Cops and Robbers: a Graph and AI based path game.

Aamishi Avarsekar, Ashkan Aleshams, Harry Duong, Dravin Nagalingam

April 16th 2021

Problem Domain and Project Question

- **Problem Domain**
  The domain of our final project focuses on the concept of the popular game: Cops and Robbers. The core of our project is the application of Graphs which establishes a network between locations in a city and finds optimal paths between locations.

  In the original version of the game, there are robbers on the run who have to steal a treasure in the city and escape the cops without being caught. To simplify the algorithms we wish to implement, we will have only one cop and one robber. However, the goal of the robber remains the same, to steal the treasure in the city and escape.

  The city in our game is New York City which consists of several locations such as parks, cemeteries, health care facilities and hospitals, tourist spots, fire stations and police stations. In the initial state of the game, the robber would be hiding at a location where he is safe from the cops. He then asses his next move based on the type of location, and the location of the cop. The robber takes a path to the treasure that is closest to the treasure and avoids all locations where is likely to get caught. After stealing the treasure, the robber follows the same principle and attempts to escape the city. The cop has to chase the robber and catch him before he escapes the city.

  There are three possible outcomes of each game. The first outcome is where the robber is able to steal the treasure and escape the city without being caught. It is crucial that the robber steals the treasure before leaving the city. This is the second outcome of each game. If the robber is unable to steal the treasure or reach the location of the treasure, the game ends in a draw. The third outcome is the most important one. In it, the cop catches the robber, and the cop wins. This happens when the cop and the robber are in the same location at the same time.

- **Project Goal**
  Our goal for the final project is to determine how we can apply Graphs to implement the game of Cops and Robbers, as well as develop different types of AIs for cops and robbers, and compute how effective they are.

The dataset and its purpose:

The datasets that we used in our project:

1. `Areas_of_Interest_Centroids.csv`: Stores the name of parks and cemeteries in NYC.

   This dataset stores the name of the location as well as its type. The dataset has the following format: the_geom, OBJECTID, Id, Name, Stacked, AnnoLine1, AnnoLine3, Borough, AnnoLine4, AnnoLine2a, Angle. For our dataset, we only required the Name column and AnnoLine3 column which stores the name and the type of the location respectively.

   This dataset can be found at this URL:

   https://data.cityofnewyork.us/d/mzbd-kucq?category=Healthview_name=Places

2. `New_York_Tourist_Locations.csv` - 348 New York Tourist Locations:

   This dataset stores the name, the address and the zipcode of tourist locations in New York City. The dataset has the following format: name of the tourist spot, its address, and its zipcode. For our dataset, we just extracted the name of the tourist spot and categorized it as 'tourist spot' in our final dataset.

This dataset can be found at this URL:

https://www.kaggle.com/anirudhmunnangi/348-new-york-tourist-locations

3. `rows`.csv - NYC Health + Hospitals Facilities - 2011:

   This data set stores the health and hospital facilities in New York City. It contains information for faculty type, the borough it is located in, the name of the facility, its phone number, location, etc. The dataset has the following format: Facility Type, Borough, Facility Name, Cross Streets, Phone, Location 1, Postcode, Latitude, Longitude, Community Board, Council District, Census Tract, BIN, BBL, NTA. For the final dataset, we only use the column that stored the the name of the facility and manually categorized them as 'health' in our dataset. Even though the NYC Health + Hospitals Facilities dataset categorizes the faculty into further types, we used an umbrella term 'health' for all the locations.

   This dataset can be found at this URL:

   https://data.cityofnewyork.us/d/ymhw-9cz9?category=Healthview_name=NYC-Health-Hospitals-Facilities-2011

4. `FDNY_Firehouse_Listing.csv` - FDNY Firehouse Listing:

   This dataset consists of Firehouse Listings for the FDNY (Fire Department of New York). It stores the name of the facility, its address, borough, postcode etc. The dataset has the following format: FacilityName, FacilityAddress, Borough, Postcode, Latitude, Longitude, Community Board, Community Council, Census Tract, BIN, BBL, NTA. The FDNY has a format for describing the name of the fire station. Each fire station states its engine or ladder type for identification. For our dataset, we extracted the name of each fire station and categorized it as 'fire station' in our dataset.

