Heuristics in the game of Connect-K

Jenny Lam
Department of Computer Science
UC Irvine, California

Abstract

Heuristics are used to cope with intractably large search spaces in adversarial games. In this paper, we examine several ways to improve performance of Minimax and Alpha-Beta search in the game of Connect-k, a generalized version of the more famous Connect-4 game. In particular, we explore the use of the concept of threats as a family of features useful in the design of the heuristic. We also explore several move ordering schemes that stimulate pruning of the search tree. Finally, we attempt to implement transposition tables to improve Minimax search.

Introduction

The game of Connect-k is an adversarial game consisting of a board of m by n cells, where m and n are greater than or equal to k. Two players, which we call p_1 and p_2 take turn in placing a piece of a color representing them on the board. If gravity is off, the piece may be placed in any empty cell. If gravity is on, we add the restrict the valid cells to empty cells that are the lowest in each column of the board. The object of the game is to form a line of k consecutive pieces of one's kind horizontally, diagonally or vertically.

The focus of this paper will be reasonable play for the game of Connect-4, because it has a non-trivial branching factor of 7, which is not too large to be impractical either. Also the game has interest in the general public as well. It is also interesting to note that this particular game has been solved (Allis 1988). More importantly, we will keep in mind that if both players play optimally, the first player should always win.

By convention in this paper, p_1 will be represented by red pieces and p_2 by blue pieces.

Basic search algorithms

The basic algorithms used in this paper are Minimax and Alpha-Beta searches as introduced in (Russell and Norvig 2010). There is an obstacle to direct implementation that must be handled. Large branching factors in the search tree can occur when the board is large or when gravity is turned off.

To cope with large branching factors which lead to impractically long searches, we add two cutoff schemes:

- a fixed time limit cutoff, used in combination with iterative deepening: when time is up, the search returns the best move found on the previous iteration;
- a fixed depth limit

We evaluate terminal boards by assigning $-\infty$ for a loss, 0 for a draw and $+\infty$ for a win—standard utility values. To evaluate non-terminal nodes of the search tree, we use a weighted linear heuristic consisting of six features as described in the following section.

To observe the pruning effect and efficiency of Alpha-Beta search over Minimax search, we have the two algorithms play against themselves and each other with a fixed time limit of 5 seconds, using the heuristic

(own winning rows) - (opponent winning rows).

The result is recorded in the next four figures.

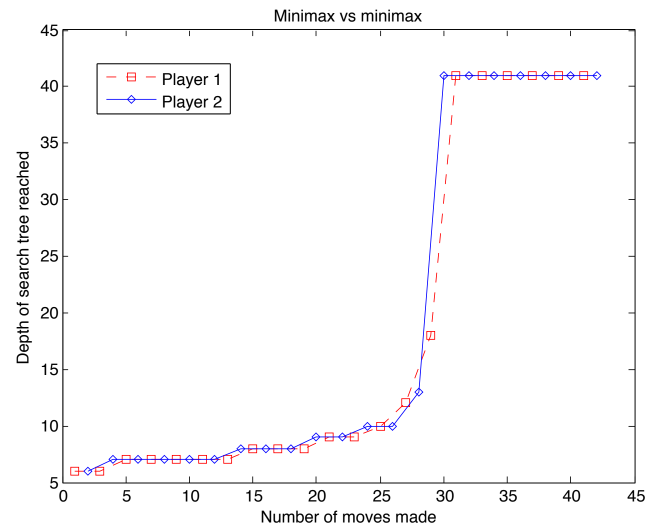

Figure 1: Depth reached in Minimax vs. Minimax play

We observe that depending on the outcome of the game, there is eventually a sharp drop or a flattening of the curve indicating a forced win or a draw, respectively. But of concern to us is what happens initially. The curves in all cases rise, indicating greater depth reached as the game progresses and the branching factor diminishes. However, the trend

