Application #1 Description

In the Module 1 Application, we will combine the mathematical analysis that we began in the Homework with the code that you have written in the Project to analyze a real-world problem: How do scientific papers get cited? This part of the module will probably be much more unstructured than you are accustomed to in an on-line class. Our goal is to provide a more realistic simulation of how the concepts that you are learning are actually used in practice. Your key task in this part of the module is to **think** about the problem at hand as you answer each question.

As part of this portion of the module, you'll need to write code that processes medium-sized datasets. You are welcome to use either desktop Python or CodeSkulptor when writing this code. To process the data in CodeSkulptor, you will need to be careful in how you implement your code and will probably need to increase the default timeout from 5 secs to around 20-30 secs. You can reset the timeout using:

```
import codeskulptor
codeskulptor.set_timeout(20)

```

Citation graphs

Our task for this application is to analyze the structure of graphs generated by citation patterns from scientific papers. Each scientific paper cites many other papers, say 20-40, and sometimes (e.g., review papers) hundreds of other papers. But, let's face it: It is often the case that the authors of a paper are superficially familiar with some (many?) of the papers they cite. So, the question is: Are the cited papers chosen randomly (from within the domain of the paper) or is there some "hidden pattern"?

Given that we will be looking at "paper i cites paper j" relationships, it makes sense to represent the citation data as a **directed graph** (a citation graph) in which the nodes correspond to papers, and there is an edge from node *i* to node *j* if the paper corresponding to node i cites the paper corresponding to node j. Since we're interested in understanding how papers get cited, we will analyze the in-degree distribution of a specific graph, and contrast it to those of graphs generated by two different random processes.

Question 1 (4 pts)

For this question, your task is to load a provided citation graph for 27,770 high energy physics theory papers. This graph has 352,768 edges. You should use the following code to load the citation graph as a dictionary. In CodeSkulptor, loading the graph should take 5-10 seconds. (For an extra challenge, you are welcome to write your own function to create the citation graph by parsing this text representation of the citation graph.)

Your task for this question is to compute the in-degree distribution for this citation graph. Once you have computed this distribution, you should normalize the distribution (make the values in the dictionary sum to one) and then compute a log/log plot of the **points** in this normalized distribution. How you create this point plot is up to you. You are welcome to use a package such as `matplotlib` for desktop Python, use the `simpleplot` module in CodeSkulptor, or use any other method that you wish. This class page on "Creating, formatting, and comparing plots" gives an overview of some of the options that we recommend for creating plots.

Since `simpleplot` does not support direct log/log plotting, you may simulate log/log plotting as shown in this example from the PoC video on "Plotting data". However, be sure to include an indication on the labels for the horizontal and vertical axes that you are plotting the log of the values and note the base that you are using. (Nodes with in-degree zero can be ignored when computing the log/log plot since $log(0) = -\infty$.)

Once you have created your plot, upload your plot into the peer assessment as a file. Please review the class guidelines for formatting and comparing plots on the "Creating, formatting, and comparing plots" class page. These guidelines cover the basics of good formatting practices for plots. Your plot will be assessed based on the answers to the following three questions:

- Does the plot follow the formatting guidelines for plots?
- Is the plot that of a normalized distribution on a log/log scale?
- Is the content of the plot correct?

Question 2 (3 pts)

In Homework 1, you saw Algorithm **ER** for generating random graphs and reasoned analytically about the properties of the ER graphs it generates. Consider the simple modification of the algorithm to generate random directed graphs: For every ordered pair of distinct nodes (i, j), the modified algorithm adds the directed edge from i to j with probability p.

For this question, your task is to consider the shape of the in-degree distribution for an ER graph and compare its shape to that of the physics citation graph. In the homework, we considered the probability of a specific in-degree, k, for a single node. Now, we are interested in the in-degree distribution for the entire ER graph. To determine the shape of this distribution, you are welcome to compute several examples of in-degree distributions or determine the shape mathematically.

Once you have determined the shape of the in-degree distributions for ER graphs, compare the shape of this distribution to the shape of the in-degree distribution for the citation graph. When answering this question, make sure to address the following points:

- Is the expected value of the in-degree the same for every node in an ER graph? Please answer yes or no and include a short explanation for your answer.
- What does the in-degree distribution for an ER graph look like? Provide a short written description of the shape of the distribution.
- Does the shape of the in-degree distribution plot for ER look similar to the shape of the in-degree distribution for the citation graph? Provide a short explanation of the similarities or differences. Focus on comparing the shape of the two plots as discussed in the class page on "Creating, formatting, and comparing plots".

Question 3 (2 pts)

We next consider a different process for generating synthetic directed graphs. In this process, a random directed graph is generated iteratively, where in each iteration a new node is created, added to the graph, and connected to a subset of the existing nodes. This subset is chosen based on the in-degrees of the existing nodes. More formally, to generate a random directed graph in this process, the user must specify two parameters: n, which is the final number of nodes, and m (where $m \leq n$), which is the number of existing nodes to which a new node is connected during each iteration. Notice that m is fixed throughout the procedure.

The algorithm starts by creating a complete directed graph on m nodes. (Note, you've already written the code for this part in the Project.) Then, the algorithm grows the graph by adding $n - m$ nodes, where each new node is connected to m nodes randomly chosen from the set of existing

nodes. As an existing node may be chosen more than once in an iteration, we eliminate duplicates (to avoid parallel edges); hence, the new node may be connected to fewer than m existing nodes upon its addition.

