Exploring the Alpha Beta Pruning In Minimax Algorithm in Tic Tac Toe

Abstract

This research paper delves into the application of the Minimax algorithm in the context of Tic-Tac-Toe, exploring its strategic decision-making process and addressing challenges posed by the game's vast solution space. Beyond the classic Minimax algorithm, the study incorporates Alpha-Beta Pruning as an optimization technique, showcasing its efficacy in streamlining decision trees while maintaining strategic depth through pseudocode and analysis. To simulate natural difficulty in AI gameplay, Depth-Limited Minimax is introduced as a method for the creation of an AI that is at the opponent's skill. Exploring modifications for natural difficulty reveals a captivating trend when AI competes against itself, converging toward a middle rating after 10 games. This trend highlights the delicate balance achieved by introducing elements such as randomness in decision-making, handicaps, and limited knowledge of the game state. Future directions propose extending these concepts to more complex games and integrating machine learning techniques for adaptive AI, promising continued advancements in gaming AI research and application. Questions remain regarding why AI converges despite equal difficulty settings or when both are on opposite ends of the spectrum. This comprehensive exploration provides valuable insights into the dynamics of the Minimax algorithm, optimization through Alpha-Beta Pruning, and the challenges and possibilities in simulating natural difficulty for AI in strategic games.

Introduction

TicTacToe is a classic game with a history that extends over 3500 years, making it a timeless pastime. From primitive sketches on ancient Egyptian rocks to strategic matches played on wooden boards or scraps of paper, the game's enduring charm emanates from its straightforward rules and easy-to-grasp nature. It continues to captivate enthusiasts, offering a simple yet engaging respite from monotony. Beneath its apparent simplicity, however, lies a vast solution space, rendering it an ideal arena for testing artificial intelligence (AI) techniques. This paper delves into the performance of the Minimax algorithm in the context of Tic-Tac-Toe, presenting the creation of an AI player designed to replicate the strategic abilities of its opponent. This is achieved through the incorporation of randomness, limited vision, and elements of human psychology into the Minimax algorithm, elevating the game to new dimensions of strategy and challenge.

Section 1: Overview of the Minimax Algorithm

1.1 Classic Tic Tac Toe Setup

In its traditional rendition, Tic-Tac-Toe unfolds as a thoughtful encounter between two players, where they strategically position symbols (X or O) on a 3x3 grid. The objective is to form a horizontal, vertical, or diagonal line of three identical symbols. Despite its initial simplicity, the game conceals a vast solution space, boasting a staggering 3^9 potential game states. Filling the nine spaces translates into navigating through 9! (362,880) possible sequences, each portraying a distinct configuration of the game pieces.

Section 2: The Minimax Algorithm

2.1 Adversarial Search

Adhering to the principles of adversarial search, the Minimax algorithm operates in a dynamic interplay with an opponent harboring opposing objectives. It encapsulates winning conditions by assigning values of (-1) for one side and (+1) for the other. The minimizing faction seeks the lowest score, while the maximizing counterpart aspires to achieve the highest score. This algorithm meticulously simulates all conceivable games stemming from the current state until a terminal state is attained, with each terminal state carrying a valuation of (-1), 0, or (+1). Tailoring its predictions to the player's turn, the algorithm systematically assesses whether the optimal action leads to a lower or higher value, progressively assigning values to every potential action.

2.2 Recursive Procedure

Within the recursive framework, the maximizing player engages in a thoughtful examination of potential actions and their subsequent outcomes. When contemplating a move, the maximizing player strategically ponders the actions the minimizing player would undertake to attain the lowest possible value. This recursive reasoning toggles between minimizing and maximizing perspectives, systematically assigning values to prospective states. The overarching objective of the algorithm is to maximize gains, whether by securing three in a row or strategically thwarting the opponent's advances. This culminates in the maximizing player strategically selecting the option with the highest assigned value.

2.4 Minimax Pseudocode Overview

Outlined is the pseudocode for the Minimax algorithm, illustrating its recursive nature. Functions include Max-Value and Min-Value, and the algorithm proceeds by iteratively selecting actions that lead to the highest or lowest values, respectively.

```
Function Minimax(state, depth, maximizingPlayer)
    If depth is 0 or the state is a terminal state
        Return the utility value of the state

    If maximizingPlayer
        BestValue = -∞
        For each child state in Actions(state)
            Value = Minimax(child, depth - 1, False)
            BestValue = max(BestValue, Value)
        Return BestValue

    Else (minimizingPlayer)
        BestValue = +∞
        For each child state in Actions(state)
            Value = Minimax(child, depth - 1, True)
            BestValue = min(BestValue, Value)
        Return BestValue
```

2.5 Min Max Analysis

While the Minimax algorithm exhibits a notable prowess in identifying the optimal move for the computer, its primary drawback resides in the imperative to traverse the entire game tree. This requirement imposes substantial demands on both time and computing power. It excels in scenarios where the human opponent adheres to optimal play; however, this assumption may not always align with reality. If the human diverges from flawless play, the computer's outcome remains, at a minimum, as favorable as it would be against a flawlessly playing opponent. This nuanced dynamic underscores the algorithm's sensitivity to the assumption of optimal human play.

