CMS/CS/EE/IDS 144 **Gurus: Emily & Sidd**

Homework 1: Starting with a Crawl

Assigned: 01/8/26 Due: 01/15/26 5:00pm PT

Collaboration is allowed for all problems and at the top of your submission, please list all the people with whom you discussed. Also, if you used AI to help you, please write a couple sentences on how you used it. Crediting help from other classmates or AI will not take away any credit from you. The details of the collaboration and AI use policies for this course are available in the Resources tab on Piazza.

You must turn in your homework electronically via Gradescope. ***Be sure to submit your homework as a single file.***

Start early and come to office hours with your questions! We also encourage you to post your questions on Piazza, as well as answer the questions asked by others on Piazza.

1 Coding + Data Analysis [80 points]

1.1 Before you can walk you must... [55 points]

A web crawler is a program that automatically traverses and collects the web pages on the Internet. Crawlers are widely used by search engines to find and retrieve what's on the web so that they can build an index of the pages for searching. However, they're also useful for many other reasons, e.g., for gathering data, for validating the links on a web page, or for gathering everything from a particular site. You are going to implement a very primitive web crawler to explore the web pages in the Caltech domain (URLs containing ".caltech.edu") and count the number of hyperlinks each page contains.

The basic idea of crawling: A web crawler starts by retrieving a web page, parsing the HTML source and extracting all hyperlinks in it. Then, the crawler uses these hyperlinks to retrieve those pages and extracts the hyperlinks in them. This process repeats until the crawler has followed all links and downloaded all pages. Therefore, a crawler requests web pages just as a web browser does, but it browses automatically, following all the the hyperlinks in each page. Although the idea is simple, there are a few issues that come up:

1. Many web pages contain links to multimedia/data files, the crawler should not waste bandwidth downloading these files.

2. Many web sites deliver dynamic content; the web page is generated dynamically based on a query string specified in the URL. A crawler can get trapped on such a site, since there are potentially 'infinitely many' pages. For example, on an online calendar, the crawler can keep following the links of dates/months and get trapped in an endless loop.

3. Given that the crawler has extracted a long queue of links to visit, which one should be selected to visit next?

4. Given that the crawler has seen a page already, when should it go back to revisit the page and check for new and changed links? (You can ignore this for this assignment.)

There are many other issues too, such as the robot exclusion standard, Javascript in web pages, ill-formed HTML and so on.

Your task: Your task is to write a crawler that, given a URL in the Caltech domain, retrieves the web page, extracts all hyperlinks in it, counts the number of hyperlinks, follows the extracted hyperlinks to retrieve more pages in Caltech domain, and then repeats the process for each successive page. During this process you must keep updating the number of hyperlinks on a web page and the number of hyperlinks which point to that page.

We highly recommend writing this crawler in Python. We are giving you Python code to aid in retrieving and parsing the web pages. We cannot guarantee that the TAs will be able to help you with programming issues if you do not use Python 2.7 or Python 3.

You may write your crawler in any programming language you prefer, but you need to turn in a program that the TAs can test on the Caltech CS cluster. You should feel free to use third party libraries, but you must specify which ones to include and how to run your code. If you find you are having trouble with these parts of the problem, contact the TAs, since this is not meant to be the focus of the problem.

Instead, your main task is to code the traversal of the web graph. Specifically, you should design your own selection policy. (You can ignore the revisit policy for this assignment and visit each page only once.) You could do this using something similar to a breadth-first-search or depth-first-search, or something more sophisticated. A key component in this will be to ensure that you do not visit pages more than once and that you do not get trapped. (To prevent your web crawler from getting trapped you will have to deal with issues related to "Non-HTML pages" and "Dynamic pages.")
You should also save your network, as it will be necessary in problem 4c.

Important notes:

- If you search the web, you will likely be able to find source code for a crawler. We expect that you will *not* look at external source code when doing this assignment and that you will write your own crawler.

