

Algorithmic Aspects of Telecommunication Networks

Project-2

Submitted by

Soumya Mukhija

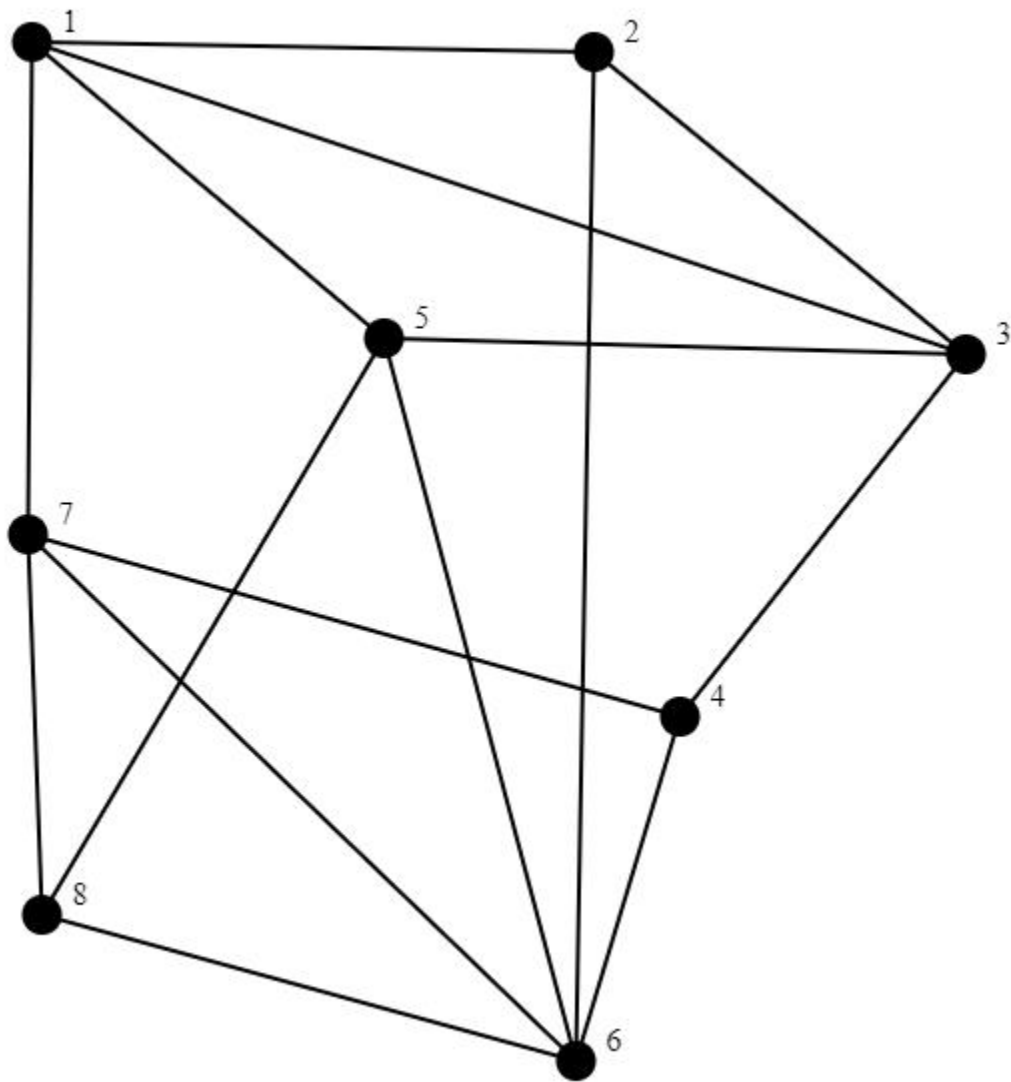
SXM200134

Table of Contents

Sr.	Name	Page
1.	Introduction	3
2.	Algorithm	5
3.	Results	8
4.	Appendix	9
5.	How to run (README)	12
6.	References	13

Introduction

According to the problem statement, the task in this project is to experimentally study how the network reliability depends on individual component reliabilities of the network for a specific use case (given below).