Section 3: Optimizations

3.1 Alpha-Beta Pruning

Alpha–beta pruning stands as a strategic optimization technique for the minimax algorithm, with the overarching goal of diminishing the number of nodes assessed within the search tree. This technique intervenes by discontinuing the evaluation of a move when there is conclusive evidence that the move is inferior to a previously scrutinized one. Illustrated through a Tic-Tac-Toe example, alpha-beta pruning steers clear of unnecessary computations, presenting a more efficient alternative to the exhaustive search mandated by the conventional minimax approach. The algorithm strategically establishes initial evidence, bypassing further exploration of actions that unequivocally prove to be less advantageous than those already thoroughly evaluated. This refinement in the search process significantly enhances the algorithm's efficiency.

3.2 Alpha Beta Pruning Pseudocode

The pseudocode for Alpha-Beta Pruning extends the Minimax algorithm by introducing alpha and beta parameters, representing the best values found for the maximizing and minimizing players, respectively. This optimization allows the algorithm to cut branches of the tree that won't positively contribute to the goal state. The algorithm efficiently utilizes space and time by avoiding unnecessary computations.

```
Function MinimaxAB(state, depth, alpha, beta, maximizingPlayer)
    If depth is 0 or the state is a terminal state
        Return the utility value of the state

    If maximizingPlayer
        BestValue = $-\infty$
        For each child state in Actions(state)
            Value = MinimaxAB(child, depth - 1, alpha, beta, False)
            BestValue = max(BestValue, Value)
            alpha = max(alpha, BestValue)
            If beta <= alpha
                Break
        Return BestValue

    Else (minimizingPlayer)
        BestValue = $+\infty$
        For each child state in Actions(state)
            Value = MinimaxAB(child, depth - 1, alpha, beta, True)
```

```
BestValue = min(BestValue, Value)
beta = min(beta, BestValue)
If beta <= alpha
    Break
Return BestValue
```

3.3 Alpha Beta Pruning Analysis

To elevate the Tic-Tac-Toe algorithm, consider incorporating the following features:

- Immediate Winning Move: The algorithm can be augmented to identify and execute an immediate winning move for the computer.
- Blocking Human Wins: Enhance the algorithm to recognize and block potential winning moves for the human on the next turn.
- Efficient Win Recognition: Implement a mechanism to cease searching and make a move if a particular action leads to a win.
- Randomized Search Order: Introduce randomness into the search order within the tree, potentially facilitating more efficient pruning.

The fourth feature, randomizing the search order, involves evaluating child nodes in a random sequence, contributing to enhanced pruning efficiency. When combined with the other features, this approach significantly reduces the total number of generated nodes in the opening move of a 3x3 Tic-Tac-Toe game, underscoring the effectiveness of the optimized Alpha-Beta Pruning.

Conducting a complexity analysis reveals that randomness has a negligible impact when forced steps are enabled in Tic-Tac-Toe. However, in the absence of pruning, randomness proves beneficial by reducing the number of tested nodes across multiple trials. The overall complexity of the Alpha-Beta Pruning, with improved features, is determined to be O((nM)^N), highlighting its efficiency in both space and time utilization.

The research concludes by presenting an enhanced Alpha-Beta algorithm for Tic-Tac-Toe. This algorithm strategically selects moves, streamlines the state space through effective pruning, and minimizes recursive calls. The calculated time complexity of the improved algorithm is O(bM), signaling reduced execution time owing to a diminished need for function calls.

Section 4: Natural Difficulty

4.1 Method for simulating matching difficulty/learning

In the realm of simulating difficulty and learning, the Depth-Limited Minimax method emerges as a strategic approach. With a staggering 255,168 possible Tic-Tac-Toe games and an astronomical 10^{2900} potential games in Chess, the conventional minimax algorithm faces computational challenges, especially in the context of chess.

Depth-Limited Minimax strategically narrows its focus by considering only a predetermined number of moves before halting, without reaching a terminal state. This limitation, however, prevents the algorithm from obtaining precise values for each action, as it doesn't reach the end of hypothetical games. To address this challenge, Depth-Limited Minimax relies on an evaluation function. This function estimates the expected utility of a game from a given state, essentially assigning values to states. For instance, in a chess game, the utility function analyzes the current board configuration, assesses its expected utility based on the pieces and their locations, and returns a positive or negative value indicating the board's favorability for one player over the other.

