CS 6385
ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

PROJECT - 2
Network Reliability using Exhaustive Enumeration

SUBMITTED BY -
YASHWANTH DEVIREDDY
NetID : yxd210008

CONTENTS

1. ABSTRACT
2. ALGORITHM
3. FLOW CHART
4. PSEUDO CODE
5. OUTPUT AND RESULTS
6. APPENDIX
7. README
8. REFERENCES

ABSTRACT

This project investigates the dependence of network reliability on individual component reliabilities in the given network topology. The network is a complete graph on 5 nodes, which means there is a single edge between every node. The triangle is a complete subgraph on 3 nodes and it may fail with respect to probability parameter 'p.' This study uses an exhaustive enumeration algorithm to systematically explore all possible states of the network, evaluating their operational viability and computing the overall network reliability. The algorithm involves generating state configurations, verifying network operability, and calculating reliability based on the failure probability 'p.' The program is executed for varying values of 'p,' and the resulting network reliabilities are graphically depicted. The findings offer insights into the sensitivity of network reliability to the failure probabilities of its constituent triangles, contributing to the broader understanding of reliability in complex network structures.

ALGORITHM

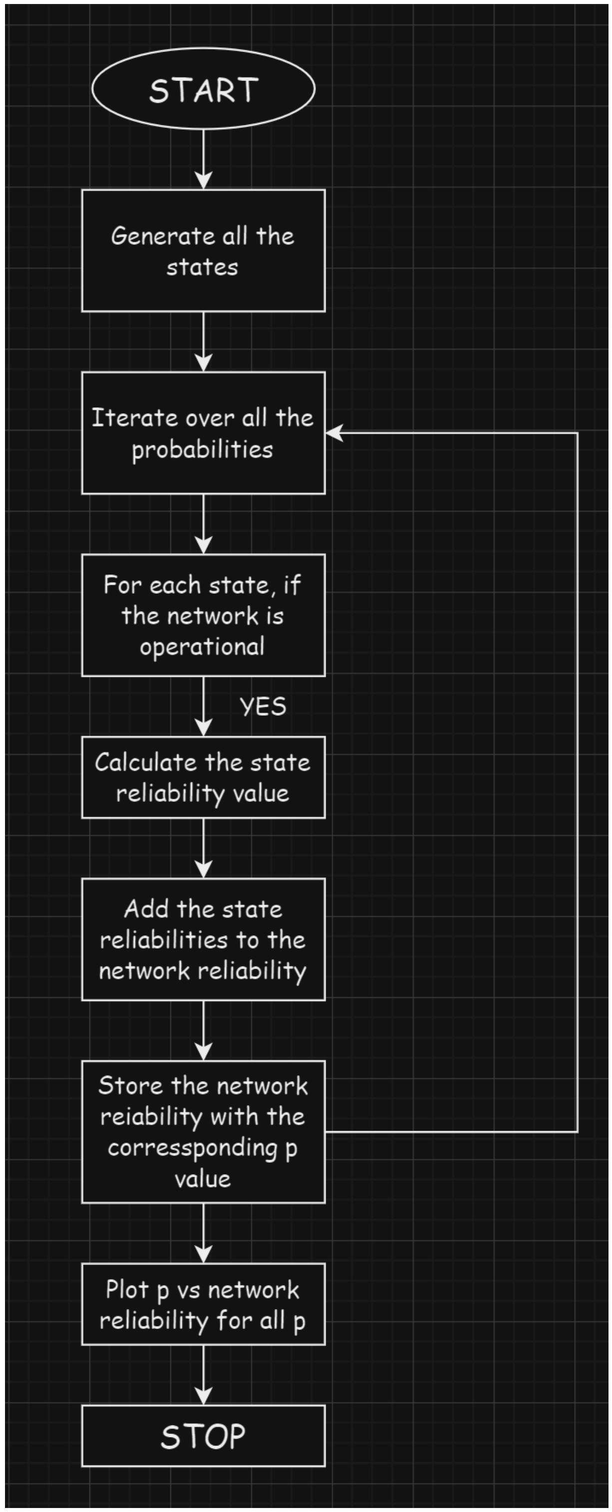
The algorithm to find network reliability of the given complete graph:

1. Generate all possible states for the triangles in the network. Each triangle can either be operational(1) or failed(0).
2. For each state generated, check if the system is operational by removing the edges of the triangles that failed. The system is operational if there are no isolated nodes in it.
3. If the state is operational, it is considered an ‘up’ state. Calculate the reliability of the state based on the ‘p’ value.
4. Calculate the network reliability by summing up the state reliabilities of all operational states.
5. Plot the probability vs network reliability graph.

