CS 6700: Connect Four

May 21, 2018

Prof. Bart Selman

Apurv Sethi (as2658)

1 Introduction / Overview

For my project, I implemented an application that runs an alpha beta pruning tree on the game Connect Four with varying heuristics for evaluation, and varying tree depths. This program was written in Python and also used numpy, in order to allow easier manipulation of the board, which was constructed as a 2D arry.

2 Goals

My goals for this project were as follows:

1. Implement an alpha beta pruning tree, which could rely on varying heuristics
2. Create an agent that would be able to play Connect Four against another such agent, relying on the alpha beta pruning tree
3. Implement a varying set of heuristics, designed with increasing complexity, and thus increasing cost in terms of time
4. Input a range of depths over which the agents would be evaluated
5. Input heuristics for each of the two adversaries, which could vary
6. Output a table showing the winner of a Connect Four match for each of the depths given as an input range

3 Conceptual

3.1 Alpha-Beta Pruning Tree

Alpha-Beta pruning trees are a relatively simple way to drastically decrease runtime of a Minimax tree. Minimax trees are a great way of representing a two person, adversarial game, as they are reliant on maximizer nodes, which try to maximize the value and represent the current player, and minimizer nodes, which try to minimize the value and represent an adversary. The values refer to an evaluation of the state of the game as it stands at any given node, as the root maximizer node tries to move towards a position with the highest possible value.

Alpha-Beta trees operate under the principle: if a move can be determined to be worse than another move already examined, then there is no need for further examination of the move. The rules can be described as follows:

1. Alpha is the best (highest) path found so far along the path for the maximizer node
2. Beta is the best (lowest) path found so far along the path for the minimizer node
3. Search below a minimizer node may be pruned if the value at the node is less than alpha, which is highest value option explored along the path to the root thus far.
4. Search below a maximizer node may be pruned if the value at the node is greater than beta, which is the lowest value option explored along the path to the root thus far

These rules allow for a specific improvement that can drastically lower runtime: ordering the tree such that "good moves" are explored early in the tree. A perfect ordering would theoretically double the depth that an alpha-beta pruning tree would be able to explore in the same amount of time, compared to a random ordering.

3.2 Heuristic

As alpha-beta pruning trees are fairly straightforward, and connect four itself is a solved game, I was most interested in the heuristics that could be used to accurately evaluate the state of the board, and the tradeoff that exists in doing so between heuristics that are less accurate, but more costly.

3.2.1 Fours

I decided that using groups of fours, referring to each consecutive set of four spaces that could be filled, would be most effective and efficient in running these heuristic. This is partially also because these groups would need to be checked to find out if the game was finished anyway, so these groups could also be easily reused to gather evaluate the state of the game.

The groups below show how a regular 6x7 Connect Four board can be split into 1D arrays of length 4:

- Horizontal splicing with a set row index but increasing column index across rows is a fairly straightforward way to gather a 1D array of length four, representing a possible winning set of positions. The Xs below represent where each splice would start, and each would continue horizontally.

$$\begin{bmatrix} X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \end{bmatrix}$$

- Vertical splicing is also very similar to the horizontal splicing described above, with a set column index but increasing row index across columns. Similarly, the Xs below represent where each splice would start, and each would continue horizontally.

$$\begin{bmatrix} X & X & X & X & X & X & X \\ X & X & X & X & X & X & X \\ X & X & X & X & X & X & X \\ O & O & O & O & O & O & O \\ O & O & O & O & O & O & O \\ O & O & O & O & O & O & O \end{bmatrix}$$

- The diagonals indicated below, starting at an X moving downwards towards the right, are one of the reasons that I chose to use numpy. Grabbing these diagonals would be an extremely frequent action, and by using numpy's diagonal function with different offsets and splicing 1D arrays afterwards, it was significantly faster in the implementation.

$$\begin{bmatrix} X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ O & O & O & O & O & O & O \\ O & O & O & O & O & O & O \\ O & O & O & O & O & O & O \end{bmatrix}$$

- The diagonals indicated below are similar to those above, starting with and X and moving upwards towards the right. Once again, using numpy made using the same function described above simple after reflecting the array vertically.

$$\begin{bmatrix} O & O & O & O & O & O & O \\ O & O & O & O & O & O & O \\ O & O & O & O & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & O & O & O \end{bmatrix}$$

By creating the 69 different groups of four described above, it became much easier to evaluate different game states against each other, with different heuristics used described below.

Additionally, in the heuristics described below, let players be denoted by Os and Xs for first and second player respectively, also corresponding to -1 and 1.

3.2.2 Random Sampling

The simplest heuristic I chose to implement was random sampling, where the heuristic function would take a random sample of 10 of the above 69 groups of fours with replacement, and set the score equal to the sum of values within the groups. I chose to implement this function because it minimized the work done by the heuristic function. Though it would have a high bias problem, this function would also be the most lightweight of the functions I chose to run.