- **Ensure that your crawler stays within the Caltech domain. Do not crawl library linked services (e.g. JSTOR, IEEE Xplore, Wiley Online Library, etc.) or systematically download academic journal articles. These are violations of the terms of service.**

- **An efficient crawler can easily be mistaken for a malicious denial of service (DoS) attack. Do not do DoS attacks on the Caltech servers!** You should limit the number of parallel requests and/or the time between requests. For a discussion on web crawling etiquette, see http://en.wikipedia.org/wiki/Web_crawler#Politeness_policy.

- You should work on this problem individually. Feel free to discuss ideas with your classmates, but you should not look at any other students' code and you should write your code by yourself.

What you turn in: You should submit the following after crawling at least 2000 HTML pages in the Caltech domain starting from www.caltech.edu. Note that if a crawled page has URLs that were never actually visited by the crawler, these URLs should not be included in the network.

1. Your code, including libraries used and how to run it.

2. An explanation of the selection policy you chose and its strengths and weaknesses.

3. Two histograms. One for the number of hyperlinks per page. One for the number of hyperlinks which point to each page.

4. Two complementary cumulative distribution functions (ccdf). One for the number of hyperlinks per page. One for the number of hyperlinks which point to each page.

5. The average clustering coefficient and the overall clustering coefficient of the graph. Treat the edges as undirected for these calculations.

6. The average and maximal diameter of the graph. Treat the edges as undirected for these calculations.

7. A comparison of the degree distribution, the clustering coefficients, and the diameters with those in Problem 2. Can you describe the similarities and differences regarding the "universal" properties between collaboration network and the web graph?

   If you calculated the maximal and average diameter using a different algorithm from that used in Problem 2, be sure to describe the algorithm that you used here.

Submit (1) as part of your HW on Gradescope.

Helpful tips: Python scripts (`fetcher3.py` and `fetcher273.py`) to extract all hyperlinks in a web page specified by an URL are provided in the zip file for this homework. It is a very simple program based only on the Python standard libraries, so you just can install Python (http://www.python.org) and run it. The code is about 100 lines long, including lots of comments. Take a look at it even if you are not familiar with Python. The script performs the following tasks.

1. Use the URL to make a connection and fetch the http header. If the Content-Type is not “text/html”, return “None”.

2. Retrieve the web page content.

3. Use an html parser to parse the page and extract all hyperlinks.

4. Refine the hyperlinks, e.g., convert relative URLs to absolute URLs, and ignore the parameters of dynamic pages.

If you use Python, you can import this file and simply call the function “fetch_links(URL)” to get a list (may be empty) which contains all hyperlinks in the web page specified by the URL. It returns “None” if the URL does not point to a valid HTML page or if some error occurred. If you use another programming language, you can still use this script by making a system call to execute it and piping the output to your program. The output is a list (may be empty "[]") of hyperlinks which looks like: "[‘http://today.caltech.edu’, ‘http://www.caltech.edu/copyright/’]". The output is “None” if the URL does not point to a valid HTML page or if some error occurred. You should catch (except) any other errors that occur.

If you use Python, we highly recommend using the package `networkx` (available in both Python 2.7 and 3) to do some of the network computation and `pickle` to save the network. You’ll also want to use this package for Problem 4.

As a reference point, our solution code in Python is about 100 lines long, including comments.

1.2 Six degrees of separation [25 points]

In most real-life network graphs, we find that we can reach from almost any node to any other node within a very small number of hops (often six). This is known as six degrees of separation and was first illustrated experimentally in 1967, when Stanley Milgram performed a famous experiment where he asked a randomly selected set of 296 people from the midwest to try to forward a letter to a *target*, a stockbroker in Boston.

The participants were given some personal information about the target (including his name, occupation and address). They were asked to forward the letter to someone they knew on a first-name basis and to pass along the same instructions, so that the letter reached the target as quickly as possible. Eventually, 64 letters made it to the target, with the median number of intermediaries for these letters being around 6 and hence six degrees of separation.

The Milgram experiment shows six degrees of separation in the social network, but it has also been shown to occur in many other networks, e.g., six degrees of Kevin Bacon for actors/actresses. In this problem, you explore six degrees of separation in a few other networks.

