Scotland Yard

November 5, 2023

Aniket Kumar Sahil (2022CSB1067) ,
Akash (2022CSB1064)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Mr. Law Kumar

Summary: Our project aims to recreate a board game known as Scotland Yard, a detective game where some detectives try to catch a criminal Mr. X who is roaming around the city. We have used the knowledge of graphs and decision tree to create a game such that you can play with the computer. In our game, we act as detectives and Mr. X is controlled using the decision tree. We have used graph to represent the city by using nodes as various stations where Mr. X and the detectives can stay and using edges as the path where they can move. We create new map every time we play the game using Barabási-Albert Model. The detectives are human controlled while Mr. X is Algorithmically controlled using the decision tree we created!

---

1. Introduction

In the subsequent section, we will implement the graph data structure and will create a decision tree to control the movements of Mr. X. Graph data structure is very useful here as it is the best possible way to represent a city.

`What is Scotland Yard?`

It is a game board invented in 1983. Following are the rules of the game:-

1. The city map is like a connected graph of 50 nodes connected with each other with various modes of transportation like taxis, buses, subways, ferries, etc. Each node is numbered from 1 to 50. Each player is assigned a random number and that is the point from where he/she starts.
2. Expect Mr. X, the location of the rest of the players is revealed.
3. Each movement can be done across the map by a taxi. Note that each movement costs a ticket.
4. You can only move to the adjacent stop of the transport used. One step at a time.
5. All detectives starts with a limited amount of tickets. However, after each move, the detectives must give their tickets to Mr. X which he can use for himself and hence we can say he has unlimited moves.
6. Mr. X secretly keeps a note of all his movements in a diary. His mode of transportation is always revealed to the detectives but his main location is revealed after every 4 moves.
7. The game starts with all of them at random locations , and then each one of them moves one step at a time.
8. The game ends with 2 outcomes Either Mr. X gets caught or the detectives run out of tickets. He gets caught if a detective lands at the same place as Mr. X.
9. If he gets caught, then the detectives win otherwise Mr.X wins.

`Why use Graphs?`

We have used the Graphs Data structure here cause it is the best way to represent a city as there can be multiple connections to a single location and it is very easy to represent it using graph and traversing the graph in search of a particular location. Graphs are also the best possible way to visualize a real city. And graphs are also helpful for the implementation of our various algorithms usd in traversal as well as making the decision tree for the movement of Mr. X.

IMPLEMENTATION

In this project, we have done the following implementaions:-

- Barabási-Albert Model: It is used to create the graph every time we play the graph.
- Decision Tree: It is the decision tree used to decide the moves for Mr. X.
- Monte-Carlo Tree Search: It is the algorithm we **tried** to use to decide the moves for the Detectives.

2. FUNCTIONS

This section tells about the various functions implemented in the code :

BARABSI ALBERT MODEL :
In this function we generate a graph randomly which eventually becomes the city of our game. The graph is forged by first making a small connected graph which act as a base for the rest of our graph. There is A parameter m associated with it which defines how dense the graph comes out to be. This is directly proprotional with the max degree of the vertex set. Here we set the max degree to be 4. Hence our graph pretty much comes out as a graph with each vertex having degree 4.The time complexity of this is O(V) where v is the number of vertices.

BFS FOR SHORTEST PATH :
In this function we find the shortest path between 2 vertices. It explores the graph level by level, ensuring that the first occurrence of the target vertex is provides the shortest path. The distance array keeps track of the distances, queue manages the order of exploration. If the destination vertex is not reached, the function returns -1, signifying that there is no path between the two vertices. The time complexity of this is O(V + E) as we are doing BFS.

DECISION TREE :
In this function we generate a decision tree that decides the next move of Mr. X. This is done by calculating the average distance of each detective from each neighbor for every neighbor. This requires the usage of BFS to find the minimum distance between two vertices. Since we are applying BFS which has time complexity of O(V+E), we can say that it has a time complexity of O(4 * (V+E)).