   This dataset can be found at this URL:

   https://data.cityofnewyork.us/Public-Safety/FDNY-Firehouse-Listing/hc8x-tcnd

5. NYPD_Hate_Crimes.csv - NYPD Hate Crimes Dataset containing confirmed hate crime incidents in NYC:

   This dataset stores the information about hate crimes in New York City. It contains information such as the ID for the complaint, its date, the precinct where the complain was lodged, the borough, etc. The dataset has the following format: Full Complaint ID, Complaint Year Number, Month Number, Record Create Date, Complaint Precinct Code, Patrol Borough Name, County, Law Code Category Description, Offense Description, PD Code Description, Bias Motive Description, Offense Category, Other Motive Description, Arrest Date, Arrest Id.

   For our dataset, we extracted the column Complaint Precinct Code and categorized it as 'police station'. As there are more complaints than precincts, we decided to use the `visited` method to add those precincts in our dataset, that were previously not added.

   This dataset can be found at this URL:

   https://data.cityofnewyork.us/Public-Safety/NYPD-Hate-Crimes/bqiq-cu78

Computational Overview

Dataset generation:

We have designed our game algorithm in such a way that the cop and robber assess their next move based on the score given to each type of location. Each location in the data set is associated with a score from 1 to 10. This score that is associated with the type of location, is crucial to our algorithm. Locations with a score closer to 1 are considered good for the robber, whereas locations closer to 10 are where the cop would most likely be. The robber will have a tendency to only traverse to locations with a lower score.So, we have generated our data set `large_location_data.csv` based on this requirement.

To accumulate our the data set, we had to first find appropriate data sets that stored the locations in the city, along with their name and their type. We were able to find several data sets that stored different types of locations. For example, `New_York_Tourist_Locations` stores the name of a tourist location in New York City, its address and zipcode. For our data set, we only require the name of the location, and as we already know that `New_York_Tourist_Locations` is data set for tourist spots in New York City, we just added the type 'tourist spots' manually in our data set.

Similarly, all the other data sets that we found contained information that was not required by our project. So

to extract only the names of the location, we identified the name column in the data set, and added it to our data set csv file.

Our final data set, `large_location_data.csv` has the following format: name of the location and its type. Each row in the data set has only two entries as described above. `large_location_data.csv` was made using the `data_set_builder.py` file. If you want to test the reproducability of `data_set_builder.py`, kindly folloe the instructions in the `data_set_builder.py` file. This should produce another folder in the folder where `data_set_builder.py` is stored.

Using Graph to store data:

Graphs play a vital role in our program. As an overview, our game takes in a data set of locations in the city of New York and creates a graph where the locations in the data set are vertices of the graph. It then generates random edges between the graph's vertices. There are two different types the of player that we have chosen to implement which are based on the concept of graph networks, network connectivity, and graph traversal. Vertices in the graph are categorized into different types of locations. We have aimed to produce a graph that can accurately represent locations in a city. The use of graphs is crucial for our project since the structure of the domain of the cops and robbers game is reliant on it.

Code and Algorithm structure:

To construct the data types that make up our program, we made classes for vertexes and graphs. The vertex class has instance attributes of item (the name of the location in string form), a set of neighbors (vertexes adjacent to the vertex), kind (a string representation of the kind of location the vertex is), and the score (an integer ranging from 1 to 10 inclusive that is associated to the location vertex). The vertex class has an initializer function and three functions called `degree()`, `check_connected()`, and `connected_distance()`. The functions respectively return the degree of the vertex, show whether the vertex is connected to the vertex associated with the target vertex, and return a path that connects the self with the target vertex with a certain length.

The graph class takes in a set of vertices and has a series of functions, both adopted from the course notes and our own. Specifically, the function `get_all_vertices` that returns a set of all the vertex items, or location names. The same purpose is also served for `edges_list`, except we return a list of all the edges in the graph. Our `assign_random_edges` function takes in vertices in the graph and randomly assigns edges, and `get_path` takes in two items and returns the shortest path between the two, returning None if a path doesn't exist.

In addition to these classes, we have classes to represent the robber and the cop. We created a Player class that serves as a parent abstract class for the cop and robber player. The Player class has a variety of functions that come in handy for the cop and robber. The RobberPlayer class represents a robber player that prioritizes the safest path to travel around the graph. The RobberPlayer takes in a `curr_location` which represents which vertex the player is currently on. In addition it takes in a `move_limit` and `move_count` which are two integers that represent the max amount of moves the RobberPlayer can take, and the current amount of moves that player has made. The RobberPlayer class also takes in a `target_location` that represents where the player is aiming to travel to. We also made a RiskyRobberPlayer class which represents the same player as the RobberPlayer except this player prioritizes the shortest path rather than the safest path that the RobberPlayer prioritizes, and therefore it takes risks to reach these locations. Our CopPlayer class represents the cop, which takes in `curr_location` and `move_count` which represent the current location of the cop and the amount of moves the cop made. The CopPlayer has a `make_move` function that makes a move for the cop based on the move count. This combination of vertexes and graphs with our three different types of players allows us to generate a cops and robbers program that is based around the concepts of graphs, with players traveling to vertices through different paths made up of edges.

