Artificial Intelligence Final Project

COMP.4200 Artificial Intelligence, Fall 2022

Miner School of Computer & Information Sciences
University of Massachusetts Lowell

Solving Connect Four with AI

Applying Machine Learning to Complex Problems

Sepehr Madani, Jacob Veber

Fall 2022

Abstract

Examining the application of artificial intelligence techniques used to solve CONNECT FOUR gives us an insightful and practical study of the AI theories in a real-world application. In this paper, we observe and examine the advantages that different optimization algorithms offer, namely, `MinMax` and `Alpha-Beta` pruning, which furthers our understanding of how we may apply AI to other fields of research.

There are also strategies specific to ensuring a victory in CONNECT FOUR that reveal how a problem can be broken apart and digested, giving a foundation to build a solution from.

Keywords: machine learning; game theory; artificial intelligence, alpha-beta pruning

1 Context and Motivation

Studying the field of Artificial Intelligence (AI) has offered an abundance of new insights into how we may approach different computational problems. One particular branch of mathematics, game theory, has exploded since its merger with AI research. By researching game theory, scientists aim to develop an understanding of the causal relationships held between decision-making and their outcomes.

Such examinations can reveal findings not only in applied mathematics and computer science, but across other fields as well, such as in economics, biology, and the social sciences. These fields have been notoriously challenging to model because of how many variables one must consider when predicting an outcome. Examining environments with fewer variables and less complicated rule-sets allows us to apply findings to more complex systems. One way mathematicians have done this is by solving games of increasing complexity, in other words, making every outcome exactly predictable given a position and assuming perfect play. Games like *nim* and *Tic-Tac-Toe* fall into this category, while Chess and Go do not yet have a guaranteed winning or drawing strategy (although they are theoretically solvable)[1][5].

In this paper, we examine the game of **CONNECT FOUR** (also known as *Captain's Mistress*, *Four in a row*, among others).

[1]There exist AIs that play *Chess* and *Go* at an exceptional level, exceeding that of their respective Grand Champions.

1.1 Game Rules

CONNECT FOUR is a two-player board game with perfect information for both sides played on a vertical board consisting of six rows and seven columns. We can classify CONNECT FOUR as an adversarial, zero-sum game.

Players each have 21 chips. First player has yellow, and second player has red chips. They make moves in turns, by dropping their chips into the board from the top, falling to the lowest unoccupied space in the column. The first player to obtain four *connected* chips horizontally, vertically, or diagonally, wins (essentially, making four in a row; hence the name). If neither of the players obtain a connected four set of chips and the board is filled (by which time neither have any more chips left), the game is a draw. An example of the winning configuration for yellow is shown below.

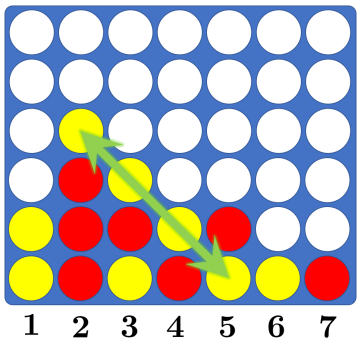

Figure 1: A winning position for yellow, achieving a diagonal four chips in a row.

1.2 Complexity

In terms of complexity, CONNECT FOUR falls somewhere between Tic-Tac-Toe and Chess, having an astounding 4,513,985,219,092 possible game configurations. [8] With Tic-Tac-Toe having already been trivially solved [6] and chess being unsolved [3], CONNECT FOUR is a perfect candidate to examine for an applied study of Artificial Intelligence.

2 Winning Strategy

2.1 History

CONNECT FOUR is shown to be a first-player-win; in other words, with perfect play, the first player can always force a win.[1]

In 1988, two people solved CONNECT FOUR through independent studies; James Dow Allen [1], and Victor Allis [2]. Computers at that time were not yet powerful enough to solve the game through sheer brute force by checking every possible move, so the two researchers detailed a series of different strategies to ensure the best chance at victory. Since then, AI methods using different optimization algorithms have emerged to solve the game with varying degrees of success and efficiency. A study of the techniques used to solve CONNECT FOUR can prove useful for gaining a better insight into AI in game theory as well as other fields.

