

CS 6385 - Algorithmic Aspects of Telecommunication Networks

Project – 1

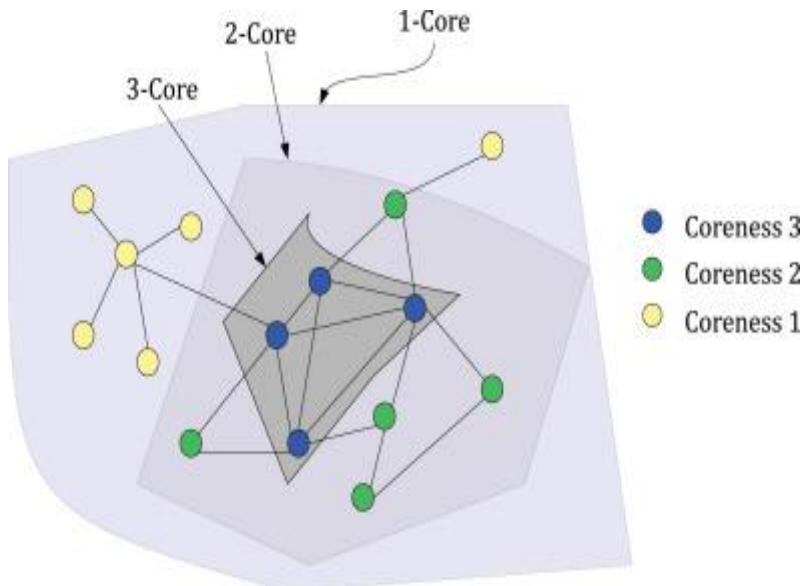
Implementation of K-Core Algorithm

PRANITHA BOJJA
NETID: PXB210063

Introduction

Random graphs are a fundamental concept in network theory and graph theory. They are used to model various real-world networks, including social networks, the World Wide Web, and biological networks. Understanding the structural properties of random graphs can provide valuable insights into the behavior of complex systems.

One essential concept in the study of random graphs is the "core number." The core number of a graph measures its structural core or central component. It helps identify highly connected regions within a network. In this project, we aim to explore how the core number of a random graph depends on the edge probability (p), which controls the likelihood of edges between nodes.



In this project, we use Python and NetworkX, a popular Python library for complex network analysis, to investigate the relationship between the core number and edge probability in random graphs.

The concept of k -core structures has several real-time applications:

Social Network Analysis: Identifying influential user groups and communities within social networks.

Biology: Understanding protein-protein interaction networks and identifying essential proteins for various biological processes.

Infrastructure Planning: Identifying critical components in transportation networks or power grids for improved planning and resilience.

Recommendation Systems: Enhancing personalized recommendations by considering user communities in collaborative filtering.

K- core Algorithm:

1. Delete all nodes of degree $< k$:

- In this step, we remove all nodes from the graph that have a degree (number of edges connected to the node) less than k . This reduces the graph by eliminating nodes that do not meet the k -core criteria.

2. Check if the remaining graph is empty:

- After the removal of low-degree nodes, we check whether the graph is empty. If it is, we output the message "the k -core is empty" and terminate. This means there is no k -core in the original graph.

3. If the remaining graph is non-empty:

- If the graph is not empty after node removal, we proceed to the next steps.

4. Check if every node in the remaining graph has degree $\geq k$:

- We examine the degrees of all nodes in the remaining graph. If every node has a degree greater than or equal to ' k ', this remaining graph is the k -core. We output this graph as the k -core of the original graph and terminate.

5. If not every node has degree $\geq k$, repeat from Step 1:

- If some nodes in the remaining graph do not have a degree of ' k ' or higher, it means the current graph is not the k -core. We go back to Step 1 to delete nodes of degree $< k$, and the process continues until we find the k -core or determine that it is empty.

This algorithm iteratively removes nodes that do not meet the k -core criteria until either we find the k -core or determine that it does not exist in the original graph. The process is repeated until the conditions for the k -core are met.

