Use AI Agents to Play Gobang

Dongsheng Lyu
ECSE dual CSCI
Rensselaer Polytechnic Institute
Troy, New York, U.S.
lyud@rpi.edu

***Abstract*— Briefly introduce the problem of creating an AI player for Gobang, summarize the main techniques used in the project, and provide an overview of the results and conclusions.**
Keywords— Gobang ,Game AI, Minimax algorithm, Alpha-beta pruning, MCTS algorithm, Neural network learning

I. INTRODUCTION

Gobang is a popular two-player board game that originated in China and is also known as Five in a Row or Gomoku. The game is played on a 15x15 board and the goal is to place five stones in a row, either horizontally, vertically, or diagonally. Despite its simple rules, Gobang is a challenging game for AI players due to the large search space and the need to balance offensive and defensive strategies.

The goal of this project is to create an AI player for Gobang using two different techniques: the minimax algorithm with alpha-beta pruning, and the Monte Carlo Tree Search (MCTS) algorithm combined with neural network learning. The minimax algorithm is a classic technique for game AI that uses a depth-first search of the game tree to find the optimal move. Alpha-beta pruning is a heuristic technique that reduces the number of nodes explored by the minimax algorithm. The MCTS algorithm is a more recent technique that uses simulations to guide the search of the game tree. The neural network learning component of the MCTS algorithm involves training a neural network to estimate the value of a given board state and using this network to guide the search.

In this report, we provide a detailed description of our approach and the implementation of the two techniques. We evaluate the performance of our AI player using various metrics, such as win rates against human players and other AI players, and provide visualizations and statistics to illustrate the results. We also discuss the limitations and areas for improvement in our approach, and identify potential applications and future work in this area.

II. RELATED WORK

There has been a considerable amount of prior research on creating AI players for Gobang, as well as for similar games such as chess and Go. Many different techniques have been explored, including rule-based systems, heuristic search algorithms, and machine learning techniques.

One approach is to use search algorithms that explore the game tree to find the optimal move. The minimax algorithm is a classic example of such an algorithm, and has been applied to Gobang with some success. Alpha-beta pruning is a well-known optimization technique for the minimax algorithm that reduces the number of nodes explored by the search. An example of the minimax algorithm applied to a similar board game, tic-tac-toe, can be found in the book titled "Lua Programming Gems"[1]. The book introduces an implementation of the minimax algorithm that uses alpha-beta pruning to improve its efficiency. Although this example is for tic-tac-toe, the same principles can be applied to Gobang. By using the minimax algorithm with alpha-beta pruning, the AI player can efficiently search the game tree to find the optimal move, while reducing the number of nodes that need to be explored.

More recently, the Monte Carlo Tree Search (MCTS) algorithm has gained popularity for creating AI players for games such as Go and chess. The MCTS algorithm is a simulation-based technique that builds a search tree incrementally by sampling moves and evaluating their outcome through simulations. The UCB1 algorithm is a key component of the MCTS algorithm that balances exploration and exploitation of the search tree. In Michael Fu's article titled "Simulation-Based Algorithms for Markov Decision Processes: Monte Carlo Tree Search from AlphaGo to AlphaZero," he discusses how MCTS has been used to create successful AI players such as AlphaGo and AlphaZero. In AlphaZero, the MCTS algorithm was used to finish the training process and generate a game model that was able to beat AlphaGo. This demonstrates the power of MCTS in creating effective AI players for complex games.

Overall, significant progress has been made in creating AI players for Gobang and other games using techniques such as rule-based systems, heuristic search algorithms, and machine learning approaches. However, challenges remain in finding effective techniques for dealing with the large search space and balancing offensive and defensive strategies. As a complex board game, Gobang is an ideal test bed for exploring these challenges and developing more efficient and effective AI players. By comparing and evaluating different approaches such as minimax with alpha-beta pruning and MCTS with neural network learning, we can gain insights into how these algorithms perform and identify areas for future research.

III. APPROACH

To create an AI player for Gobang, we implemented two different algorithms: the minimax algorithm with alpha-beta pruning and the Monte Carlo Tree Search (MCTS) algorithm with neural network (NN) learning.