Allis' analysis consists of discussing the ineffectiveness of a brute-force approach (citing the weakness of computational power at the time, as mentioned before) and proposing a *knowledge-based* approach. He formally outlines 9 rules named `Claimeven`, `Baneinverse`, `Vertical`, `Aftereven`, `Lowinverse`, `Highinverse`, `Baseclaim`, `Before`, `Specialbefore`, the first four outlining the most elementary rules that any experienced CONNECT FOUR player will observe. The author implements all said rules in the solver program `VICTOR`, however, a detailed discussion of this approach is beyond the scope of this paper. Therefore, in the interest of time, we will leave it to the reader [2].

Lastly, it is worth noting that CONNECT FOUR has since been solved with brute-force by creating an 8-ply database. The reader is free to explore the findings on John Tromp's Connect Four Playground [7].

2.2 Our Approach for the Strategy

2.2.1 Notation

For the discussion of the strategy, we first denote the first player playing yellow chips as A and the second player playing red chips as B. We also define a numerical coordinate system on the board, assigning 1 through 7 to their respective **columns** from left to right. Every position in the board is referenced to as a **square**.

We also describe any game position after n moves as a sequence of digits, $a_1a_2a_3\cdots a_n$, where $a_i \in \{1, 2, \cdots, 7\}$ describing the played column. For instance, the position 4435 describes the board shown to the right.

One can also verify that the positions in figure 1.1 can be described as 12345765431243322, or 34125547624332152. Note that we do not care that a position cannot be described uniquely, since our purpose is to easily feed any position to the program to begin the search algorithm from.

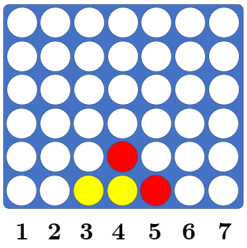

Figure 2: Position described by 4435.

3 Applying Artificial Intelligence Techniques

3.1 Heuristic Score Function

For evaluating each position for whether A or B can win, we set a scoring system as follows.

For any non final position reflecting the outcome of the game for the player to play, considering that both players play perfectly and try to win as soon as possible or lose as late as possible. A position has:

- a positive score if the current player can win. 1 if he wins with her last chip, 2 if she wins with her second last chip, and so on.
- a null score if the game will end by a draw.
- a negative score if the current player would lose regardless of what she plays. -1 if her opponent wins with his last chip, -2 if her opponent wins with his second last chip, and so on.

Positive (winning) score can be computed as 22 minus number of chip played by the winner at the end of the game. Negative (losing) score can be computed as number of chip played by the winner at the end of the game minus 22. Note that 22 is one plus the number of initial chips with each player.

For instance, position "4455" has a score of 18, because the A can win in two moves with his 4th chip, yielding $22 - 4 = 18$.

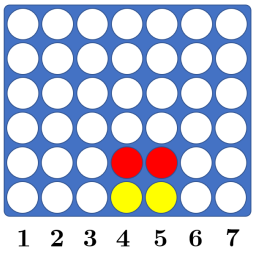

Figure 3: A winning position for A, using her 4th chip.

3.2 The `MinMax` Algorithm

The most simple technique for solving CONNECT FOUR is the `MinMax` algorithm. In this technique, we are finding the shortest path possible in the game's decision tree where each node is a position, and each move is an edge connecting two different nodes.

As previously mentioned, each possible move is given a score that describes whether the new position after that move gives the player a better or worse chance at victory, assuming best play by both players. In essense, `MinMax` aims to *min*imize the *max*imum loss that a player would sustain after each move.

This study uses a variant of the `MinMax` algorithm called **`NegaMax`** that condenses the opponent and player scores into a single value. We are leveraging that the opponent's positional value is always the additive inverse (negative) of the player's value (in other words, $S_A + S_B = 0$).

While `MinMax` provides the foundation for how the computer makes its moves, it is still a rather primitive approach to finding a solution. On its own, the algorithm is highly inefficient and will explore unnecessary pathways, such as those with a lower score than an already discovered node. In practice this leads to a disappointing performance and memory usage.

This is observed in Pascal Pons' study, in which test sets of 1,000 different board positions were separated into 6 different groups depending on the number of moves that had been played, and how many were left to play to win. All test sets with less than 29 moves played (which is considered to be well into the endgame) were unable to be calculated within 24 hours of the `MinMax` algorithm being run [4].

To make improvements to the time and unnecessary computing, we turn to Alpha-Beta Pruning for optimization.

