CSC111 Final Project: Analyzing Connectivity of Wikipedia Pages to Kevin Bacon

Aidan Ryan, Lorena Buciu, Luke Kuo, Kevin Yang

Friday, April 16, 2021

1 Problem Description and Research Question

1.1 Wikipedia

Over the past 20 years, Wikipedia has steadily grown into one of the most widely accessed non-academic sources of information in the world. Surpassing the Ming Dynasty *Yongle Encyclopedia*—the previous record holder since 1408—as the largest encyclopedia to ever be compiled. One major difference that the digital age has brought in the formatting of encyclopedias is the distancing from an outwardly obvious ordering (i.e. alphabetical) as we move from pages in a book to web-pages. Wikipedia provides clear in-text links to other articles (hyperlinks called *anchors*), and as such allows for traversal through a massive variety of topics starting from just one page.

1.2 Wikiraces and the Six Degrees

The "Six Degrees of Separation" hypothesis describes the idea that every person is within six social links from any other person. In pop culture it has most notably morphed into the famous "Six Degrees of Kevin Bacon" game, the goal being to connect any Hollywood actor to Kevin Bacon through film and television appearances between linked actors. A Wikirace is another example of how this idea has been extended; a Wikipedia game where starting from a random page, the player needs to reach a target destination page in the least amount of clicks. This is done by strategically navigating through different hyperlinks found on each page. Our goal was to merge these two games together and automate it, allowing us to answer an interesting question:

Can you connect Kevin Bacon not only to any Hollywood actor in under 6 links, but also to *all of Wikipedia*?

Most people have likely at least heard of some variation of these games, and the interlinking web of articles that comprises Wikipedia would seem to translate smoothly to the concept of graphs covered in this course. Exploring these connections and analyzing them follows intuitively, and this is simply one fun way of doing so. As is described in the computation plan, some interactivity is also provided to the user in both how the data is analyzed and in visualization.

2 Dataset Description

The dataset used in this project is named `graph_data.csv`. This file was generated by using webscraping and retrieving the articles directly linked to Kevin Bacon from the Wikipedia backlinks page. Wikipedia's stance on web scraping is largely positive, with them stating that they are fully alright with scraping as long as pages aren't being accessed too quickly.

This dataset is very small, but if the scraping functions were left to run for longer while unrestricted it would allow for a comprehensive analysis of wikipedia. The small dataset is enough to succinctly demonstrate the ways the code can be used. This `.csv` file contains two columns, one for the article and another for its "anchors" (the articles that are directly linked to it). This entire file is used for the creation of the graph. The dataset is generated upon running the function, so there is no downloading required.

3 Computational Overview

3.1 Data Representation

The data used in this project consists of the connections between different Wikipedia articles. The graph data structure overlaps with the structure of Wikipedia very well, as graphs can easily take on the interconnected web shape that Wikipedia is. By modelling the data with graphs, we have the vertices of the graph representing each article web page and the edges being associated anchor hyperlinks between the vertices. This allowed us to establish the links between pages.

3.2 Computations

The graph that represents Wikipedia was built recursively. This graph needed to be modelled before any other computations/analyses/visualizations were made. First, the data was scraped from the Wikipedia backlinks page with the use of the `get_direct_links` function in data.py. In graph.py, the `make_graph` function and its recursive helper build the graph branching out from Kevin Bacon by calling `get_direct_links` on each of the vertices and creating edges in the graph.

As the goal is to find the shortest path from one random vertex in the Wikipedia graph to the other, a breadth-first search (BFS) algorithm was used. This was implemented iteratively instead of recursively to easily facilitate the recording and returning of this shortest path in a list. This implementation also handles cycles and the case where no path is found adeptly. Customizability is built into this function (and many throughout the program), allowing the end user to experiment with the final function in the `main.py` file. This is one of the main computations that allows us to analyze the connections to the target page.