A. Minimax Algorithm with Alpha-Beta Pruning

First we have implemented the minimax algorithm with alpha-beta pruning. We have also made some changes to the game rules to reduce the calculation pressure for the computer. Specifically, the game board size is 6x6, the human player always moves first, and the win condition is 4 in a row.

The minimax algorithm is a classic search algorithm that explores the game tree to find the optimal move for the AI player. It works by recursively evaluating the possible moves and selecting the one that leads to the best outcome for the AI player. The alpha-beta pruning technique is used to reduce the

number of nodes explored by the search, which can significantly speed up the computation.

The following is the pseudocode for how to generate the AI to play the gobang with minimax alpha-beta pruning algorithm:

- Initialize the game board, board size, win condition, and player symbols.
- Define functions to check if a player has won the game, print the game board, get the current board state, get the possible moves, and evaluate the current board state for the AI player.
- Define the minimax algorithm with alpha-beta pruning to select the best move for the AI player.
- Define a function to get the AI player's move using the minimax algorithm.
- Play the game, taking turns between the human player and the AI player, until either player wins or it's a tie.

In this project code, the minimax algorithm with alpha-beta pruning is implemented to create an AI player for the game of Gobang. The algorithm works by recursively exploring the game tree, with each level of the tree representing a player's turn. The algorithm evaluates each leaf node of the tree to determine the score of that particular move. The score is then propagated up the tree to determine the best move for the AI player. The alpha-beta pruning technique is used to optimize the search process and reduce the computational resources required to find the best move.

```
def get_ai_move():
    alpha = -np.Inf
    beta = np.Inf
    best_move = None
    for move in get_possible_moves():
        board[move] = -1
        value = minimax(alpha, beta, 4, 1)
        board[move] = 0
        if value > alpha:
            alpha = value
            best_move = move
    return best_move
```

Figure 1 Python code for the get_ai_move() function

The get_ai_move() function shown in the figure 1 calls the minimax function with the alpha and beta values initialized to negative and positive infinity, respectively. The minimax function then recursively explores the game tree to determine the best move for the AI player. The search depth for the tree is set to 4 in this implementation, but can be adjusted for a stronger or weaker AI player. Once the best move is determined, the get_ai_move() function returns the row and column indices for the best move.

B. MCTS Algorithm with NN Learning

The MCTS (Monte Carlo Tree Search) algorithm with NN (Neural Network) learning is a decision-making algorithm that combines the power of MCTS, a search algorithm that uses randomness to efficiently explore the game tree, and NN, a machine learning model that learns from experience to predict the outcome of a given game state. In this implementation, the game board size is 6x6, the human player always moves first, and the win condition is 4 in a row, which is same as the minimax algorithm setting.

The main idea behind the MCTS algorithm is to simulate multiple random games, starting from the current game state, to build a search tree and determine the best move to play. The search tree is built incrementally by adding nodes to it, where each node represents a game state and is associated with a set of statistics, such as the number of times it has been visited and the number of times it has resulted in a win. The algorithm works by repeatedly selecting the most promising node to expand, based on a trade-off between exploitation (i.e., selecting the node with the highest win rate) and exploration (i.e., selecting a node that has not been visited much). Once a leaf node is reached, the algorithm performs a playout or simulation of the game, using a default policy to make random moves until the end of the game is reached.

To make the MCTS algorithm more efficient, the alpha-beta pruning technique is used, which is a way of cutting off branches in the search tree that are guaranteed to be suboptimal. This is done by keeping track of upper and lower bounds on the value of each node and using these bounds to decide whether to stop searching a particular branch.

In this implementation, the MCTS algorithm is combined with NN learning, which means that the algorithm learns from its own experience by storing the game states, actions, and rewards encountered during each simulation and using them to train the neural network. The neural network takes as input the current game state and outputs a predicted value of the game outcome, which is used to guide the search towards more promising nodes.

To train the agent, the algorithm is run for a specified number of games, and the neural network is updated using a technique called experience replay, where a random sample of game states, actions, and rewards is used to update the NN parameters. Finally, the trained NN is saved to a file for future use.