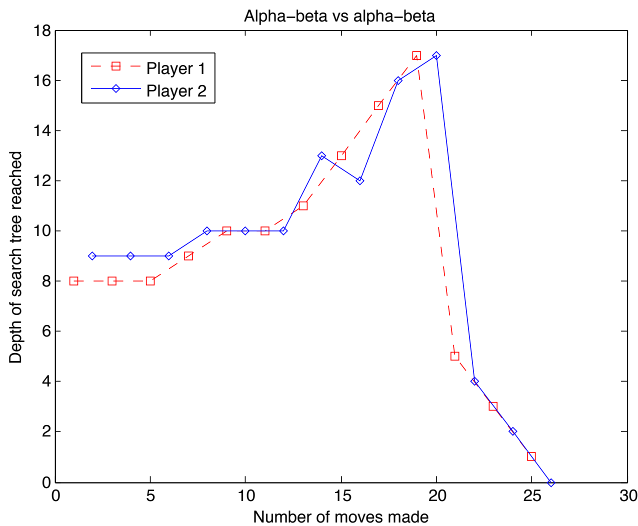

Figure 2: Depth reached in Alpha-Beta vs. Alpha-Beta play

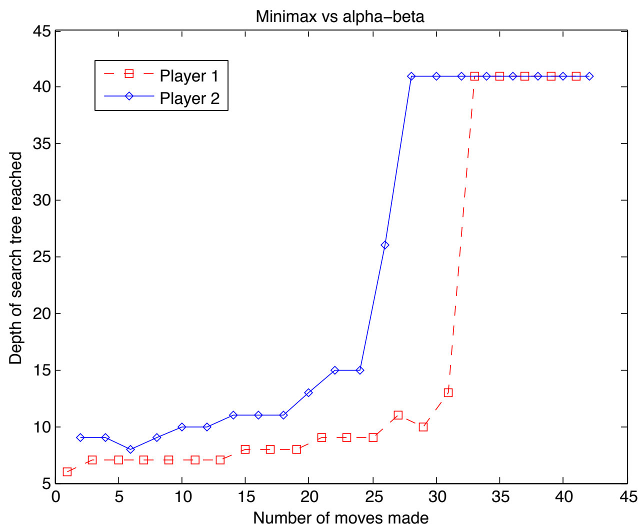

Figure 3: Depth reached in Minimax vs. Alpha-Beta play

over the four graphs is that the rise is quicker with Alpha-Beta (reaching depth 18 in 20 moves) than with Minimax (reaching depth 18 in 30 moves).

We can also compare the performance of Minimax versus Alpha-Beta using a fixed depth comparison, and recording the time it took each to make the first move. More specifically we have the two algorithms play themselves with a depth limit of 8.

According to Table 1, there is a clear performance improvement from Minimax to Alpha-Beta. Note that both still play the same game which leads to the following end board:

Heuristics

The selection of features, and in particular the odd and even threats is motivated and explained in (Allis 1988). In particular, consider a line of k board cells containing exactly $k-1$ of p_1's pieces, and suppose that the remaining cell that is part of this line is empty. We do not count immediate threats, which can be blocked on the next move, and therefore require that the empty cell not be on the bottom row and not have a filled cell immediately underneath it. Such an empty cell is called a threat for p_1. A similar definition

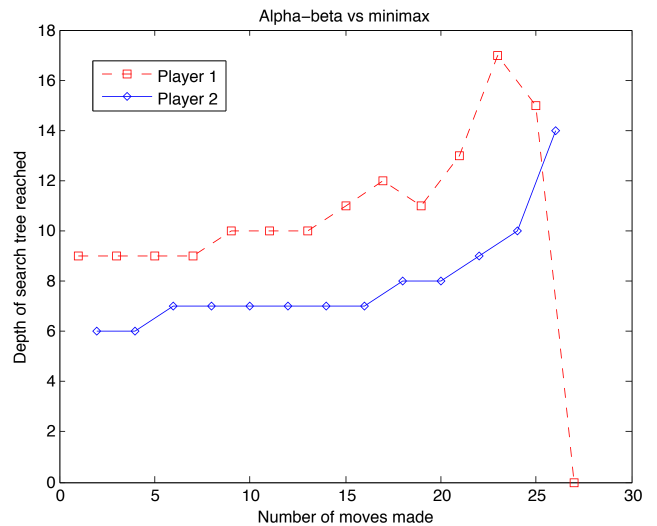

Figure 4: Depth reached in Alpha-Beta vs. Minimax play

Algorithm	Minimax	Alpha-Beta
p_1 time (s)	29.566	1.408
p_2 time (s)	28.172	0.776
p_1 max-calls	5,793,333	134,154
p_2 max-calls	5,721,974	154,624
p_1 min-calls	840,693	42,231
p_2 min-calls	818,879	38,852

