CS4423-Assignment-2-Part-2

March 28, 2025

Table of Contents

1 CS4423 Assignment 2: Part 2

This is a template for your solution to the `networkx` questions on Assignment 2 (Part 2).

1.0.1 Instructions and Collaboration Policy

This is a homework assignment. You are welcome to collaborate with class-mates if you wish. Please note: * You may collaborate with at most two other people; * Each of you must submit your own copy of your work; * In Cell `[1]`, choose your own node colour in `opts`. It should not be the same as given here (`#ABCDEF`), or the same as your collaborators. For more, see https://matplotlib.org/stable/users/explain/colors/colors.html * If the question asks you to construct an example, then that example should be unique to you (and your collaborators). If copied from anybody else, all involved will score zero. * The file(s) you submit must contain a statement on the collaboration: who you collaborated with, and on what part of the assignment. * *The use of any AI tools, such as ChatGPT or DeepSeek is prohibited, and will be subject to disciplinary procedures.* * Upload your file, in either **PDF or HTML** formats, to https://universityofgalway.instructure.com/courses/31889/assignments To convert your notebook to `pdf` the easiest method maybe to first export as 'html', then open that in a browser, and then print to pdf. * Your file must include your name and ID number.

1.1 Preliminaries

1.1.1 Task 1.1: Give you name, ID, and list of collaborators

Your name goes here: Andrew Hayes

Your ID number goes here: 21321503

Place your collaboration statement here:

1.1.2 Task 1.2: Load any Python modules, and choose your own colour for nodes

[1]:
```
import networkx as nx
### Change the next line so nodes appear in your favourite colour.
opts = { "with_labels": True, "node_color": '#654321' } # show labels;␣
↪IMPORTANT: Choose your own colour here
```

Other ones that Niall used when preparing solutions. Add any you need:

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd
import math
import statistics
```

1.2 Centrality Measures

Before you do this set of tasks, it may be helpful to review the example at the end of Week 7 Part 2

Adjacency Lists.
One way of representing a graph is an as adjacency list. It has one row per node. That row starts with the node label, followed by a colon, followed by a list of its neighbours. For an undirected graph, one does not list an edge twice.

Consider the following list, for a graph, G_1, on the nodes $\{1, 2, 3, \ldots, 10\}$: 1: 2 3 4 6 7 2: 3 3: 4 4: 5 8 5: 6 6: 7 7: 8: 9 10 9: 10:

So, in the adjacency list for G_1, no neighbours of Node 7 are listed, because the associated edges are already accounted for in the neighbour lists on Nodes 1 and 7.

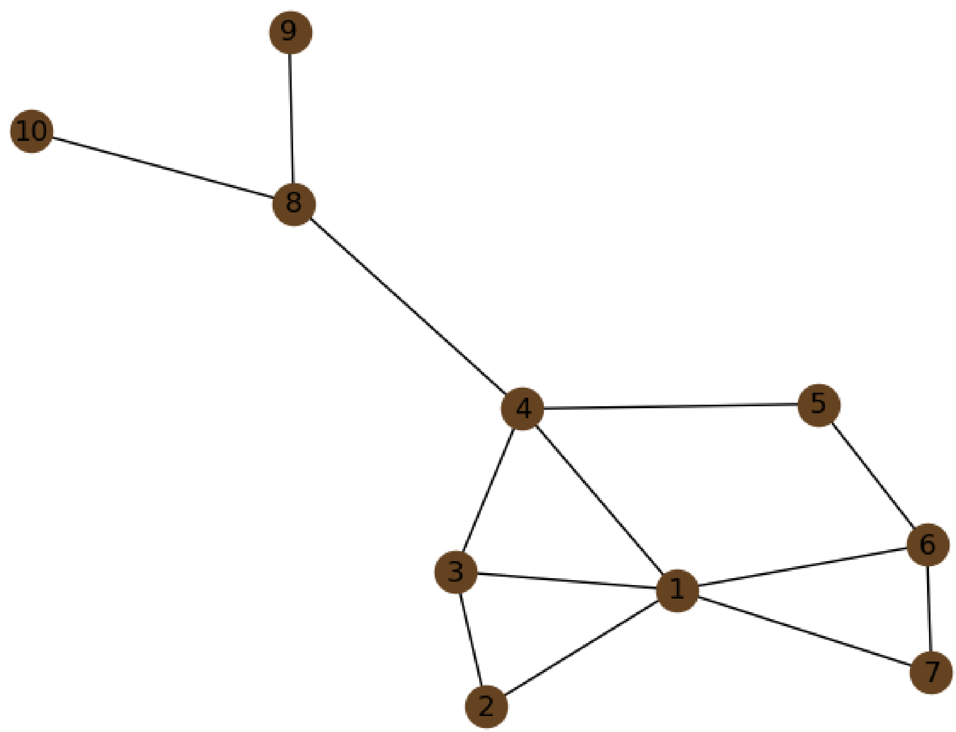
1.2.1 TASK 2.1: Define G_1 in `networkx` and draw it.

Let G_1 be the network prescribed by the adjacency list above. Define it as a `networkx` network, and draw it.

[3]:
```
G1 = nx.Graph()
```