3. Figures, Tables and Algorithms

3.1. Figures

Following images are a visual representation of the algorithms we have implemented.

Below this is the Barabasi Albert model of graph making. Notice how we start from a complete graph of 3 vertices and as we keep on adding vertices one by one it gets bigger and more complex.

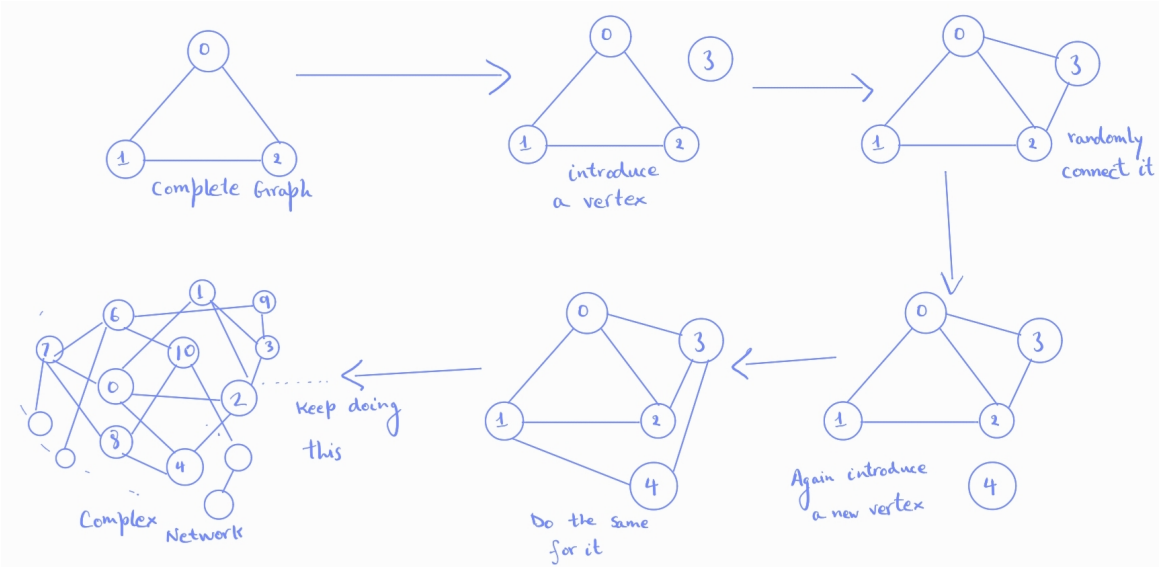

Figure 1: Visualizing Barabasi Albert Model

This image below shows the how the decision tree makes decision after analyzing each vertex.

