Intelligent Agent for Connect 4

Anna Vasilchenko & Jen Johnson

Middlebury College

Abstract

The aim of this project is to propose and compare different approaches for an imperfect Connect 4-playing agent. We aim to examine the success of purely-planning agents vs hybrid agents. Even though Connect 4 is a solved game, an imperfect agent for a game is more applicable for a consumer market. Furthermore, the fallibility of an imperfect agent allows for more meaningful comparisons of the different methods. The two algorithms that were proposed were the (1) Hardcoded-Planning Hybrid Algorithm and (2) the Chained-Based Planning Algorithm. The Hardcoded-Planning Hybrid Algorithm first detected 3-in-a-rows and either took the win or blocked the opponent. Then, it detected 2-in-a-rows and either extended the chain or blocked the opponent's chain. Finally, it chose a column using a Zero-Sum heuristic. The Chained-Based Planning Algorithm chose a column using a chain-based heuristic. This evaluates the board based on the number of pieces that the agent and the opponent have in a row, giving more weight to the opponent's chains. Both algorithms used a minimax search algorithm with a maximum depth of five and alpha-beta pruning.

The success is evaluated in terms of number of wins over 10,000 iterations when the algorithms were run against each other and against a random agent. The random agent always chose a random column at every turn. It was found that the Chained-Based Planning Algorithm performed better than the Hardcoded-Planning Hybrid Algorithm.

Introduction

The goal of Connect 4 is to get four pieces of the same color in a row horizontally, vertically, or diagonally before your opponent does. The placement of pieces is restricted by gravity--a piece will always fall to the lowest unoccupied space of any column.

Connect 4 is a solved game. This was first proven by James Allen on October 1, 1988 and later by Victor Allis on October 16, 1988. Allen and Allis solved the game independently of one another and they both used knowledge-based approaches (Sarnan *et. al*, 2009). They discovered that if the first player chooses the middle column as the first move, the first player will always, with perfect play, be able to win the game. Our motivation was to create an intelligent that did *not* abide by perfect play. It is unrealistic to design games for the consumer market with the intention of causing the consumer to always lose to the AI agent. We were therefore motivated to develop algorithms that could be realistically competitive for the average consumer. More specifically, we wished to design algorithms that would be competitive and win most of the time but would not unrealistically difficult for the human player to beat at times.

We wished to explore the success and efficiency of two methods: hard-coding and planning. The hard-coding method of a game-playing agent depends on the enumeration of all possible scenarios that the agent may encounter over the course of the game; the planning method relies on the agent being able to apply a set of rules to its current situation (Nareyek, 2002). We proposed two algorithms. The first was a hybrid approach of both the hard-coding and the planning method, while the second algorithm was purely the planning method.

The work from this project is important because it seeks to examine the efficiency of two very different approaches. It explores the benefits and drawbacks of using a hybrid approach as

opposed to a planning approach. This exploration of the balancing act between pure and hybrid methods is not limited in importance only to our specific project, but has wider significance in other AI applications as well.

Our paper will first examine the different types of algorithms that exist for Connect 4 agents. It will then give a detailed account of our approach in developing algorithms and how it relates to previous approaches. The modifications made on our algorithms and their resulting effects on the agent's success rate will be presented chronologically. Finally, there will be a discussion of the results and their significance.

Related Work

Ahmad M. Sarhan, Adnan Shaout and Michele Shock took a simple and defensive approach in creating their algorithm for a Connect 4 intelligent player (Sarnan *et al.*, 2009). Their algorithm's evaluation function was based on board rankings: boards where the agent has three pieces in a row were ranked very highly and boards where the opponent has three pieces in a row were ranked very lowly. Otherwise, all other boards were ranked as equal in value. Their algorithm would then have the agent attempt to complete one of its own three-in-a-rows (if there are any) or block one of its opponent's three-in-row (if there are any), then play randomly. They next explored a more advanced algorithm that attempted to extend (or block) two-in-a-rows as well as three-in-a-rows.