Project Pseudo Code:

Task – 1:

Input:

The algorithm for Task 1 receives the following input:

A real number 'p' ($0 < p < 1$), represents the edge probability for the random graph.

Output:

The output of the algorithm is as follows:

The value of Core (G_p), which is the core number of the generated random graph G_p .

This value is averaged over 10 independent runs of creating and analyzing G_p .

The resulting average core number is denoted as Core(p), representing the obtained average Core (G_p) value while using the specified edge probability ' p '.

The output when given $p = 0.6$ as input is Core(p) = Core (0.6) = 11.3 for 10 independent runs:

Enter the edge probability ($0 < p < 1$): 0.6

core values for run 1:11

core values for run 2:12

core values for run 3:11

core values for run 4:12

core values for run 5:10

core values for run 6:12

core values for run 7:12

core values for run 8:11

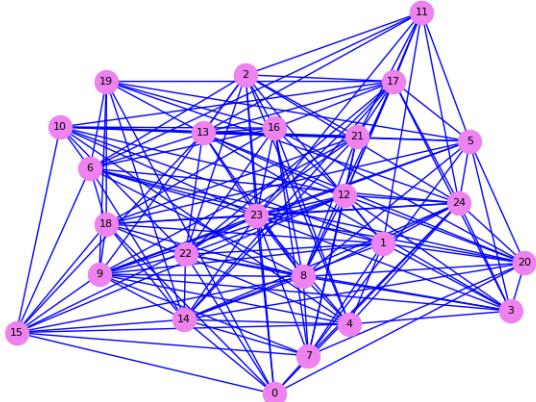
core values for run 9:11

core values for run 10:11

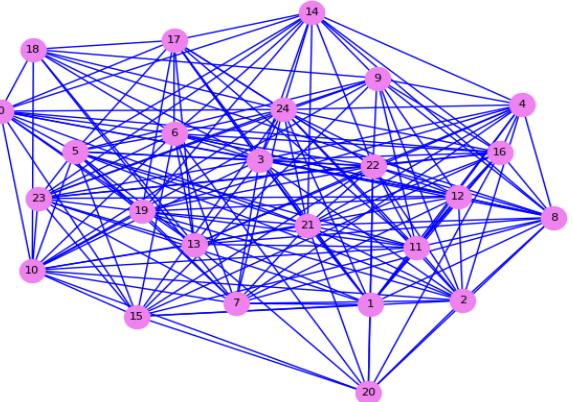
Core (0.6) = 11.3

The 10 different graphs generated for edge probability $p = 0.6$ are:

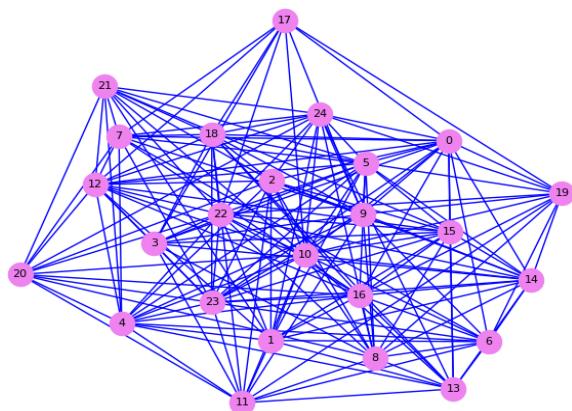
The output graph for run – 1 with $p=0.6$



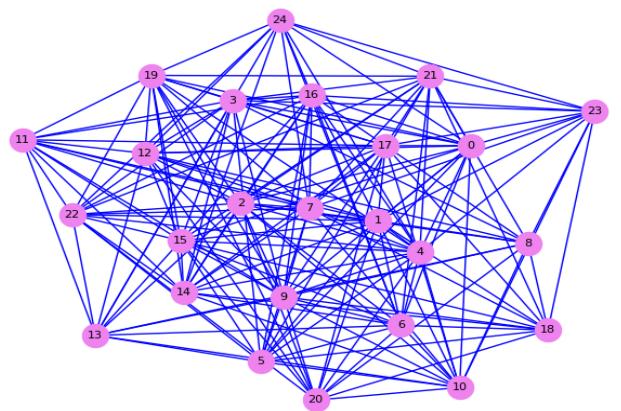
The output graph for run - 2 with $p=0.6$