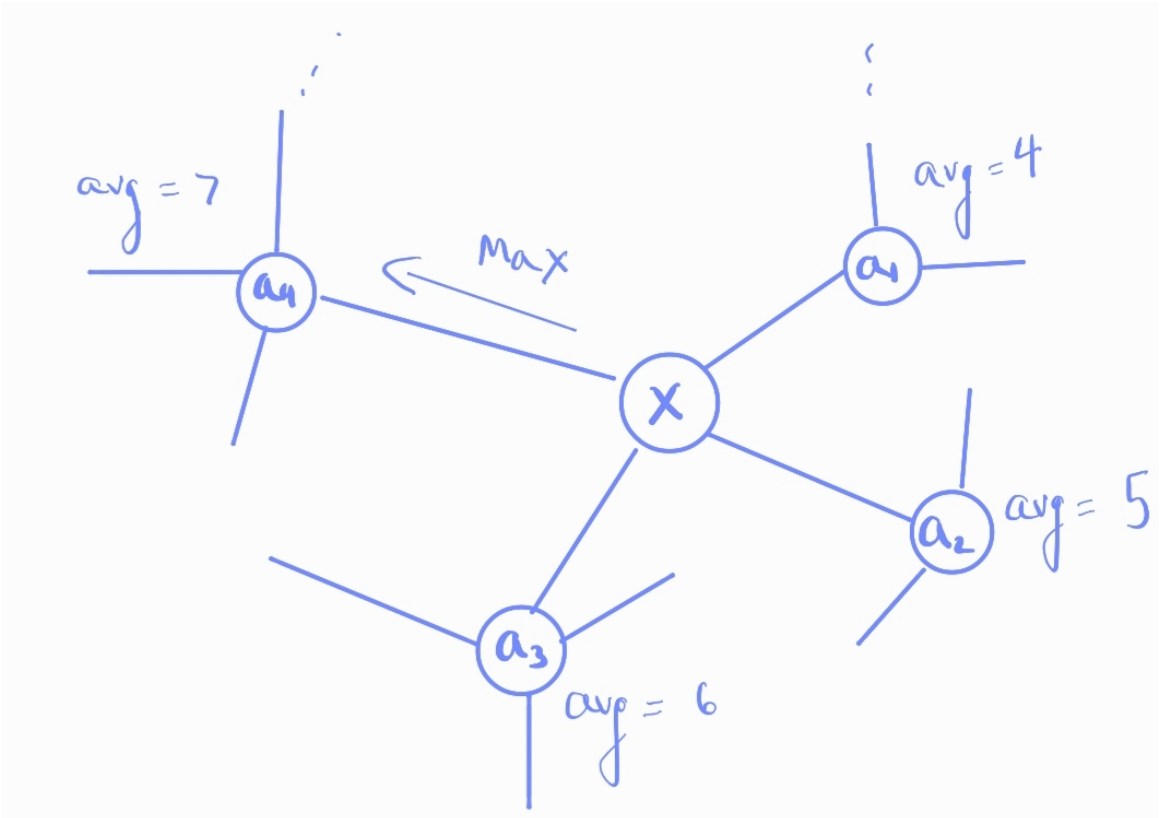

Figure 2: Decision Tree making a decision by seeing the average

3.2. Tables

The table below show the worst case time complexities of the various functions used : 1.

Example of Table (optional)

	Worst Case
Decision tree	O(4*(V+E))
Barabasi Albert Model	O(V)
Shortest path using BFS	O(V+E)

Table 1: Worst case Time complexities

3.3. Algorithms

This is the

Algorithm 1 Barabási Albert Model

Generate a small complete graph (like of 3 vertices)
for $i = 3$ *to* $(n-1)$ **do**
    Generate 2 random numbers r1 and r2
    if $r1 == r2 || degree\ of\ any\ is > 4$ **then**
        regenerate
    end if
    connect all the vertices and update the degrees
end for

3.

Algorithm 2 Decision Tree for Mr. X

Set disAvg = 0.0
Set MaxdisAvg = -1.0
Set next = -1
Set Bestnext = -1
for *Each neighbour of Mr.X in the graph*[*mrx*] **do**
    Calculate the value of disAvg ie the average distance between the Mr. X and the Detectives if he moves there
    disAvg = (sum of the distance at each vertex)/3.0
    Find the maximum average and set it for maxdisAvg and the corresponding vertex in bestnext
end for
if bestnext != -1 **then**
    set the mrx as bestnext
end if
append mrx to mrxmoves wich keep track of the MR. X moves

This algorithm is using queues for BFS to find the shortest path between any 2 nodes :

Algorithm 3 Finding the shortest distance

```
if start == end then
    return 0
end if
Initialize distance as an array of graph.size() with all the elements as -1
create an empty queue q
enqueue start into q
while q is not empty do
    Current = dequeue from q
    if current == end then
        return distance[current]
    end if
    for each neighbour in graph[current] do
        if distance[neighbor] == −1 then
            enqueue neighbour into q and set distance[neighbour] to distance [current] + 1
        end if
    end for
end while
return −1
```