Additional steps:

1. Initialize the number of nodes, the range of p values.
2. Check the operability of the network for each state by creating an adjacency matrix for that state. It can be done as follows:
    - Create a 5x5 matrix(for our network) with all ones.
    - Check if any triangle is down in the current state. Then, for that triangle make all the edges a ‘0’ in the matrix.
    - Check if there exists an isolated node, by checking if there is an empty column or row in the matrix.
    - If isolated node exits the network is not operational
3. Calculate the network reliability with varying probabilities from 0.05 to 1.0.
4. Plot the probabilities vs network reliability graph.

FLOW CHART


START
Generate all the states
Iterate over all the probabilities
For each state, if the network is operational
YES
Calculate the state reliability value
Add the state reliabilities to the network reliability
Store the network reiability with the corressponding p value
Plot p vs network reliability for all p
STOP

PSEUDO CODE

Function: network_reliability(p, num_nodes)

reliability_val = 0

Iterate through each state generated by the all states function.

For each state in all possible states, do

    If the state is operational then,

        state_probability(state,p)

        add the probability of this state to the reliability value

Return reliability value

Function: all_states(num_nodes)

Generate all the combinations of triangle states and return them

Generate all possible combinations of up/down states for triangles

Return list of all possible states

Function: if_operational(state, num_nodes)

Convert the given state to an adjacency matrix

If there are isolated nodes then,

    Return False

Else

    Return True

Function: state_to_adjacency_matrix(state, num_nodes)

Initialize an adjacency matrix with all values equal to 1

Set the diagonal of the adjacency matrix to 0

For each triangle in the network, do

    If triangle fails in the particular state then,

Make the edges of triangle as ‘0’ in the matrix
Return the adjacency matrix

Function: state_probability(state, p)
Probability = 1.0
For each triangle state, do
If triangle state is 1 then,
Probability = Probability * p
Else,
Probability = Probability * (1-p)

#MAIN
num_nodes = 5
probabilities = []
net_reliabilities = []
cur_p = 0.05
While cur_p < 1.05, do
Append cur_p to probabilities
net_reliability = Calculate network reliability for p and num_nodes
Append net_reliability to net_reliabilities
Increment cur_p by 0.05

