Greedy Saboteur Agent with Perfect Information Monte Carlo

Yixiong Sun, Zhizun Wang

1 INTRODUCTION

In the project, we aim to design a saboteur player that can successfully beat the random player the majority of the time. To achieve the goal, we developed two different agents for the game: an agent combining a greedy strategy with perfect information monte carlo (we will call it greedy agent for convenience), and a minimax agent with alpha-beta pruning. The former one had an outstanding performance when it played against the random player, so it was selected for submission.

This report presents the detailed motivation and development process for the greedy agent. It explains the logic and algorithms supporting the greedy agent. It also goes over the minimax agent, and briefly analyzes its weakness. Finally, it demonstrates how to improve the agent's performance in the future.

2 BEST APPROACH AND MOTIVATION

2.1 Motivation

Monte Carlo Tree Search (MCTS) is a popular AI search algorithm that is widely used throughout many games, notably Go to a high success, beating high level professionals. Thus, we were motivated to use MCTS in our AI agent for Saboteur. The main principle behind MCTS involves exploring the most promising moves by randomly simulating games and keeping track of the score. With a careful balance between exploration of new moves and exploitation of known moves to work, MCTS can select moves with a strong possibility of winning.

2.2 Theoretical Basis

There were 2 main issues however when implementing MCTS for the game Saboteur. First, Saboteur is a game with hidden information and non-deterministic outcomes. Our agent cannot see the hand of the other player nor the remaining cards in the deck. Consequently, the card drawn each turn from the deck is random due to the hidden order of the cards. Second, the number of goal states, ie. the positions of the board that leads to a win for our agent is vastly outnumbered by the number of non goal terminal states. In this setup of Saboteur, most games will terminate in draws for both players assuming random play.

To address the first issue, we decided to implement a modified version of MCTS called Perfect Information Monte Carlo (PIMC) [1]. Here, a determinization is defined as an instance of the game where the information is fully observable and moves are deterministic. The hidden information is chosen from the possible set of states, so for Saboteur, the opponent's hand and deck order is chosen from all the possible configurations. Then by

repeated sampling of determinizations, the algorithm performs MCTS on the sampled deterministic game. Finally, the results are combined and the best overall move across the samples is chosen. Although there are some flaws in simply averaging over random samples, in practice, PIMC has worked well for many game applications.

We address the second issue by incorporating a naive greedy agent combined with expert knowledge. Due to the large search space and high number of terminal states that do not result in a win, we decided to begin play with a greedy agent and apply PIMC when the greedy agent fails to suggest any reasonable move. The greedy agent attempts to play tiles that minimize the Manhattan distance to the goal state, with vertical distance taking priority. Combined with expert knowledge on when to play Maps, Malus and Bonus, the combined strategy was effective in winning most games against a random AI.

2.3 Implementation

The implementation of the agent involves three main classes: *GameState*, *DeterministicGameState* and *MCTS*. Each round, we retrieve the Saboteur board, cards in hand and Malus status, and we update our known information in the *GameState* class. We can determine what move the opponent made with the exception of Maps and Drops and we keep track of possible remaining cards in the deck. Following this, we use our greedy strategy to suggest a move combined with expert knowledge. The expert knowledge prioritizes Map cards to reveal the hidden goals, Bonus cards to remove Malus effects and aggressive Malus play to hinder the opponent. This is followed by a greedy play for Tile and Destroy cards that minimize distance towards the goal. If the greedy agent cannot suggest any reasonable moves, we begin PIMC. While the time limit is not exceeded, we determinized the game state into the *DeterministicGameState* class and performed 100 rounds of MCTS on the sampled game state. Finally we take the move that returns the most number of wins.

3 ADVANTAGES AND DISADVANTAGES

3.1 Advantages

Throughout our tests of the greedy agent, we found that although the greedy strategy only aims to reach the goal as fast as possible, it successfully beats the random AI a majority of the time. Notably, the time taken for a greedy decision is very fast and does not run into problems of exceeding the time limit at all. Combining the greedy move with PIMC, we have a contingency plan for when the best move is not obvious to the agent, simulating multiple possible playthroughs and taking the best move.

Moreover, the implementation of the greedy strategy is easy to understand, because the agent is greedily looking for the moves which are closest to the hidden objective. As a mainstream

measure of the proximity to the destination, the Manhattan distance is quite intuitive when being used as the evaluation function of the greedy agent.

3.2 Disadvantages

The main disadvantage of our approach is that the agent is too greedy. Although we added PIMC as a backup plan, as long as the greedy strategy can find a move that minimizes the distance to the goal, the agent takes the greedy move. As a result, the agent never wins against Human opponents since the Human can always bait the agent to set up the win for themselves. There was the possibility of swapping purely to PIMC after a certain threshold of distance or cards remaining in the deck was achieved but we found that in our experiments, the greedy strategy comes out on top when playing against a random AI.

The second disadvantage is that our PIMC implementation does not sample enough determinizations from the game state, mainly due to the limited time constraint. We only manage to achieve from 10 to 100 samples, depending on the stage of the game, and as a result, our best move is still lacking information about many possible game configurations. Furthermore when simulating game play in MCTS, we assumed random play, however there is some domain specific knowledge that could be applied to shrink the search space.