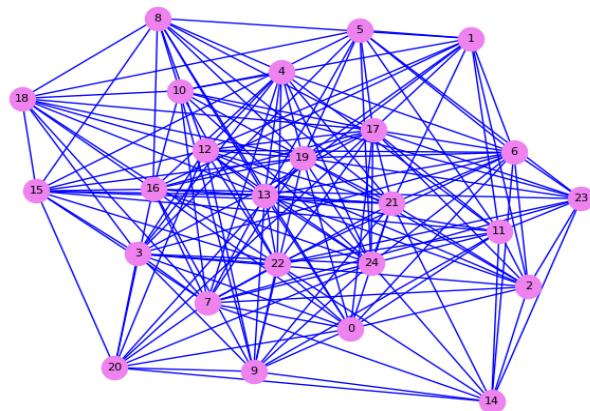
The output graph for run – 3 with $p=0.6$



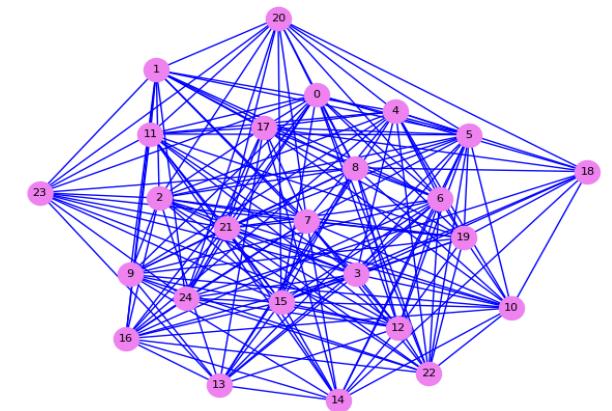
The output graph for run - 4 with $p=0.6$



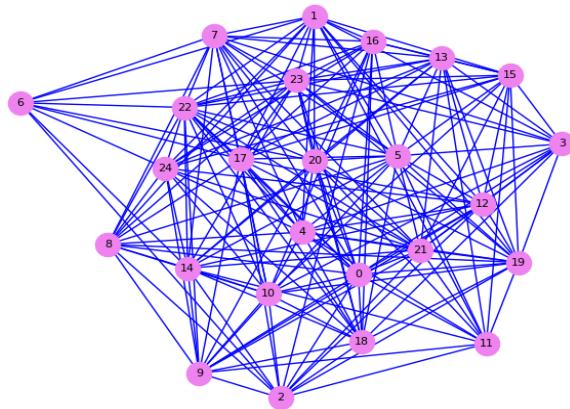
The output graph for run – 5 with $p=0.6$



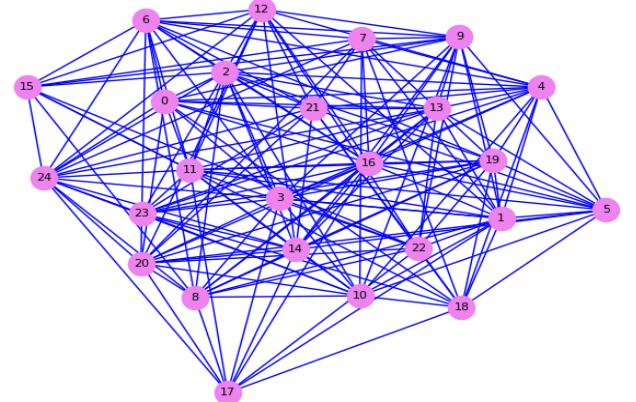
The output graph for run - 6 with $p=0.6$



