CS 6385 Project 2 Report

Algorithm Description

The method we use is like trying every possible combination of working and failing connections in our network to see how often the network stays connected. We focus on groups of three points (triangles) and check if these triangles are working or not.

We first list all the possible triangles. Then, for each way these triangles can be up (working) or down (not working), we pretend to remove the not working parts and see if all the points are still connected. We calculate how likely each situation is and add up the chances of the network staying connected.

Details of the Code

Each part of the code does a specific job: setting up the network, identifying the triangles, simulating different scenarios of triangles working or not, and then calculating how likely the network is to stay connected in each scenario.

Network Representation and Triangle Identification

Graph Creation and Triangle Enumeration

In the first step, we create a graph representing our network. This graph is a set of five points (or nodes), where each point is connected to every other point. This setup is known as a 'complete graph' because all possible connections between points are present. The next task is to identify all the 'triangles' within this graph. A triangle here refers to a set of three interconnected points. Since our graph is complete, every combination of three points forms a triangle. We use a Python function to list all these triangles, which are crucial for our later calculations.

Simulation of Network States

Generating and Testing States

For each triangle, we consider two states: operational (up) and non-operational (down). The core part of our program is to simulate every possible combination of these states across all triangles. In each simulation, we adjust our graph to reflect the state of each triangle: if a triangle is down, we remove the connections (edges) that make up that triangle. This step is crucial because it shows us how the network looks in every possible scenario of triangle failures.

Operational Check

Assessing Network Connectivity

After simulating each state, we need to check if the network is still 'operational'. In our context, operational means that there are no isolated points - every point is still connected to the others in some way. Our Python function does this by checking if, after removing edges for down triangles, there are any points left with no connections. If every point still has at least one connection, the network in that state is considered operational.

Probability Calculations

Computing Network Reliability

The final step is to calculate the network's reliability. Reliability here is defined as the likelihood that the network remains operational. For each network state simulated, we calculate the probability of that state occurring based on a given probability 'p' that each triangle is operational. We then add up the probabilities of all states where the network is operational. This sum gives us the overall reliability of the network for a specific probability 'p'.

Iteration over Probability Values

Experimentation and Plotting

We repeat the reliability calculation for different values of 'p', ranging from 0.05 to 1 in steps of 0.05. This step is important as it shows us how the network's reliability changes as the likelihood of each triangle being operational changes. After calculating reliability for each value of 'p', we use these data points to plot a graph. This graph visually represents the relationship between the probability of triangle functionality and the overall reliability of the network.

Conclusion

Implementation Summary

In conclusion, the implementation of this project involved creating a network model, enumerating possible failure states, assessing the network's operational status in each state, and calculating the network's overall reliability based on these states. The iterative calculation over a range of probabilities 'p' and the subsequent graphical representation provides a comprehensive view of how individual component reliability impacts the overall network reliability. This methodical approach highlights the importance of component reliability in complex network systems.

Analysis of the Graph

The graph shows a curve that starts very low when 'p' is at 0.05 and grows significantly as 'p' increases. This indicates that when the probability of each triangle being operational is very low, the network's reliability is also low. However, as the probability of the triangles being operational increases, so does the reliability of the network.

The curve appears to be sigmoidal, with a slower increase in reliability from 'p' = 0.05 to about 'p' = 0.65, after which the curve steepens. This suggests that there is a threshold area where the reliability of the network is particularly sensitive to changes in 'p'. After a certain point, increases in 'p' yield larger increases in reliability until it begins to plateau as it approaches 'p' = 1.

As 'p' approaches 1, the reliability of the network also approaches 1, indicating that when it is almost certain that all triangles will be operational, the network is almost guaranteed to be reliable. This is expected since if all components (triangles) are functioning, the network should not have any failures.

At very low 'p' values, the reliability of the network remains close to zero, which means that when the probability of triangles being operational is very low, the network is almost certainly going to be non-operational.

In summary, the output graph provides a clear visual representation of the relationship between the reliability of individual components within a network and the overall network reliability. It underscores the importance of component reliability in network design and maintenance.