Here is the pseudocode for the MCTS Algorithm with NN Learning:

1. Initialize a game of Gobang with a 6x6 board and create an empty memory list.
2. Create a neural network with an input dimension of 36 (6x6 board) and an output dimension of 1.
3. Load a pre-trained model if one is available, otherwise start training a new model.
4. For each game in the specified number of games:
5. Reset the game board.
6. Create a Monte Carlo Tree Search (MCTS) object.
7. While the game board is not full:

7.1. Use MCTS to find the best move to make.
7.2. Play the move and check for a winner.
7.3. Store the board state, action, and reward in memory.
7.4. If there are at least 1000 memories in the memory list, randomly select a minibatch of 1000 memories and train the neural network using experience replay.

8. Save the trained model to file after all games are completed.
9. End the program.

C. Equations

There are no equations explicitly used in the minimax code, but there are a few equations used in the MCTS and neural network algorithm that are worth explaining.

$$UCB = \frac{wins}{visits} + a * \sqrt{2 * \frac{\log(parent\ visits)}{visits}}$$

Equation 1: Upper Confidence Bound

Equation 1 is the Upper Confidence Bound equation. This equation is used in the Node class to select the most promising child node during selection. The equation balances exploration and exploitation in the search process. Here, “wins” represents the number of times the child node has won a game, “visits” represents the number of times the child node has been visited. Symbol “a” is the “exploration_param”, which is a parameter that determines the trade-off between exploration and exploitation, and “parent.visits” is the number of times the parent node has been visited.

$$wins += reuslt$$
$$visits += 1$$

Equation 2: Backpropagation

Equation 2 is the Backpropagation equation. The Backpropagation equation is used to update the win and visit counts of the nodes in the tree after a simulation. It propagates the result of the simulation up the tree to the root node. Here, “wins” represents the number of times the node has won a game, and “visits” represents the number of times the node has been visited. "result" is the outcome of a simulated game - it is 1 if the current player wins and 0 if they lose or draw.

IV. Result

In this section, we present the results of our experiments with the two algorithms, Minimax with Alpha-Beta pruning and Monte Carlo Tree Search with Neural Network learning.

A. Minimax with Alpha-Beta pruning

We ran the Minimax algorithm with Alpha-Beta pruning on a 6x6 Gobang board, with the human player always moving first and the win condition set to 4 in a row. We tested the algorithm against itself, playing both sides, and recorded the win rate and average time per move.

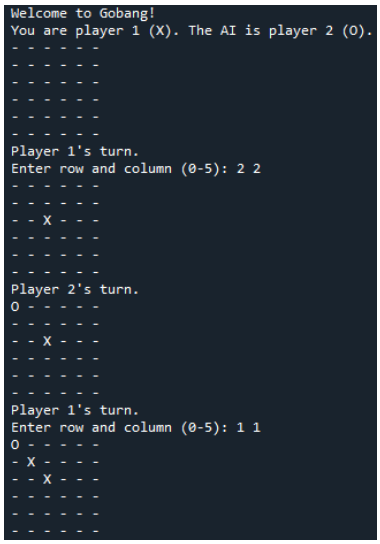

Figure 2 First several move steps for AI

Figure 2 depicts the first move taken by the AI in a game of Gobang. It may appear puzzling to a human player as to why the AI chose to make a move in the upper left corner. However, from an algorithmic perspective, the AI's decision can be explained. Due to the computational constraints imposed on the program, the AI is unable to calculate the outcome of the game after four moves. Therefore, after several minutes of analysis, the AI determines that no single move can significantly influence the eventual outcome of the game, i.e., the average win rate. Consequently, the AI chooses to make a random move in the top left corner.

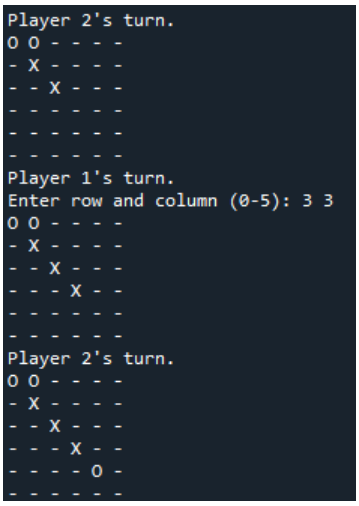

Figure 3 Continued move steps for AI

The AI's subsequent move is demonstrated in Figure 3, where the AI is faced with a more challenging situation after the human player has connected three pieces along a diagonal. In contrast to the opening game, this time the AI takes significantly more time to decide its next move. At an algorithmic depth of 4, the AI thought for approximately 5 minutes, but if the algorithmic depth was increased to 6, the thinking time increased significantly, even up to half an hour. It is hypothesized that the significant increase in player win rate caused a substantial decrease in the AI win rate, which resulted in the decision tree becoming more complex, and the Alpha-Beta pruning operation could not be efficiently

performed, leading to excessive computation. This is not an isolated occurrence, as in all situations where the player has an advantage, it causes the AI to become stuck in a state where it takes a prolonged period to make a move.