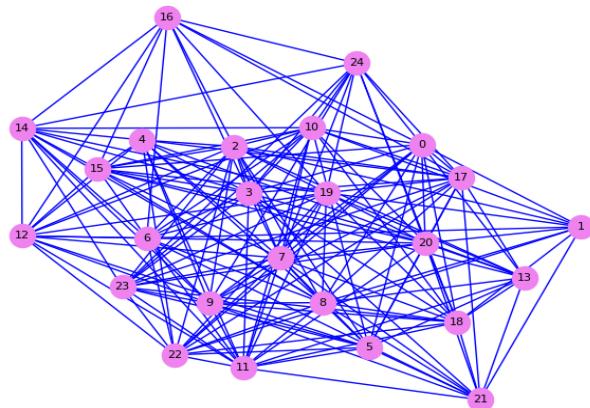
The output graph for run – 7 with p=0.6



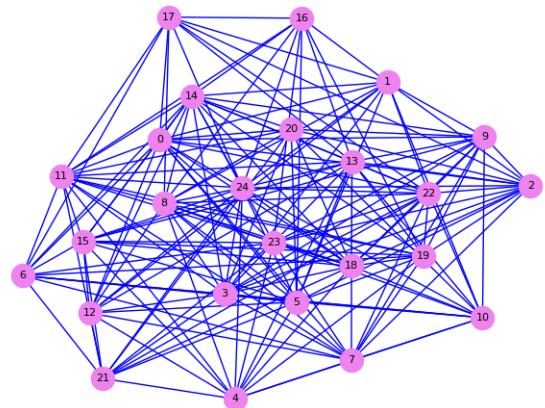
The output graph for run - 8 with p=0.6



The output graph for run – 9 with p=0.6



The output graph for run - 10 with p=0.6



Task – 2:

Task 2 involves creating graphical representations of three generated random graphs with different 'p' values to demonstrate the core numbers and corresponding k-core subgraphs. Here's a step-by-step description of the task:

Input:

- The 'p' values for the three random graphs: p=0.15 (sparse), p=0.5 (medium density), and p=0.85 (dense).
- The corresponding k values are determined by $k = \text{Core}(G)$, where G is the graphically shown graph.

Execution with Output:

1. Generate Random Graphs:

- Create three random graphs using the specified 'p' values (0.15, 0.5, and 0.85). These graphs represent varying levels of edge probability, resulting in different densities: sparse, medium, and dense.

2. Calculate Core Numbers:

- For each of the three random graphs, calculate the core number (k) using the algorithm described in Task 1.

The output of core values for 10 different runs when $p = 0.15$ and $k = \text{Core}(G)$ is as follows:

core values for run 1:3

core values for run 2:3

core values for run 3:2

core values for run 4:3

core values for run 5:3

core values for run 6:3

core values for run 7:3

core values for run 8:3

core values for run 9:3

core values for run 10:3

$\text{Core}(0.15) = 2.9$

The output of core values for 10 different runs when $p = 0.5$ and $k = \text{Core}(G)$ is as follows:

core values for run 1:9

core values for run 2:9

core values for run 3:9

core values for run 4:8

core values for run 5:8

core values for run 6:9

core values for run 7:9

core values for run 8:8

core values for run 9:9

core values for run 10:9

$\text{Core}(0.5) = 8.7$

The output of core values for 10 different runs when $p = 0.85$ and $k = \text{Core}(G)$ is as follows:

core values for run 1:16

core values for run 2:16

core values for run 3:18

core values for run 4:19

core values for run 5:17

core values for run 6:17

core values for run 7:18

core values for run 8:18

core values for run 9:16

core values for run 10:17

$\text{Core}(0.85) = 17.2$

3. Highlight k-Core Subgraphs:

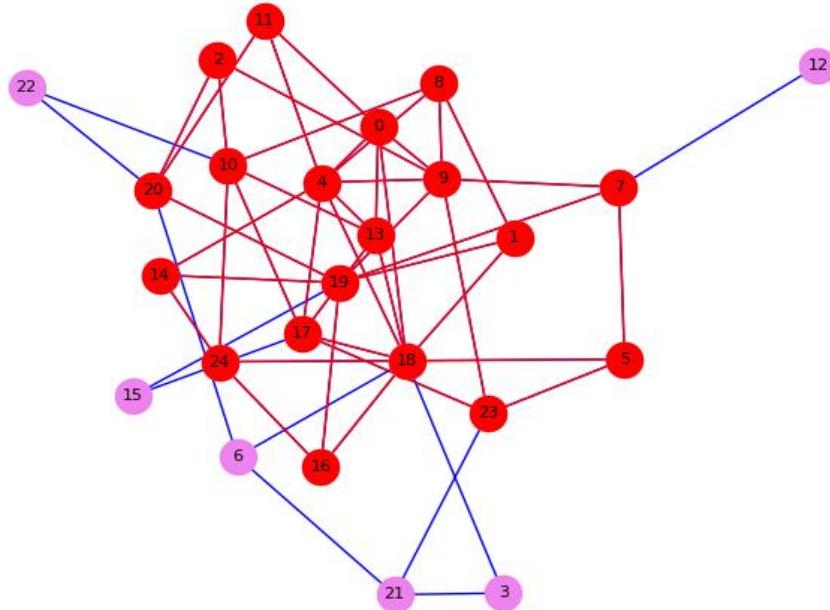
- Based on the calculated core numbers, identify the k-core subgraph in each random graph.
The k-core subgraph consists of nodes and edges that belong to the k-core.

4. Graphical Representation:

- Create visual representations of the random graphs and highlight the k-core subgraphs.

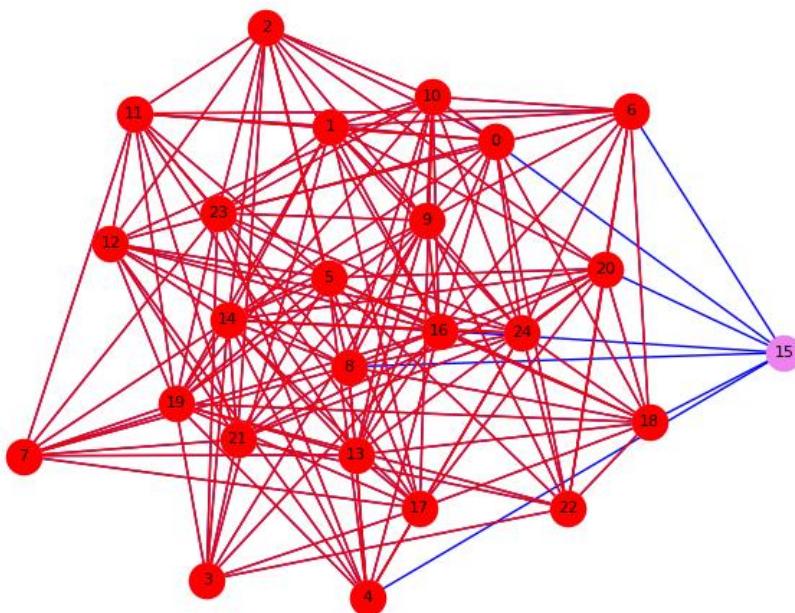
- For the graph with $p=0.15$ (sparse), draw the entire graph and highlight the k-core subgraph ($k=\text{Core}(G)$).

