Modelling Scotland Yard

Fredderf and Ed0G

December 29, 2021

1 Introduction

We concentrate on a popular search game called Scotland Yard involving one player, 'MisterX' and 1-5 'Detective' players. The game is played on a graph with vertices representing locations in London and 3 edge types representing taxi, bus and underground travel. The players take it in turns to move to an adjacent vertex. The aim of the Detectives is to 'capture' MisterX by landing on the same vertex as him or not allowing him to move and the aim of MisterX to stop this from happening. If MisterX is captured the detectives win, whereas if MisterX can last 24 turns without being captured they win.

The game is begun by all players choosing at random a starting tile. The detectives locations are at all times common knowledge however the location of MisterX is only revealed after his 3rd, 8th, 13th, 18th and 24th turns. Each detective receives 4 underground tickets, 8 bus tickets and 11 taxi tickets.

On each turn MisterX moves to an adjacent vertex and records it covering it up with the ticket used to get there (taxi, bus, underground). If this is not possible due to surrounding detectives the game is over and the detectives win. The detectives then move in the same way on either a taxi, bus or underground edge spending a respective ticket in the process.

Furthermore, MisterX receives 5 blank tickets, which allow MisterX to hide his mode of transport on a turn, and 2 double move tokens ,which allow him to take 2 turns before the detective get a chance to move.

There are always at least 4 searchers in the game. When there are less than 4 detective players then up to 2 of the 4 players are 'Policemen' and do not have the ticket restriction that detectives have. These players are controlled jointly by the searcher players.

2 Model of the game

We will model a simplified version of the game in order to improve the performance of the player algorithms. We will remove rules involving travel

tickets for the detectives, essentially making all detectives policemen (we will continue to refer to all searchers as detectives throughout). We will further simplify the game by removing all double move tokens and blank tokens. To balance the disadvantages applied to the player MisterX, we will reduce the detectives to 3.

We make an undirected graph with vertex set $V = \{1, 2, \ldots, 199\}$ and 3 edge sets E_t, E_b and E_u to represent the board and taxi, bus and underground edges respectively.

Definition 2.1. We define the location variables of MisterX and Detectives and as follows (for $k \in 0, 1, 2, \ldots, 24$ and $j \in \{1, 2, 3\}$):

- LM_k is the location (vertex) of MisterX at the end of turn k.
- LD_k^j is the location (vertex) of Detective j

Definition 2.2. We define the possible moves for MisterX and Detectives and the transport mode used by MisterX as follows (for $k \in 1, 2, \ldots, 24$ and $j \in \{1, 2, 3\}$):

- M_k is the set of possible moves that MisterX can move to on turn k.
- D_k^j is the set of possible vertices that detective j can move to on turn k.
- $XTransMode_k$ is the transport mode used by MisterX on

The game runs as follows:

Algorithm 1: SCOTLAND YARD GAME

Input: LM_0, LD_0^1, LD_0^2, LD_0^3
Output: Winner

for *k in 1,2,...,24* **do**
    LM_k =MisterXMove
    Update $XTransMode_k$ LD_k^1=DetectiveMove
    LD_k^2=DetectiveMove
    LD_k^3=DetectiveMove
    if $LM_k \in \{LD_k^1, LD_k^2, LD_k^3\}$ **then**
        STOP: Detectives win
STOP: MisterX Wins

2.1 Modelling in Python

We use python to model the game. The file generate_graphs.py contains functions relating to drawing out the graph in the form of a Graphviz object. When it adds nodes, it also highlights the positions of MisterX and the Detectives.

The player.py file makes the classes MisterX and Detective which includes initialising each by choosing an initial location from the starting locations. The each class also includes a function to find a list of possible moves from their current location.

We will use play_game.py to run the model of a single game. The initialise_game function creates a MisterX and 4 Detectives and ensures all Detectives have different starting locations. It then contains the code to run the game.

3 Initial Strategies

As an initial strategy we will select the move of every player by choosing a vertex uniformly at random from the set of unoccupied vertices adjacent to his current vertex location. For a finite set S, we call $random(S)$, the element chosen uniformly at random from S.

Algorithm 2: MISTERX RandomMove STRATEGY

Input: k=TURN, LM_{k-1}, LD^1_{k-1}, LD^2_{k-1}, LD^3_{k-1}
Output: LM_k

Calculate M_k
if $M_k = \emptyset$ **then**
    Stop Detectives win
else
    Return Randomchoice(M_k)

We ran the game with the RandomMove strategy for both MisterX and Detectives 100,000 times. Detectives won 36309 times and MisterX won 63691. We expected MisterX to win the majority of the time however were expecting a larger majority.

