RL and planning for Scotland Yard

Amokh Varma
Dept. of Mathematics
IIT Delhi

Ayush Srivastava
Dept. of Mathematics
IIT Delhi

Shauryasikt Jena
Dept. of Electrical Engg
IIT Delhi

***Abstract*—This paper is for the Term Project of ELL729: Stochastic Control and Reinforcement Learning , under Prof. Arpan Chattopadhya, IIT Delhi. Our work is based on training and developing intelligent agents in Scotland Yard, a board game based on multiplayer and episodic hide-and-seek. For this, we employ various learning and planning methods, *including Deep Q-Networks and Monte Carlo Tree Search*, to increase the victory rate for Mr X: the killer, one type of player in the game.**

I. Introduction

Scotland Yard is a multiplayer board game in which a group of detectives (player type 1) try to chase down a criminal called Mr X (player type 2). The game is based on a map of Old England. The detectives and Mr X move along the map using different means of transportation: bus, cab, and underground. Any detective that lands on the same location as Mr X will end the game in detectives' favour. The location of Mr X is kept secret from detectives but the order of modes of transport used is common knowledge, as are the locations of the detectives. The aim of Mr X is to stay safe for up to 20 turns and the aim of the detectives is to collectively try and catch them before the turns run out.

II. Related Work

There have been several works on playing board games using reinforcement learning and planning. The work in [1] uses Monte Carlo Tree Search to roll out out different possible scenarios and find the best move at each step. There have been other papers aimed at solving different board games like Monopoly, Backgammon, Chess etc. [2] uses Q-learning to find the optimal policy in a setting of monopoly. The state space representation plays a big role when applying reinforcement learning methods. [2] uses a multi-dimensional array with some information carried forward from the past and some of the current state of board. Such a representation fully represents the belief of current state as perceived by the user, and even more than that as the computer has the ability to memorise past flawlessly. The board is represented as a graph connecting different nodes . It is a multi graph denoting different vehicular transportations and and is unweighted. [3] uses Graph Neural Networks to solve graph-based RL problems using deep learning.

III. Problem Definition and Challenges

A. Rules of the game

We have used 'n' and a killer called Mr X in our game. The game start with random initial positions for all the (n+1) players. Going on ahead, all our work is based on the default setting of the game with n=4 detectives. The detectives do not know the positions of Mr X except on move numbers 3,8,13 and 18. In each turn, Mr X and detectives have one move each. They are given certain number of cards at the beginning of the same. For each move, they have to give up a card and use the transportation provided by it. The tickets used by the killer (also known as Mr X or X) are known to the detective except when he uses a special ticket called 'hide ticket' which allows him to take a move without declaring the mode of transport. The map has 200 nodes and each of them are connected using different combination of modes of transport, chosen among bus, taxi and underground. The game ends if 20 turns are over, or one or more detectives land on the same position as the Mr X. The game also ends if the all the detectives can't move but Mr X can resulting in victory for Mr X, and vice-versa leading to a victory for detectives.

B. The Challenges

Under these set of rules, the game is asymmetrically difficult and biased against Mr X as the mere survival for Mr X for 20 games is extremely difficult in this case. Despite the environment being same for both of them, the observed state is different for both. Mr X has access to the location of the detectives whereas the detectives do not have the access to the location of Mr X except at the end of certain moves, making the game fully observable for Mr X and partially observable for detectives. The action spaces for both detectives and Mr X are not only dependent upon the state in which they are in but also on the number of cards left, making it extremely difficult to apply standard Reinforcement learning algorithms like Q-learning without major modifications.

IV. The game as a decision making problem

- At each step, Mr X knows the locations of detectives and the cards he has as well as the cards carried by the detectives.
- The detectives know their own position and have a list of possible locations, based on probabilities, of Mr X calculated using last seen location of Mr X and the sequence of tickets used.
- We use the following reward functions for our agents. These rewards are awarded after each step. The detectives carry a collective reward instead of individual rewards.

This particular model of reward is based on the reward used in [2].

$$R_x = \min_i(d(p_x, p_{d,i})) \tag{1}$$

$$R_d = \frac{1}{(4 * \min_i(d(\tilde{p}_x, p_{d,i})))} \tag{2}$$

where R_x is the reward given to Mr X, R_d is the reward given to detective and $\tilde{p}$ and p refer to predicted set of positions and actual position respectively. $d(x, y)$ is the shortest graph distance between the 2 locations, calculated using Dijkstra's Algorithm.