```
edges = [
    (1, 2), (1, 3), (1, 4), (1, 6), (1, 7),
    (2, 3),
    (3, 4),
    (4, 5), (4, 8),
    (5, 6),
    (6, 7),
    (8, 9), (8, 10)
]

G1.add_edges_from(edges)
nx.draw(G1, **opts)
```



1.2.2 TASK 2.2: Compute Centralities

Compute the Degree, Eigenvector, Closeness, and Betweenness Centralities of the nodes in this graph. Display all four in a table (using a pandas `DataFrame`) sorted by the eigenvector centrality.

[4]:
```
degree_centrality = nx.degree_centrality(G1)
eigenvector_centrality = nx.eigenvector_centrality(G1)
betweenness_centrality = nx.betweenness_centrality(G1)
```

```
closeness_centrality = nx.closeness_centrality(G1)

centrality_df = pd.DataFrame({
    "Degree": degree_centrality,
    "Eigenvector": eigenvector_centrality,
    "Closeness": closeness_centrality,
    "Betweenness": betweenness_centrality
})

centrality_df_sorted = centrality_df.sort_values(by="Eigenvector",␣
 ↪ascending=False)
```

[5]:

```
centrality_df_sorted
```

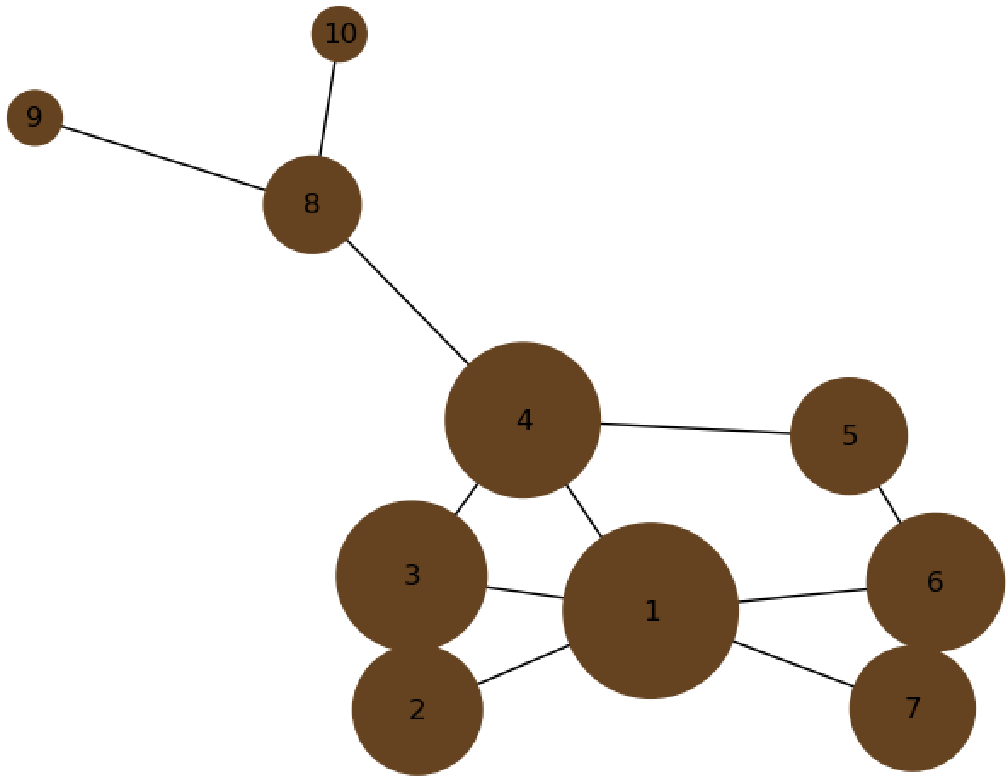
[5]:

	Degree	Eigenvector	Closeness	Betweenness
1	0.555556	0.544133	0.600000	0.351852
4	0.444444	0.424131	0.642857	0.560185
3	0.333333	0.398195	0.529412	0.064815
6	0.333333	0.333310	0.450000	0.050926
2	0.222222	0.296646	0.409091	0.000000
7	0.222222	0.276219	0.428571	0.000000
5	0.222222	0.238443	0.473684	0.055556
8	0.333333	0.166524	0.500000	0.416667
9	0.111111	0.052423	0.346154	0.000000
10	0.111111	0.052423	0.346154	0.000000

1.2.3 TASK 2.3: Draw the graph with node size proportional to eigenvector centrality

[6]:

```
node_sizes = [10000 * eigenvector_centrality[node] for node in G1.nodes()] #␣
 ↪multiplying by 10000 because the nodes aren't visible otherwise
nx.draw(G1, **opts, node_size=node_sizes)
```



1.2.4 TASK 2.4: Make your own example

If TASK 2.3 went as intended, you should find that, for G_1, Node **4** had both the greatest closeness and betweenness centrality. Make up an example of a graph, G_2, which the node with the greatest closeness centrality is different from the one with the greatest betweenness centrality. Define and draw the graph in `networkx`, and compute the centralities to verify your result.

WARNING You have to construct this example yourself. Do not try to use the same graph as anyone other than a listed collaborator.

[7]:
```
G2 = nx.Graph()

edges = [
    # define two separate star graphs, wherein the highest betweenness will be␣
 ↪the central node
    (0,1), (0,2), (0,3), (0,5),
    (7,8), (7,9), (7,10), (7,11),

    # create a bridge node between the two star graphs, thus having the highest␣
 ↪betweenness
    (12,7), (12,0)
```

```
]

G2.add_edges_from(edges)
```

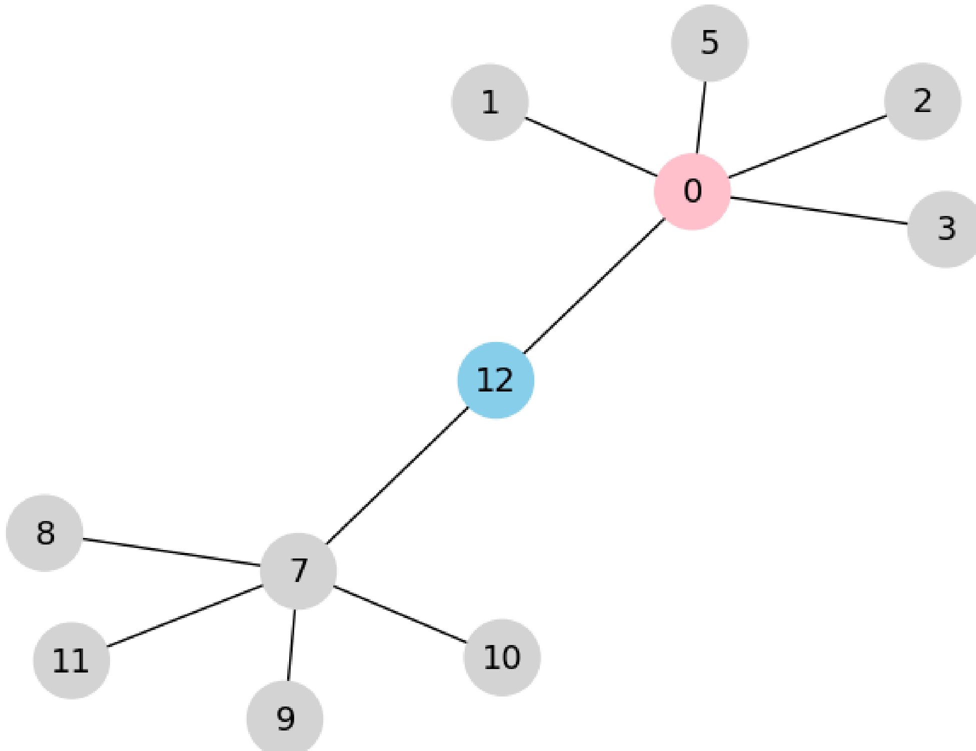
Don't remove the following cell. It will plot your network, G2, and identify the nodes with greatest betweenness and closeness centralities.

[8]:
```
### Do not delete this cell. It has an overly-elaborate way of showing the
↪difference in centralities.
CC = nx.closeness_centrality(G2)
BC = nx.betweenness_centrality(G2)

max_betweenness = max(BC, key=BC.get)
max_closeness = max(CC, key=CC.get)
print(f"Node {max_betweenness} has the greatest betweenness, and Node
↪{max_closeness} has the max closeness")
# Node colors: highlight key nodes
node_colors = []
for node in G2.nodes():
    if node == max_betweenness:
        node_colors.append("pink")  # Highest betweenness
    elif node == max_closeness:
        node_colors.append("skyblue")  # Highest closeness
    else:
        node_colors.append("lightgray")

nx.draw(G2, with_labels=True, node_color=node_colors, edge_color="black",
↪node_size=1000, font_size=14)
```