3.3 Visualizations

For the visualizations there are two options for the desired graph analysis mode which can be chosen by the user:

1. **"random"**: *Use a randomly chosen vertex in the graph as the starting point to connect to the target page.*
    - This mode displays a dash webpage that contains the graph with the full connections to Kevin Bacon, but also a graph that highlights the shortest path found by the BFS algorithm when selecting a random vertex in the generated graph. There is a button that when clicked updates the graph to show the shortest path for a new randomized vertex to the target. The function responsible for the graph with the shortest path is `shortest_path` in `visualizations.py`.
2. **"full"**: *Use all vertices in the graph to analyse the widespread connectivity to the target page.*
    - This mode runs the BFS algorithm on every single vertex in the graph to comprehensively analyze connectivity. This data is represented in a bar graph, where the x axis represents the minimum number of edges required to connect the vertex to the target article, and the y axis is the amount of vertices (that required those minimum amount of edges). This bar graph is created by `connectivity_bar_graph`, which ropes in many other functions in `visualizations.py`. It has a slick colour gradient based off of the y value. One interesting feature is the ability to hover over any of the bars to see the specific wikipedia page names that are included in each bar's count.

`Dash`, `Plotly`, and `networkx` were used to produce these visualizations.

3.4 Usage of New Libraries

The most important part of being able to analyze the article connections is being able to get a list of articles that are directly connected to a target article.

This was accomplished through the use of the `requests` and `BeautifulSoup` package. Outside of Wikipedia's own database dumps, which are not only massive storage-wise but incredibely complicated to parse, the availability of useable wikipedia datasets is sparse. As such, we used web scraping to get what we needed. Wikipedia has a backlinks page that displays the articles that directly link to the target. `requests` and `BeautifulSoup` allowed us to scrape this page in order to retrieve these links by using `requests.get()` to send a GET request to the API and parsing the HTML response with `BeautifulSoup`. The links were extracted with `BeautifulSoup`'s `.findall('a')` function to get the HTML $< a >$ (link) tags. These links were then filtered by a loop to get exactly what we needed.

The `dash` library also allows for a more interactive user experience. By putting the `shortest_path` `plotly` figure into the dash framework we could increase user interactivity by having a button that updates the graph and displays the shortest path from a new randomized article on each click, allowing the user to explore more possibilities. This takes the use of `plotly` farther than in class content, allowing for a more involved and engaging user experience.

4 Setup and Operation Instructions

Running this project as intended requires the installation of multiple libraries. Instructions on how to properly organize downloaded files and install these libraries are included here.