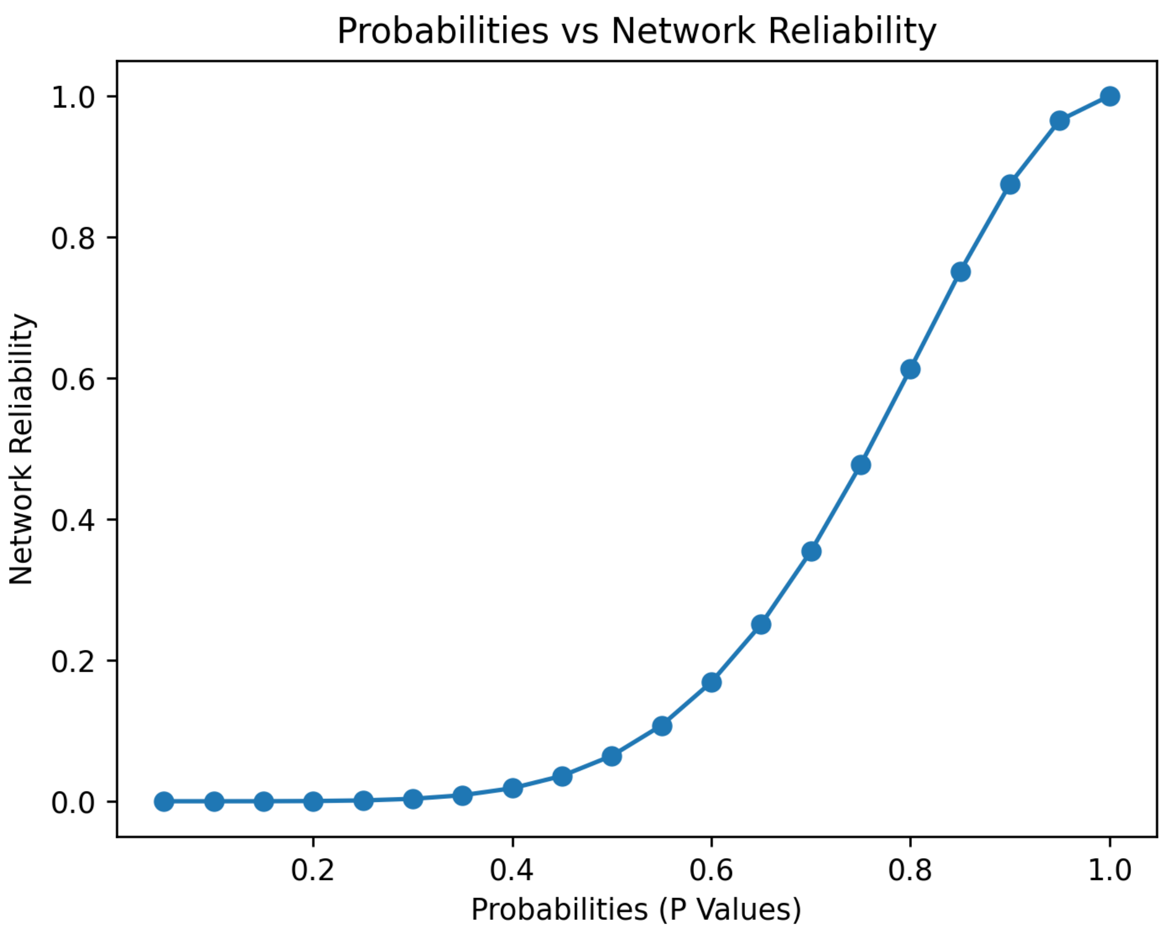
For each probability value in the list, do
Print the probabilities and the corresponding reliabilities
Display the plot with probabilities on the x-axis and net_reliabilities on the y-axis

OUTPUT AND RESULTS

The Network Reliability values for varying probabilities are as follows:

Probability	Network Reliability
p = 0.05	7.810605468750008e-08
p = 0.1	4.990600000000003e-06
p = 0.15	5.660329042968748e-05
p = 0.2	0.0003154944
p = 0.25	0.0011882781982421875
p = 0.3	0.0034835993999999997
p = 0.35	0.008568684579492185
p = 0.4	0.018487705600000008
p = 0.45	0.03599505943417966
p = 0.5	0.06445312499999999
p = 0.55	0.10754724701230454
p = 0.6	0.16878274559999995
p = 0.65	0.25075178481386734
p = 0.7	0.35419407940000003
p = 0.75	0.4769268035888674
p = 0.8	0.6127878144000004
p = 0.85	0.7508245413373051
p = 0.9	0.8750707506000004
p = 0.95	0.9653869913091799
p = 1.0	1.0

```
PS C:\Users\Yash> & C:/Users/Yash/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Yash/Desktop/CS6385_Project2_YXD210008.py
Probability      Network Reliability
p = 0.05         7.810605468750008e-08
p = 0.1          4.990600000000003e-06
p = 0.15         5.660329042968748e-05
p = 0.2          0.0003154944
p = 0.25         0.0011882781982421875
p = 0.3          0.003483599399999997
p = 0.35         0.008568684579492185
p = 0.4          0.018487705600000008
p = 0.45         0.03599505943417966
p = 0.5          0.06445312499999999
p = 0.55         0.10754724701230454
p = 0.6          0.16878274559999995
p = 0.65         0.25075178481386734
p = 0.7          0.35419407940000003
p = 0.75         0.4769268035888674
p = 0.8          0.6127878144000004
p = 0.85         0.7508245413373051
p = 0.9          0.8750707506000004
p = 0.95         0.9653869913091799
p = 1.0          1.0
```