```
Node 0 has the greatest betweenness, and Node 12 has the max closeness
```



1.3 Random Networks

We'll learn in class that one way in which ER models don't reflect some real-world networks in that they tend to have fewer triangles (3-cycles) than real-world graphs. Here a *triangle* in G means a subgraph that is isomorphic to C_3. So, for example, C_3 has 1 triangle, and the Wheel Graph, W_n has $n-1$.

One way to count all the triangles in a graph is as folllows: 1. Compute the adjacency matrix, A, of G 2. Compute $B = A^3$. Note that b_{ij} is twice the number of paths of length 3 from i to j. And, in particular, b_{ii} is the number of 3-cycles involving Node i. Note: b_{ii} double counts the number of triangles involving i because, if $i \to j \to k \to i$ is a 3-cycle, so too is $i \to k \to j \to i$. 3. Compute the *trace* of B (i.e., the sum the diagonal entries in B), and divide by 6 to calculate the number of 3-cycles.

Asides: * It is not a homework question, but work out why you should divide the trace of B by 6. * Well done: you've just come up with a proof that the trace of the cube of any $0-1$ matrix is divisible by 6.

1.3.1 TASK 3.1: Count Triangles

Write a function (**with some sensible name of your own choosing**) that takes as its input a graph, and returns the total number of triangles in G. Tip: `np.trace()` returns the trace of a 2D numpy array.

[25]:
```
def num_triangles(G):
    A = nx.adjacency_matrix(G).todense()
    A_bin = (A != 0).astype(int) # handle weighted edges

    B = np.linalg.matrix_power(A_bin, 3)
    trace = np.trace(B)
    return int(trace / 6)
```

Verify that your function works by checking that, e.g., the graph returned by `nx.wheel_graph(5)` has 4 triangles.

[26]:
```
num_triangles(nx.wheel_graph(5))
```

[26]:
```
4
```

1.3.2 TASK 3.2: Comparing $G_{ER}(n, m)$ with graphs from social science

`networkx` comes with some generators from graphs that are much-studied in the network science. In Week7: Part 2 we considered the *Florentine Families* graph, which is generate by `nx.florentine_families_graph()`. There are others such as * The Karate Club Graph which is generated using `nx.karate_club_graph()` * The (Les Miserables network)[https://networkx.org/documentation/stable/reference/generated/networkx.generators.social.les_miserables generated by `nx.les_miserables_graph()`

For each of the three networks mentioned above: * Generate the graph, and output the number of order and size of the network, and the number of triangles it has. * Use `nx.gnm_random_graph()` to make a graph drawn from $G_{ER}(n, m)$ that has the same size and order. Output how many triangles it has.

[27]:
```
florentine = nx.florentine_families_graph()
order = florentine.number_of_nodes()
size = florentine.number_of_edges()
triangles = num_triangles(florentine)

print("Florentine order: " + str(order))
print("Florentine size: " + str(size))
print("Florentine number of triangles: " + str(triangles))

ger = nx.gnm_random_graph(order, size)
print("GER number of triangles: " + str(num_triangles(ger)))
```

```
Florentine order: 15
Florentine size: 20
Florentine number of triangles: 3
GER number of triangles: 4
```

[28]:
```python
karate = nx.karate_club_graph()
order = karate.number_of_nodes()
size = karate.number_of_edges()
triangles = num_triangles(karate)

print("Karate Club order: " + str(order))
print("Karate Club size: " + str(size))
print("Karate Club number of triangles: " + str(triangles))

ger = nx.gnm_random_graph(order, size)
print("GER number of triangles: " + str(num_triangles(ger)))
```

```
Karate Club order: 34
Karate Club size: 78
Karate Club number of triangles: 45
GER number of triangles: 21
```

[30]:
```python
mis = nx.les_miserables_graph()
order = mis.number_of_nodes()
size = mis.number_of_edges()
triangles = num_triangles(mis)

print("Les Miserables order: " + str(order))
print("Les Miserables size: " + str(size))
print("Les Miserables number of triangles: " + str(triangles))

ger = nx.gnm_random_graph(order, size)
print("GER number of triangles: " + str(num_triangles(ger)))
```

```
Les Miserables order: 77
Les Miserables size: 254
Les Miserables number of triangles: 467
GER number of triangles: 45
```

1.4 Extras

The following isn't part of the assignment, but you might find it interesting: 1. Use `np.linspace(0,1,100)` to create an array of probabilities. 2. For $n = 100$ make a $G_{ER}(n, p)$ graph with the values of p drawn from above, and count the number of triangles. Call this $T(G)$. 3. I conjecture that $T(G)/m(G) \approx Cp^2$, for some constant C that depends on n. Try to produce a plot that supports (or refutes) this conjecture, and try to estimate C