The k-core subgraph is highlighted with red nodes and red vertices for $p=0.15$



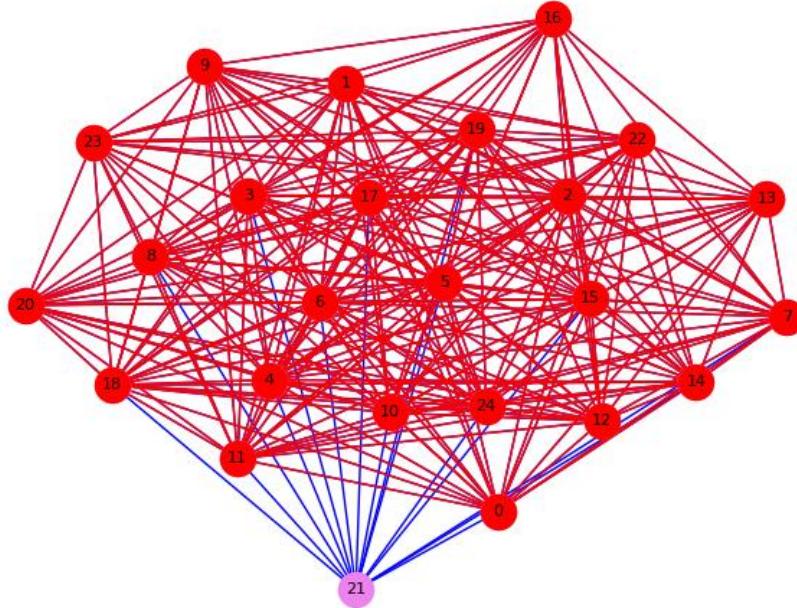
- For the graph with $p=0.5$ (medium density), draw the entire graph and highlight the k-core subgraph ($k=\text{Core}(G)$).

The k-core subgraph is highlighted with red nodes and red vertices for $p=0.5$



- For the graph with $p=0.85$ (dense), draw the entire graph and highlight the k -core subgraph ($k=\text{Core}(G)$).

The k -core subgraph is highlighted with red nodes and red vertices for $\mathbf{p=0.85}$



Task – 3:

Input:

Range of edge probabilities, 'p,' from 0.05 through 0.95 in increments of 0.05.

Output:

A graphical diagram illustrating how the core number ('Core(p)') depends on 'p'.