1. Download all included files and extract them all into the same empty folder. Note that the dataset slice is generated when running the top-level These files should include:
   i. `graph.py`
   ii. `data.py`
   iii. `processing.py`
   iv. `visualizations.py`
   v. `main.py`
   vi. `requirements.txt`
   vii. `links.csv`
2. Launch PyCharm and open the project folder.
3. Navigate to the file `requirements.txt` and install any requirements or dependencies that appear after opening it.
   (a) This can be done by clicking the `'install requirements'` popup that appears at the top of the text editor when opening `requirements.txt`.
4. Open and run the file `main.py` in PyCharm.
5. Call the function `six_degrees` in the text editor, experimenting with different `depth_cap`, `analysis_type`, and `target` arguments that meet the preconditions in the documentation of the function.
   i. `depth_cap` controls the maximum amount of connections deep that the search algorithm can dig into. This is set by default to 6 due to the *'Six Degrees of Kevin Bacon'* that the project is based on.
   ii. `analysis_type` determines which of the two options for analyses will be performed on the dataset. The possible values and their meanings can be found here.
   iii. `target` controls the article that the search algorithm tries to make connections towards. By default this is set to Kevin Bacon, once again because of the original intent of the project.
6. If you either don't specify the `analysis_type` or specify it to `'random'`, you should see a message in the console that the graph is being created. Eventually, the console will provide a link to the visualizations, which you should click. Sample images of both the visualization link and final graph are included here.
   i. In the opened webpage, you can rerun the program with a new random starting point by clicking the "Update Graph" button!
7. If you specify the `analysis_type` to `'full'`, the graph should open immediately. A sample image is included here.

```
Creating Graph...
Graph has been created.
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "main" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

Figure 1: Console Output

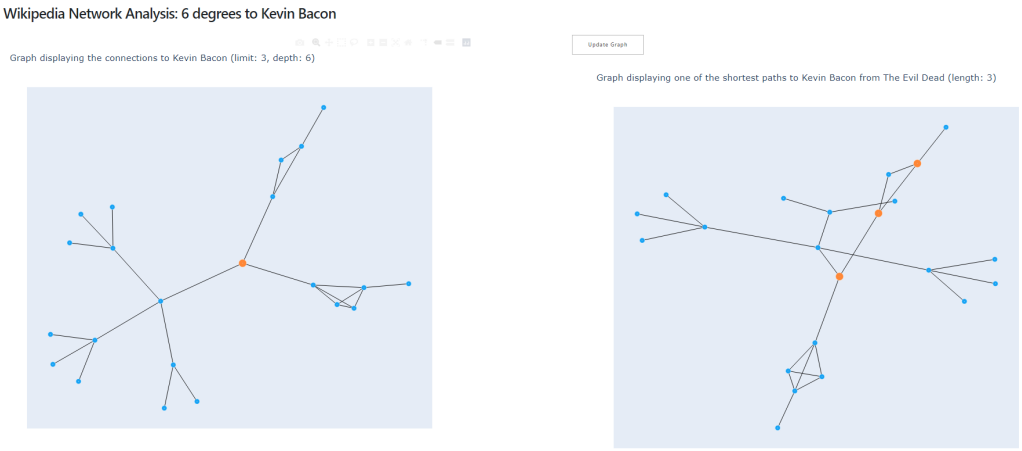

Figure 2: "Random" Setting Analysis

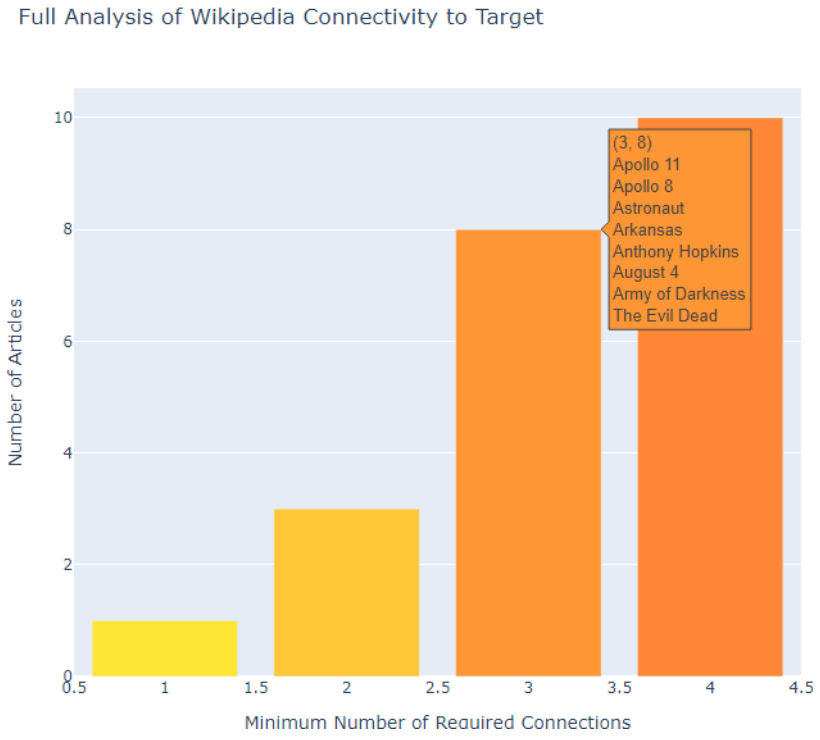

Figure 3: "Full" Setting Analysis

5 Changes

Regarding the higher-level objectives and general end-user-experience of the project, changes have been minimal. Most of the changes from the original plan come in the collection of the data, visualization, and in small minutia of how particular functions were implemented. Originally the method for collecting data had not been decided, but here we decided it would be best to use web scraping and provide a slice of what would be a much more comprehensive dataset.

There were a few changes and additions to the plan for visualization. Firstly, originally we planned to mimic the first assignment's visualization of linked lists when showing the path between the starting article and target article. Instead of this, we decided it was much more visually engaging to render the entire graph in our standard visual representation, but highlight the vertices that are part of the path with a complimentary colour. Secondly, we decided to add the re-run button for the ease of the viewer, as it is much more satisfying to click that button than constantly re-run the code in the console. The third and final change with the visualization was deciding to use hovering text boxes in the **full** analysis visuals, instead of the original idea of clicking on the bars to open a separate page. This new method looks *much* cleaner, and had a much more elegant approach in terms of code.

Finally, there are some smaller back-end changes that the user would never notice. As was expanded on further in the computation section, it was simpler to use an iterative approach for the breadth-first search algorithm, rather than the recursive implementation that I had originally planned to use. Another small change is the decision to use `Dash` for some of our visualizations. This back-end change allowed us to implement the aforementioned re-run button, as well as display multiple items on that webpage at once. Overall, the changes were largely made to improve the user-experience while not sacrificing the core goal of the project.

