The Effects of Alpha-Beta Pruning on Minimax Search in Connect 4

Alpha-Beta Pruning

Alpha-Beta pruning is an optimization technique for the minimax algorithm and is used to reduce the computation time by a huge factor. By doing this it allows us to search much faster and search into deeper levels of the tree without taking as long as the pure minimax algorithm would.

The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of at that level or above and the maximum score that the minimizing player is assured of at that level or above respectively. Alpha is initialized to negative infinity and beta is initialized to positive infinity, so both players start with their worst possible score. Whenever the maximum score that that the minimizing player (beta) is assured of becomes less than the minimum score that the maximizing player (alpha) is assured of (beta <= alpha), the maximizing player doesn't need to consider the descendants of this node as they will never be reached in actual play. This is why the technique is known as "Alpha-Beta Pruning", as subtrees which yield the value of an equivalent subtree or worse, and so cannot influence the final result, aren't explored, hence reducing computation time.

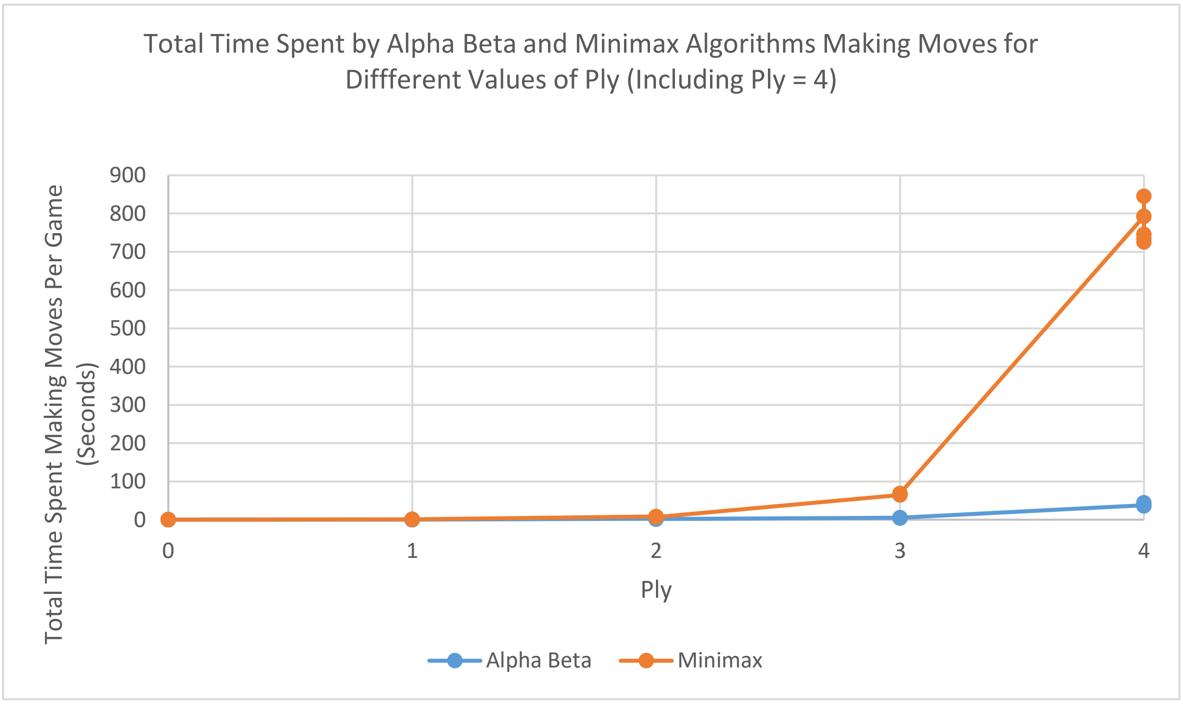
However, in terms of its ability to find the optimal move, Alpha-Beta Pruning is no better than pure Minimax search. By pruning the search tree, Alpha-Beta pruning eliminates the branches which are known to return worse scores than other searched branches, preventing calculations being performed on branches which are guaranteed not to contain the optimal move for the current player. However, this doesn't mean that Alpha-Beta Pruning will return a different value for the optimal move to pure Minimax search. Alpha-Beta Pruning will only rule sub-optimal branches out earlier than the Minimax algorithm, but will have no effect on the branch containing the optimal move. Therefore, given the same state and searching to the same ply, a pure Minimax algorithm and an Alpha-Beta algorithm will both return exactly the same optimal move, only the Alpha-Beta algorithm will return the optimal move much more quickly.