$$\text{Core}(0.05) = 1.3$$

$$\text{Core}(0.10) = 2.1$$

$$\text{Core}(0.15) = 2.8$$

$$\text{Core}(0.20) = 3.5$$

$$\text{Core}(0.25) = 4.4$$

$$\text{Core}(0.30) = 4.9$$

$$\text{Core}(0.35) = 6.0$$

$$\text{Core}(0.40) = 6.9$$

$$\text{Core}(0.45) = 8.0$$

$$\text{Core}(0.50) = 9.4$$

$$\text{Core}(0.55) = 10.1$$

$$\text{Core}(0.60) = 11.4$$

$$\text{Core}(0.65) = 12.2$$

$$\text{Core}(0.70) = 13.6$$

$$\text{Core}(0.75) = 14.7$$

$$\text{Core}(0.80) = 15.8$$

$$\text{Core}(0.85) = 17.2$$

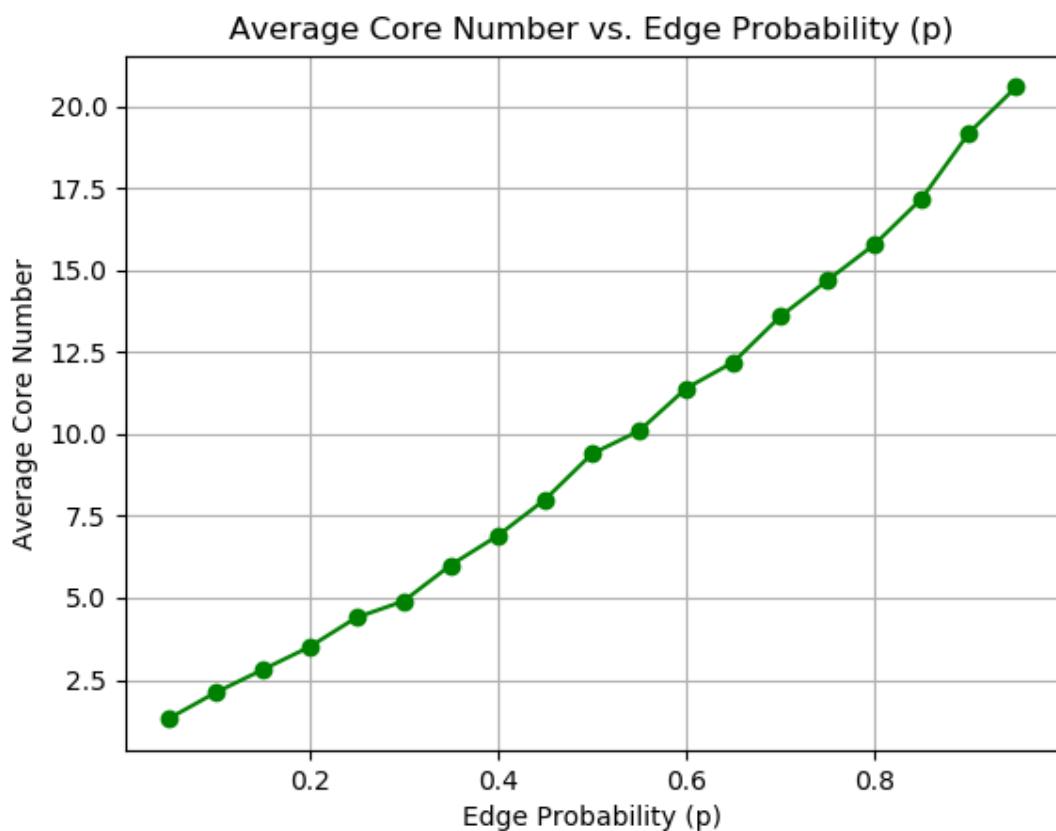
$$\text{Core}(0.90) = 19.2$$

$$\text{Core}(0.95) = 20.6$$

$$\text{Core}(G) = [1.3, 2.1, 2.8, 3.5, 4.4, 4.9, 6.0, 6.9, 8.0, 9.4, 10.1, 11.4, 12.2, 13.6, 14.7, 15.8, 17.2, 19.2, 20.6]$$

Analysis:

The below graph depicts how $\text{Core}(p)$ depends on edge probability(p).



Observations:

- For low values of edge probability ('p') in the range 0.05 to 0.20, the core number ('Core(p)') shows a gradual increase. This suggests that with low 'p', the graph contains small k-cores, but they become more prominent as 'p' rises.
- There is a noticeable inflection point in the graph around 'p' = 0.25. Below this point, the core number remains relatively low, indicating that k-cores are not well-formed. Beyond this point, the core number starts to increase more rapidly.
- The graph exhibits a sharp increase in core number for 'p' values in the range 0.30 to 0.60. This indicates that as 'p' approaches this range, denser k-cores become more prevalent.
- For 'p' values above 0.80, the core number remains consistently high, indicating that the graph is highly dense, and almost all nodes are part of well-formed k-cores.
- The core number is highly sensitive to the 'p' range used. Small changes in the range can lead to different patterns. Adjusting the range can help focus on specific density transitions in the graph.
- The relationship between 'p' and 'Core(p)' is nonlinear. Small changes in 'p' can lead to significant changes in the core number, particularly in the transition regions.
- As 'p' increases, dense clusters of nodes, represented by k-cores, start to emerge. These clusters are highly interconnected, resulting in higher core numbers.
- The graph illustrates several transition points where the graph undergoes shifts in density. Identifying these transition points can be crucial for understanding the graph's core structure.
- Averaging over 10 runs, as done in Task 1, helps smooth out the variations due to the randomness of the graph generation process. It provides a more robust measure of the core number for each 'p'.
- These observations highlight the dynamic relationship between edge probability ('p') and the core number in random graphs. Understanding how k-core structures change with graph density can be valuable for various applications in network analysis and complex systems.