Collaboration policy: In this problem, you are not allowed to collaborate with anyone. You should also attempt to do these searches without consulting more information about the topic. You can use other tools for this problem but please cite any tools you end up using.

Grading policy: In this problem, **you will not be graded on finding a correct path**. For each part, try playing around for a path for around 10 minutes (of course, you are welcome to explore for more time), but we don't intend this problem to be too time-consuming! If you are unable to find a path from start point to endpoint, **briefly** describe what you attempted and the results.

Please choose three of the four parts of this problem to do. You won't be given extra credit for doing all four!

(a) **Flight to the Queen Bee**

The existence of short paths between nodes is a characteristic feature of a number of networks around us. In this exercise, we will look for them in the music-map graph.

We have listed below ordered pairs of 'entities', each having an entry on music-map.com. For each pair, starting from the web-page for the first entity, you have to find a sequence of links (only to other music map pages) that will take you to the music-map web-page corresponding to the second. Write down (i) the first path that you found, and (ii) the shortest path that you found. In your submission, specify your paths precisely, i.e., write the sequence of links you used. (It is okay if the shortest path you find ends up being your first one). You can include the title of the entity to represent the page links that you used.

- Nikolai Rimsky-Korsakov $\rightarrow$ Beyoncé
- Lyle Mays $\rightarrow$ Taylor Swift

You get 1 bonus point each if your shortest path is the shortest among all the homework submissions. Note that you are *not* allowed to submit or create any music-map pages to 'create' your path!

(Lyle Mays, American jazz musician, performed at TEDxCaltech in 2011. Here's a video of the performance he gave with other musicians, in which they "explore music based on physics equations, Feynman's speech patterns and more, using improvisation, algorithmic composition, live video mixing, and a custom designed linked laptop network.")

(b) **Know your professors**

In this problem, let us consider the graph formed by co-authorship in published papers. We say person a is connected to person b with an edge if and only if there is a published paper with both a and b as authors. This graph is commonly referred to as the *co-authorship network* and has been the subject of a lot of analysis for the research community.

Your job in this task is to find paths between professors of CS at Caltech and other well-known names. The shorter the path you find, the better it is! As before, write down both the first path that you found, as well as the shortest path that you found between:

- Eric Mazumdar $\rightarrow$ Rudolf E. Kálmán
- Georgia Gkioxari $\rightarrow$ Paul Erdős

You get 1 bonus point each if your shortest path is the shortest among all the homework submissions. Include the names of the papers with the list of authors that form the path between the ordered pairs given above. Additionally include any tools you used to complete this search. Hint: Google "Erdos number".

(c) **YouTube**

In 2017, concern arose about YouTube Kids, the app that purports to show only child-friendly videos. Some investigation showed that YouTube's content filtering can be manipulated through title and thumbnail choice. For a more on this, optionally read this NYTimes article or this Medium discussion.

In this task, find a path between the given videos on *www.youtube.com*. An ordered pair of videos $a \rightarrow b$ shares a directed edge if and only if a link to b appears on the YouTube page for video a (the location of this changes depending on platform). Your task is to find a path from:

- Adam's Watson Lecture $\rightarrow$ Never Gonna Give You Up
- Cassie Goes For A Walk $\rightarrow$ What Does the Fox Say

along edges as described above. These videos are on YouTube. The path should only go through videos on YouTube and not ads from external websites. Record the date and time at which you discovered this path since the links in the graph change over time.

Note that if you are signed in to your YouTube account, or even Google Chrome, chances are the recommendations are customized to you. It is required that you use the 'Incognito mode' in Chrome, 'Private browsing' in Firefox/Safari, or 'InPrivate browsing' in IE. You might also want to turn off 'Autoplay'. You are *not* allowed to create any videos of your own to artificially 'create' a path! Please note that in order to "reset" your incognito session, you must exit **all** private browsing sessions (tabs).

You might have noticed that compared to music-map and co-authorship, it is easier to get "stuck" when clicking from video to video, and harder to get to more varied videos. Describe why this might occur more in Youtube than in the other examples we have considered.