4 OTHER APPROACHES

4.1 Minimax Agent with Alpha-Beta Pruning

Minimax is one of the approaches that we first came up with. In any search algorithm, searching the best possible solution from the pool of every possibility known can lead to the construction of the whole state search space, popularly called a minimax algorithm [2]. This may cause impractical time complexities which may not be suitable for real time searching operations. For example, in this saboteur game with a 14 by 14 board, we have a fairly large state space. Given the time limit, minimax might not finish the third level of the search tree or below. Alpha-beta pruning, therefore, is applied in order to reduce the computational time.

To start pruning, beta and alpha are initialized to infinity and negative infinity, respectively. At a maximizing player's node, we update alpha if it is less than the alpha of the new child of the node. At a minimizing node, beta is updated if it is greater than that of the new child. Whenever alpha is larger than beta at a node after being updated, the rest of its sub-tree will be pruned. Instead of making the agent search for the whole state space, we skip the unnecessary branches, which reduces the time by a significant amount.

4.2 Utility Functions of Minimax Agent

This minimax agent with alpha-beta pruning depends on the crucial utility functions, which can be completely different in various games. The principle in the design of utility functions, in this game, is assigning greater values to better moves and smaller values to worse ones. For instance:

- When the opponent attacks our agent using a malus card, a bonus card is considered optimal by the utility function.
- After the nugget is revealed, the value of a map card becomes very small, and the value of dropping a map card becomes extremely high.
- The closer the player is to the target, the greater importance a malus card has, because the player would like to block the opponent and increase the chance of winning.
- A tile card can have no path going through it, such as {{0,1,0}, {1,0,0}, {0,0,0}} and {{0,1,0}, {0,0,0}, {0,1,0}}, so the player can be blocked by playing it.The utility function will assign lower values to such inferior tile cards. Destroying one of these cards will also get a higher preference over destroying others.

4.3 Analysis of Minimax Agent

The performance of our minimax agent is disappointing. Tested by the autoplay for hundreds of turns, the agent has an overall draw rate of 95% and an overall winning rate of 5%. Although it never loses a game, it can only win 1 game out of 20, on average. On the other hand, the average running time is very fast. Each game can finish within 1 second, because 100 games only take approximately 90 seconds in total.

Theoretically, the minimax algorithm is optimal against an optimal opponent, because it maximizes the worst-case outcome for the maximizing player. However, in the saboteur game, the agent plays against a random player that never tries to maximize its own benefit. In this case, it is possible that the minimax agent fails to select a better move with respect to the random opponent, thereby having sub-optimal performance. This means that some other superior strategies may be applied in this saboteur game.

Moreover, the design of the utility functions is difficult in that good moves and bad moves are sometimes hard to define. How to determine the actual value of a move is another problem without a clear answer.

5 FUTURE IMPROVEMENTS

5.1 Possible Improvements On the Greedy Agent

The simplest improvement to the main disadvantage of the greedy agent is to change the style of play when the goal is near. A passive strategy can be adopted, where the agent attempts to wait for the opponent to set up the second last Tile before victory or to only continue along

after an analysis of the probability of remaining cards. This however can still result in many draws and would only work against optimal or close to optimal opponents.

Concerning the PIMC, we can use an evaluation function instead of fully simulating moves in MCTS. By designing an evaluation function, we can increase the number of samples taken by PIMC and obtain a move that better represents the possible game configurations. Combining this with domain specific knowledge, we can minimize the number of states MCTS has to search through and further reduce time taken to analyze the best move.

5.2 Possible Improvements On Minimax Agent

Now that we have learnt the importance of greedy strategy in the saboteur game, we can try to develop a player whose first part is greedy algorithm and second part is minimax with alpha-beta pruning. This player will try to greedily compete in the game based on Euclidean distance or Manhattan distance. Only if it gets stucked will it use the minimax algorithm.

Another possibility is the combination of PIMC and minimax. In the future we may first try to obtain a list of preferred moves by monte carlo tree search, and then perform minimax with alpha-beta pruning on these moves. We do not know if the order makes a difference, so alternatively we may use the minimax agent first, then the monte carlo tree search after pruning.

6 CONCLUSION

The lectures particularly focus on the explanations for the minimax algorithm and MCTS, therefore we initially tried to implement them. Nevertheless, the outcomes of the project demonstrate that both of them are incapable of winning the game at a high rate, and we have to utilize a greedy strategy as well as the PIMC, which are not introduced in this course. This experience indicates that what we have learnt in the course should not be the only resource which we rely on, and that independent research plays a significant role in such applied projects.

7 REFERENCES

[1] Whitehouse, Daniel. Monte Carlo tree search for games with hidden information and uncertainty. Diss. University of York, 2014.

[2] Singhal, Shubhendra Pal, and M. Sridevi. "Comparative study of performance of parallel Alpha Beta Pruning for different architectures." arXiv preprint arXiv:1908.11660 (2019).