V. Prerequisites

A. Reinforcement Learning

Consider a Markov Decision Process , with reward function $R(s, a)$. Let $\Pi(.|s)$ be the policy space. Reinforcement learning deals with solving the following problem:

$$\max_{\pi \in \Pi} \sum_{i=1}^{T} \gamma^i R(s_i, a_i) \quad a_i \sim \pi(.|s_i) \tag{3}$$

γ is a constant that defines how further into future we would look ahead before making a decision.

B. Q-Learning

Q-Learning is an iterative method used to solve the above problem. $Q(s, a)$ is the expected reward of taking action a from step s. Consider a state sequence $\{X_n\}$ and action sequence $\{A_n\}$. Q-Learning works using the following iteration

$$Q_{n+1}(s, a) = Q_n(s, a) + \mathbf{I}(X_n = s, A_n = a)(R(s, a) + \gamma \max_{a \in A} Q(s', a) - Q_n(s, a)) \tag{4}$$

C. Monte Carlo Tree Search

As introduced in [6], Monte-Carlo Tree Search (MCTS) is a best-first search technique which uses stochastic simulations. MCTS can be applied to any game of finite length.
Its basis is the simulation of games where both the AI controlled player and its opponents play random moves, or better, pseudo-random moves. From a single random game (where every player selects his actions randomly), very little can be learnt. However, by simulating a multitude of random games, a good strategy can be inferred. The algorithm builds and uses a tree of possible future game states. It has four stages , Selection, Expansion, Simulation and Back-propagation. For selection, we use the UCB1 algorithm:

$$a_{next} = \arg\max_{a \in A} (r_a + \sqrt{\ln(N)/n_a})$$

where n_a is the number of times action a is taken from given state and N is number of times the state is visited.

VI. Our Methodology

Defining a proper feature vector is very important in all Reinforcement Learning regime. Ideally it should contain most of the useful information that can be observed from a given state of the game, and Scotland Yard being a fairly complex game, using a proper feature space especially for detectives is very important. As mentioned in **Section IV** the detectives have a list of possible locations, based on probabilities, of Mr X calculated using last seen location of Mr X and the sequence of tickets used we incorporate which as per the best of our knowledge have never been done before.

We implemented a MCTS planning based model for both detectives and Mr. X, We also implemented a DQNN based model which was inspired from [7], and we changed a standard DQNN a bit and instead of only using state space as input we also append the action as an input and output a single value scalar and applied Adadelta optimizer with MSE loss and Relu activation in a 4 layer neural network.

We then went on to adversarial training and trained various model against each other like DQNN for X vs MCTS for detective, DQNN for X vs DQNN Detective , DQNN for X vs Trainned DQNN Detective etc for for improved results.

We also went on to train a DQNN + MCTS based model for our agents which as per the best of our knowle

VII. Results

Going through with our proposed techniques for improving the player model for Mr X, we observed the following. Further discussion is based on the win rate of Mr X over a large number of games.
NOTE: For any algorithm involving Deep Q-Networks for training the model agent of Mr X, it was observed $40000 - 50000$ *iterations of the game were required. Given time constraints and limited computation power, we provide the starting arms of all plots which conform to the observed trend of following up to the ideally expected results.*

A. Baseline

Both Mr X and the detectives were allowed to play with random moves to set up a baseline for our following observations. The results are as shown in Fig 1.
We observe that the win rate for Mr X is roughly 9-10%.

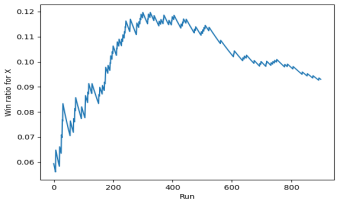

Fig. 1. Baseline set with Win Rate for Mr X as 9-10%

B. DQN Player Against Random Policy Opponent

As mentioned, we first trained Mr X as a DQN agent against all the detectives following random policy movements. The results are as shown in Fig 2.

As can be observed, the training win rate reached approxi-

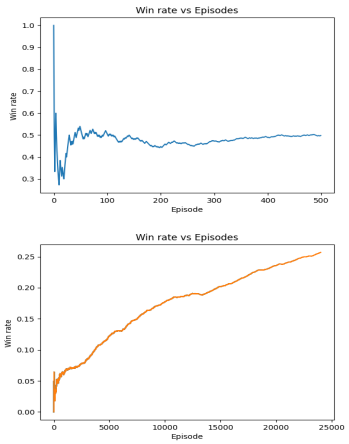

Fig. 2. Win Rate of X during training and testing respectively

mately 50%.