Our program uses a variety of major computations in order for the project to run. One major computation in our program is how we build our data set. As we said earlier, our `large_location_data.csv` data set is made up of subsets of a variety of different data sets we found. We loop through each of the data sets and extract the subsets we want, and then add them to our data set that we are making by making each line a name of a location and its type of location, such as park, cemetery, or tourist spot for example. Another big computation our program does is loading the location graph and assigning scores to each location in our data

set. We assigned certain scores to certain types of locations, for example parks have a score of 3, cemeteries have a score of 0, and health locations have a score of 7. The function iterates through each line of our data set and assigns each location the corresponding score based on its location type. Our main function of running the game is also a vital important part of our program. We call the location graph and initialize the robber, cop, and their respective start locations. Then we loop through the robber's path and update its location all while updating the cop's location. In our loop we have a set of if statements that correspond to each outcome of the game, if the robber wins or the cop wins or when the robber reaches the mid location which is when we change its target location to the end location.

How program reports results in visual/interactive way using new libraries

Our program uses new libraries in order for the Cops and Robbers game to work. The first new library that we used was NetworkX. This library was primarily used for drawing our visualization of the program, specifically the paths in our graph. We used the functions `nx.draw()` and `nx.draw_networkx_nodes()` to draw our graph and our nodes. We used `nx.draw_networkx_edges()` to draw our edges by taking in the graph, a specified position that was required by the `networkx` library, a set of the robber and cop's edges, color, and width as our parameters. We use that function twice separately to draw the edges for the robber and the cop. Additionally, once the edges are drawn we used `nx.draw_networkx_labels()` to draw on the labels on the visualization. With NetworkX being useful in helping us draw our visualization, we used the library MatPlotLib to implement and visualize our map or graph. Naming our figure "plt", we used the `show()` function, or in our code we called it on plt as `plt.show()` to visualize our program. We used `plt.axis('equal')` to make the axis of our figure the same. Then at the end of our visualization part of our code, we called `plt.show()` to visualize our figure or graph, printing our robber and cop after.
We have provided users three options for visualising the path. These instructions are specified in the doctstring of the `data_set_builder.py` file.

How to obtain the datasets and run our program

- **Instructions**

  Running this project as intended requires the installation of both the datasets and multiple libraries. Instructions for this as well as how to properly run the main program are explained below:

  1. Download the all the provided files and datasets.
  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. This will install libraries including PythonTA, Networkx and Matplotlib.
  4. Our project runs on the `csv` file `large_location_data.csv` which was built using the file `data_set_builder.py`. To run this file, download all the datasets in the `data.zip` file and save it in the same folder as the rest of the files.
  5. Open and run the file main.py in PyCharm.
  6. Run the function `run_program` in main.py, experimenting with different number of games and the two types of RobberPlayer (RobberPlayer and RiskyRobberPlayer) that meet the preconditions and documentation of the function.
  7. The `run_program` function would first produce the path that the given robber took in a game and the winner of that game. Additionally, it also outputs a visualization of the graph and the path which the robber took in a given game.
  8. For analysis purposes, focus the path that the given robber took in a game and the winner of that game.
  9. For visualization purposes, focus on the visualization of the graph and the path which the robber took in a given game.
  10. Because of our algorithms and the randomization nature of the graph, if the visualization is too hard to interpret or does not show up, please re-run the function `run_program` until a clearer graph is produced.

Changes since the proposal

Originally we had decided that we would one single robber and two types of cops. The cops would be divided into detectives and patrolling officers. The patrolling officers would walk on a fixed path, while the detective would chase the robber. When designing our algorithm, we realised that running three players at the same time, and keeping a track of their location would be computationally complex. So, we decided to change the design so that we have only one cop and one robber, where we give the user the option between choosing the type of robber they want to play with instead, i.e, a RobberPlayer or a RiskyRobber. Analysing the outputs of the two different types of robbers would be more achievable.

