Project Part B Report

Shuyuan Gao & Hong Thuan Ta

Introduction

In this report, we discuss our approach to creating a competitive game-playing program for Infexion. We will describe the methodology, performance evaluation, and supporting work behind our program, which combines the Monte Carlo Tree Search algorithm with Upper Confidence Bounds and a heuristic evaluation function to make informed decisions in the game.

Approach

We are using MCTS, Monte Carlo Tree Search algorithm to drive our agent. It was tested for a couple of times and the result are not satisfying. Therefore, we improved our algorithm by introducing UCB, Upper Confidence Bound. In Performance Evaluation section, I have listed table showing the other agents we created to compete with our agent.

Why UCB?

At the beginning, we implemented a basic version of the Monte Carlo Tree Search (MCTS) algorithm. This involved enumerating all possible actions from the current board state, and simulating games where both players took random actions until the game ended or a certain number of steps were taken. We ran each simulation 10 times and stopped after a maximum of 10 steps, calculating the win rate at each node. We used a total time limit of 180 seconds, and could only spend up to 1 second per step. To ensure accuracy, we needed at least 10 simulations per node, and determined through experimentation that 10 steps were ideal. We chose the action corresponding to the node with the highest win rate as the optimal result. This basic version of MCTS performed well in competitions against random agents.

However, the basic MCTS algorithm had a major flaw: it assumed that both players took random actions during simulations, which is not realistic in actual gameplay. To address this issue, we added Upper Confidence Bounds (UCB) optimization, which maintains a tree of all possible moves. At each step, we traverse the tree by selecting the node with the highest UCB score, until we reach a leaf node. We then simulate random moves until the game ends and update the win-loss statistics for the entire path, from the leaf node to the root of the tree. After the entire learning process, we choose the action corresponding to the node with the highest win rate among the root node's children. This improved version of MCTS performed even better in testing.

Upper Confidence Bound

The exploitation factor is calculated as the ratio of the number of times an action wins to the total number of times it is selected in simulations. Using this ratio, the value of each action is estimated based on historical data from previous simulations. The exploration factor is calculated using the UCB algorithm, which balances the exploitation of good moves with the exploration of unexplored moves. The UCB algorithm uses a combination of the exploitation factor and a measure of the exploration potential of each action, which is inversely proportional to the number of times the action has been selected in previous simulations. The exploration potential of each action is represented by the UCB score, which is calculated as follows:

exploration bonus = C × √(log(total number of simulations) / number of times the action is selected)

The parameter **C** controls the balance between exploitation and exploration and is typically set to a value between 1 and 2. In our program, we set **C** to 1.414, which has been shown to be effective in practice.

The evaluation function we used in our program is based on a combination of heuristic features that capture game strategic motivations. These features include the number of pieces controlled by each player, the distance of pieces to important areas of the board, and control of important positions on the board. The weights of these features are learned through trial and error. We found that using a combination of these features in our evaluation function resulted in a more robust and effective function than using individual features alone.

Overall, our game program combines the strengths of MCTS and UCB to select actions that balance exploration and exploitation, and uses an evaluation function based on heuristic features to estimate the value of game states. The combination of these techniques results in a strong player that can compete with human players in complex game environments.

Performance Evaluation

We assessed the effectiveness of our program by conducting test matches between our three agents -- MCTS with UCB, MCTS without UCB and random generator. The random generator is taking random steps without the support of an algorithm.

More Agents:

In order to analyze the performance and effectiveness of our MCTS algorithm with UCB, we created two additional agents, one using the standard MCTS algorithm without UCB and the other taking random moves. These agents were then set to compete with each other in multiple games, and we calculated the win rates for each agent. Through this process, we were able to demonstrate that our algorithm was the most optimal and effective approach for playing the game.

Here is an example on how we let the agents compete and record the results:

The code:

```
python -m referee <red module\> <blue module\>
```

Results:

Red: rand VS Blue: final(MCTS with UCB)

```
python -m referee agent_rand agent_final
```

Game Number	Red Player	Blue Player	Winner
1	rand	MCTS with UCB	Draw
2	rand	MCTS with UCB	Blue
3	rand	MCTS with UCB	Red
4	rand	MCTS with UCB	Blue
5	rand	MCTS with UCB	Blue
6	rand	MCTS with UCB	Blue
7	rand	MCTS with UCB	Blue
8	rand	MCTS with UCB	Blue
9	rand	MCTS with UCB	Blue
10	rand	MCTS with UCB	Blue

The full results are listed here with the code of MCTS without UCB and random:
https://github.com/GuanshiyinPusa/AI-2023-Part-B/blob/main/readme.MD

Results:

The table below shows the number of games won, lost, and drawn for each pair of agents:

Matchup	Games Won by Blue	Win Rate for red	Win Rate for Blue
Random (Red) vs MCTS with UCB (Blue)	8	20%	80%
MCTS with UCB (Red) vs Random (Blue)	0	100%	0%
Random (Red) vs MCTS without UCB (Blue)	10	0%	100%
MCTS without UCB (Red) vs Random (Blue)	0	100%	0%
MCTS without UCB (Red) vs MCTS with UCB (Blue)	3	70%	30%
MCTS with UCB (Red) vs MCTS without UCB (Blue)	1	90%	10%

From the table, it is evident that the win rate for MCTS, with or without UCB, against a random agent is quite high. However, it is crucial to note that the win rate for MCTS with UCB slightly decreases when competing against a random agent. Interestingly, the playing order also impacts the win rate when MCTS with UCB faces MCTS without UCB. MCTS without UCB achieves a 70% win rate when it plays as the red player.

A possible explanation for this phenomenon is the limited time resources required by the algorithm. With 180 seconds allocated to each player, which translates to nearly 1 second per turn, there is insufficient time for the algorithm to determine the optimal next move.

Attempted Improvements and Runtime Analysis

During the implementation of the MCTS with UCB algorithm, several improvements were made to address issues related to efficiency and accuracy. These improvements were made in a series of iterative versions, which are summarized below and put up in Github:https://github.com/GuanshiyinPusa/AI-2023-Part-B/

- V1: This was the original version of the MCTS with UCB algorithm. However, it was found that the algorithm was not performing well when playing as the blue player, resulting in a low win rate of only 20%.
- V2: To improve the accuracy of the algorithm, the maxstep parameter in the simulation function was reduced. This helped to increase the win rate, but the issue with poor performance as the blue player persisted.

- V3: To address the low win rate issue for the blue player, a modification was made to the algorithm to randomly select half of the possible actions, thus reducing the time spent on suboptimal actions. This resulted in a significant improvement in the win rate for the blue player.

- V4(agent_final): In this version, the algorithm was further optimized for efficiency. Specifically, the action list was saved when optimizing UCB, reducing the computational overhead. Additionally, the speed function was updated, resulting in a further improvement in runtime performance.

By making these iterative improvements to the algorithm, the efficiency and accuracy of the MCTS with UCB algorithm were significantly improved, resulting in a more competitive game-playing program.

Time analysis:

command	user	system	total
time python3 -m referee agent_rand agent_final	167.00s	1.19s	3:06.85
time python3 -m referee agent_rand agent_basic_MCTS	359.34s	1.72s	6:13.22
time python3 -m referee agent_rand agent_v1	146.67s	1.81s	3:10.13
time python3 -m referee agent_rand agent_v2	173.73s	0.84s	3:04.36
time python3 -m referee agent_rand agent_v3	180.92s	0.60s	3:0327

command	user	system	total
time python3 -m referee agent_final agent_rand	181.95s	0.67s	3:05.86
time python3 -m referee agent_basic_MCTS agent_rand	361.41s	0.93s	8:40.83
time python3 -m referee agent_v1 agent_rand	182.89s	0.61s	3:05.46
time python3 -m referee agent_v2 agent_rand	180.49s	0.76s	3:04.67
time python3 -m referee agent_v3 agent_rand	181.94s	0.65s	3:04.49

After analyzing the runtime of the different agents, it is clear that the agent_final has the best balance of runtime and win rate among all the tested agents. The results demonstrate that the agent_final consistently performed well in terms of runtime and was able to achieve a high win rate against the random agent. This suggests that the combination of Monte Carlo Tree Search with Upper Confidence Bounds and a heuristic evaluation function used in agent_final is an effective approach for playing the Infexion game. It is worth noting that agent_basic_MCTS had the longest runtime and agent_v2 had the shortest runtime. However, agent_v2 did not perform as well in terms of win rate as agent_final. Thus, the results suggest that agent_final is the most optimal agent among the tested agents.

Conclusion

In this report, we presented our approach to designing a competitive game-playing program for the 2023 COMP30024 Artificial Intelligence tournament. Our program utilized the Monte Carlo Tree Search algorithm enhanced with Upper Confidence Bounds and a heuristic evaluation function to balance exploration and exploitation in decision-making. The performance evaluation demonstrated the effectiveness of our approach, as our program outperformed both random agents and agents using MCTS without UCB. As a result, we believe that our game-playing program is well-suited.

Future work could focus on improving the heuristic evaluation function to better estimate game states or exploring alternative search algorithms for game tree traversal. Additionally, refining the balance between exploration and exploitation with dynamic tuning of the exploration constant could further improve the agent's performance. By continuing to develop and optimize our game-playing program, we aim to contribute to the advancement of artificial intelligence in the field of game-playing agents and beyond.