3.3 Alpha-Beta Pruning

With Alpha—Beta Pruning technique, we "prune" the decision tree by limiting the scores examined by the program to the bound $[\alpha, \beta]$. This reduces the nodes explored in the search tree; that is, if we, for instance, have explored a move with a score of 10, there is no need to explore nodes with scores lower than 10 in the tree. Furthermore, we tighten $[\alpha, \beta]$ as we continue the search, beginning with $[\alpha, \beta] = [-\infty, \infty]$.

Another benefit of the Alpha-Beta approach is that we can easily implement a weak solver that only tells us the win/draw/loss outcome of a position by calling the evaluation function a node with the $[-1, 1]$ score window.

With the addition of this technique, the number of positions that are searched is greatly reduced. This allows for the test sets of 1,000 cases, each with more than 14 moves already played to be solved in under 24 hours [4]. This is a direct improvement from strictly using the `MinMax` function, solving the set with over 28 moves played faster, and finishing calculation on sets that took over 24 hours without Alpha-Beta Pruning.

A documented implementation of the `NegaMax` function with **Alpha-Beta Pruning** is available on page 7. For a full source code, refer to `https://github.com/1ssepehr/connect4solver`.

4 Concluding Thoughts

As we saw, even a unconvoluted technique such as the alpha-beta pruning can be employed to solve a game as sophisticated as Connect Four. Such a program can even be scaled enough to be employed as a viable computer bot for children to practice against. Furthermore, alpha-beta pruning provides the basis for more advanved techniques such as transposition tables and iterative deepening which have higher performances. However, we leave for future studies. For a more detailed reference, see [2], [4], and [7].

Appendix A. Reference Code

```
/**
 * Reccursively score connect 4 position using negamax variant of alpha-beta algorithm.
 * @param: alpha < beta, a score window within which we are evaluating the position.
 *
 * @return the exact score, an upper or lower bound score depending of the case:
 * - if actual score of position <= alpha then actual score <= return value <= alpha
 * - if actual score of position >= beta then beta <= return value <= actual score
 * - if alpha <= actual score <= beta then return value = actual score
 */
int negamax(const Position &P, int alpha, int beta) {
  if(P.nbMoves() == Position::WIDTH*Position::HEIGHT) // check for draw game
    return 0;

  for(int x = 0; x < Position::WIDTH; x++) // check if current player can win next move
    if(P.canPlay(x) && P.isWinningMove(x))
      return (Position::WIDTH*Position::HEIGHT+1 - P.nbMoves())/2;

  int max = (Position::WIDTH*Position::HEIGHT-1 - P.nbMoves())/2; // upper bound of our score as we cannot win immediately
  if(beta > max) {
    beta = max;                     // there is no need to keep beta above our max possible score.
    if(alpha >= beta) return beta;  // prune the exploration if the [alpha;beta] window is empty.
  }

  for(int x = 0; x < Position::WIDTH; x++) // compute the score of all possible next move and keep the best one
    if(P.canPlay(x)) {
      Position P2(P);
      P2.play(x);               // It's opponent turn in P2 position after current player plays x column.
      int score = -negamax(P2, -beta, -alpha); // explore opponent's score within [-beta;-alpha] windows:
      // no need to have good precision for score better than beta (opponent's score worse than -beta)
      // no need to check for score worse than alpha (opponent's score worse better than -alpha)

      if(score >= beta) return score;  // prune the exploration if we find a possible move better than what we were looking for.
      if(score > alpha) alpha = score; // reduce the [alpha;beta] window for next exploration, as we only
      // need to search for a position that is better than the best so far.
    }
  return alpha;
}
```

The full source code implementation along with instructions on how to run is available at `https://github.com/1ssepehr/connect4solver`.

References

[1] J. D. Allen. *The Complete Book of Connect 4: History, strategy, puzzles*. Puzzle Wright Press, 2010.

[2] L. V. Allis. A knowledge-based approach of connect-four. *J. Int. Comput. Games Assoc.*, 11(4):165, 1988.

[3] N. Kumar et al. Can chess ever be solved. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(2):2814–2819, 2021.

[4] P. Pons. Solving connect 4: How to build a perfect ai. May 2016.

[5] J. Rodriguez. A crash course in game theory for machine learning: Classic and new ideas, Mar 2020.

[6] S. Schaefer. Tic-tac-toe (naughts and crosses, cheese and crackers, etc.). *MathRec*, Jan 2002.

[7] J. Tromp. John's connect four playground, 1995.

[8] J. Tromp. Number of legal 7 x 6 connect-four positions after n plies., May 2012.