Additionally, we decided to have a live visualisation for the path that the robber and the cop use. However, this was not supported directly by the `networkx` module. Manually updating each frame of the animation to show the increase path created multiple outputs instead of updating the same output. `networkx` is only able to update each frame if the graph has a fixed shape. However, our graph creates random edges each game, and so weren't able to do this. We decided that we should simplify our visualisation, such that the final output highlights the path the cop and the robber took.

Discussion Section

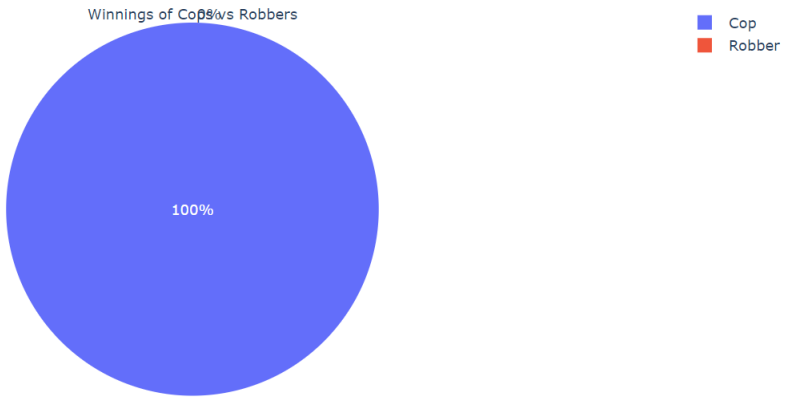
Expectations

For this project, our expectations were to first implement a game of Cops and Robbers which utilizes the structure of a graph to map out a city and locations of the players. We expected that a graph would be an appropriate data structure to present a city because of its network applications and the connectivity of its vertices. Secondly, we wanted to develop different Player AIs (RobberPLayer, RiskyRobberPlayer and CopPlayer) that would represent the robbers and cops players. For this project, we focused on developing two algorithms for the robber such that the robbers take paths that avoid the cops, and win most of the games. We also designed an algorithm for a cop to efficiently catch the robber.

Moreover, with further calculations and visualization, we wanted to determine which Robber algorithm is the more efficient. Finally, based on the result of our project, we wanted to draw a conclusion on whether graph is a reasonable data structure to use for a game such as Cops and Robbers.

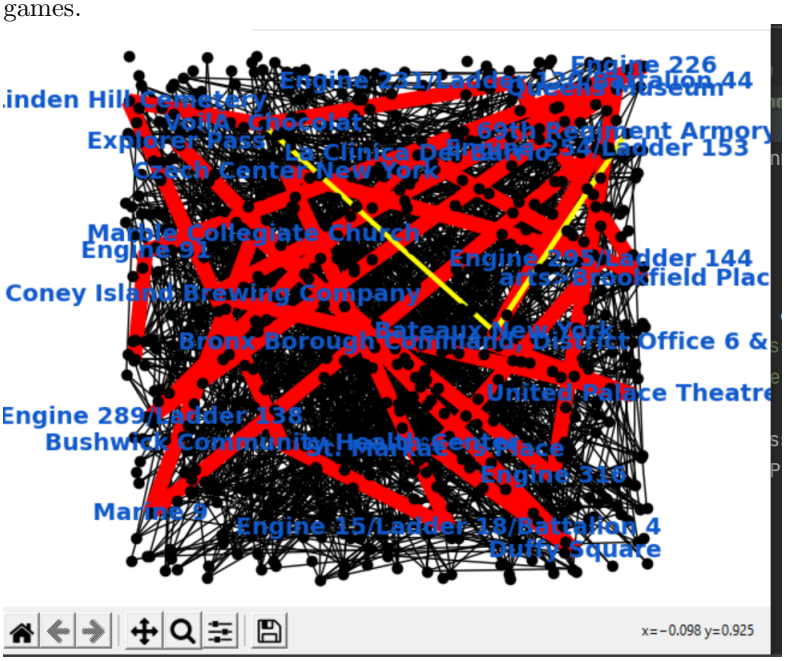
Our Results and Explanation

Our results from our program are not what we had expected at the beginning of this project. Although we initially thought there would be a clear distinction between the percentage of times the two types of robbers and cops won, when we ran our program the cop ended up winning every single time. We ended up running the program many times and each time the cop would win. Part of our visualization includes a pie chart that depicts the percentage of times each type of player won in a set amount of games. We attached below a pie chart that shows the cop won every single game that we ran.