Pseudo Code:

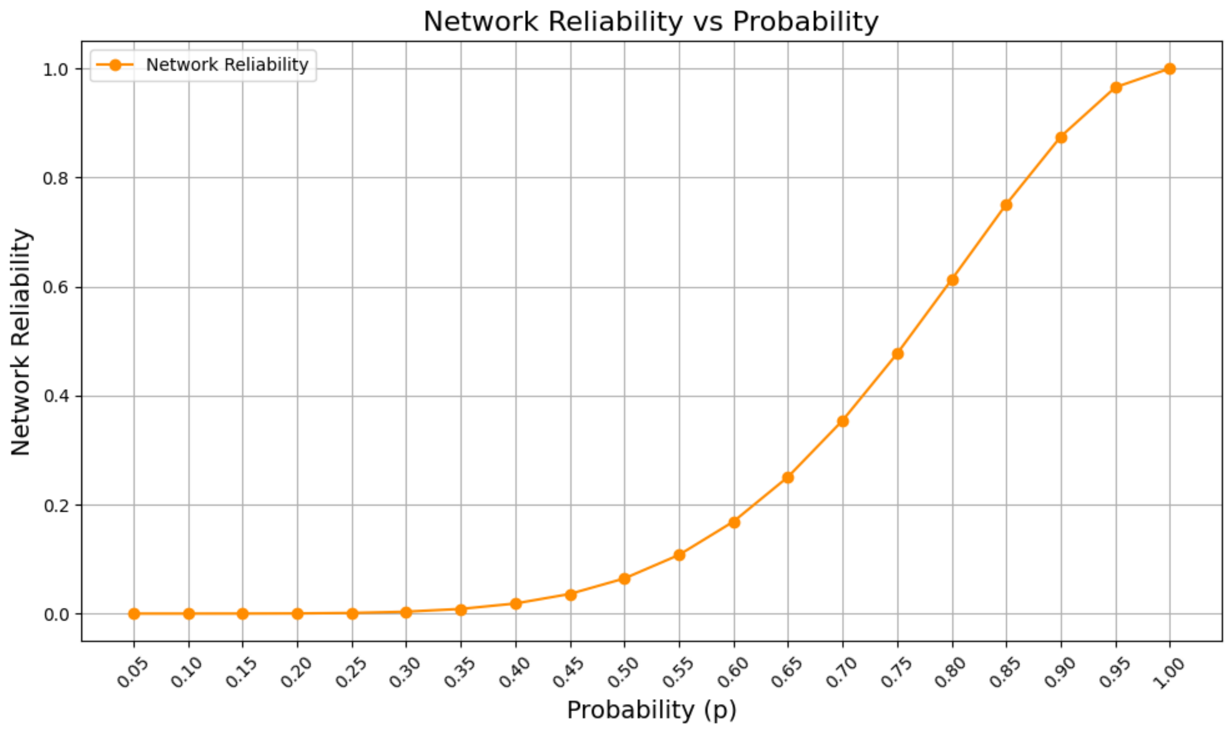
```
IMPORT itertools
IMPORT matplotlib.pyplot

DEFINE CLASS NetworkGraph
    DEFINE FUNCTION __init__ with parameter size
        SET self.size to size
        SET self.graph to a complete graph of size 'size'
        SET self.triangles to the result of _find_triangles()

    DEFINE FUNCTION _find_triangles
        INITIALIZE empty set for triangles
        FOR EACH node in graph
            FOR EACH pair of neighbors of node
                IF pair forms a triangle
                    ADD triangle to set
        RETURN set of triangles

    DEFINE FUNCTION is_operational with parameter simulated_graph
        FOR EACH set of neighbors in simulated_graph
            IF set is empty, RETURN False
        RETURN True

    DEFINE FUNCTION simulate_network with parameter state
        INITIALIZE simulated_graph as a copy of self.graph
        FOR EACH triangle and its operational state in state
            IF triangle is not operational
                REMOVE connections of triangle from simulated_graph
        RETURN simulated_graph
    DEFINE FUNCTION calculate_reliability with parameter p
        INITIALIZE reliability to 0
        FOR EACH possible state of triangles
            IF simulate_network(state) is operational
                CALCULATE probability of this state and ADD to reliability
        RETURN reliability

END CLASS

DEFINE FUNCTION plot_reliability with parameters p_values and reliabilities
    SETUP matplotlib plot
    PLOT p_values against reliabilities
    SET plot labels, title, grid, legend, and layout
    SHOW plot

MAIN
    SET network_size to 5
    INITIALIZE network as NetworkGraph with network_size
    INITIALIZE p_values as a range of probabilities
    INITIALIZE reliabilities as an empty list

    FOR EACH p in p_values
        CALCULATE reliability for p and ADD to reliabilities
        CALL plot_reliability with p_values and reliabilities
```