Exhaustive enumeration is a technique in which we consider all possible cases and evaluate each one to determine a holistic value. For example, in the case of determining network reliability for a network that stays operational as long as it's connected, we consider all the possibilities in which the individual nodes can go down without affecting the 'up' state of the entire network. This means that despite one or more nodes being 'down', the rest of the network would still stay connected.

Consider for a simple graph with 2 nodes, A and B:



Considering all possible cases using exhaustive enumeration technique, we have $R(A)R(B)$, $R(1-A)R(B)$, $R(B)R(1-B)$, and $R(1-A)R(1-B)$. However, in the last case, the network does not stay operational since there is no link (considering self-links a valid case for an operational network). Therefore the reliability of the network is:

$$R(N) = R(A)R(B) + R(A)R(1-B) + R(1-A)R(B)$$

In the problem statement, it is given that each node is up with a probability p , where $p \in [0.05, 1]$, experimented with a step of 0.05. Therefore the probability of a node being down will be $(1-p)$. We have implemented this case using Python and explained it in the following sections.

Algorithm

Overview

This algorithm is used for calculating the reliability of a network given a probability parameter for each node's reliability.

The first step is to create a graph or adjacency matrix to visualize the topology of the network. Then, the exhaustive enumeration method is used to get all possible combinations of failed nodes that could result in the network becoming disconnected. For each combination in the exhaustive list, we check if the remaining network is still connected.

Next, a reliability map is created to calculate and store reliability values for the varying reliability parameter. The reliability value for the network being connected is calculated by summing over the product of probabilities of nodes being up and not being up for all possible combinations of nodes that result in a connected network.

Finally, the obtained network reliability values are plotted graphically to see how they depend on the probability parameter.

Pseudocode

The implementation of the algorithm will be done in the following steps:

1. Create a graph or adjacency matrix to visualize the topology of our graph.

```
graph_original = nx.Graph()
edge_list =
[(1,2), (1,3), (1,5), (1,7), (2,3), (2,6), (3,4), (3,5), (4,6), (4,7), (5,6), (
5,8), (6,7), (6,8), (7,8)]
graph_original.add_edges_from(edge_list)

nx.draw(graph_original, with_labels=True)
plt.show()
```

2. Next, we get a list of all nodes in the original graph and run an exhaustive enumeration to store all combinations of nodes that can fail. Using itertools combinations, we have 2^n lists, where $n = \text{num_of_nodes}$.

```
exhaustive_enumeration = []
for i in range(len(nodes_of_graph)+1):
    exhaustive_enumeration.extend(combinations(nodes_of_graph, i))
```

3. For each combination in the exhaustive list, we check if the remaining network is still connected. The logic is that the network is connected as long as there is only one connected component which is the entire graph/network itself.

```
for failed_nodes in exhaustive_enumeration:
    remaining_network = graph_original.copy()
    remaining_network.remove_nodes_from(failed_nodes)
    if nx.number_connected_components(remaining_network)==1:
        connected_network.append(list(remaining_network.nodes()))
```

```
len(connected_network) #This gives 201.
```

- Now we create a reliability map to calculate and store reliability values for the varying reliability parameter. For a network that is connected, the node_list is used to find the nodes that are up, and we multiply the product of probabilities of those nodes being up with the product of probabilities of the other nodes not being up, or failing. The reliability value for the network being connected is calculated by summing over this product for all possible combinations of nodes that result in a connected network.

```
reliability_map = {}
for prob in np.arange(0.05, 1.05, 0.05):
    p = round(prob, 2)
    r = 0
    for node_list in connected_network:
        n = len(node_list)
        r += p**n * (1-p)**(8-n)
    reliability_map[p] = r

print(reliability_map)
{0.05: 0.31038729832031237, 0.1: 0.48585817000000002, 0.15: 0.5786467264453122, 0.2: 0.62529792000000006, 0.25: 0.6502227783203125, 0.3: 0.66865856999999995, 0.35: 0.6891039889453131, 0.4: 0.71529471999999989, 0.45: 0.7477800233203126, 0.5: 0.78515625, 0.55: 0.8250084764453083, 0.6: 0.86460671999999954, 0.65: 0.9013984733203112, 0.7: 0.93333457000000025, 0.75: 0.9590606689453125, 0.8: 0.97800191999999996, 0.85: 0.9903636483203102, 0.9: 0.9970661700000002, 0.95: 0.9996271264453147, 1.0: 1.0}
```

- The resultant values seem to increase as the reliability parameter increases. We can plot this on a graph to visualize graphically how the obtained network reliability value depends on p.