Table 1: Alpha-Beta vs Minimax searches to depth limit 8

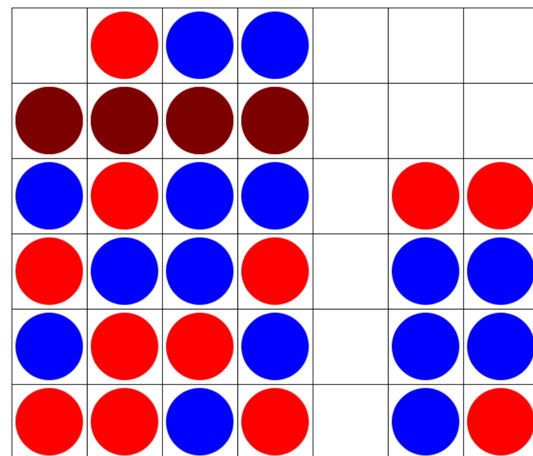

Figure 5: Final board for Alpha-Beta and Minimax searches to depth limit 8

of a threat for p_2 applies *mutatis mutandis*.

The heuristic used in the Minimax and Alpha-Beta algorithms is a linear combination of the following six features, in this particular order:

1. p_1 winning rows: a line of k board cells is considered a winning row for p_1 if it contains at least one of p_1's pieces and none of p_2's pieces;
2. p_2 winning rows: defined as above but interchanging the roles of p_1 and p_2;
3. p_1 odd threats: threat cells for p_1 that are located on an odd row
4. p_2 odd threats: threat cells for p_2 that are located on an odd row;
5. p_1 even threats: threat cells for p_1 that are located on an even row;
6. p_2 even threats: threat cells for p_2 that are located on an even row.

The computation of these threats can be improved from a naive implementation by an algorithm inspired by digital image processing that computes the distance transform for which fast algorithms exist (Fabbri et al. 2008)

The weights are set differently depending on whether the game's gravity setting is turned on or not. More specifically, when gravity is off, threats lose their meaning and so they are not computed and their respective weights are set to 0. When gravity is on, however, various weights are set.

We have found the following weight vectors (which correspond to the order of features in the list above) matched certain playing tendencies:

- The set of weights most used for testing throughout this paper was $(1, -1, 0, 0, 0, 0)$ which corresponds to:

  (p_1 winning rows) - (p_2 winning rows)

  These weights offer a good balance between blocking opponent winning lines and creating one's own.
- $(1, 0, 0, 0, 0, 0)$ reflects a tendency to want to create new winning lines
- $(0.1, -0.1, 1, -1, 1, -1)$ emphasizes the use of threats. The algorithm has the tendency to fill in the winning lines it has built. Setting weights for winning rows to 0 has shown poor performance.

These weights are intended for p_1. The corresponding weights for p_2 are obtained by interchanging the values of each of the three consecutive pairs of weights.

Move ordering in Alpha-Beta

The alpha-beta algorithm can be improved with a non-arbitrary move ordering scheme. By default, at each step of the alpha-beta search, the next available moves are examined in order of the board from left to right (and bottom-up when gravity is turned off). An arbitrary move ordering scheme should yield performance comparable to this algorithm.

Since alpha-beta takes advantage of pruning lesser-valued moves, we would like to order the moves so that the best ones are first examined. We propose two such ordering schemes.

The first ordering scheme uses the linearly weighted heuristic

$$\text{(own winning rows)} - \text{(opponent winning rows)}$$

that is also used to estimate the value of a non-terminal state at cutoff search levels. By this ordering scheme, alpha-beta search first examines the move with the best heuristic value.

The second ordering scheme is motivated by the game of Connect-4, and works well for versions of Connect-k where gravity is turned on. It orders moves by their distance to the vertical axis at the center of the board. This scheme gives preference to moves closer to the center, in agreement with one's intuition about the game, since pieces near the central vertical axis potentially contribute to winning rows on either side of the board.

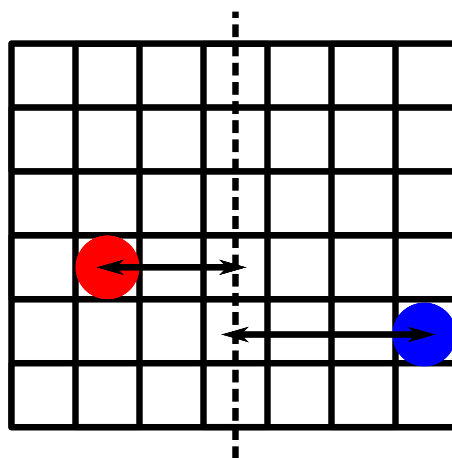

Figure 6: Move ordering preferring to move to the center: the red piece has higher precedence over the blue piece because it is closer to the central vertical axis.

