Solutions to Assignment 1
Network Science Data II
PHYS 7332, Fall 2024

Mostafa Alkady*

Question 1.

Short description of the four types of network data:

- **.csv:** CSV files are plain text files where the values in each row are separated by commas, and each row is ended by a line break.
- **pickle:** is a data serialization format that allows for the conversion of data structures into byte stream and vice versa. Often useful for preserving the state of the objects or exchanging python objects between programs.
- **.gml:** Short for Graph Modeling Language. GML files are stored using a 7-bit ASCII encoding (which is the standard English letters, digits and common punctuation marks) with any extended ASCII characters (e.g. iso8859-1, which includes letters such as: é) appearing as HTML character entities (e.g. `&#233;` for the case of é).
- **Elasticsearch:** is a distributed search engine that can store and index large amounts of data, uses Graph API feature to visualize data in graph structures and discover connections between nodes.

1. Advantages:
   - **.csv:**
     - readable, editable and viewable in any text editor.
     - platform-independent, and is supported by most software applications.
     - faster in parsing due to its simplicity and usually smaller in size than other data types.
     -
   - **pickle:**
     - supports complex data structures as it can serialize any python object.
     - easy to pickle and unpickle.
   - **.gml:**
     - Easy to read and edit by humans.
     - The HTML encoding ensures compatibility with systems or text parsers that may not handle extended characters natively.
     - widely supported by many popular network tools such as Networkx and Gephi.
     - portability, simple syntax, extensibility and flexibility.

---

* GitHub Repo: https://github.com/MostafaAlkady/student-network-science-data-book

- **Elasticsearch:**
  - It can handle massive amounts of data, as well as finding connections between nodes based on how it's linked in elasticsearch.s

2. Disadvantages:

   - **.csv:**
     - inconsistently formatted, which can cause issues with parsing or data import.
     - not ideal for complex data structures, such as nested or hierarchical data.
   - **pickle:**
     - not secure as will be explained in part 3 below.
     - it's python-specific and cannot be processed by systems written in different languages.
     - pickle files are not as optimized as other serializers in terms of file size and speed of serialization and deserialization.
   - **.gml:**
     - limited character support since it only supports basic ascii characters.
   - **Elasticsearch:**
     - optimized for search operations, not graph operations.

3. The **pickle** data type can pose a significant security issue. As explained above, the pickling process converts python objects or data structures into streams of bytes. The unpickling process runs without restrictions, and does not validate the data in any way. Therefore, it can execute any malicious inputs and harm our data/device. It's highly recommended to only unpickle data from trusted sources, or use a safer choice such as JSON.

4. - **.csv:** ideal for simple tabular data.
   - **pickle:** ideal for data that involves complex python data structures, where one needs to save and load the state of an analysis mid process.
   - **.gml:** ideal for complex graph structures that include metadata for both nodes and edges (e.g. a social network with node attributes such as age and gender, and/or edge weights..etc).
   - **Elasticsearch:** ideal for massive data that needs real-time analysis.

Question 2.

a) A few scenarios with no allowable edge exchange:

- A complete graph: in which case line 9 will always be true.
- A complete bipartite graph.
- A graph with number of nodes $n < 4$.
- A graph with number of edges $|E| < 2$.

b) The algorithm is modified below to avoid the infinite loop. Modifications to the code are colored in red. The idea is to count all failed attempts, and break the while loop when it reaches a given value.

```
1:  procedure EDGE_EXCHANGE(G, t, max_attempts)
2:      s ← 0
3:      count ← 0
4:      while s < t do
5:          if count = max_attempts then
6:              Break
7:          end if
8:                  (i, j) ← get_random_edge(G)
9:                  (u, v) ← get_random_edge(G)
10:                 if i = u or i = v or j = u or j = v then
11:                     count ← count + 1
12:                     continue
13:                 end if
14:                 if has_edge(G, i, v) or has_edge(G, j, u) then
15:                     count ← count + 1
16:                     continue
17:                 end if
18:                 remove_edge(G, i, j)
19:                 remove_edge(G, u, v)
20:                 add_edge(G, i, v)
21:                 add_edge(G, j, u)
22:                 s ← s + 1
23:             end while
24:             return G
25:         end procedure
```

c) Modified below in cyan to preserve the overall connectivity of the graph.

```
1:  procedure EDGE_EXCHANGE(G, t)
2:      s ← 0
3:      C = number_of_components(G)
4:      while s < t do
5:          (i, j) ← get_random_edge(G)
6:          (u, v) ← get_random_edge(G)
7:          if i = u or i = v or j = u or j = v then
8:              continue
9:          end if
10:         if has_edge(G, i, v) or has_edge(G, j, u) then
11:             continue
12:         end if
13:         remove_edge(G, i, j)
14:         remove_edge(G, u, v)
15:         add_edge(G, i, v)
```

```
        add_edge(G, j, u)
        if number_of_components(G) != C then
            add_edge(G, i, j)
            add_edge(G, u, v)
            remove_edge(G, i, v)
            remove_edge(G, j, u)
            continue
        end if
            s ← s + 1
        end while
        return G
    end procedure
```