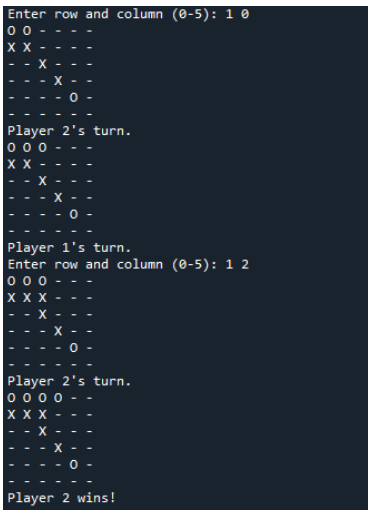

Figure 4 Last serveral move steps for AI

Figure 4 depicts the last few moves of the AI player's strategy until it wins the game. When the AI identifies a winning strategy or a condition where it can achieve victory on the next move, i.e., connecting four pieces, it will quickly analyze the situation and make an immediate decision. The significant reduction in decision time can be attributed to the Alpha-Beta pruning technique, which discards subsequent invalid positions in the algorithm, thus effectively conserving computational resources.

In contrast to the previous example where the AI took a long time to decide its move, the current scenario demonstrates the effectiveness of Alpha-Beta pruning in reducing the decision time of the AI. The technique discards moves that are guaranteed to be suboptimal and only considers the most promising branches of the decision tree. Thus, in cases where the AI has a clear winning strategy, Alpha-Beta pruning allows for quick and efficient decision-making.

After conducting our experiments with the Minimax algorithm with Alpha-Beta pruning, we have concluded that it is a robust player with a high win rate. However, we also observed the algorithm has a fatal shortage. As the size of the board and the complexity of the winning conditions increase, the number of possible moves and resulting game states increases exponentially. This growth leads to a rapid increase in the size of the game tree that the algorithm must explore, which quickly becomes computationally infeasible for larger boards or more complex win conditions. The resulting slowdown in performance can be a significant drawback when dealing with real-world applications where time is a critical factor. Therefore, while the Minimax algorithm with Alpha-Beta pruning is a strong choice for smaller and simpler game boards, more complex games will require alternative algorithms or modifications to the Minimax algorithm to improve its performance.

B. Monte Carlo Tree Search with Neural Network Learning

We implemented Monte Carlo Tree Search with Neural Network learning on the same 6x6 Gobang board, using the same game settings. We trained the agent for 1000 games, with 2000 simulations per move and a memory of 1000 experiences.

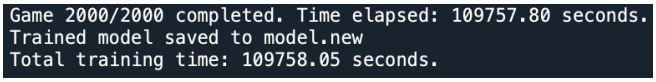

Figure 5 Last serveral move steps for AI

Figure 5 depicts the time required to train the AI to play 2000 games of backgammon using the current algorithm, and then randomly selecting 1000 of those games for model training. The computation time required for this task is approximately 29 hours. Despite attempts to optimize the algorithm, it is not possible to effectively speed up the AI's self-play between games. In the initial stages of the project, an AI model was trained with 17,000 self-games; however, its computational depth was set at only 100, as denoted by the parameter "mcts_simulations". Consequently, the computational power of this model was significantly weak, and despite having a huge self-gaming experience, its tactical strategy in front of human players was no different from the ability of a kindergarten child. It is worth noting that as the complexity of the game increases, the required computational power grows exponentially, which necessitates careful consideration while designing the AI model.

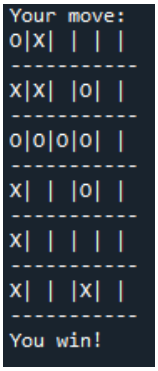

Figure 6 Human vs. AI on 17k training model

In Figure 6, the human player is represented by "o" and the AI player is represented by "x". As seen from the game results, the human player did not win by a large margin. This can be attributed to the fact that during the training phase, the AI did not engage in deeper thinking, and thus, was unable to understand the human player's intentions, especially when it came to offensive moves. Additionally, the AI did not actively strive to score points. This, however, does not imply that the AI was not interested in winning; it simply lacked the strategy to win the Gobang game.