For the Monte Carlo tree search algorithm, we had to generate a sample space that contained thousand of test runs. But before that, we need to come up with a scoring idea. If we observe, in each step we are moving 1 edge hence at max we can go 4 vertex ahead. Therefore we find all the possible vertices that are reachable and store them in an array. Then we choose any one of them randomly. Then we also move from the main vertex randomly and check the distance of the randomly reached vertex to the randomly selected vertex and since it can't be more than 4 we give (5 - distance) as a score. In this way, we can run several test over it keeping the randomly selected vertex fixed for some amount of time. In this way we can do it by selecting different vertices for different amount of time and hence we feed all that in the tree nodes. Using this scoring scheme we were ready to build a Monte Carlo Tree search algorithm. Still, it would have some drawback since the data is generated randomly, it won't always make proper sense but if we do these test using real people's mind as input it would make alot of progress and will be able to correctly predict the outcome.
Now that we know how to generate score suppose you take one node and randomly and then you have the randomly selected final destination with you. You start from the main vertex by moving to any one of the neighbours. Since they haven't been visited it first incriments its visiting variable. Then since it is a leaf node of the tree (as there is no other node), we do a roll out or more specifically generate an estimate of a score and score generation has been discussed as above. Then we again do a test run. If the vertex is visited and it is a leaf node so we do an explansion by exploring further in that vertex. Then in a similar fashion we generate a score and then pass on that score by adding it to the currently existing score uisng the formula $V = V_i + c \cdot \sqrt{\frac{\log(N)}{n_i}}$. $We always give the visiting i.e., n_i the priority. If it is visted less times we tend to visit it it more. If both have their n_i same then we chec$

4. Some further useful suggestions

Although the game we made is fully complete but there can be many improvements to the game that can still be made:-

1. Monte-Carlo Tree Search:
One can use the Monte-Carlo Tree Search Algorithm to make the detectives Algorithmically controlled too.
2. Using visual Graphics:
One can only graphics to represent the map and the players visually and make the game more appealing.
3. Increasing Complexity:
The Map can be made more complex by making more stations and also by increasing the number of means of transportations.

5. Conclusions

In our project to implement Scotland Yard in C++, we leveraged various aspects of graph theory, such as queue and BFS for pathfinding and distance of Mr X from the detectives, the Barabasi-Albert model for generating a dynamic network of nodes for the city graph, and decision trees for the logic of Mr X movement. However, our attempt to incorporate Monte Carlo Tree Search (MCTS) into the game's strategy for detectives faced challenges. Despite our efforts to simulate and optimize decision-making for the detectives, MCTS implementation proved to be a complex task.

We found that the dynamic nature of Scotland Yard, where the location of Mr. X evolves over time, made it challenging to effectively apply MCTS. This strategic shortcoming could be attributed to issues related to defining proper state spaces and action spaces for the detectives, as well as the lack of a clear scoring mechanism. Nonetheless, the project provided valuable insights into the complexities of implementing MCTS in real-world games and underscored the need for more robust strategies in dynamic game environments.

While we were unable to fully integrate MCTS, our project laid the groundwork for future improvements and underlines the complexity of creating effective game-playing agents. We hope that our work will inspire future research and development in this fascinating field.

6. Bibliography and citations

Acknowledgements

We would like to thank our instructor, Dr. Anil Shukla and Teaching Assistant Mr. Law Kumar to give us their valuable time and help us in the completion of the project.

References

1. https://scholar.afit.edu/cgi/viewcontent.cgi?article=5334context=etd
2. https://www.ijcai.org/Proceedings/99-1/Papers/084.pdf
3. https://dke.maastrichtuniversity.nl/m.winands/documents/Cig2011pape42.pdf
4. https://www.geeksforgeeks.org/barabasi-albert-graph-scale-free-models/
5. https://arxiv.org/pdf/1611.05990.pdf