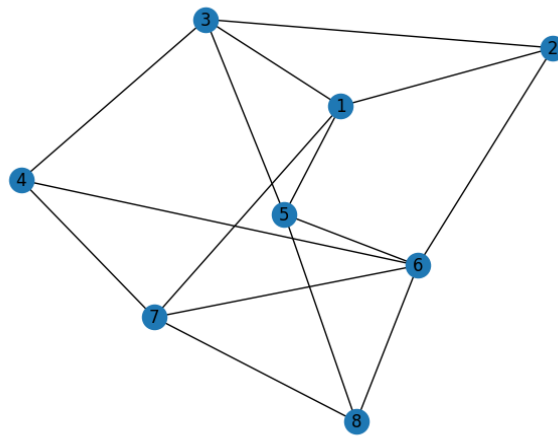
```
plt.plot(list(reliability_map.keys()), list(reliability_map.values()))
```

Debugging

There are some ways in which debugging has been implemented in this program. This is by

- Visualizing the original graph to ensure what we are working on is the same as expected.

```
nx.draw(graph_original, with_labels=True)
plt.show()
```



2. Asserting the value of exhaustive enumeration to check if it is correct. Since there are 8 nodes, the number of possible cases should be $2^8 = 256$, since each node can have 2 cases - UP or DOWN. If the assertion is invalid, i.e, the length of exhaustive_enumeration is not equal to the possible length, then the program exits with the message, "Number of nodes of the graph are not equal to the input."

```

num_nodes = len(nodes_of_graph)
possible = 2**num_nodes
exhaustive_enumeration = []
for i in range(len(nodes_of_graph)+1):
    exhaustive_enumeration.extend(combinations(nodes_of_graph, i))

assert len(exhaustive_enumeration) == possible, "Number of nodes of
the graph are not equal to the input."

