Media Library Graph Generation using IMDb, NetworkX, and Python

Zack Fravel

Portland State University
ECE508 - Spring 2021

1. Project Motivation and Objective

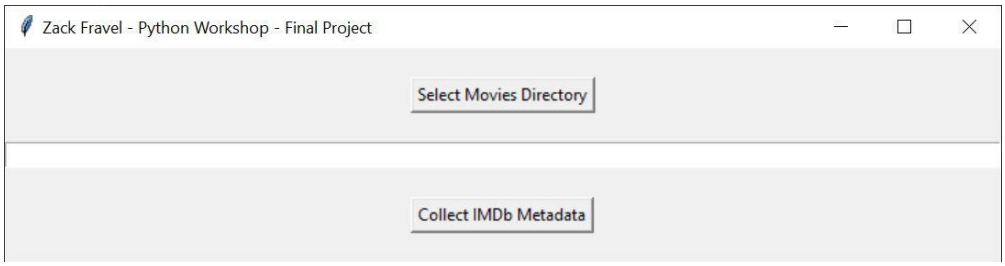
Throughout the Coronavirus pandemic one of the activities that took up a lot of my time was maintaining and setting up a consolidated media server for my friends and I (sort of like a personal netflix). In doing so I became very interested in media organization methods and thought a graph-based media organization tool would be a useful and fun Python project to complete. One could argue my inspiration for the project stems all the way back to playing the 6 degrees of kevin bacon game with my friends.

The objective of this project was to create a program that accepts a user's directory of movies (with proper formatting) and collects IMDb metadata on each movie that then can be used to generate different interesting graph structures and visualizations of the media library based on the user's desired organization filter. This project can produce graphs based on the following filters: *Year*, *Actors*, *Directors, Writers, Cinematographers, Composers, Producers, Genres,* and finally a special case*: Actors and Directors*. The project is intended to be run in a Python console environment so the user can continue to directly manipulate and analyze the graph structure using built in library functions beyond what the program initially outputs.

2. Results

Before diving into the implementation details, provided is a brief summary of the results achieved. I will go into greater detail on many aspects of the results in the following 'Design Process' section.

Overall the project was very successful in achieving the objective that I laid out above. The best way of showing the results is by outlining and showing a typical flow. When the program is first run, the user is greeted with a file selection dialogue:

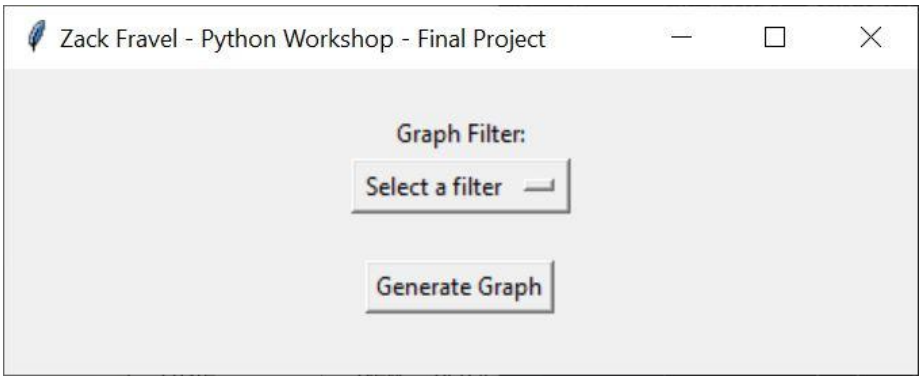


When the user clicks 'Select Movie Directory', the program provides a file explorer dialogue where the user can select their folder. The path is then filled in the text box and the user can press 'Collect IMDb Metadata' to begin scraping details from IMDb

for each movie. I have provided users with a loading screen in the python console window, as it takes about 3 seconds per movie to download all the metadata (shown below).

```
Collecting IMDb Data . . .
Folder:  C:/Users/zpfra/Documents/Graduate School/Course Materials/2021/Spring 2021/ECE 508 - Python Workshop/Project/Movies
 11%|          | 3/28 [00:11<01:38,  3.93s/it]
```

Once the program has finished creating Movie class objects and gathering IMDb metadata, the user is presented with another dialogue box:



Here is where the user can select their filter (listed in Objective) using the drop down menu. Once the user clicks 'Generate Graph', the program provides another loading screen, as some large graphs can take some time:

```
Collecting IMDb Data . . .
Folder:  C:/Users/zpfra/Documents/Graduate School/Course Materials/2021/Spring 2021/ECE 508 - Python Workshop/Project/Movies
100%|██████████| 28/28 [01:25<00:00,  3.06s/it]
Generating Graph. . .
Filter:  Actors and Directors
 54%|█████     | 15/28 [00:04<00:03,  3.50it/s]
```