You get 1 bonus point if your path is the shortest among all the homework submissions. Clearly list the exact names of the intermediate videos and video links in your submission.

(d) **Citations**

In this problem, let us consider the graph formed by citations in published papers. An ordered pair of papers $a \rightarrow b$ shares a directed edge if and only if paper a cites paper b. We say that paper a is connected to paper b if there is a directed edge between either a and b or b and a.

Your job in this task is to find paths between seminal papers in different fields, the shorter the better. As before, write down both the first path that you found, as well as the shortest path that you found between:

- Zipf (1936): "The psycho-biology of language" $\rightarrow$ Granovetter (1973): "The Strength of Weak Ties"
- Kalman (1960): "A New Approach to Linear Filtering and Prediction Problems" $\rightarrow$ LeCun, Bottou, Bengio, and Haffner (1998): "Gradient-Based Learning Applied to Document Recognition"

You get 1 bonus point each if your shortest path is the shortest among all the homework submissions. Include the names of the papers that form the path between the ordered pairs given above. Additionally include any tools you used to complete this search. Hint: consider using arxiv.org's LitMaps functionality, explained here. For example, you might use this starting point. There are a number of similar tools, feel free to try out different ones and report which you like best!

2 Theory [20 points]

2.1 Random Bids

You are participating in a sealed bid auction for your dream car. In sealed bid auctions, all participants submit bids without knowing the bids from other participants. We will learn more about strategies for auctions later in the course. For now, each of the N bidders (including you) choose a bid value following the continuous uniform distribution between 0 and $v, v > 0$. It follows that the bid for each participant is i.i.d. The winner of this auction is the bidder with the maximum value.

(i) What is the probability that you win the auction?

(ii) As the bidders submit their sealed bids in an envelope, an auction worker opens the sealed envelops one at a time to look at the value. The worker records the current highest bid privately in a notebook. What is the expected number of times that the worker would need to update the highest bid as they go through the N envelopes?

The first bid that will be recorded is the value in the first envelope opened, this corresponds to 1 update of the highest bid.

(iii) Does your answers to the previous 2 parts change if the distribution that the bidders sampled from were not uniform, but still i.i.d. and continuous for all bidders?

(iv) What is the expected amount of money that you will pay for participating in the auction? If you lose the auction, you will pay 0 and if you win, you will pay the amount that you have bid. As a followup, how much does your dream car have to be worth to you in order for your expected payoff from the auction to be non-negative? For this problem, if you win, you can compute the payoff as the value of the car minus the amount you pay. If you lose, the payoff is 0.

Hint: A possible approach is to find the probability density function of the amount to pay if you win through the cumulative distribution function.

2.2 The Friendship Paradox

The idea of the friendship paradox was first posed by Scott L. Field in 1991, and it's a fairly universal phenomenon across many types of social networks that has been studied broadly since then. Simply put, the friendship paradox is that people consistently have fewer friends than their neighbors empirically, but believe that they have more friends than their neighbors!

In this problem, we'll investigate this paradox mathematically.

(a) **The classic statement of the Friendship paradox.** The first (and simplest) mathematical formulation of the friendship paradox was stated as follows. Consider an undirected graph $G = (V, E)$ with $V = \{1, \ldots, n\}$. Let μ be the average degree of a node in the graph and σ^2 be the variance of the degree of a node in the graph. Then, the friendship paradox can be stated as *the average degree of neighbors* *is strictly more than the average degree of a node in the graph whenever the degree distribution has non-zero variance.* By "the average degree of a neighbor" we mean $E[d_i]$ where d_i is the degree of a node i that is selected by picking an edge uniformly at random and then picking a random endpoint of that edge.

Prove the classic statement of the friendship paradox by showing that the average degree of a neighbor is $\mu + \sigma^2/\mu$. That is, show that when the variance is nonzero, the average degree of a neighbor is larger than the average degree of a node.

(b) **Averages can be deceiving.** As we saw in class, one has to be careful when interpreting averages. The friendship paradox stated above sounds general, but is less powerful than it seems.

Give an example of a graph where the variance of degrees is non-zero, but every node has the same degree as its neighbors.