6 Discussion

The main source of experimentation in this project was with the data collection portion. The concept of web scraping has not been covered in the course, and the use of it in this project was both a great learning experience but also the cause of the largest issue with the results of the project. The amount of time required to scrape from Wikipedia at a comfortable rate does not fit snugly in this project's timeframe; as such, taking a small slice was not only practically the right move in terms of runtimes, but essentially a requirement. With this slice, Kevin Bacon could be connected every other page in the dataset underneath the depth cap of 6 connections. However, taking this small slice inherently invalidates the results of our project relative to the discussion question. Thus, on the code side the underlying goal of the project somewhat changed to demonstrating that what was written here could be applied to a much larger dataset. This is not a problem, though it is somewhat disappointing. The scalability of our code should be good enough, so perhaps in the future we can revisit this project and re-evaluate our method of data collection.

Outside of this point of contention, there was a massive amount of time looking into search algorithms such as the Dijkstra and Bellman-Ford algorithms. That time was wasted as these algorithms are only more effective than the much simpler Breadth-First Search in weighted graphs, whereas our graph here is unweighted. As such, a significant amount of time could have been saved just going straight to using BFS. In the future, it may be interesting to see if the efficiency of the program can be improved through some form of weighting.

There are many different ways to expand from this project, the most obvious being to simply allow the scraping to continue for longer and run the same code on a more accurately representative dataset. As we allowed for quite a bit of customization on the user's side, this project could already be used to explore connections between any start point and end point with an ideal dataset. In terms of user functionality, linking the actual Wikipedia webpages to the resulting graph to allow for easy-click access would be something obvious to implement; this would require some changes to the scraping code, but would largely not be too difficult. One interesting way that this project could be used would be to create a Wikirace game. Randomly choose a starting article and target, use the work done here to determine if they can be connected in a reasonable number of connections, and have users compete to match the algorithm's pace. There is a Wikirace game online already, but it does not implement this additional competitive aspect, so it is an interesting alteration that uses the code developed here.

In conclusion, this project was one more based in personal interest than in a social cause. It takes a wide span of concepts covered in the last year and utilizes them to explore a fun, simple what if question. While the results with the produced dataset are not that impressive, the project is a springboard for more interesting analysis and provided an opportunity to learn how to work with web scraping and large datasets.

7 References

Morse, G. (2003, February). The Science Behind Six Degrees. *Harvard Business Review.* Retrieved March 11, 2021 from https://hbr.org/2003/02/the-science-behind-six-degrees

Requests: Http for Humans. requests. Retrieved March 12, 2021 from https://requests.readthedocs.io/en/master/

Six degrees of Kevin Bacon. (2021, February 23). Retrieved March 11, 2021 from https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon

Six degrees of Wikipedia. (2021, March 04). Retrieved March 11, 2021 from https://en.wikipedia.org/wiki/Wikipedia:Six_degrees_of_Wikipedia

"Wikispeedia Navigation Paths." SNAP, Retrieved March 12, 2021 from https://snap.stanford.edu/data/wikispeedia.html

Wikirace. (2021, February 19). Retrieved March 11, 2021 from https://en.wikipedia.org/wiki/Wikipedia:Wikirace

"Writing CSV Files in Python." Programiz, Retrieved April 15, 2020 from https://www.programiz.com/python-programming/writing-csv-files