4 Updated Strategy for Detectives

We next improve the strategy of the detectives to improve their performance. To do this we will introduce a set of possible positions for MisterX based on common knowledge in the game. We will call the set '$PossXLocation$'. The following algorithm will be run by Detectives just before their turn.

Algorithm 3: DETECTIVES RandomMove STRATEGY

Input: k=TURN, LD^1_{k-1}, LD^2_{k-1}, LD^3_{k-1}
Output: LD^1_k, LD^2_k, LD^3_k

move=[ ]
for j *in 1,2,3* **do**
    Calculate D^j_k
    if $D^j_k = \emptyset$ **then**
        $LD^j_k = LD^j_{k-1}$
    else
        $LD^j_k = Randomchoice(D^j_k)$
Return LD^1_k, LD^2_k, LD^3_k

Algorithm 4: PossXLocation SET UPDATE

Input: k=TURN, $PossXLocation$, LD^1_{k-1}, LD^2_{k-1}, LD^3_{k-1}, $XTransMode$
Output: $PossXLocation$

if $k \in \{3, 8, 13, 18\}$ **then**
    $PossXLocation = \{LX_k\}$
else
    $temp = \{\}$
    for *vertex in* $PossXLocation$ **do**
        Add to $temp$ all vertices adjacent to $vertex$ in $G(V, E_{XTransMode})$
    $PossXLocation = temp \backslash \{LD^1_{k-1}, LD^2_{k-1}, LD^3_{k-1}\}$
Return $PossXLocation$