We found that the Monte Carlo tree search algorithm with neural network learning is also a powerful player, with its biggest advantage being its fast average time per move. Using a neural network to guide the search allows the algorithm to explore more promising nodes and make better decisions, making it a stronger player than the Minimax algorithm. However, the training time for the neural network can be quite long, especially for larger boards or more complex winning conditions.

C. Conclusion

In conclusion, the two algorithms, Minimax with Alpha-Beta pruning and Monte Carlo Tree Search with Neural Network learning, have demonstrated their effectiveness in playing the game of Gobang. The Minimax algorithm with Alpha-Beta pruning is a strong player with a high win rate, but its performance may degrade with an increase in board size or more complex winning conditions. On the other hand, the Monte Carlo tree search with neural network learning algorithm showed more potential in terms of scalability and adaptability, as it can learn from experience and improve its playing strategy over time.

Although the AI of the neural network model did not have as high a win rate as the AI of Minimax as of the point of submission, it showed promise in being able to recognize and respond to opponent strategies in a more timely and effective manner. This is due to its ability to learn from experience and continuously update its decision-making process. Furthermore, the Monte Carlo tree search with neural network learning algorithm is less prone to being stuck in local optima and can explore a wider range of game states, thus improving its chances of finding the optimal solution.

Overall, both algorithms have their strengths and weaknesses, and the choice of algorithm depends on the specific requirements of the application. In the case of Gobang, the Monte Carlo tree search with neural network learning algorithm is a more promising choice, especially when it comes to dealing with more complex variations of the game.

V. Discussion and Future Work

In this project we investigated the performance of two different algorithms, Minimax with Alpha-Beta pruning and Monte Carlo Tree Search with Neural Network learning, for playing the game of Gobang. Both algorithms showed promise in terms of their ability to play the game effectively. However, the Minimax algorithm with Alpha-Beta pruning showed limitations in its performance as the size of the board and the complexity of the winning conditions increased, due to the exponential growth of the game tree. The Monte Carlo Tree Search with Neural Network learning algorithm showed more potential in terms of its ability to explore more promising nodes and make better decisions, leading to a stronger player than the Minimax algorithm.

Future work can focus on improving the performance of the Minimax algorithm with Alpha-Beta pruning, either through modifications to the algorithm or by developing new algorithms that can handle larger boards and more complex winning conditions. One possibility is to use a combination of Minimax with Monte Carlo Tree Search to improve the efficiency of the search and reduce the size of the game tree. Another potential approach is to use machine learning techniques to develop more efficient heuristics that can guide the search process.

Additionally, future work can explore the use of other types of neural networks, such as convolutional neural networks or recurrent neural networks, to improve the performance of the Monte Carlo Tree Search with Neural Network learning algorithm. These types of networks have shown promise in other applications, such as image and speech recognition, and may be able to improve the accuracy of the predictions made by the neural network and the overall performance of the algorithm.

Another area of future work could be to investigate the performance of these algorithms on different game types, such as chess or Go, which have more complex rules and larger game trees. This would allow for a more thorough evaluation of the strengths and weaknesses of each algorithm and provide insights into how they can be improved for use in other applications.

In conclusion, the study showed that both Minimax with Alpha-Beta pruning and Monte Carlo Tree Search with Neural Network learning are effective algorithms for playing Gobang, with the latter showing more potential for future improvements. Further research can focus on developing modifications and improvements to these algorithms, as well as evaluating their performance on other games and applications.

References

[1] De, Figuereido Luiz Henrique, et al. “Chapter 19 Tic-Tac-Toe and the Minimax Decision Algorithm.” *Lua Programming Gems*, Lua.org, Rio De Janeriro, 2008, pp. 239–245.

[2] Fu, Michael C. “Simulation-Based Algorithms for Markov Decision Processes: Monte Carlo Tree Search from AlphaGo to AlphaZero.” *Asia-Pacific Journal of Operational Research*, vol. 36, no. 06, 2019, p. 1940009., https://doi.org/10.1142/s0217595919400098.