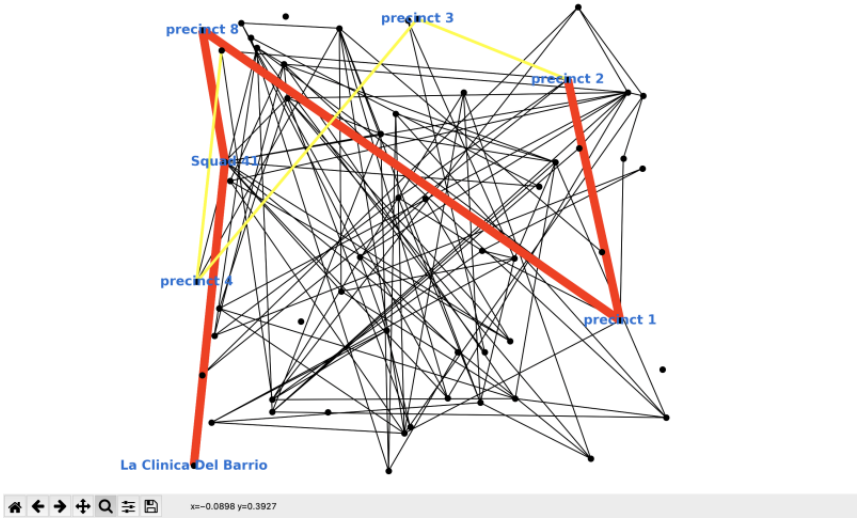


In the pie chart above we can see that the cop won 100% of the time.

We tried testing our program out on a large dat set and a small data set. Below are two images of the graph visualization of the small data set and the large data set. In both of these scenarios, the cop won all games.



Above is the visualization of the graph using the large data set.



Above is the visualization of the graph using the small data set.

After looking through our project we realized that our algorithm and program structure was biased towards the cop. In order for a robber to win, the robber must be able to steal the treasure and make it to the end location without being caught by the cop or running out of moves. The cop however, had more opportunities to win. Our program structure made cops have unlimited moves, whereas robbers only had a certain number of moves. With this restriction, robbers had to travel to a mid location to steal treasure and then traverse their way to the end location. There were cases in our program results where the robber would run out of moves and not be able to reach the target location, thus making the cop win by default. Additionally, there were cases where the robber would traverse to a location vertex and become trapped there because there was only one edge connecting that vertex, thus allowing the cop to easily catch it. There was also the expected way a cop could win, which was by catching a robber. By this analysis it is clear that there are many more opportunities for the cop to win in this game than the robber, which may explain why our results show the cop winning all the time.

Reflection - What we could have done differently

To further improve our implementation of the project, there are a few things that we could have done differently.

1. We could have the start and the end locations fixed for every game. With this change, the algorithm would possibly work better since the randomization factor is now minimized.

2. It was not a good idea to base our algorithms solely on the output on the graph. Instead, we could corporate the use of graph for the game and the tree structure for implementing the algorithm. Similar to what we have done in Assignment 2, using a tree to store all the path that robber players took for every game and from there develop a learning algorithm (since the start point and end location of the game are now fixed). This way, the robber player could potentially make better moves which leads to them winning the game.

3. As for the visualization, we were a little ambitious trying to store and graph out all the locations in a city. Consequently, our final visualization was full of nodes and edges and quite hard to see the path which the robbers and cops took. However, we could simplify our target to a university or a campus that has less locations than a crowded location-filled city. This change would make the visualization easier to interpret and more "human-friendly".

References

Networkx. (Aug 22, 2020). Networkx v2.5. Retrieved April 16, 2021, from Networkx:
https://networkx.org/documentation/stable//reference/index.html

Matplotlib. (n.d.). Mathplotlib v3.4.1 Retrieved April 16, 2021, from Matplotlib:
https://matplotlib.org/stable/citing-matplotlib

Assignment 2: Trees, Chess, and Artificial Intelligence. (Mar 6, 2021). Retrieved April 16, from CSC111 course materials:
https://www.teach.cs.toronto.edu/ csc111h/winter/assignments/a2/handout/

Assignment 3: Graphs, Recommender Systems, and Clustering. (Mar 27, 2021). Retrieved April 16, from CSC111 course materials:
https://www.teach.cs.toronto.edu/ csc111h/winter/assignments/a3/handout/

Tutorial 7: Graphs, Graphs, Graphs. (n.d.) Retrieved April 16, from CSC111 course materials:
https://www.teach.cs.toronto.edu/ csc111h/winter/tutorials/07-graphs/