The additional constraint does not make the edge exchange more difficult. However, it increases the running time of the algorithm significantly since it will check the connectivity of the graph (at best $\mathcal{O}(V + E)$) after each edge exchange, accepting the changes if they maintain the connectivity, and rejecting if they change it. [1]

d) Modified below in purple to preserve the correlations on the edges of the graph.

```
procedure EDGE_EXCHANGE(G, t)
    s ← 0
    while s < t do
        (i, j) ← get_random_edge(G)
        (u, v) ← get_random_edge(G)
        if i = u or i = v or j = u or j = v then
            continue
        end if
        if has_edge(G, i, v) or has_edge(G, j, u) then
            continue
        end if
        if x_j = x_v and x_i = x_u then
            remove_edge(G, i, j)
            remove_edge(G, u, v)
            add_edge(G, i, v)
            add_edge(G, j, u)
        else
            continue
        end if
        s ← s + 1
    end while
    return G
end procedure
```

[1] An important question here is the following: Will my algorithm access all configurations of the graph with the same degree sequence and number of components? In other words, is there any configuration that can only be reached through a series of edge swaps that will inevitably change the connectivity during the process? No idea.

Same comment as the previous part. Except the increase in the running time will not be as significant as before because the complexity of the extra condition is only $\mathcal{O}(1)$.

Question 3. (Check the attached Jupyter Notebook!)

Question 4.

a) Check the attached Jupyter notebook!

b) Check the attached Jupyter notebook!

c) The anthropology department was found to have the most prerequisites per course. This is indeed surprising. However, after inspection, it was found that the anthropology department lists many courses as alternatives in their webpage. And our implementation of the network treats all of these courses as equal prerequisites.

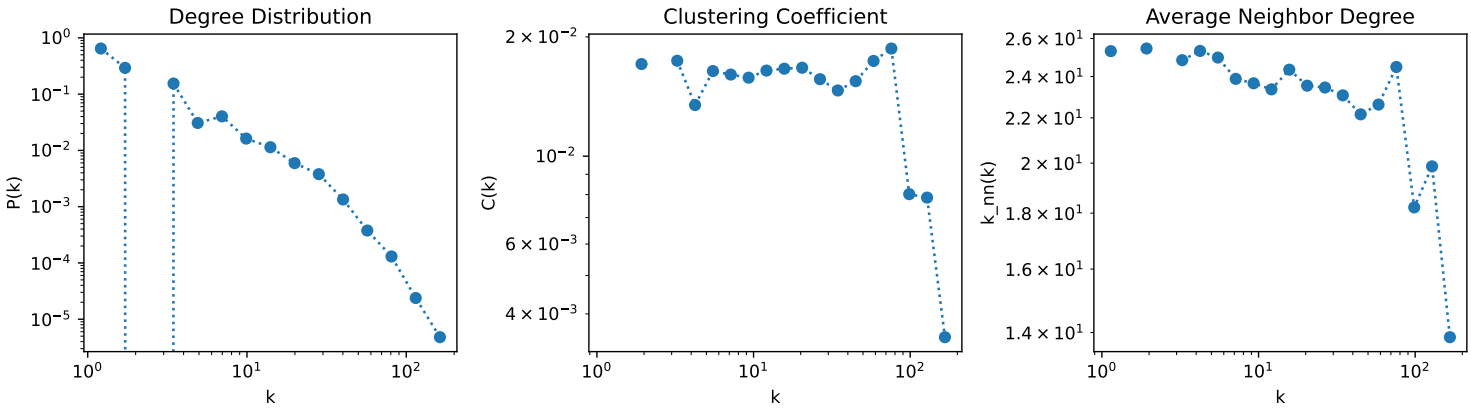
d) The longest chain of prerequisites we can find is the following:
`['BIOL 1107', 'BIOL 1113', 'PHSC 2302', 'PHSC 2303', 'HSCI 1105', 'NRSG 2220', 'NRSG 3323', 'NRSG 3320', 'NRSG 3420', 'NRSG 4502', 'NRSG 4995']`

e) Answering research questions requires identifying the relevant subsets of the data to construct the correct network that helps answer the questions. For example, the NEU course descriptions webpage contains the data of all courses which include: description, prerequisites, corequisities (if applicable), and attributes. Questions such as: What's the longest chain of prerequisites? What's the course with the most number of prerequisites? .. etc, require having a network of courses connected together if they are prerequisites. However, questions such as: what words are most present in the description of a certain department requires a co-occurence matrix and is not related to the previously described network.