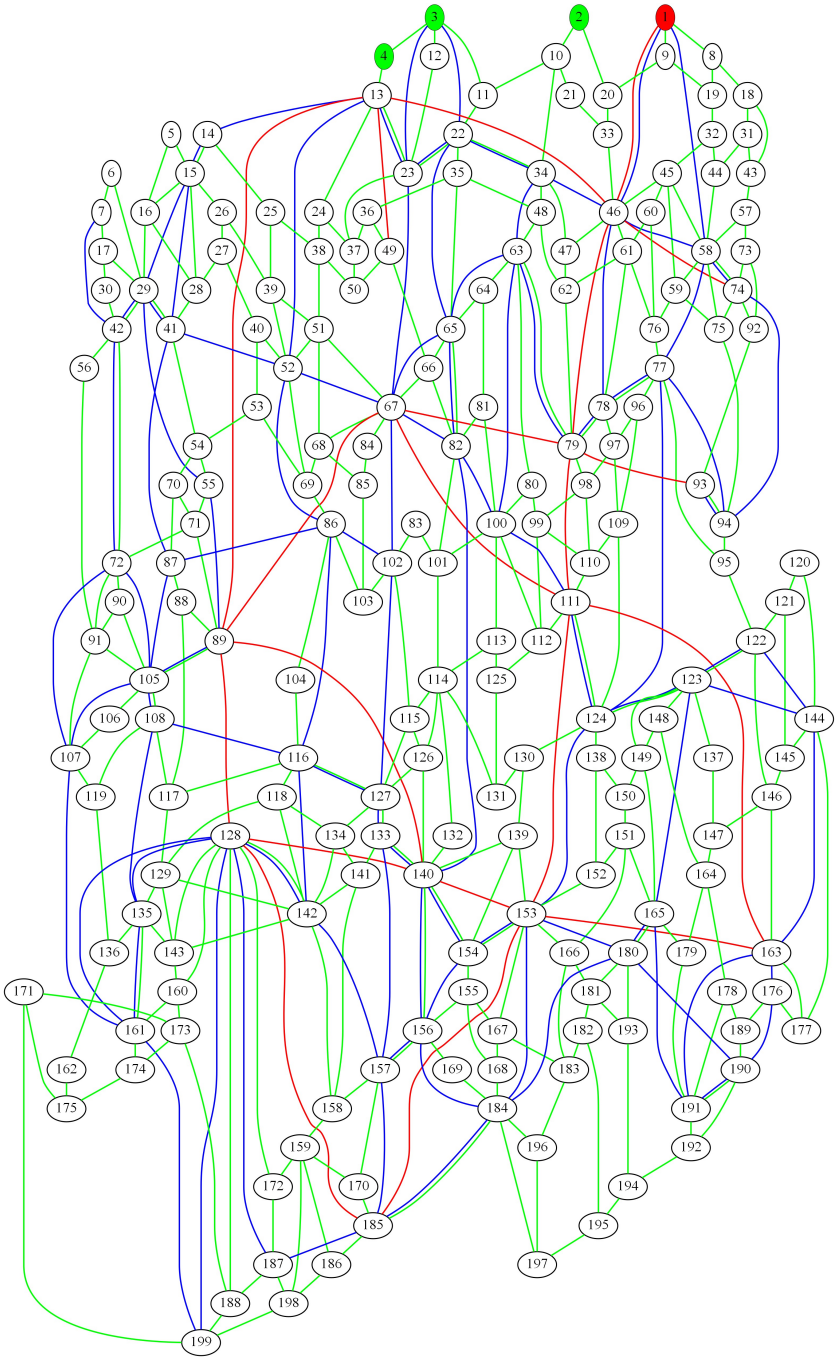

Figure 1: The graph $G(V, E_t, E_b, E_u)$ with edges of E_t, E_b and E_u represented in yellow, blue and red respectively. MisterX position and those of the detectives are highlighted in red and green respectively and orange when a detective has landed on MisterX

Following their turn the Detectives will run the single operation

$$PossXLocation = temp \backslash \{LD_k^1, LD_k^2, LD_k^3\}$$

Using this we update the detectives strategy on their turn to one where all detectives move towards the possible location of MisterX. For this we use the function $distance(x, y)$ to be the graphical distance between 2 vertices (ie. the minimum length of a path from a vertex of set X to a vertex of set Y).

Definition 4.1. We define Let u, v be vertices. We define $distance(u, v)$ to be the minimum length of a path from u to v on the graph $G(V, E_t \cup E_b \cup E_u)$.

Let N be a set of vertices. We define:

$$distance(u, N) := \min\{distance(u, v) : v \in N\}$$

Algorithm 5: DETECTIVES RUSH STRATEGY

Input: k=TURN, $PossXLocation$, LD_{k-1}^1, LD_{k-1}^2, LD_{k-1}^3
Output: $\{LD_k^1, LD_k^2, LD_k^3\}$

```
if k < 3 then
    LD_k^1 = RandomMove
    LD_k^2 = RandomMove
    LD_k^3 = RandomMove
else
    for j in 1,2,3 do
        d=distance(LD_{k-1}^j,PossXLocation)
        for move in D_{k-1}^j do
            if distance(move,PossXLocation)< d then
                LD_k^j = move
Return {LD_k^1, LD_k^2, LD_k^3}
```

We ran the game Detectives rush strategy and MisterX random strategy 1000 times. Detectives won every time. ***add collection of graphs to demonstrate closing in strategy***

5 Updating Strategy for MisterX

We now aim to improve MisterX's performance.

We first try a simple strategy to maximise the distance between MisterX and the position of the detectives. The results for this were generally unsuccessful with MisterX winning only 1-2% of games. We optimize the strategy

Algorithm 6: MISTERX RUN STRATEGY 1

Input: k=TURN,LM_{k-1}, LD^1_{k-1}, LD^2_{k-1}, LD^3_{k-1}
Output: LM_k

```
d = distance(LM_{k-1}, {LD^j_{k-1} : j = {1,2,3}})
LM_k = 0
find possible moves M_k for move in M_k do
    if distance(move, {LD^j_{k-1} : j = {1,2,3}}) < d then
        LM_k = move
if LM_k = 0 then
    LM_k = RandomMove
Return LM_k
```

by considering the distances to all detectives. We clearly want nearer detectives to take priority so take a sum of $3^{-distance(LM_k,LD^j_k)}$. This a potential difference in distance to a nearer detective always takes precedence over a change in distance between the other 2 farther detectives.

Algorithm 7: MISTERX RUN STRATEGY 2

Input: k=TURN,LM_{k-1}, LD^1_{k-1}, LD^2_{k-1}, LD^3_{k-1}
Output: LM_k

```
LM_k = arg min_{move∈M_k}(Σ_{j=1}^{3}(3^{-distance(LM_k,LD^j_k)}))
Return LM_k
```

This algorithm hence chooses the move out of MisterX's available moves that minimizes the value described above.