The research of Kulev and Wu (2009) found that evaluating a board based on number of unblocked chains resulted in a more effective heuristic than evaluating a board based on the number and length of neighboring pieces for each individual piece. A drawback of this approach

was that the AI agent would sometimes prioritize extending its own chain over blocking an opponent's win.

We used Sarnan *et al.*'s approach as inspiration for the hardcoded component of our Hardcoded-Planning Hybrid Algorithm. A key difference between our algorithm and Sarnan *et al.*'s algorithm is that if there are no three-in-a-rows to extend or block, our algorithm performs a search based on a simple Zero-Sum heuristic rather than simply choosing a random column.

Kuvlev and Wu's agent and its drawbacks inspired our second Chained-Based Planning Algorithm. Our improvement was to double the weight of opponent's chains in comparison to the weight of one's own chains. So, if 16 points were added for one's own two-in-a-row, an opponent's two-a-row would subtract 32 points.

Problem Statement

There are two traditional approaches to AI agents, the hard coding method and the planning method. The hard-coding method uses numerous statements to encode all of the possible actions to which the agent will have to respond (Nareyek, 2002). The agent is given a list of priorities to follow (Introduction to AI Techniques, 2009) based on the state of the board. The first priority is to win, the second priority is to block the opponent from winning, and so on. A hard-coded agent has a fast response time, but one if its major drawbacks is that it is unable to do long-term planning; it is simply a reactive agent.

The planning method is not only able to make decisions for itself using information from its current environment, but also adapt to many different situations. It enumerates possible moves ahead of the current board and then selects the best long-term move. Like the hard-coding

method, it also possess flaws: it takes a significant amount of time to search for the next best move. As new information is obtained, the agent must constantly recompute its path to a winning state; this may significantly increases the response time of the agent.

Because both methods have their flaws, another alternative approach is to create a hybrid of both (Nareyek, 2002). A hybrid agent combines the fast speed of the hard-coding method and the long-term reasoning of the planning method. By implementing a max depth, the recomputation of the planning will be limited and thereby reduce the running time.

Our project goal is to compare two different algorithms based on different approaches--one a hybrid approach and the other a purely planning approach. The first hybrid agent will perform a pre-search detection of any 2 and 3 in-a-rows. If no two or three -in-a-rows are detected, the algorithm performs an adversarial search based on a Zero-Sum heuristic. The second purely-planning agent will always perform an adversarial search based on a heuristics of weighing chainings of 2, 3, and 4 in-a-rows heuristic. Detailed descriptions and some pseudocode will be shown in the Methods and Results section.

Both of our approaches will have a planning component that will be made up of a heuristic function, a minimax algorithm, and alpha-beta pruning. As both the AI player and the opponent add pieces to the board, the board's value changes. This value is also called a utility and it is determined by a heuristic function. Therefore, a player's goal on each turn is to add a piece such that a board of better value (for the player) is obtained. For the AI agent to accomplish this goal, it will perform an alpha-beta search with the assumption that its opponent will always play optimally. In this minimax search, the algorithm is defined as the MAX player and the opponent as the MIN player. At every choice for MAX or MIN, it is assumed that MAX

will try to maximize the utility of the board while MIN will always play optimally and will try to minimize the utility.

The alpha-beta pruning method will be used to decrease the response time of the agent by optimizing its search algorithm. First, alpha is initialized to negative infinity, and beta to positive infinity. For each node, keep track of best value so far: alpha is highest value for MAX, while beta is lowest value for MIN. If it is MAX's turn, look at the alpha value. If alpha is greater than or equal to the beta value of child, MAX will always choose this node and its subsequent board states over the other child's board. Therefore, it is unnecessary to look at the rest of the board states following from this board. This assumption is called pruning because it reduces the number of board states the agent needs to look at, and decreases the running time of the search algorithm. If it is MIN's turn, look at the beta value. If beta is less than or equal to the alpha value of the child, prune.

Methods & Results

Implementation of Minimax Search and Alpha-Beta Pruning

The code for our GUI environement was borrowed from http://www.cs.unm.edu/~bchenoweth/cs152/. This code already had a basic implementation of a minimax algorithm that we used as a starting point. Our first modification was to add alpha-beta pruning that we modeled after the code from https://github.com/mattnenterprise/LWJGLConnectFour. We found that our pruning modification significantly improved the response time our AI agent (Figure 1).