The full description of the algorithm for generating random directed graphs with this process is given below, and is called Algorithm DPA (note that the m in the input is a parameter that is specified to this algorithm, and it does not denote the total number of edges in the resulting graph). The notation $\sum_{x \in S} x$ means the "sum of all elements x in set S." For example, if $S = \{1, 7, 12\}$, then $\sum_{x \in S} x \equiv 1 + 7 + 12 = 20$.

Algorithm 3: DPA.

Input: Number of nodes n ($n \geq 1$); integer m ($1 \leq m \leq n$).
Output: A directed graph $g = (V, E)$.

1 $V \leftarrow \{0, 1, \ldots, m-1\}$; // Start a graph on m nodes
2 $E \leftarrow \{(i, j) : i, j \in V, i \neq j\}$; // Make the graph complete
3 **for** $i \leftarrow m$ **to** $n - 1$ **do**
4 $totindeg = \sum_{j \in V} indeg(j)$; // sum of the in-degrees of existing nodes
5 $V' \leftarrow \emptyset$;
6 Choose randomly m nodes from V and add them to V', where the probability of choosing node j is $(indeg(j) + 1)/(totindeg + |V|)$; // The m nodes may not be distinct; hence $|V'| \leq m$
7 $V \leftarrow V \cup \{i\}$; // new node i is added to set V
8 $E \leftarrow E \cup \{(i, j) : j \in V'\}$; // connect the new node to the randomly chosen nodes
9 **return** $g = (V, E)$;

Notice that this algorithm is more complex than the ER algorithm. As a result, reasoning about the properties of the graphs that it generates analytically is not as simple. When such a scenario arises, we can implement the algorithm, run it, produce graphs, and visually inspect their in-degree distributions. In general, this is a powerful technique: When analytical solutions to systems are very hard to derive, we can simulate the systems and generate data that can be analyzed to understand the properties of the systems.

For this question, we will choose values for n and m that yield a DPA graph whose number of nodes and edges is roughly the same to those of the citation graph. For the nodes, choosing n to be the number of nodes as the citation graph is easy. Since each step in the DPA algorithm adds m edges to the graph, a good choice for m is an integer that is close to the average out-degree of the physics citation graph.

For this question, provide numerical values for n and m that you will use in your construction of the DPA graph.

Question 4 (3 pts)

Your task for this question is to implement the DPA algorithm, compute a DPA graph using the values from Question 3, and then plot the in-degree distribution for this DPA graph. Creating an efficient implementation of the DPA algorithm from scratch is surprisingly tricky. The key issue in implementing the algorithm is to avoid iterating through every node in the graph when executing Line 6. Using a loop to implement Line 6 leads to implementations that require on the order of 30 minutes in desktop Python to create a DPA graph with 28000 nodes.

To avoid this bottleneck, you are welcome to use this provided code that implements a `DPATrial` class. The class has two methods:

- `__init__(num_nodes)`: Create a `DPATrial` object corresponding to a complete graph with `num_nodes` nodes.
- `run_trial(num_nodes)`: Runs `num_nodes` number of DPA trials (lines 4- 6). Returns a set of the nodes, computed with the correct probabilities, that are neighbors of the new node.

In the provided code, the `DPATrial` class maintains a list of node numbers that contains multiple instances of the same node number. If the number of instances of each node number is maintained in the same ratio as the desired probabilities, a call to `random.choice()` produces a random node number with the desired probability.

Using this provided code, implementing the DPA algorithm is fairly simple and leads to an efficient implementation of the algorithm. In particular, computing a DPA graph with 28000 nodes should take on the order of 10-20 seconds in CodeSkulptor. For a challenge, you are also welcome to develop your own implementation of the DPA algorithm that does not use this provided code. However, we recommend that you use desktop Python as your development environment since you are likely to encounter long running times.

Once you have created a DPA graph of the appropriate size, compute a (normalized) log/log plot of the **points** in the graph's in-degree distribution, and upload your plot into the peer assessment. (Note that you do not need to upload or machine-grade your DPA code.) Your submitted plot will be assessed based on the answers to the following three questions:

- Does the plot follow the formatting guidelines for plots?
- Is the plot a log/log plot of a normalized distribution?
- Is the content of the plot correct?

Question 5 (3 pts)

In this last problem, we will compare the in-degree distribution for the citation graph to the in-degree distribution for the DPA graph as constructed in Question 4. In particular, we will consider whether the shape of these two distributions are similar and, if they are similar, what might be the cause of the similarity.

To help you in your analysis, you should consider the following three phenomena:

- The "six degrees of separation" phenomenon,
- The "rich gets richer" phenomenon, and
- The "Hierarchical structure of networks" phenomenon.

If you're not familiar with these phenomena, you can read about them by conducting a simple Google or Wikipedia search. Your task for this problem is to consider how one of these phenomena might explain the structure of the citation graph or, alternatively, how the citations patterns follow one of these phenomena.

When answering this question, please include answers to the following:

- Is the plot of the in-degree distribution for the DPA graph similar to that of the citation graph? Provide a short explanation of the similarities or differences. Focus on the various properties of the two plots as discussed in the class page on "Creating, formatting, and comparing plots".
- Which one of the three social phenomena listed above mimics the behavior of the DPA process? Provide a short explanation for your answer.
- Could one of these phenomena explain the structure of the physics citation graph? Provide a short explanation for your answer.

Marquer comme terminé