3.2.3 Sum Cubed

The next heuristics I chose to implement was one that would focus on the distance between places occupied by a player. For each group of four, the heuristic would sum the values within it, and then cube the result before adding this value to the score. The purpose of this heuristic function was to highly value groups of four that a single player managed to gain a large portion of. However, there are shortcomings. Namely, a portion of the array that looks like:

$$\begin{bmatrix} X & X & O & X \end{bmatrix}$$

Should certainly not be valued as highly as a group that looks like:

$$\begin{bmatrix} X & X & X & O \end{bmatrix}$$

While this issue is mitigated when surrounding groups of four are taken into account and should definitely be more accurate than the first heuristic described above, the shortcomings can be made up with some modifications, which were used for the third heuristic described below.

3.2.4 Valuing Singular Holdings

The heuristic I wrote hoping to be the most accurate was one that focused on groups of four spots where one player had a monopoly. Given a monopoly, the score was raised by $10^{abs(n)-1} \times sign(n)$ where n is the sum of the elements in the group of fours places. As a result, evaluation would be conducted mainly based on the number of viable groups a player could hope to use to win the game, with more priority being given to groups where the player had less work left to do.

This does still leave some issues. For example, a situation like:

$$\begin{bmatrix} - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \\ - & O & X & O & X & O & - \\ - & X & O & O & O & X & - \\ X & X & X & O & O & O & X \end{bmatrix}$$

The spot available in the middle of the board will doubly count well for X, though the player does not “win twice” by making two such groups of 4 on the board. As a result, the heuristic would largely over-count in this situation.

4 Results

4.1 Tables of results

The tables in the appendix indicate the results that were found with these setups. The left side refers to player 1, and this agent would also be the first to play. The top refers to player 2, who would play second. H1, H2, and H3 refer to Random Sampling, Sum Cubed, and Valuing Singular Holdings respectively as different heuristics. Additionally, D1 through D5 refer to the depth of the alpha beta tree when running for a given agent, from a depth of just 1 to a depth of 5.

5 Analysis

5.1 Observations

While implementing code for the agent and the different heuristics, I often printed each connect four board as moves were made, allowing me to catch obvious mistakes (and later fix them in my heuristic function). Generally, these agents were run with a depth of 4, which seemed to perform adequately well, particularly in regards to the third heuristic, valuing singular holdings. I could understand the logic behind such moves and also felt that they were largely reasonable as I finished implementing the project.

The first and second heuristics often seemed to miss an obvious choice, such as not choosing the center to start a game. Additionally, the second heuristic, sum cubed, in particular would often target the sides of a board, which seems strange to me as I don't see how the implementation for this algorithm led to this effect.

However, the first heuristics also surprised me at times, as I expected it to perform relatively poorly in compared to the other two heuristics, but found that it seemed to make more reasonable decisions then the second heuristic in particular. I believe that this may have been the result of taking 10 samples out of 69 highly interconnected groups. Though this could lead to poor decisions at times, the heuristic was clearly able to evaluate the game state as a whole much better than I had expected.

5.2 Quantitative Results

The results of these tests were surprising to me, particularly in regards to the first heuristic, random sampling. Though I had higher expectations of the first heuristic than the second heuristic after having watched the agent play against itself, I did not expect the first heuristic to perform so well. The first heuristic did not truly take into account the number of tiles placed "in a row" and this leads me, to some extent, to believe that the results given in the tables above are somewhat skewed, as there are only seven columns to choose from at most, and randomness can easily play a part in the sampling process used for random sampling. However, the results above show that random sampling was largely able to perform well against both of the other heuristics in most depth settings. Additionally, other individual tests focused on the random sampling heuristic obtained similar results.

The depth settings themselves would also have helped skew the results, as a maximum depth of 5 is not necessarily able to show the strength of a more accurate heuristic function. Unfortunately, the heuristics I chose, and particularly the way in which they were implemented, forced the constant creation of the groups of four described in section 3.2.1 and made the runtime for each of the tree searches somewhat slow. This was the case across heuristics, as I still relied on groups of four for my first and second heuristic as well.

6 Conclusion

In conclusion, I believe I was able to accomplish each of the goals that I originally set, though I was somewhat surprised by the results myself. I would have liked to build something a little more user-friendly, as I think a GUI would likely make it easier to track the changes on a Connect Four board and the wins and losses accumulated under different settings.

The alpha-beta pruning tree performed as expected, and while the tree search was somewhat slow in regards to runtime, it was still able to output reasonable decisions.

6.1 Future Work

In order to gain a more detailed understanding of the tree search and heuristic function in particular, I think that focusing on more efficient heuristic functions and exploring the tree search with greater max depths would be helpful. While Connect Four is a "solved game", I felt that there were still lessons to be learned through this implementation. The main change I would like to have made would be a speedup through a streamlining of getting groups of four out from a connect four board. Using some version of pass by reference would be very valuable here, as would the possibility of only updating the heuristics for the groups of four that have changed from the previous time the heuristic was run. I believe this would greatly decrease the cost of running any of the heuristic functions described. Another possibility would be to save the results of the evaluation function on each possible board in a separate file and using a hash of the board in order to access the score given to the board. While this would take a significant amount of time initially, later uses would be faster.

