**Assignment 3**

**Aryaman Srivastava 2210110206**

Generate random graphs of n = 106 nodes using the following three models and compare their diameters and average clustering coefficients:

1. ER Model - g(v,p) with p = 0.5
2. Small world model
3. Kronecker model

Submit this word file with the answers filled in.

**Code for ER Model**

import networkx as nx

import numpy as np

import time

def generate\_er\_graph(n: int, sample\_size: int = 500):

start\_time = time.time()

# Generate ER graph with p = 0.5

G = nx.erdos\_renyi\_graph(n, p=0.5)

# Find largest connected component

largest\_cc = max(nx.connected\_components(G), key=len)

subgraph = G.subgraph(largest\_cc)

# Estimate diameter

nodes = list(subgraph.nodes())

max\_distance = 0

num\_samples = 100

for \_ in range(num\_samples):

source, target = np.random.choice(nodes, 2, replace=False)

try:

distance = nx.shortest\_path\_length(subgraph, source, target)

max\_distance = max(max\_distance, distance)

except nx.NetworkXNoPath:

continue

# Calculate clustering coefficient

sample\_nodes = np.random.choice(

list(G.nodes()),

min(sample\_size, G.number\_of\_nodes()),

replace=False

)

clustering = nx.average\_clustering(G, nodes=sample\_nodes)

return {

'diameter': max\_distance,

'clustering': clustering,

'time': time.time() - start\_time

}

**Code for Small world Model**

import networkx as nx

import numpy as np

import time

def generate\_small\_world\_graph(n: int, sample\_size: int = None):

start\_time = time.time()

# Generate small world graph

k = max(2, int(np.log2(n))) # Number of nearest neighbors

p = 0.1 # Rewiring probability

G = nx.watts\_strogatz\_graph(n, k, p)

# Find largest connected component

largest\_cc = max(nx.connected\_components(G), key=len)

subgraph = G.subgraph(largest\_cc)

# Estimate diameter

nodes = list(subgraph.nodes())

max\_distance = 0

num\_samples = 100

for \_ in range(num\_samples):

source, target = np.random.choice(nodes, 2, replace=False)

try:

distance = nx.shortest\_path\_length(subgraph, source, target)

max\_distance = max(max\_distance, distance)

except nx.NetworkXNoPath:

continue

# Calculate clustering coefficient with improved sampling

if sample\_size is None:

if n <= 20000:

# Use all nodes for smaller networks

sample\_nodes = list(G.nodes())

else:

# Use 20% of nodes for larger networks

sample\_size = int(n \* 0.2)

sample\_nodes = np.random.choice(

list(G.nodes()),

sample\_size,

replace=False

)

else:

sample\_nodes = np.random.choice(

list(G.nodes()),

min(sample\_size, G.number\_of\_nodes()),

replace=False

)

clustering = nx.average\_clustering(G, nodes=sample\_nodes)

return {

'diameter': max\_distance,

'clustering': clustering,

'time': time.time() - start\_time

}

**Code for Kronecker Model**

import networkx as nx

import numpy as np

import time

def generate\_kronecker\_graph(n: int, sample\_size: int = 500):

def create\_kronecker\_graph(initiator: np.ndarray, k: int) -> nx.Graph:

n = 2\*\*k

edges = []

for i in range(n):

for j in range(i+1, n):

bits\_i = format(i, f'0{k}b')

bits\_j = format(j, f'0{k}b')

prob = 1.0

for bit\_i, bit\_j in zip(bits\_i, bits\_j):

prob \*= initiator[int(bit\_i)][int(bit\_j)]

if np.random.random() < prob:

edges.append((i, j))

G = nx.Graph()

G.add\_edges\_from(edges)

return G

start\_time = time.time()

# Initiator matrix

initiator = np.array([

[0.9, 0.5],

[0.5, 0.3]

])

# Generate Kronecker graph

k = int(np.log2(n))

G = create\_kronecker\_graph(initiator, k)

# Find largest connected component

largest\_cc = max(nx.connected\_components(G), key=len)

subgraph = G.subgraph(largest\_cc)

# Estimate diameter

nodes = list(subgraph.nodes())

max\_distance = 0

num\_samples = 100

for \_ in range(num\_samples):

source, target = np.random.choice(nodes, 2, replace=False)

try:

distance = nx.shortest\_path\_length(subgraph, source, target)

max\_distance = max(max\_distance, distance)

except nx.NetworkXNoPath:

continue

# Calculate clustering coefficient

sample\_nodes = np.random.choice(

list(G.nodes()),

min(sample\_size, G.number\_of\_nodes()),

replace=False

)

clustering = nx.average\_clustering(G, nodes=sample\_nodes)

return {

'diameter': max\_distance,

'clustering': clustering,

'time': time.time() - start\_time

}

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Random graph model** | **Diameter** | **Avg. Clustering coefficient** | **Time taken to construct the graph (s)** |
| **ER Model** | **12** | **0.4705** | **7453.89** |
| **Small world Model** | **40** | **0.3515** | **3850.74** |
| **Kronecker Model** | **18** | **0.0005** | **389.42** |