2.3 Disconcerting Diameter Definitions

The diameter of a graph is a fundamental concept, but different communities often use different ways of measuring it, which can lead to confusion. In this problem, we'll look at the contrasts between two of the most common notions of diameter: the maximal diameter and the average diameter.

All definitions of diameter depend on the notion of distance in a graph. In a connected, undirected graph G, the *distance* between two vertices is length of the shortest path connecting them. Given this, the *(maximal) diameter* of G is the greatest distance between two vertices. In contrast, the *average diameter* is the average distance between two nodes in the graph. For disconnected graphs, both the average distance and maximal diameter are ∞. While these two definitions seem similar, they can actually be very different!

Your task: Construct an unweighted graph where the maximal diameter is more than 2 times the average distance between nodes.

2.4 Understanding Clustering

We will see in class that many networks exhibit "homophily", i.e., if a vertex A is connected to vertices B and C, then it is very likely that B and C are connected as well. An example of this is that if B and C are both friends of A, then it is likely that B and C are friends too.

The notion of "clustering coefficient" attempts to provide a measure of this tendency for the formation of triangles in the network. In this exercise, we introduce two different definitions of clustering coefficient and see that, despite the fact that they appear very similar, they can yield very different results.

Consider an undirected graph G with vertices labeled $1, 2, \cdots, n$. We define the *average clustering coefficient* of G as follows.

$$Cl^{avg}(G) := \frac{\sum_{i=1}^{n} Cl_i(G)}{n},$$

where

$$Cl_i(G) := \frac{\text{number of triangles centered on vertex } i}{\text{number of triples centered on vertex } i},$$

where a triple centered at vertex i is an unordered pair of vertices that are connected to i. Intuitively, $Cl_i(G)$ is the probability that two 'friends' of i are 'friends' with each other. For vertices i with degree 0 or 1, define $Cl_i(G) = 0$.

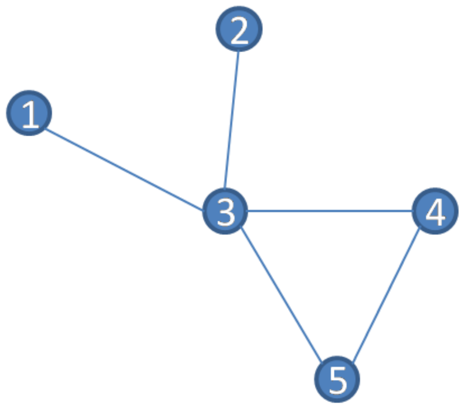

Figure 1: *Consider the graph G depicted above. Since vertex 3 has 4 neighbors, the number of triples centered at vertex 3 is 6, implying* $Cl_3(G) = 1/6$. $Cl_1(G) = Cl_2(G) = 0$, $Cl_4(G) = Cl_5(G) = 1$, *implying* $Cl^{avg}(G) = 13/30$. *The number of triangles in G is 1, node 3 contributes 6 connected tuples, nodes 4 and 5 contribute 1 each, implying* $Cl(G) = 3/8$.

We define the *overall clustering coefficient* of G as follows.

$$Cl(G) = \frac{3 \times \text{ number of triangles in the graph}}{\text{number of connected triples of vertices}},$$

where a connected triple refers to a vertex connected to an unordered pair of vertices. Note that $Cl(G)$ is the fraction of connected triples that have the third edge filled in to complete the triangle. See the example in Figure 1 for an illustration of the two definitions.

Note that for any graph, $Cl^{avg}(G), Cl(G) \in [0, 1]$, with a greater value indicating more 'clustering.' However, despite the fact that these two measures seem very similar, they are not. Your task in this exercise is to illustrate that these two definitions of the clustering coefficient can be very different. You will construct examples (families of graphs) showing that the average and overall clustering coefficients can be as different as possible.

Your task: Construct an example where, as the number of nodes in the graph becomes large, $Cl^{avg}(G_n) \to 1$ and $Cl(G_n) \to 0$. You must justify that your graph family satisfies these properties in the limit. Your graph family need not be connected.