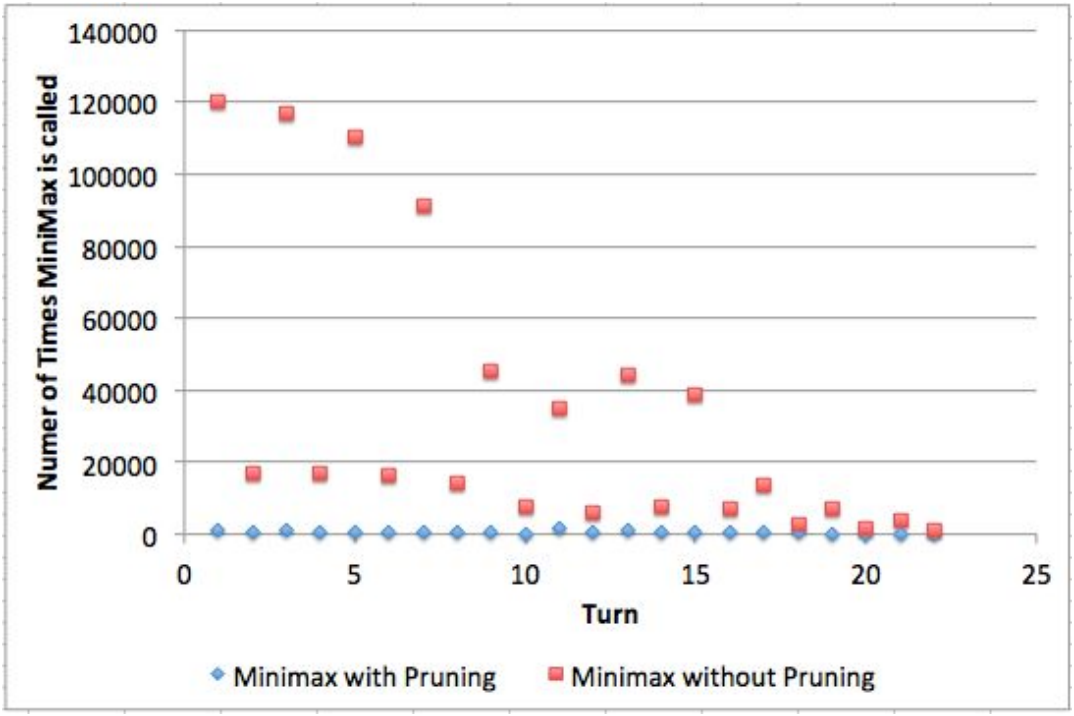

Figure 1: The red data points depict the number of times per turn that the mutually recursive Min and Max functions of the minimax algorithm were called before pruning was implemented. The blue data points are the number of calls after pruning was implemented.

1. Hardcoded-Planning Hybrid Algorithm: Pre-Search detection of 2,3 in-a-rows followed by an adversarial search based on a Win-Lose-Draw heuristic with alpha-beta pruning.

1a. Pre-Search Board Evaluation for Hardcoded-Planning Hybrid Algorithm

This is an algorithm for a hybrid agent that first attempts to make a move based on the current configuration of the board. The first possible move that the agent looks to make it extending one of its own three-in-a-rows to a winning four-in-a-row. If this winning move cannot be made, the agent sees if there are any of its opponent's three-in-rows that it can block. If still no moves can be made, it then either extends one of its two-in-a-rows or blocks one of its opponent's two-in-a-row (if both moves are available at the same time, the choice is random).