To compare the amount of pruning induced by these ordering schemes, we run the original and enhanced versions of the alpha-beta algorithm with a depth limit of eight levels in the game of Connect-4. We record the amount of time taken by the first and second players (each running the same version of the algorithm) to decide their first move on the board. We also record the number of recursive calls made to the min-value and max-value functions, which correspond to the number of nodes of the search-tree that have been visited above level eight, also for the first move made.

The trends for the first and second players are similar. Specifically, ordering by distance to the center shows a time improvement over the default by a factor of 3 for p_1 and a factor of 10 for p_2. By contrast, ordering by heuristic value is comparable to if not better than the default move ordering by time.

However, both these orderings show a reduced number of calls to the min-value and max-value functions. In the case of ordering by distance to the center, this improvement is particularly dramatic, as the reduction is by a factor of 100.

Ordering	default (none)	scheme 1 (best-heuristic)	scheme 2 (center dist)
p_1 time (s)	1.408	0.843	0.054
p_2 time (s)	0.776	0.966	0.076
p_1 max-calls	134,154	17,057	7,771
p_2 max-calls	154,624	29,157	14,302
p_1 min-calls	42,231	7,227	5,156
p_2 min-calls	38,852	8,670	3,781

Table 2: Move ordering schemes in Alpha-Beta

The more modest performance—still an improvement by a factor of 2—of ordering by heuristic value, coupled with the time involved in the computation of the heuristic value itself could explain why no time improvement was exhibited.

Transposition Tables with Minimax

During the Minimax search, game states (or board states) are repeatedly examined. To eliminate redundant work, we implement a version of Minimax that records all explored states and their respective computed values. This way, if a state has already been explored, we can use its value and avoid searching its subtree again.

Algorithm	Minimax	Minimax + transposition
p_1 time (s)	29.566	35.244
p_2 time (s)	28.172	3.521
p_1 max-calls	5,793,333	201,866
p_2 max-calls	134,154	201,163
p_1 min-calls	840,693	59,367
p_2 min-calls	818,879	58,363

Table 3: Minimax and Minimax with transpositions searches to depth limit 8

We notice that while the first player is slower with a transposition table than without, player 2 gains a clear advantage when equipped with the transposition table.

Performance on large boards

While currently described algorithms (such as Alpha-Beta with iterative deepening and heuristic evaluation at cutoff nodes) play nearly perfectly against humans (specifically the author), and plays perfectly in the game of tic-tac-toe, it is still a challenge and a problem for the algorithm to play 5×5 boards when gravity is turned off, because of the large branching factor.

While the time-limited versions of Minimax and Alpha-Beta will always return a move within 5 seconds (or more if specified), they do not play reasonably. The main challenge is to reach a large enough depth: ideally it would reach depth k to block an opponent winning row and $2k$ to build winning rows of its own.

The Horizon Effect

We have several versions of Alpha-Beta with a fixed depth time limit of 5 seconds to simulate reasonable fairness rules. The different versions of the algorithm differ only by the weights set which are as follows (from the point of view of p_1):

- $w_1 = (1, -1, 0, 0, 0, 0)$: standard weights
- $w_2 = (2, -1, 0, 0, 0, 0)$: slight preference for creating winning rows, but still interested in minimizing opponent winning rows
- $w_3 = (1, 0, 0, 0, 0, 0)$: only concerned with creating winning rows
- $w_4 = (0.1, -0.1, 1, -1, 1, -1)$: introduce threat concept
- $w_5 = (0.1, -0.1, 1, 0, 0, 0)$: only concerned with building odd threats
- $w_6 = (0.1, -0.1, 0, 0, 1, 0)$: only concerned with even threats

The result of the games are demonstrated in Table 4 and Table 5. In these tables, the left column represents the weights of the first player and the top row represents the weights of the second player. For each pair of players, the outcome of the game is summarized with the first number representing the number of moves/pieces on the board at the end of the game and the second value indicates the winner of the game.

