii. Else multiply node reliability with (1-p)

```
else:
        connProb *= (1-p)
```

    d. If the network is connected in the current state, then take summation of all the nodes reliability and assign it to total reliability.
4. Finally, return the value of total reliability.

The isNetworkConnected function uses the depth first search method and checks whether the nodes in the entire network are connected or not. For each node the function calls getNeighbors function to find the adjacent nodes to the current node.

Pseudo code for isNetworkConnected() function:

1. First get the indices of the nodes that are up from the node state.
2. If there are no nodes that are up, then the network is not connected.
3. Now, Initialize the set of checked nodes to 0 and stack with the first node.

```
Checked = set()
stack = [vrtx[0]]
```

4. Iterate until stack is empty:
    a. nd = stack.pop() - pop the top node from stack
    b. if the node is not checked:
        i. checked.add(nd) – add the node to checked as it is visited now.
        ii. neighbors = get_neighbors(nd) – get all the neighbors of the current node.
        iii. For each neighbor check if the state of the neighbor node is 1 and add it to the stack.
5. Return “True” if the length of visited nodes is equal to all the nodes in network.

The generate states function is used to assign the up/down condition to the nodes based on the binary state that is computed with the length of 2^n.

Pseudo code for genStates function:

1. Initialize the binary string and convert the index of node to binary using bin function
2. Remove 0b prefix using string slicing.
   Binary_str = bin(i)[2:]
3. Now left pad the string with zeros until the string is of length n that is number of nodes.
   Padded_str = binary_str.zfill(n)
4. Convert the padded binary string into a numpy array of integers using int function.
   States = [int(x) for x in padded_str]
5. Finally, return the state of the node as the output of the function.

README SECTION

To run this program, the below steps need to be followed:

1. Install Python version 3.x or later on your computer, if it is not installed.
2. Create a new file such as networkReliability.py in a directory on your computer.
3. Copy the code in the Appendix section and save it to the networkReliability.py file.
4. Open command prompt and navigate to the directory where the file is saved.
5. Type the following command to run the program

   'Python networkReliability.py'

6. The program will compute the reliability of the network for different values of node probabilities and plot the results.
7. The output of the program will be displayed in a graph.

Note: I have named the file as networkReliability.py, you can name the file with different name but make sure to use the correct file name while executing the code. Also make sure that all the necessary packages such as NumPy, matplotlib are installed in the python environment. If any package is not installed, then they can be installed using following commands:

- For NumPy package: 'pip install numpy'
- For matplotlib package: 'pip install matplotlib'

The networkReliability.py python file consists of four functions. The 'calcNetworkReliability' function computes the reliability of total network for the given node probability p. The 'genStates' function is used to generate the state of each node in the network. The 'isNetworkConnected' function checks if the entire network is connected or not. The 'getNeighbors' is used to find all the neighbor nodes to the current node. The main program uses these functions to compute the network reliability for different values of node up probability p and plot the results. The program can be modified to use a different network topology or a different set of node probabilities.

RESULTS

5.1 Output:

For this project a network topology is given, and the graph is taken as input in the form of a matrix. The input graph is taken as:

```
ntkGraph = [[0, 1, 1, 0, 1, 0, 1, 0],
            [1, 0, 1, 0, 0, 1, 0, 0],
            [1, 1, 0, 1, 1, 0, 0, 0],
            [0, 0, 1, 0, 0, 1, 1, 0],
            [1, 0, 1, 0, 0, 1, 0, 1],
            [0, 1, 0, 1, 1, 0, 1, 1],
            [1, 0, 0, 1, 0, 1, 0, 1],
            [0, 0, 0, 0, 1, 1, 1, 0]]
```

As given in the project description the values of p are taken from 0.05 to 1 in steps of 0.05.

```
pValues = [0.05 + i * 0.05 for i in range(0, 20)]
```