APPENDIX
PROGRAM

```
# Import the required libraries
import itertools
import numpy as np
import matplotlib.pyplot as plt

# Function for calculating the final reliability value of the network
def network_reliability(p, num_nodes):
    reliability_val = 0
    for state in all_states(num_nodes):
        if if_operational(state, num_nodes):
            reliability_val += state_probability(state, p)
    reliability = reliability_val
    return reliability

# Function for returning all the triangle states possible
def all_states(num_nodes):
    # Generate all possible combinations of up/down states for triangles
    # Each triangle is represented as 1 for up and 0 for down
    num_triangles = num_nodes * (num_nodes - 1) * (num_nodes - 2) // 6
    triangle_states = list(itertools.product([0, 1], repeat=num_triangles))

    return triangle_states

# Function to check if the given state is operational
def if_operational(state, num_nodes):
    # Check if the network is operational for a given state
    # Return True if operational, False otherwise
    adjacency_matrix = state_to_adjacency_matrix(state, num_nodes)

    # Check if there are any isolated nodes
```

```
    return not any(np.sum(adjacency_matrix, axis=0) == 0)

# Function to convert the given state to an adjacency matrix
def state_to_adjacency_matrix(state, num_nodes):
    # Initializing the adjacency matrix
    adjacency_matrix = np.ones((num_nodes, num_nodes), dtype=int)

    for i in range(num_nodes):
        adjacency_matrix[i][i] = 0

    triangles = [(0, 1, 2), (0, 1, 3), (0, 1, 4), (0, 2, 3), (0, 2, 4), (0, 3, 4), (1, 2, 3), (1, 2,
4), (1, 3, 4), (2, 3, 4)]

    for l in range(len(state)):
      (i, j, k) = triangles[l]
      if state[l] == 0:
        adjacency_matrix[i][j] = adjacency_matrix[j][i] = 0
        adjacency_matrix[j][k] = adjacency_matrix[k][j] = 0
        adjacency_matrix[i][k] = adjacency_matrix[k][i] = 0

    return adjacency_matrix

# Function to calculate the probability of the given state
def state_probability(state, p):
    # Calculate the probability of a given state based on the failure probability p
    probability = 1.0

    for triangle_state in state:
        probability *= p if triangle_state == 1 else (1 - p)

    return probability
```

```
# Main Program
if __name__ == "__main__":
  num_nodes = 5
  probabilities = []
  net_reliabilities = []

  # Run the program for different values of p and collect reliability values
  cur_p = 0.05
  while cur_p < 1.05:
    probabilities.append(round(cur_p, 2))
    net_reliabilities.append(network_reliability(cur_p, num_nodes))
    cur_p += 0.05

  # Print the reliability values for all the p values
  for i in range(len(probabilities)):
    print(f"p = {probabilities[i]} \t {net_reliabilities[i]}")

  # Plot the results (Probability vs Network Reliability)
  plt.plot(probabilities, net_reliabilities, marker='o')
  plt.xlabel('Probabilities (P Values)')
  plt.ylabel('Network Reliability')
  plt.title('Probabilities vs Network Reliability')
  plt.show()
```

README

1. Make sure that python is installed (version 3.x recommended).
2. Install the required libraries (itertools, numpy and matplotlib) if they are not installed using the commands:
   - pip install networkx
   - pip install matplotlib
   - Pip install numpy
3. Copy the whole program given in the previous section and save it. Run the program.
4. The code outputs the network reliability values for the corresponding probabilities. It also displays the plot of the dependency between probabilities and network reliability.

REFERENCES

- Lecture Notes
- https://app.diagrams.net/
- https://docs.python.org/3/library/itertools.html
- https://numpy.org/devdocs/user/index.html#user
- https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html