Then for the sake of completeness of results, we trained the detectives as DQN agents against a random policy user Mr X. The agents of detectives were trained in conjunction to a collective reward. The results are as shown in Fig3.

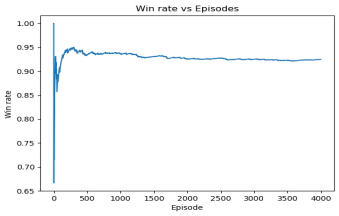

Fig. 3. Win rate of Detective during training

C. Adversarial DQN Agents

For this method, both Mr X and detective agents were trained against each other from scratch, again with all detectives acting in regards to a collective reward. The win rate of X is seen in Fig 4.

The observed win rate is approximately 10%, which is expected to reach more than 40% upon completion of 40000 runs.

D. MCTS Player Against Random Policy Opponent

Here, we simply trained the player under lens to take the best decision as ascertained by the MCTS algorithm, i.e., shown for Mr X Fig. 5.

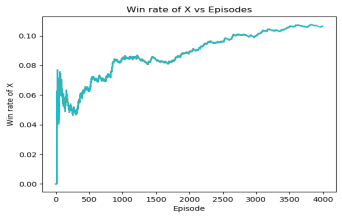

Fig. 4. Win rate of X during training

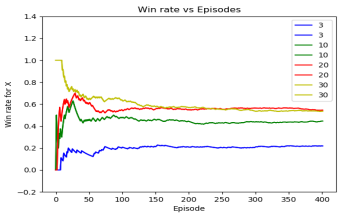

Fig. 5. MCTS with varied playouts, maxmimum win rate of $> 60\%$ reached

E. DQN Mr X with MCTS-boosted Decisions

Here we train our X agent using Deep Q-learning. However, for the update step, we run a tree based search and use that action instead of using maximum of $Q_n(s, a)$.

.

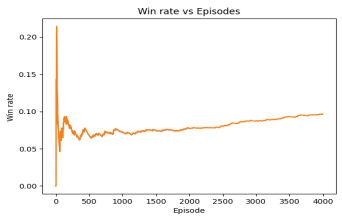

Fig. 6. DQNN trained for X, with best action chosen using MCTS.

This shows lower results because it couldn't be trained to completion, which requires ~ 40000 runs.

CONCLUSIONS

As has been established in previous papers, the game is NP-hard. Hence Reinforcement Learning is a reasonable choice for approaching this problem. Moreover, as was expected from our setting and consolidated by our observational results, the game is asymmetrically biased against Mr X.

Generally speaking, Tree Search based algorithms showed more promising results due to

- It is based on actual simulations
- DQNN training to completion requires more computational power than what was at our disposal

Training against adversaries gives slightly better results than normal case, but at the cost of much higher training time. A point worth noting is that training with adversary gives higher testing win rate. Thus, if adversarial training was run to completion then results are expected to improve noticeably.

VIII. Future Work

Due to limited resources and computational power at hand at the time of our work, we could not realise the following methods. However, following our work mentioned here, we

- aim to run above mentioned algorithms to their full for obtaining the true expected results, given enough computation power/time
- aim to implement a Graph Neural Network to model the positions
- employ an asynchronous Actor-Critic model instead of Q-learning
- employ Double Q-learning for stable convergence of loss

References

[1] "Monte Carlo Tree Search for the Hide-and-Seek Game of Scotland Yard" by A.M Nijsen and Mark H.M Winands, IEEE transactions on Computational Intelligence and AI in games.
[2] "Learning to play Monopoly: A Reinforcement Learning Approach" by Panagiotis Bailis, Anestis Fachantidis and Ioannis Vlahavas
[3] " Deep Reinforcement Learning using Graph-Based State Representation " by Vikram Varadpande, Daniel Kudenko and Megha Khosla
[4] "Deep Reinforcement Learning using Mixed Convolutional Network " by Yanyu Zhang and Hairuo Sun.
[5] " Asynchronous methods for deep reinforcement learning" by Volodymyr Mnih, Adrià Puigdomènech Badia1, Mehdi Mirza1, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, Koray Kavukcuoglu, ICML 2016
[6] " Monte Carlo Tree Search, A New Framework for Game AI " by Guillaume Chaslot, Sander Bakkes, Istvan Szita and Pieter Spronck, Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference
[7] "Adversarial neural networks for playing hide-and-search board game Scotland Yard" by Tirtharaj Dash, Sahith Dambekodi, Ajith Abraham, Neural Computation and Application (2020)