Additionally, changing the number of random samples that random sampling is taking with replacement would also be a parameter that could greatly affect the results in Table 1.

Similarly, trying other tree search algorithms, specifically those that would let the user use a timeout of sorts would also greatly affect results, as the tree search would stop at low depths when forced to do so, but would continue to higher depths when it was able to efficiently prune of larger portions of the tree. I think iterative deepening would be an interesting tree search to implement on top of the alpha beta tree as it stands.

6.2 Closing

Overall, I felt that this was a fairly successful project. I built an alpha beta tree that could be used in other situations as well, and gained a deeper understanding of the heuristic functions that drive the tree. Additionally, the project could also be modified slightly to play against a user, and change the difficulty level based on the depth the tree was allowed to search to challenge the player based on their individual level. A mobile or desktop app based on this project would also be feasible.

7 Acknowledgments

My understanding of each piece of the project grew through online publications and blogs that others had written. Specifically, the following sites / publications were especially helpful:

1. L.V. Allis, A knowledge-based approach of Connect Four: The game is over, white to move wins, M.Sc.Thesis, Vrije Universiteit Report No. IR-163, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1988.

2. Lukas Tommy et al, The Analysis of Alpha Beta Pruning and MTD(f) Algorithm to Determine the Best Algorithm to be Implemented at Connect Four Prototype, 2017 IOP Conf. Ser.: Master. Sci. Eng. 190.

3. https://www.gimu.org/connect-four-js/jQuery/alphabeta/index.html

These sites, along with a variety of other answers from users of StackOverflow, were used to construct the code and gain a better conceptual model for the heuristic functions.

8 Appendix

In the tables below, the results of a game between two agents with the given heuristics and depths, are given. H1, H2, and H3 refer to random sampling, sum cubed, and valuing singular holdings respectively. D1 through D5 refer to tree searches with max depths set to 1 through 5 respectively. The player on the left (also -1) plays first, while the player on top (also 1) plays second.

		H1				
		D1	D2	D3	D4	D5
	D1	1	1	-1	1	1
	D2	1	1	1	1	1
H1	D3	-1	-1	1	-1	1
	D4	-1	-1	1	-1	1
	D5	1	-1	-1	-1	1

Table 1: Results of Heuristic 1 vs. Heuristic 1, Depth Tests

		H2				
		D1	D2	D3	D4	D5
	D1	-1	-1	1	1	1
	D2	-1	-1	1	-1	-1
H1	D3	-1	1	1	-1	1
	D4	-1	0	1	1	1
	D5	-1	1	-1	0	1

Table 2: Results of Heuristic 1 vs. Heuristic 2, Depth Tests

		H3				
		D1	D2	D3	D4	D5
	D1	-1	1	1	-1	-1
	D2	-1	-1	-1	0	1
H1	D3	-1	1	1	1	0
	D4	-1	1	1	1	1
	D5	-1	-1	-1	1	-1

Table 3: Results of Heuristic 1 vs. Heuristic 3, Depth Tests

		H1				
		D1	D2	D3	D4	D5
	D1	1	1	1	1	1
	D2	-1	1	1	-1	1
H2	D3	0	-1	-1	1	1
	D4	-1	1	1	1	1
	D5	0	-1	-1	-1	-1

Table 4: Results of Heuristic 2 vs. Heuristic 1, Depth Tests

		H2				
		D1	D2	D3	D4	D5
	D1	0	1	0	1	1
	D2	-1	-1	1	0	1
H2	D3	-1	1	0	1	1
	D4	-1	0	0	-1	1
	D5	0	1	0	-1	0

Table 5: Results of Heuristic 2 vs. Heuristic 2, Depth Tests

		H3				
		D1	D2	D3	D4	D5
	D1	0	1	1	1	0
	D2	-1	-1	1	0	1
H2	D3	0	1	0	1	0
	D4	-1	0	1	1	1
	D5	0	1	1	1	0

Table 6: Results of Heuristic 2 vs. Heuristic 3, Depth Tests

		H1				
		D1	D2	D3	D4	D5
	D1	1	1	1	1	1
	D2	-1	1	1	1	1
H3	D3	-1	1	1	0	1
	D4	1	-1	-1	1	1
	D5	1	-1	1	-1	-1

Table 7: Results of Heuristic 3 vs. Heuristic 1, Depth Tests

		H2				
		D1	D2	D3	D4	D5
	D1	-1	-1	1	1	1
	D2	-1	-1	1	-1	-1
H3	D3	-1	-1	1	-1	-1
	D4	-1	-1	1	-1	-1
	D5	-1	-1	1	-1	-1

Table 8: Results of Heuristic 3 vs. Heuristic 2, Depth Tests

		H3				
		D1	D2	D3	D4	D5
	D1	1	1	0	1	1
	D2	-1	1	1	1	1
H3	D3	0	0	1	-1	0
	D4	-1	0	0	-1	1
	D5	1	0	0	0	-1

Table 9: Results of Heuristic 3 vs. Heuristic 3, Depth Tests