Finally, if there are still no possible moves for the agent to take, it performs a minimax search with alpha-beta pruning where the evaluation function is:

```
if (STATE is WIN for ALG): return 1
else if (STATE is LOSS for ALG): return -1
else: return 0
```

The pseudocode for the pre-search evaluation of the board is as follows:

```
(1)  Scan the board for any three-in-a-rows. If this could be made
     into a win (four-in-a-row) with a legal move, do so and exit.
     (Game ends).
(2)  Else, find all columns where a legal move would block a
     current three-in-a-row of the opponent. Store these column
     numbers in arrayList R.
(3)  Then find all columns where a legal move would either build
     upon a current two-in-a-row belonging to the algorithm or block
     a current two-in-a-row belonging to the opponent. Sort these
     columns in arrayList W.
```

To put the pre-search evaluation and the search components together, the following is done for every move:

```
(1)  If (R is not empty): choose a random col from R.
(2)  Else if (R is empty, but W is not empty): choose a random col
     from W.
(3)  Else: perform the "simple" alpha-beta search with depth limit
     of 5.
```

With our pre-search modification, an average of 13% of the turns could be made without calling the recursive minimax function. Therefore, the response time of the agent was further improved from what it had been before the modification.

1b. Running Time for Hardcoded-Planning Hybrid Algorithm

The implementation of our algorithm used a 2D array for the representation of the board. Our Hardcoded-Planning Hybrid Algorithm scans the board once for every time it detects two-in-a-rows and three-in-a-rows for those that are horizontal, vertical, left-diagonal, and right-diagonal. In the worse cases, it also needs to call the minimax algorithm with alpha-beta pruning that has a running time O(b^(d/2)). B is the branching factor, in this case the number of columns on the board, since on each turn, there are up to 7 possible moves. D is the depth of the search tree, which was set to 5 to balance between the long-term planning ability and response time. Therefore, the overall running time is O(7^ (5/2)).

1c. Results of the Hardcoded-Planning Hybrid Algorithm

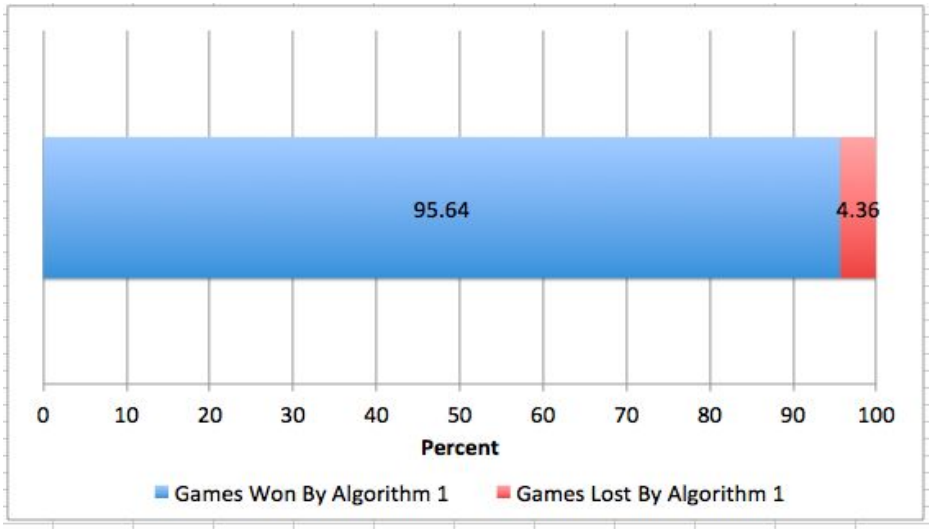

Figure 2: The success of the Hardcoded-Planning Hybrid Algorithm against an agent that plays randomly when tested over 10,0000 iterations.

2. Chained-Based Planning Algorithm: Alpha-beta pruning search based on a weighted-chainings of 2,3,4, in-a-rows heuristic

2a. Evaluation Function for Chained-Based Planning Algorithm

Our second algorithm is concerned with calculating a state's utility based on a heuristic that increases the utility for all the 2, 3, and 4 -in-a-rows belonging to the agent while drastically decreasing the utility for all 2, 3, and 4, -in-a-rows belonging to the opponent.

Similarly to the first algorithm, our second algorithm scans through the board four times--once for each time it wishes to detect 2, 3, and 4 -in-a-rows of horizontal, vertical, left-diagonal and right-diagonal nature. A chain of the opposite player is only valid in calculating the state's utility if can be blocked by the current player (ie, every valid chain must be adjacent to an empty slot that the opponent player could legally choose in order to block the chain in question). See Figure 3 for an illustration of a valid vs. invalid n-in-a-row.