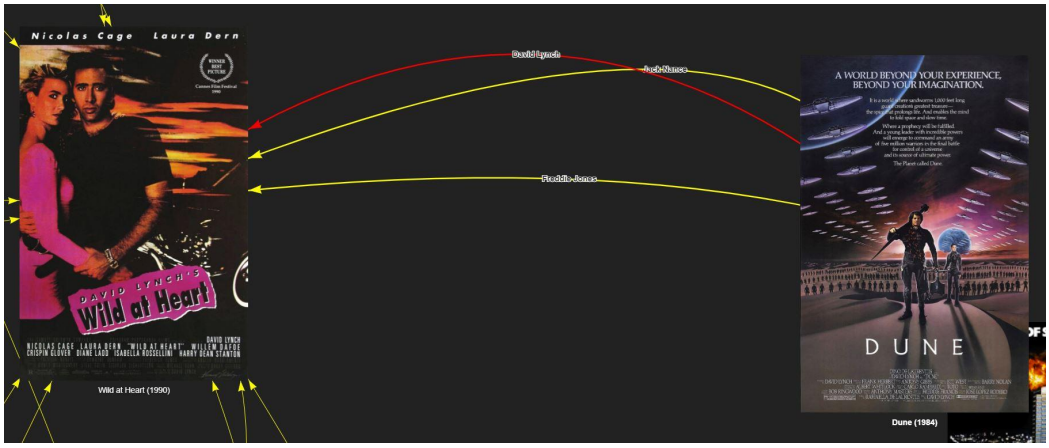
Upon completion, the program prints out graph analysis of all the nodes. Currently multigraphs (graphs with multiple edges between nodes) aren't supported for many consolidated analysis functions in NetworkX yet. Therefore analysis is performed on the node level. The screenshot on the right shows the analysis provided. Each node has a provided connectivity factor, degree centrality (fraction of nodes in graph connected to it), and closeness centrality (distance to all other nodes in the graph).

```
Fargo (1996)

    Connectivity:
        No Country for Old Men :  1
        O Brother, Where Art Thou :  2
        Prisoners :  1
        Raising Arizona :  3
        Saving Private Ryan :  1
        Schindler's List :  1
        Sicario :  2
        The Rock :  1
        The Shawshank Redemption :  1
        Wild at Heart :  3

    Degree Centrality:  0.5185185185185185

    Closeness Centrality:  0.08333333333333333
```

The most valuable and interesting output however comes in the form of an HTML file that is launched at the end of the program execution. This HTML file contains a pyvis visualization of the graph we created. The visualization shows the movie posters of each graph as the nodes with labeled edges connecting them all. The user can use their mouse to hover over each node or edge to glean more information (i.e. edge source/destination and node neighbors). The pyvis graphs have physics modeling and can be manipulated with a mouse. Below I've provided an example graph using the 'Actors and Directors' filter. Movies connected by actors use yellow edges where movies connected by directors use red edges. I find this graph especially interesting to see which actors work with which directors most frequently.

Coen Brothers Movies

David Lynch Movies



I have provided a video that shows more examples of the results in the submission.

3. Design Process

In the following sections I will lay out in detail what pre-existing code and ideas I leveraged, how my code is organized, and challenges I had to overcome as well as the total work put into the project.

a. Pre-existing code and ideas

The very start of the project involved some research on what pre-existing python libraries were out there that would be useful for this project. Thankfully, I found a fantastic collection of libraries that enabled me to complete this project in about 500 lines of code. First, and most importantly, is the IMDbPY library which is what enables everything that comes after it. IMDbPY, written by Davide Alberani and H. Turgut Uyar, provides a quick solution for searching and retrieving movies from IMDb. Another really useful library was tqdm which provided a quick and easy solution for showing a loading bar in the console window. Tkinter was used for GUI development. Finally networkX and pyvis provided a plethora of graph and visualization related functions I was able to readily access and use for my project.

b. Code Organization

- **movie.py**

Movie.py was the first file I completed for this project, it implements the Movie class that stores all the IMDb information to be used by the rest of the program. Each Movie object contains the list of actors, crew members, genres, plot summary, cover art URL, runtime, as well as the entire returned IMDb object itself. Upon initialization, the class takes a folder name and parses the name and year from it. Folders need to be organized the following way: “Movie Name (Year)”. The program then queries IMDb using the movie name and stores the top 3 results and compares the year of all the results and stores the one that matches (this is error checking for remakes, duplicate names, etc.). Once the year is matched, the movie object is retrieved and fields are set. The Movie class imports IMDbPY.

- **promptGUI.py**

promptGUI.py implements the initial GUI the user is presented with for choosing a folder. Inside there are two functions: ask_for_folder() and collect_data() that are respectively called on each button press. Tkinter has a built-in 'filedialogue' function that makes this very simple. Once the metadata starts collecting, the promptGUI is destroyed. PromptGUI.py imports Tkinter.