Appendix:

Source Code:

```
import networkx as nx
import random
import matplotlib.pyplot as plt

# To compute the core number of a graph
def compute_core_number(graph):
    try:
        core_numbers = nx.core_number(graph)
        if core_numbers:
            return max(core_numbers.values()) # Return the maximum core number
        else:
            return 0
    except nx.NetworkXError:
        return 0

# To generate a random graph with edge probability p
def generate_random_graph(n, p):
    G = nx.Graph()
    for u in range(n):
        for v in range(u + 1, n):
            if random.random() < p:
                G.add_edge(u, v)
    return G

# To compute the average core number for a given edge probability p
def average_core_number(p, num_runs, num_nodes):
    core_sum = 0
    for i in range(num_runs):
```

```

random_graph = generate_random_graph(num_nodes, p)
core_value = compute_core_number(random_graph)
#print(f'core values for run {i + 1}:' + str(compute_core_number(random_graph)))
core_sum += compute_core_number(random_graph)

# Highlight the k-core subgraph
k_core = nx.k_core(random_graph, k=core_value)

# Visualize the random graph with k-core subgraph highlighted
plt.figure()
pos = nx.spring_layout(random_graph)
nx.draw(random_graph, pos, with_labels=True, node_color='violet', edge_color='blue',
node_size=300, font_size=8)
nx.draw(k_core, pos, node_color='red', edge_color='red', node_size=300, font_size=8)
plt.title(f'Random Graph (p={p}), k-core (k={core_value})')
plt.show()

return core_sum / num_runs

```

```

p =[0.15, 0.5, 0.85] # Edge probability for Task-2
p = float(input("Enter the edge probability (0 < p < 1): ")) #reading the p value from user

```

```

num_runs = 10 # Number of runs
num_nodes = 25 # Number of nodes

```

```

# Task-1
#Compute the average core number for the given edge probability
average_core = average_core_number(p, num_runs, num_nodes)
print(f'Core({p}) = {average_core}')

```

```

# Task-2
average_core_list = []

```

```

n = 0
for i in p:
    average_core_list.append(average_core_number(i, num_runs, num_nodes))
    print(f"Core({i}) = {average_core_list[n]}")
    n += 1

# Task-3

import numpy as np
p_values = np.arange(0.05, 1.0, 0.05)
average_core_p_list=[]
v=0
for i in p_values:
    average_core_p_list.append(average_core_number(i, num_runs, num_nodes))
    print(f"Core({i}) = {average_core_p_list[v]}")
    v +=1

import matplotlib.pyplot as plt
print(average_core_p_list)
#Show graphically in a diagram how Core(p) depends on p.
plt.figure()
plt.plot( p_values, average_core_p_list, marker='o', linestyle='-', color='g')
plt.title('Average Core Number vs. Edge Probability (p)')
plt.xlabel('Edge Probability (p)')
plt.ylabel('Average Core Number')
plt.grid(True)
plt.show()

```

Read me:

Programming Language used: Python

Editor used: Visual Studio Code

Libraries used: NetworkX, Random, Matplotlib

To run code:

- Import the libraries (NetworkX, Random, Matplotlib) to a python(.py) file.
- Execute the file in the IDE.

References:

Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., & Vespignani, A. (2005). Large scale networks fingerprinting and visualization using the k-core decomposition. In Advances in neural information processing systems (pp. 41-50).

Dorogovtsev, S. N., Goltsev, A. V., & Mendes, J. F. (2006). k-core organization of complex networks. Physical Review Letters, 96(4), 040601.

<https://www.altcademy.com/blog/calculate-the-k-core-of-a-graph/>

https://www.sciencedirect.com/science/article/pii/S037015731930328X?casa_token=GybZar37AJwAAAAA:YGAgX1vSmxTdKws0JOX72aMBn5Zv6qc1chiF-fV09ml768dNEI6szWWez-ozgeDm3N6Rkikr

<https://i11www.iti.kit.edu/extr/publications/bggkw-acgpa-08.pdf>

<https://www.geeksforgeeks.org/find-k-cores-graph/#>

<https://scholarworks.wmich.edu/cgi/viewcontent.cgi?article=1507&context=dissertations>

<https://networkx.org/>

<https://www.educative.io/answers/networkx-library-in-python>