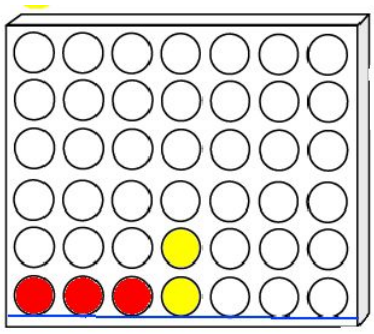

Figure 3: The red's 3-in-a-row would not affect the utility evaluation of the board since it has no adjacent blank space that yellow could go to in order to block it. On the other hand, the yellow's 2-in-a-row would affect the utility of the board since the red player can choose column 4 to block it.

The function evaluation value for a chain of n-in-a-row is as follows: `return 4^n`

If a chain is valid, it contributes to the total value of a player's chains. Because we wanted to ensure that the AI agent sees the opponent's chains as a significant danger, we chose to have the value of an opponent's chains be double the value of the agent's chains. Therefore, the total utility of a board (the output of the evaluation function) is equal to:

```
return [Value of current player's valid chains] - 2*[Value of opponent's valid chains]
```

To find its next best move, the AI agent will use an alpha-beta search to do the following:

- Assume that the opponent will play optimally by trying to minimize the utility value
- Will attempt to maximize the utility value
- Continue searching until winning state is found or max depth of 5 is encountered

2b. Comparison of Running Times Between Algorithm 1 and Algorithm 2

The running time of the purely-planning algorithm has the same running time of the hybrid algorithm during its worst case (since the purely-planning algorithm always calls an alpha-beta search while the hybrid only calls it during its worst case). The worst case for the hybrid algorithm is when there are no 2 or 3 in-a-rows of either player.

Despite the differences in running times, we did not notice a significant difference in response time between the two algorithms.

2c. Results of the Chained-Based Planning Algorithm

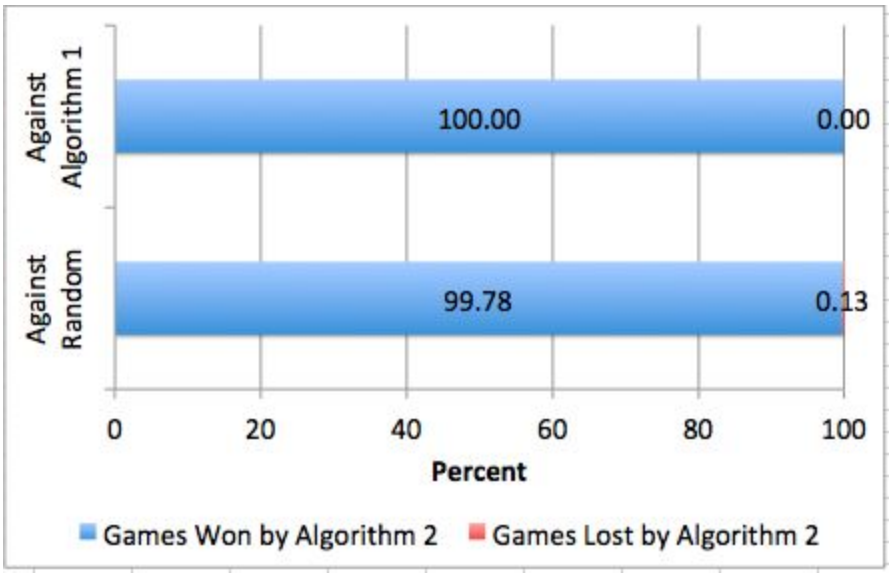

Figure 4: It was found that the Chained-Based Planning Algorithm was much more successful against the Hardcoded-Planning Hybrid Algorithm and a random agent when run for 10,0000 iterations.

