The Minimax Algorithm for Game Playing

Divyam Gupta, 2201AI48

April 19, 2024

Abstract

The Minimax algorithm is a crucial concept in game theory and a part of the adversarial search. The mini-max algorithm helps players in two-player games choose their moves optimally. It looks at all possible outcomes and picks the move that gives the best chance of winning and minimizes the maximum possible loss.This paper provides an overview of Minimax, its implementation, pseudo-code, principles, and its applications.

1 Introduction

The mini-max algorithm is a strategic decision-making tool often employed in game theory. Through recursive or backtracking methods, it determines the best possible move for a player, taking into account the optimal responses of their opponent. In games such as chess, checkers, and tic-tac-toe, where two players take turns making moves in a zero-sum environment, the Minimax algorithm plays a crucial role in determining the optimal strategy for a player. Among the two players playing the game, one is called MAX and the other is called MIN.

2 Minimax Algorithm

The Minimax algorithm is a recursive search algorithm used to determine the best possible move for a player in a two-player zero-sum game. It operates by exploring the game tree to evaluate the possible outcomes of each move. The algorithm alternates between maximizing and minimizing the player's score at each level of the tree, assuming that the opponent also plays optimally.

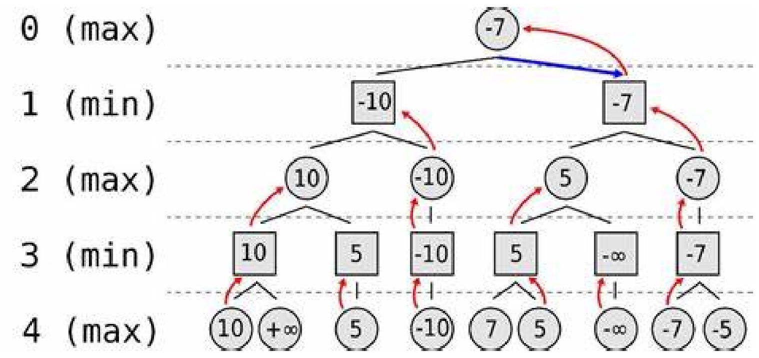

Figure 1: Sample Minimax Game Tree, credits: moreware.org

2.1 Strategy Used

For a game tree, the minimax value of each node, MINIMAX(n), can be used to determine the optimal strategy. If both players play optimally from there to the end of the game, The minimax value of a node is the payoff value (the final numeric value for a game that ends in a terminal state for the player) of being in the corresponding state. Given a choice, MAX will prefer to move to a maximum value state, while MIN prefers a minimum value state.

The Minimax algorithm can be defined as follows:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Even if one of the players (say MIN) does not play optimally, then MAX will do even better. For suboptimal opponents other strategies may do better than the minimax strategy, but will surely do worse against optimal opponents.

2.2 Pseudo Code

The following is the pseudo-code for the Minimax algorithm:

Algorithm 1 Minimax Algorithm

```
function MINIMAX-DECISION(state)
    return $\text{argmax}_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state, a$))
end function
function MAX-VALUE(state)
    if TERMINAL-TEST(state) then
        return UTILITY(state)
    end if
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s, a$)))
    end for
    return $v$
end function
function MIN-VALUE(state)
    if TERMINAL-TEST(state) then
        return UTILITY(state)
    end if
    $v \leftarrow \infty$
    for each $a$ in ACTIONS(state) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s, a$)))
    end for
    return $v$
end function
```

2.3 Explanation

- S_0: The initial state, which specifies how the game set-up at the start.
- PLAYER(s): The player which has the move in state s.
- ACTIONS(s): Returns the set of legal moves in state s.
- RESULT(s, a): Defines the result of applying action a in state s.
- TERMINAL-TEST(s): Returns true when the game is over, else false. States where the game ends are called terminal states.
- UTILITY(s, p): A utility function (objective/ payoff function), defining the final numeric value for a game ending in the terminal state s for player p.

The Minimax algorithm evaluates the possible outcomes of each move by recursively exploring the game tree. It begins at the current state of the game and considers all possible moves that the player can make. For each possible move, it simulates the opponent's response and continues this process until a terminal state is reached or a specified depth limit is reached. At each level of the tree, the algorithm alternates between maximizing the player's score and minimizing the opponent's score. The player chooses the move in such a way that their score is maximised, assuming that the opponent will choose the move that minimizes the player's score. This process continues until the entire game tree is explored, and the optimal move for the player is determined.

2.4 Properties

For the branching factor, b, of the game tree, and the maximum depth of the tree, m.

- **Complete** - Min-Max algorithm is Complete. It will definitely find a solution (if it exists), in the finite search tree.
- **Optimal** - The Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity** - The time complexity of the Min-Max algorithm is $O(b^m)$
- **Space Complexity** - The space complexity of the Mini-max algorithm is $O(b^m)$.

3 Limitations

The major drawback of the algorithm is that it becomes slow and requires a lot of computation for complex games like Go, chess, etc. These games have a large branching factor, and the player has to decide from many choices.

4 Variations

To address its limitations and improve its performance, the following variations of the algorithm are used:

- **Alpha-Beta Pruning:** It reduces the search space of the MinMax algorithm by removing branches that are not required to be investigated.
- **Monte Carlo Tree Search:** It explores the game tree using random sampling. The method can determine the best move by randomly choosing moves by exploring more nodes in less time.
- **MinMax with Iterative Deepening:** The combination of MinMax algorithm and iterative deepening the technique can probe further down the game tree via iterative deepening without needing more memory.

5 Applications

The Minimax algorithm is widely used in various board games and video games (GameAI), where decision-making is crucial for success. For example, chess, tic-tac-toe, poker, checkers, etc. The Minimax algorithm is also used in other applications, such as financial planning, resource allocation, auction, negotiation, where decision-making in adversarial environments is required.

6 Conclusion

The Minimax algorithm is a powerful tool in the field of artificial intelligence, particularly in decision-making for adversarial games. Its ability to determine the optimal strategy for a player in a two-player zero-sum game makes it indispensable for game-playing AI systems. By understanding and implementing the Minimax algorithm, AI developers can create intelligent agents capable of competing with human players in a wide range of games and other applications.

References

1. Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, 2016.
2. https://www.scaler.com/topics/artificial-intelligence-tutorial/min-max-algorithm/
3. https://www.javatpoint.com/mini-max-algorithm-in-ai
4. Class notes of CS249 AI-1 course provided by Dr. Chandranath Adak