For the given inputs when the code is executed the following output is obtained:

```
For Probability p = 0.05, Value of reliability = 0.3104
For Probability p = 0.10, Value of reliability = 0.4859
For Probability p = 0.15, Value of reliability = 0.5786
For Probability p = 0.20, Value of reliability = 0.6253
For Probability p = 0.25, Value of reliability = 0.6502
For Probability p = 0.30, Value of reliability = 0.6687
For Probability p = 0.35, Value of reliability = 0.6891
For Probability p = 0.40, Value of reliability = 0.7153
For Probability p = 0.45, Value of reliability = 0.7478
For Probability p = 0.50, Value of reliability = 0.7852
For Probability p = 0.55, Value of reliability = 0.8250
For Probability p = 0.60, Value of reliability = 0.8646
For Probability p = 0.65, Value of reliability = 0.9014
For Probability p = 0.70, Value of reliability = 0.9333
For Probability p = 0.75, Value of reliability = 0.9591
```

For Probability p = 0.80, Value of reliability = 0.9780

For Probability p = 0.85, Value of reliability = 0.9904

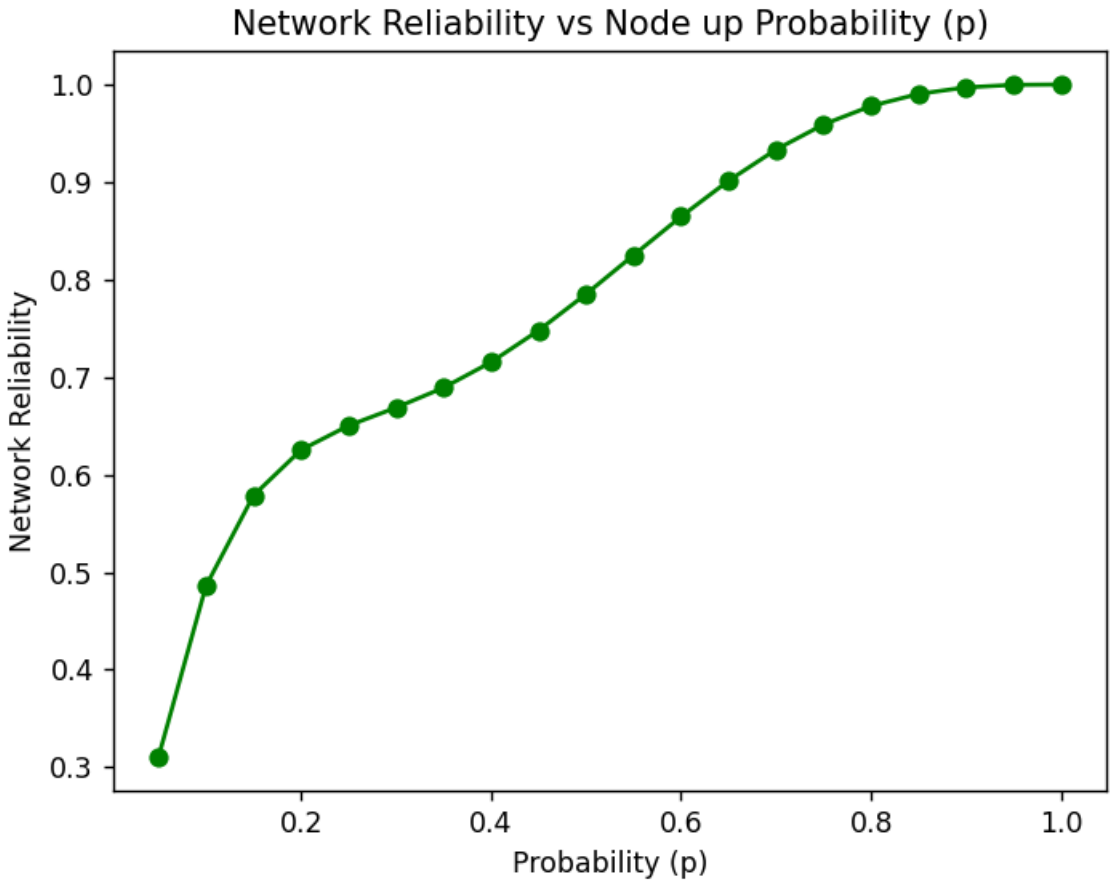
For Probability p = 0.90, Value of reliability = 0.9971

For Probability p = 0.95, Value of reliability = 0.9996

For Probability p = 1.00, Value of reliability = 1.0000

5.3 Graphical Representation:

For all the node up probabilities from 0.05 to 1 in steps of 0.05 and their corresponding values of network reliability the below graph is plotted and observed.