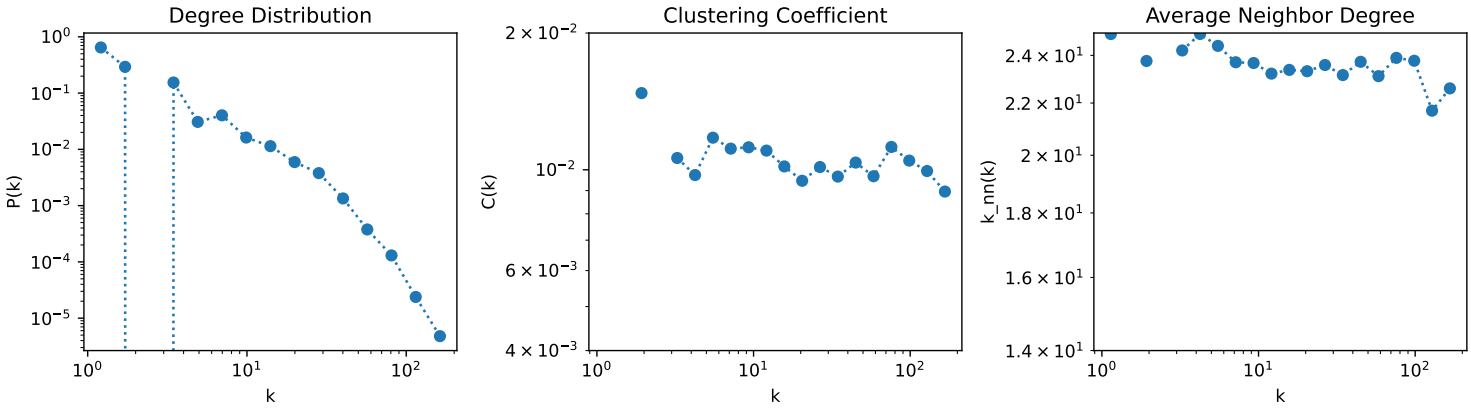
Question 5.

a) Bio-dmela: a biological network, where nodes represent proteins, and links represent protein-protein interactions. [1]

b) The following three distributions are plotted for **the original bio-dmela network**: Degree distribution $P(k)$, clustering coefficient $c(k)$, and average neighbor degree $k_{NN}(k)$, all log binned.



The same distributions are now plotted for the **degree-preserving randomized version of the network**:



c) It's observed that the degree distribution is the same for both networks (as expected), while the clustering coefficient and the average neighbor degree behave differently. In the original network, the hubs have lower clustering and average neighbor degree than other nodes. While in the randomized version, the hubs have almost as much clustering and average neighbor degree as all other nodes. This might suggest that hubs serve as a connector for different parts of the network as opposed to being part of a densely connected group.

Comparing our results with a degree-preserving randomized version of the network allows us to determine whether the network properties are due to inherent factors that have to do with the wiring diagram, or if they could be explained through chance and randomness and therefore not very significant.

Question 6.

Feedback on the assignment:

- Q3: part (d): I was confused about what this test function should output exactly. An accompanying quick example of such a function would have been a great help to resolve any confusion.
- Q4: part(e): if I understand it correctly, then the answer seems straightforward and doesn't require extensive thinking.
- Q4: In part (a), we wrote a function to obtain **links** for the prerequisites. In the following part, I needed the a similar function that return the **titles** of the prerequisites instead of the links. Therefore the former function was not used at any point during the exercise.
- Q3: In part (a), we wrote a function that takes the node as input and returns its centrality. We were then required to compare with nx.centrality, which returns dictionaries instead of a number. So I had to make another version of my function that returns a dict for ease of comparison.
- The last two points are not very important in themselves but they serve to make it easy for students to have a **flow** to the question and build on previous parts piece by piece. This provides more clarity and eliminates any ambiguity students can have.

- Some terms needed to be defined exactly to avoid confusion, e.g. connectivity (Q2,Q3), robustness (Q3).

This is my estimated time it took me for each question for your reference:

- Q1: ~ 5 hours.
- Q2: ~ 2 hours.
- Q3: ~ 7 hours.
- Q4: ~ 6 hours.
- Q5: ~ 4 hours.

Feedback on the course:

- I suggest to have a separate class for getting to know GitHub and Linux terminal well. These are very essential tools and I feel like I'm still not comfortable with them as I need to be.
- Juypter notebooks are well written and easy to follow. No complaints!
- The coding exercises in class is a great idea, but I suggest also posting the correct codes after the lecture for students who couldn't figure it out during class.
- A five-minute break in the middle of the session would help a lot, and we can take extra five minutes for the class if everyone is okay with this.
- Finally, I learned so much so far and I appreciate your efforts in making this class. Thank you!

---

[1] R. Singh, J. Xu, and B. Berger, PNAS **105**, 12763 (2008).