- **graphGUI.py**

graphGUI.py implements the secondary GUI the user is presented with after IMDb metadata is collected. Very simple drop-down menu is provided with the graph filter options laid out earlier in the report. A generate_graph() function is called when the button is clicked that destroys the GUI. GraphGUI.py imports Tkinter.

- **functions.py**

Functions.py contains the meat of the implementation. I wanted to separate some of the main program functions to make the top level program more readable. Inside this file contains create_movie_list(), count_folders(), addNodes(), addEdges(), analyzeGraph(), and visualizeGraph() functions. I organized it in this manner to make debugging much easier on myself as well as making it easier to expand in the future. Functions.py imports the Movie class, os, and NetworkX.

- **program.py**

Finally, program.py implements the top level program. Program.py imports function, followed by promptGUI. Once promptGUI has done its job, the program calls create_movie_list() from function.py to create the list of IMDb objects from folder names. Once done, the program then imports graphGUI to ensure they appear in order and variables are set between them. Once graphGUI is closed and the filter is set, the rest of program.py executes where I import networkX and pyvis and initialize their graphs by calling addNodes() and addEdges(). Once the graphs are created, the program finishes by calling analyzeGraph() and visualizeGraph(). analyzeGraph() computes interesting network properties on all the nodes using NetworkX built in functions where visualizeGraph() uses pyvis to generate an interactive graph with a Barnes-Hut physics model in HTML that opens in a web browser.

c. Challenges and Workload

There were quite a bit of challenges in getting this complete with the results I wanted. The largest roadblock was visualization and the limits put on by NetworkX and Pyvis. In pyvis, for multiple edges to show between nodes the graph *has* to be a directed multigraph. A multigraph presents challenges because networkX doesn't support multigraphs in a lot of their algorithm functions, so analysis is done only on nodes. Because of this directed limitation, if I wanted to create edges between all movies appropriately there would be 2x the amount of edges necessary (i.e. to and from instead of just one line). The result of this is when you hover over a title to see the list of neighbors, only the node earliest in the alphabetical list will list all the neighboring nodes because it's the one that created an edge to the other movie. If this were able to be an undirected (bidirectional) edge, both nodes would show each other as neighbors. This is just a limitation of the current software.

Another challenge I ran into was how to execute the code in the correct order. I kept running into issues where both GUIs would display at the same time, or one would freeze while the other showed up. I finally discovered through research that I could call a destroy() function on each instance and import them in the top level program in order so these conflicts wouldn't occur.

The IMDbPY library presented its own set of challenges as well. The first was learning how to properly search for a movie and store correct results. I kept having issues with remakes (i.e. Dune 2021 vs Dune 1984) and having incorrect movies being stored. To get around this, I store the first 3 results (very small chance it won't be in those first 3 results) and then do some error checking on which year the title was released. This ensures that the correct movies are picked. The last issue with the IMDb library is some movies don't have all the attributes in my code (i.e. Pulp Fiction doesn't have a composer, it has a soundtrack.) I didn't have time to implement this massive amount of error checking code so I just stuck with movies that had the typical fields. Documentation on the IMDb was also limited.

The time spent on this project was somewhere between 25 and 30 hours.

4. How to run

Install the following libraries using pip (refer to https://pypi.org/ for specifics):

- IMDbPY
    - Used for sending requests and receiving data from IMDb

- Tkinter
    - Used for GUI
- Tqdm
    - Used for console progress bars
- NetworkX
    - Used for creating and manipulating graphs
- Pyvis
    - Used for visualizing graphs in browser

The program as of now is designed to be run in PyCharm or a similar IDE with virtual python console environment capabilities. To run the project, build and run **program.py** in the python console with the following files in the same directory:

- functions.py
    - Contains a multitude of functions used by program.py
- movie.py
    - Contains movie class definition
- promptGUI.py
    - Implements the initial prompt GUI
- graphGUI.py
    - Implements the secondary graph filter GUI

Once the program is started, the user will be presented with a dialogue box asking for a directory path. Once selected the user clicks “Collect IMDb Metadata'', the console shows the current progress of the IMDb data collection. Once complete, a secondary GUI shows that allows the user to select a specified graph filter. Once that’s been chosen, the graph is built (progress bar shown in console for long runs) and is displayed in the user’s web browser along with analysis statistics printed to the console. The benefit of running in a console environment is the ability to play with and manipulate the structures after the code has been run (i.e. run more analysis functions, check attributes of movies or nodes in list, etc . . .)

5. Future Work

There are a lot of places this can go for future work. For one, all the error checking issues can be solved so all movies work. Second, this can be expanded to include TV Shows and Episodes. Third, as of now the program only generates one graph, I think in the future it would be great if the user had the ability to generate multiple graphs and run analysis on multiple graphs (this is why I included the ‘actors and directors’ filter). Other improvements to the GUI such as nicer aesthetics would also greatly enhance the program.