Discussion & Conclusion

The Chained-Based Planning Algorithm performed significantly better than the Hardcoded-Planning Hybrid Algorithm. We believe there are a couple of reasons for this. One, it was heavily penalized for the opponent's chains: it was always deducted double the points for its opponent's chain than what it gained for its own chains. This forced the algorithm to play a very defensive game. On the other hand, our first algorithm, the Hardcoded-Planning Hybrid Algorithm, did not always prioritize defensive play over offensive play. For example, when it had multiple options between extending its own two-in-a-rows or blocking the opponent's

two-in-a-rows, it made its choice randomly. (Although it is true that it always prioritized blocking an opponent's three-in-a-row over extending its own two-in-a-row).

Another reason why the purely-planning method fared better than the hybrid method is because it was capable of performing long-term planning at every move. The alpha-beta search allowed the Chained-Based Planning Algorithm to choose the next best move not only based on the current board's evaluation but future ones as well. Although it is true that the Hardcoded-Planning Hybrid Algorithm also performed long-term planning whenever it called its own alpha-beta search, it only did so in the cases where there were no 2 or 3-in-a-rows to block or extend. Although the Hardcoded-Planning Hybrid Algorithm had a quicker response time than the Chained-Based Planning Algorithm, it was a lot more impulsive and short-term driven which significantly affected its success rate.

Future Work

Indirect Actions. Future work could be improving the hybrid algorithm further. Its current heuristic is not able to consistently detect the case where the agent could block or complete a non-consecutive three-in-row. This is a major weakness. For example, in the board presented in Figure 5, the agent will randomly choose between:

a) blocking the opponent's 2-in-a-row on the right by choosing column 5;

b) extending its own 2-in-a-row by choosing column 0; or

c) blocking the opponent 2-in-a-row on the left by choosing column 2.

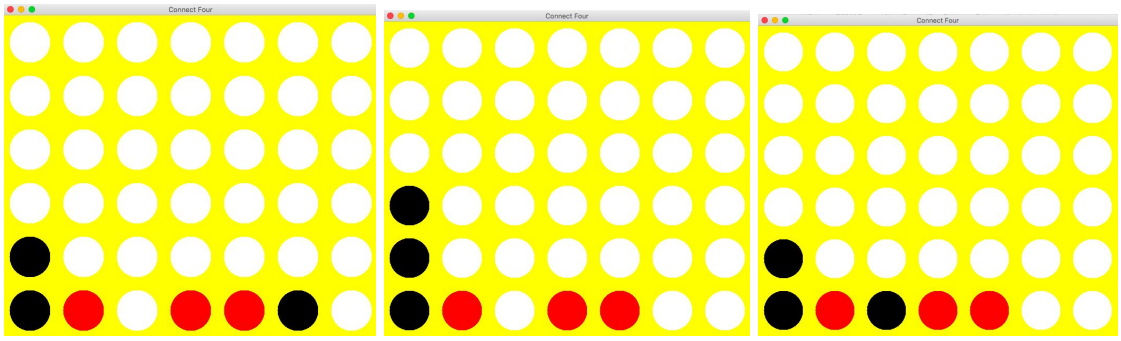

Figure 5: Out of the 7 possible moves, there are 3 moves that are beneficial to the agent: (from left to right) (a), (b), and (c) as explained above.

The optimal agent would always choose option (c), because choosing any other option allows the opponent to win on its next turn. A more advanced heuristic or even a "hard coded" method would consistently detect this type of threat or advantage and increase its success rate.

Max-Depth Optimization. The depth is currently set to 5 to retain the quick response time of the agent. A higher depth could potentially increase the long-term reasoning of the agent but could also sacrifice the response time. It would be useful to experiment with different max-depth value to obtain the best compromise between response time and planning capabilities.

Bibliography

Kulev, V. and Wu, D. “Heuristics and Threat-Space-Search in Connect 5.” 2009.

MIT Game Search. “Introduction to AI Techniques.” 2009.

Nareyek, A. “Intelligent Agents for Computer Games.” 2002.

Sarnan, A.M., Shaout, A., and Shock, M. “Real-Time Connect 4 Game Using Artificial Intelligence.” Journal of Computer Science 5(4):283-289, 2009.