$p_1 \backslash p_2$	w_1	w_2	w_3
w_1	20/p_2	20/p_2	30/p_2
w_2	15/p_1	25/p_1	20/p_2
w_3	23/p_1	17/p_1	30/p_2

Table 4: Alpha-Beta with various winning row weights

$p_1 \backslash p_2$	w_4	w_5	w_6
w_4	22/p_2	22/p_2	22/p_2
w_5	35/p_1	22/p_2	18/p_2
w_6	29/p_1	16/p_2	28/p_2

Table 5: Alpha-Beta with various threat weights

These results do not agree with the initial observation that given optimal play, player 1 should always win. More intriguingly, not a single game resulted in a draw. Instead, player 2 seems to have a distinct advantage in almost all cases. Several explanations can account for this discrepancy.

First, because the games are played with a cutoff, the players are not playing optimally. Perhaps the weights are not set to optimal values. And perhaps, the search algorithm is not given enough to reach a deep enough level.

Second, because player 2 is always searching starting from a node whose corresponding board has one more piece than the board on which player 1 searched on, player 2 will always be able to search to a deeper level, and see further explained. Therefore, when given equal search time, player

2 has in fact always a clear advantage over player 1. The issue is thus always a matter of reaching deep enough in the search tree, that is, the so-called horizon effect.

Implementation details

We now discuss a few implementation issues that were somewhat subtle. Given that the program was written in Java, the performance of the same algorithm would change depending on whether it was run the first or the second time. This is because Java only compiles the code the first time it is run. For reliable measurements, we ran each algorithm twice and only recorded the second run's timed performance.

Transpositions tables did not produce a noticeable speed up in the Minimax algorithm until it was initialized to a large capacity. This is because the table was implemented as a Hashmap. With a initial capacity of 16 (the default), the program would have to continually resize the table as it grew. This operation would slow the algorithm considerably, thus leaving no noticeable effect overall.

Suggestions for further improvements

In this paper, the linearly weighted heuristic was assigned weights by hands. The performance of the differently heuristics was assessed qualitatively, and chosen based on the strategies presented in (Allis 1988). A more systematic approach can be taken, such as the approach taken in (Veness et al. 2009). In this paper, the weights are found through a learning algorithm that aims to match the heuristic value of a search state with the value that is propagated to it from deeper levels in the search tree. This simple idea should also eliminate the horizon effect that was discussed earlier.

To successfully play on large game boards, we could try to switch to a different AI paradigm based on a knowledge base. Searches as the ones presented in the paper would become subordinate to this knowledge base. For example, to cope with large branching factors at the beginning of the game, we use experience to suggest a preference towards placing move towards the center of the board or towards its edges, and then perform a forward search.

In this paper, transposition tables were implemented for the Minimax algorithm, and demonstrated dramatic performance improvement. We therefore propose to also implement them for the alpha-beta algorithm. The implementation would be different in that, rather than maintaining a single transposition table for the whole algorithm, we would maintain two, one for the min-values and one for the max-values.

Conclusion

We have implemented improvement features over standard Minimax and Alpha-Beta searches. For Minimax, we found that the addition of a transposition table added significant pruning. Similarly, move ordering schemes contributed to pruning of Alpha-Beta search.

The best algorithm we have found to play the game of Connect-4 is an alpha-beta search with move ordering towards the center of the board. It was not clear whether taking into account threat features into the heuristic significantly improved performance. While an efficient algorithm was found to compute the number of threats, we preferred to have a standard (own winning rows) - (opponent winning rows) heuristic to minimize the time taken to evaluate heuristic values.

While this algorithm just described performs well when played against a human player on a game of Connect-4, it does not perform reasonably when gravity is turned off or on larger boards. We believe that simple improvements as suggested in this paper may impact the performance of our algorithm in a positive way.

Acknowledgements

I would like to thank Michael Bannister for his intellectual contributions to this paper. This paper was part of a project for the course CS 271 (Artificial Intelligence) taught by Richard Lathrop at the University of California, Irvine, in Fall 2010.

References

Allis, L. 1988. A knowledge-based approach of connect four: The game is over, white to move wins. Master's thesis, Vrije Universiteit.

Fabbri, R.; da F. Costa, L.; Torelli, J. C.; and Bruno, O. M. 2008. 2d euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys* 40(1):2:1–2:44.

Russell, S., and Norvig, P. 2010. *The Artificial Intelligence*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

Veness, J.; Silver, D.; Uther, W.; and Blair, A. 2009. Bootstrapping from game tree search. In Bengio, Y.; Schuurmans, D.; Lafferty, J.; Williams, C. K. I.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 22*. 1937–1945.