```

3. Printing values such as connected_network, reliability map, etc. to determine if there are any unexpected values.

Results

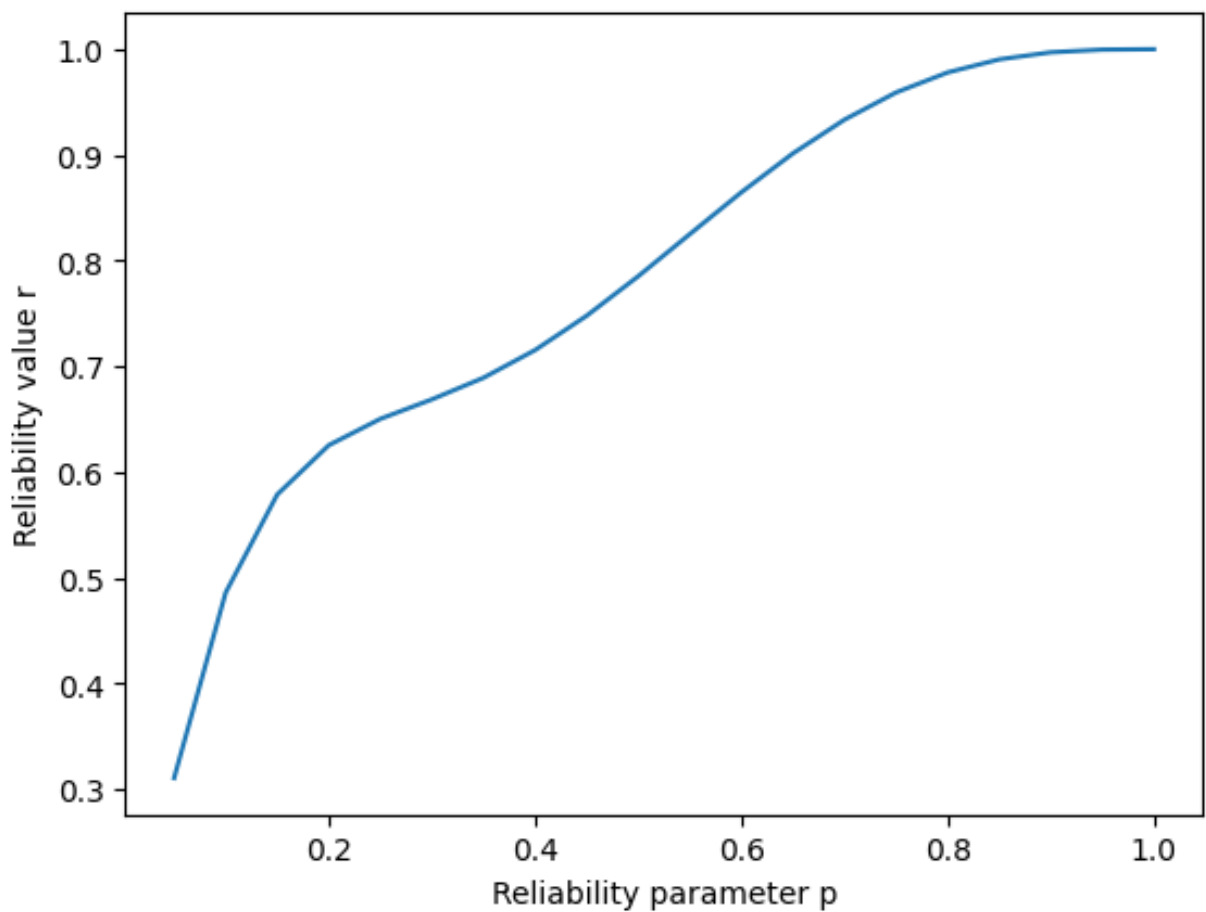
The result of this experiment is a reliability map that shows how the reliability of a network (r) depends on the probability parameter for each node's reliability (p).

The reliability values are calculated using an exhaustive enumeration method to get all possible combinations of failed nodes that could result in the network becoming disconnected.

The reliability value for the network being connected is calculated by summing over the product of probabilities of nodes being up and not being up for all possible combinations of nodes that result in a connected network.

The resulting reliability values are then plotted graphically to show how they depend on the probability parameter.

After running this experiment, we determined that the reliability of the network (r) increases as the reliability of individual nodes (p) increases.



Appendix

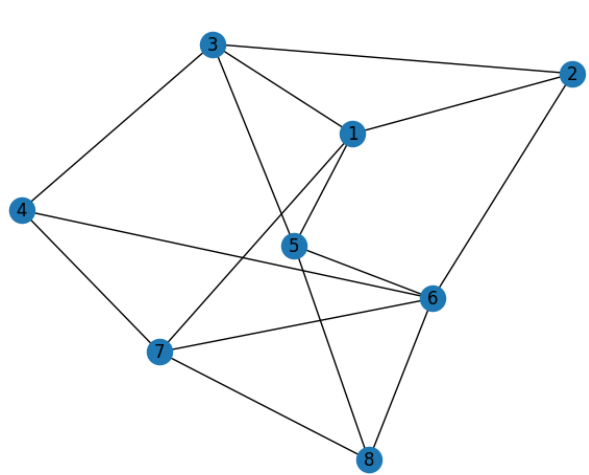
```

import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from itertools import combinations

# First we see the topology of the graph and create an adjacency matrix
adj_mat = [
    [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]
]

# Alternatively, we create a NetworkX graph by adding edges
graph_original = nx.Graph()
edge_list =
[(1,2), (1,3), (1,5), (1,7), (2,3), (2,6), (3,4), (3,5), (4,6), (4,7), (5,6), (5,8), (
6,7), (6,8), (7,8)]
graph_original.add_edges_from(edge_list)
nx.draw(graph_original, with_labels=True)
plt.show()