Appendix : Source Code

```python
import itertools

import matplotlib.pyplot as plt

class NetworkGraph:
    def __init__(self, size):
        # Initialize the network graph with the specified number of nodes.
        self.size = size  # Size refers to the number of nodes in the graph.
        # Create a complete graph, where each node is connected to all other
nodes (except itself).
        self.graph = {i: {j for j in range(size) if j != i} for i in range(size)}
        # Identify all unique triangles within the graph. A triangle is a set of
three interconnected nodes.
        self.triangles = self._find_triangles()

    def _find_triangles(self):
        # This private method identifies all unique triangles in the graph.
        # It uses itertools.combinations to find all possible pairs of neighbors
for each node.
        # Then checks if those pairs are interconnected, indicating a triangle.
        return {tuple(sorted((node, *pair)))
                for node in self.graph
                for pair in itertools.combinations(self.graph[node], 2)
                if pair[0] in self.graph[pair[1]]}

    def is_operational(self, simulated_graph):
        # This method checks if the network is operational.
        # A network is operational if each node has at least one active
connection to another node.
        return all(neighbors for neighbors in simulated_graph.values())

    def simulate_network(self, state):
        # Simulate the network based on the operational state of triangles.
        # 'state' is a sequence indicating which triangles are operational (True)
or not (False).
        simulated_graph = {node: set(neighbors) for node, neighbors in
self.graph.items()}
        for triangle, operational in zip(self.triangles, state):
            if not operational:
                # If a triangle is not operational, remove its connections from
the graph.
                # This simulates the effect of a triangle (set of connections)
being down.
```

```
                simulated_graph = {node: neighbors - set(triangle)
                                   if node in triangle else neighbors
                                   for node, neighbors in
simulated_graph.items()}
        return simulated_graph

    def calculate_reliability(self, p):
        # Calculate the overall reliability of the network given the probability
'p'.
        # Reliability is computed over all possible states of the triangles.
        # Uses the principle of Bernoulli trials for each triangle's operational
state.
        return sum((p ** sum(state)) * ((1 - p) ** (len(state) - sum(state)))
                   for state in itertools.product([True, False],
repeat=len(self.triangles))
                   if self.is_operational(self.simulate_network(state)))

def plot_reliability(p_values, reliabilities):
    # Function to plot the relationship between probability and network
reliability.
    plt.figure(figsize=(10, 6))
    plt.plot(p_values, reliabilities, 'o-', color='darkorange', label='Network
Reliability')
    plt.xlabel('Probability (p)', fontsize=14)  # Label for the x-axis.
    plt.ylabel('Network Reliability', fontsize=14)  # Label for the y-axis.
    plt.title('Network Reliability vs Probability', fontsize=16)  # Title of the
plot.
    plt.grid(True)  # Enable grid for better readability.
    plt.legend()  # Show legend.
    plt.xticks(p_values, rotation=45)  # Rotate x-axis labels for clarity.
    plt.tight_layout()  # Adjust layout for no overlap.
    plt.show()  # Display the plot.

# Main execution block
network_size = 5  # Define the size (number of nodes) for the network graph.
network = NetworkGraph(network_size)  # Instantiate the NetworkGraph class.
p_values = [i / 20 for i in range(1, 21)]  # Create a range of probability values
from 0.05 to 1.0.
reliabilities = [network.calculate_reliability(p) for p in p_values]  # Calculate
reliability for each probability value.

# Call the function to plot the network reliability against different probability
values.
plot_reliability(p_values, reliabilities)
```

Result:



References:

1. Graph Theory and Network Analysis:
    - Basic concepts of graph theory, including nodes, edges, and connectivity, are essential for understanding how networks are structured and analyzed.
    - Reference: Graph Theory - Basic Concepts
2. Python itertools Library:
    - The itertools library in Python, especially functions like combinations, is used extensively in your code to generate combinations of graph nodes.
    - Reference: Python itertools
3. Network Reliability:
    - Understanding network reliability, including concepts like redundancy and connectivity, is key to grasping the purpose of your simulation.
    - Reference: Network Reliability
4. Matplotlib for Data Visualization:

- Matplotlib is a Python library used for plotting graphs and visualizations, as seen in your code for plotting the reliability graph.
- Reference: Matplotlib Tutorial

5. Bernoulli Trials and Probability:
    - Bernoulli trials and probability calculations play a significant role in calculating the reliability of the network based on the probability of each triangle being operational.
    - Reference: Bernoulli Trials

Readme:

Requirements:

- Python 3.x
- Matplot Lib

Set Up:

```
pip install matplotlib
```

To run the script, execute the following command in the project directory:

```
python rxc210069_project2.py
```