In the above graph it is observed that as the probability of the node working increases the reliability of network is also increased. In the above graph, the reliability is increased from 31.04% at p = 0.05 to 100% at p = 1.0. This indicates that the network becomes highly reliable as the probability of each node being functional is near to 1. For the values of p from 0.05 to 0.25 there is a significant increase in reliability and for the higher values of p from 0.25 to 1 the increase in reliability becomes more gradual. Also from p = 0.75 the reliability is closer to 1, so it is recommended to make sure that the probability of each node is above 0.75 for a highly reliable network.

APPENDIX

```
import numpy as ny
import matplotlib.pyplot as fig

def calcNetworkReliability(p, ntkGraph):

    n = len(ntkGraph)
    totReliability = 0.0
    for i in range(2**n):
        current_state = genStates(n, i)
        connProb = 1.0
        for j in range(n):
            if current_state[j] == 1:
                connProb *= p
            else:
                connProb *= (1-p)
        if isNetworkConnected(current_state, ntkGraph):
            totReliability += connProb

    return totReliability

def genStates(n, i):
    states = ny.array([int(x) for x in bin(i)[2:].zfill(n)])
    return states

def isNetworkConnected(state, ntkGraph):
    vrtx = ny.nonzero(state)[0]
```

```
    if len(vrtx) == 0:
        return False
    checked = set()
    stack = [vrtx[0]]
    while stack:
        nd = stack.pop()
        if nd not in checked:
            checked.add(nd)
            nghResult = getNeighbors(nd)
            for neighbor in nghResult:
                if state[neighbor] == 1:
                    stack.append(neighbor)
    return len(checked) == len(vrtx)

def getNeighbors(i):
    nbr = []
    for j in range(len(ntkGraph)):
        if ntkGraph[i][j] == 1:
            nbr.append(j)
    return nbr



if __name__ == "__main__":

    ntkGraph = [[0, 1, 1, 0, 1, 0, 1, 0],
                [1, 0, 1, 0, 0, 1, 0, 0],
                [1, 1, 0, 1, 1, 0, 0, 0],
```

```
        [0, 0, 1, 0, 0, 1, 1, 0],
        [1, 0, 1, 0, 0, 1, 0, 1],
        [0, 1, 0, 1, 1, 0, 1, 1],
        [1, 0, 0, 1, 0, 1, 0, 1],
        [0, 0, 0, 0, 1, 1, 1, 0]]


relValues = []
pValues = [0.05 + i * 0.05 for i in range(0, 20)]
for p in pValues:
    rel = calcNetworkReliability(p, ntkGraph)
    relValues.append(rel)

for p, r in zip(pValues, relValues):
    print(f"For Probability p = {p:.2f}, Value of reliability = {r:.4f}")

fig.plot(pValues, relValues, marker = 'o', color = 'g')
fig.xlabel('Probability (p)')
fig.ylabel('Network Reliability')
fig.title('Network Reliability vs Node up Probability (p)')
fig.show()
```

REFERENCES

[1] https://en.wikipedia.org/wiki/Reliability_(computer_networking)

[2] https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.81

[3] https://www.pslightwave.com/how-to-measure-network-reliability/#:~:text=Methods%20to%20Measure%20Network%20Reliability&text=You%20can%20calculate%20MTBF%20or,service%20does%20not%20include%20downtime.

[4] https://nrelabs.io/resources/introduction/

[5] Lecture Notes from e-Learning