```



```

connected_network = []
nx.number_connected_components(graph_original)

nodes_of_graph = list(graph_original.nodes())
Nodes_of_graph      #1

num_nodes = len(nodes_of_graph)
possible = 2**num_nodes
exhaustive_enumeration = []

```

```

for i in range(len(nodes_of_graph)+1):
    exhaustive_enumeration.extend(combinations(nodes_of_graph, i))
assert len(exhaustive_enumeration) == possible, "Number of nodes of the
graph are not equal to the input."

for failed_nodes in exhaustive_enumeration:
    remaining_network = graph_original.copy()
    remaining_network.remove_nodes_from(failed_nodes)
    if nx.number_connected_components(remaining_network)==1:
        connected_network.append(list(remaining_network.nodes()))
len(connected_network)          #201

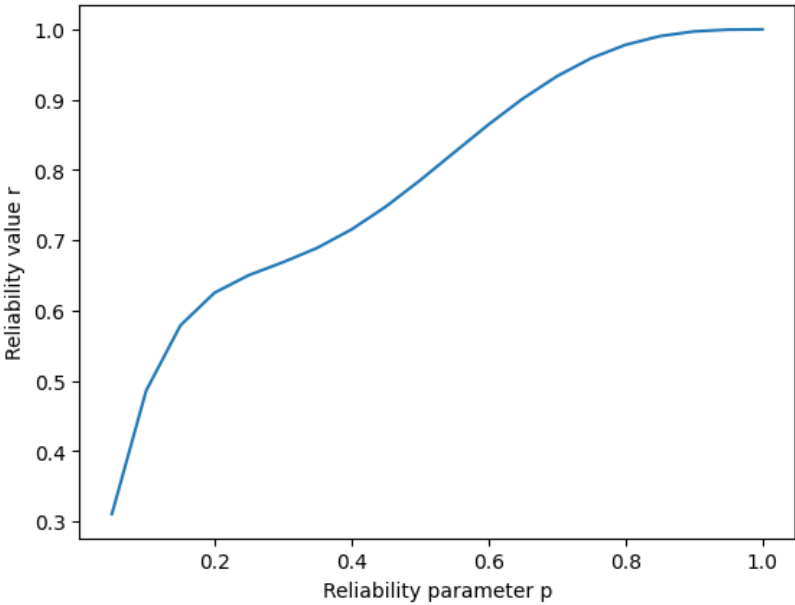
reliability_map = {}

for prob in np.arange(0.05, 1.05, 0.05):
    p = round(prob, 2)
    r = 0
    for node_list in connected_network:
        n = len(node_list)
        r += p**n * (1-p)**(8-n)
    reliability_map[p] = r

print(reliability_map)
#{0.05: 0.31038729832031237, 0.1: 0.48585817000000002, 0.15:
0.5786467264453122, 0.2: 0.62529792000000006, 0.25: 0.6502227783203125, 0.3:
0.66865856999999995, 0.35: 0.6891039889453131, 0.4: 0.71529471999999989, 0.45:
0.7477800233203126, 0.5: 0.78515625, 0.55: 0.8250084764453083, 0.6:
0.86460671999999954, 0.65: 0.9013984733203112, 0.7: 0.93333457000000025, 0.75:
0.9590606689453125, 0.8: 0.97800191999999996, 0.85: 0.9903636483203102, 0.9:
0.9970661700000002, 0.95: 0.9996271264453147, 1.0: 1.0}

plt.plot(list(reliability_map.keys()), list(reliability_map.values()))
plt.xlabel('Reliability parameter p')
plt.ylabel('Reliability value r')
plt.show()

```



How to run (README)

1. Copy and paste the code in the [Appendix](#) section to a file on your system. Name that file something you like and add '.py' to the end, for example, **network_reliability.py**.
2. Open terminal and navigate to the directory you have saved this file in. Once you are there, type **python network_reliability.py**.

References

1. Lecture notes.
2. Networkx documentation.
3. Matplotlib documentation.
4. Python documentation.