These assigned values become instrumental in determining the right action, and the efficacy of the Minimax algorithm relies heavily on the quality of the evaluation function. A superior evaluation function enhances the algorithm's decision-making prowess, emphasizing the pivotal role of accurate state valuation in the learning process. To enhance the dynamics of AI decision-making and evaluate player skill, consider implementing the following elements:

- Randomness in AI Decisions: Instead of consistently choosing the move with the highest score, inject a degree of randomness. The AI could randomly select from among the top few moves, introducing an element of unpredictability into its decision-making process.
- Handicaps for Player Advantage: Provide players with advantages, such as extra pieces in games like chess or additional turns. Handicaps can level the playing field and add strategic depth to the gaming experience.
- Limited Knowledge of Game State: Restrict the AI's awareness of the game state. This could involve making the AI "forget" some of the player's past moves or limiting its consideration to a subset of the game board. This limitation adds an interesting challenge and requires the AI to adapt to partial information.

In addition to previously mentioned metrics, consider these for gauging player skill:

- Consistency:Assess how much the player's performance varies from game to game. More consistent players are likely to possess higher skill levels, as their results are less influenced by chance.
- Improvement Over Time:Monitor whether the player's performance is improving over successive games. A player demonstrating continuous improvement is likely learning and enhancing their skills.

- Performance Against Other Players: In a multiplayer setting, evaluate the skill level of opponents the player has triumphed over or lost to. This provides context on the player's proficiency relative to others.

By incorporating these elements and metrics, you can create a more dynamic and adaptive gaming experience while gaining valuable insights into the skill development and performance of players.

4.2 Analysis of modifications for natural difficulty

The exploration of modifications to enhance natural difficulty in AI gameplay has yielded insightful observations. Notably, when AI faces off against itself, a discernible trend emerges toward a middle rating, typically ranging between 4 and 7, after 10 games. This convergence suggests a dynamic equilibrium in which the introduced elements, such as randomness in decision-making, handicaps, and limited knowledge of the game state, contribute to a balanced and competitive gaming environment. The presence of variability in AI decisions, player advantages through handicaps, and selective awareness of the game state collectively shape a nuanced playing field. The observed trend toward a middle rating underscores the effectiveness of these modifications in fostering engaging and closely contested matches, revealing the delicate balance struck between challenging gameplay and an adaptive AI learning experience. This analysis informs the ongoing refinement of AI algorithms to continuously enhance the natural difficulty and overall gaming experience.

Section 5: Conclusion and Future Directions

5.1 Summary of Findings

In summary, our investigation into the enhancement of AI gameplay has yielded valuable insights. The integration of modifications, such as introducing randomness in decision-making, implementing handicaps, and limiting the AI's knowledge of the game state, has demonstrated a compelling impact on the natural difficulty of AI gameplay. Notably, when AI competes against itself, a consistent trend emerges toward a middle rating between 4 and 7 after 10 games. This convergence signifies a delicate equilibrium achieved through the dynamic interplay of these modifications, fostering engaging and competitive matches.

5.2 Future Directions

Looking ahead, several promising avenues beckon for further exploration. One compelling direction involves extending these concepts to more complex games beyond the scope of our current investigation. Scaling up to games with intricate strategies and larger decision spaces will unravel new challenges and insights. Additionally, the integration of machine learning techniques presents an exciting opportunity for developing adaptive AI that can evolve and learn from gameplay experiences. Exploring reinforcement learning and neural networks can potentially elevate the sophistication of AI decision-making, enabling it to adapt to diverse gaming scenarios. Future endeavors should aim to harness these advancements to create more nuanced and intelligent AI systems, pushing the boundaries of gaming AI research and application.

SOURCES

1. B, S., R, V., & T, S. S. (2020). Analysis of Minimax Algorithm Using Tic-Tac-Toe. ResearchGate. Retrieved from https://www.researchgate.net/publication/346813363_Analysis_of_Minimax_Algorithm_Using_Tic-Tac-Toe
2. Samuel, S. (n.d.). Improved technique in Tic-Tac-Toe game to minimize the condition of draw using min-max over optimal strategy. ResearchGate. Retrieved from https://www.researchgate.net/publication/365944370_Improved_technique_in_Tic-Tac-Toe_game_to_minimize_the_condition_of_draw_using_min-max_over_optimal_strategy
3. Felstiner, C. (2019). Alpha-Beta Pruning. Whitman College. Retrieved from https://www.whitman.edu/Documents/Academics/Mathematics/2019/Felstiner-Guichard.pdf