Therefore, when playing a Minimax algorithm against an Alpha-Beta algorithm on the game of Connect 4 they will both make the same moves if searching to the same ply. Therefore, each game will be played out in exactly the same manner no matter whether the Alpha-Beta algorithm is playing as the red player or blue player; it is impossible to distinguish the two algorithms from just seeing the moves they play. This means that, when playing against each other for the same value of ply, no matter how many games you play them against each other for, the same colour player will win the game, independent of whether it is using the Alpha-Beta algorithm or Minimax algorithm. Therefore, if you swap the colour player of each algorithm every game, each algorithm will win 50% of the games, which was backed up from the results of my experiments.

Data Collection

In order to collect data to examine the effects of Alpha-Beta Pruning on the pure Minimax search I played the Minimax AI against the Alpha-Beta modified AI on the game of connect 4 for different values of ply. For each ply value I played the two AIs against each other 6 times, and for each game recorded the total time spent by each AI making moves in the game, the mean time spent by each AI per move in the game, the times spent by each AI per move in the game, and the winner of the game. In each game the AI which was assigned red tokens played first, so to account for any advantage/disadvantage to be had from being the first to play I let each AI be the red player 3 times for each value of ply. The range of values I set for the ply were 0-4 inclusive, as the games with a ply of 4 each took over 10 minutes, so higher values of ply would have taken too long due to the computational restrictions imposed by my laptop. After collecting the data I then plotted a scatter graph, with each point representing the total time spent by one of the AIs making moves during a game with a given ply, enabling me to clearly see the performance improvements that Alpha-Beta pruning caused. I also created a table showing the mean time taken per move by each AI for different values of ply and a table showing which colour won each game for different values of ply. **The figures from which the scatter graphs and tables were generated can be seen at the end of this document under the "Data" header.**

Scatter Graphs



Tables of Results

Ply	0	1	2	3	4
Mean Time Spent by Alpha-Beta per Move (seconds)	0.00692	0.01243	0.09474	0.12181	0.97349
Mean Time Spent by Minimax per Move (seconds)	0.00668	0.03084	0.38205	1.66738	19.31999

Please note, the mean time spent by each algorithm per move was calculated by dividing the total amount of time the algorithm spent searching for moves throughout the game by the number of moves it actually played in the game. For example, if in a game the Alpha-Beta algorithm spent 100 seconds in total calculating which moves to play, and ended up playing 20 moves before the game came to an end, then the mean time spent by it per move would be 5 seconds. However, figures 1-4 illustrate the distribution of the Minimax algorithm's time spent on individual moves throughout the game. The average is a misleading performance indicator as the time spent per move decreases considerably after 20 moves, as demonstrated in figures 1-4.

Ply	0	1	2	3	4
Winner in Every Game, Independent of Algorithm	Red	Blue	Blue	Red	Blue

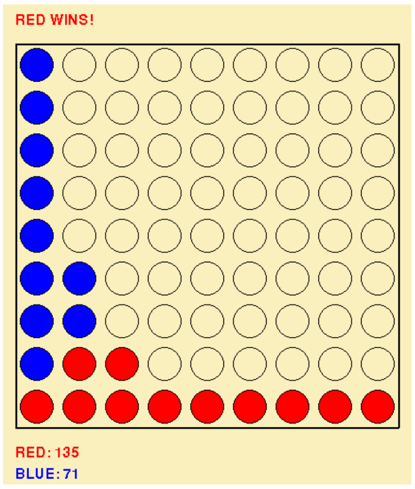
Performance Analysis

Looking at the graphs showing the total time spent by the two AIs making moves per game and the table showing the mean time spent by the two AIs per move it is clear that the Alpha-Beta AI is much more efficient at finding the optimal move for different values of ply. For smaller values of ply, this difference isn't as large as for bigger values of ply. For example, with a ply of 1, the mean time spent making each move by the Alpha-Beta AI is 40.3% of the mean time that the Minimax AI spends taking each move. Although this is a significant improvement, when the ply is 4 this improvement is even more significant, with the mean time spent making each move by the Alpha-Beta AI being 5% of the mean time that the Minimax AI spent taking each move. This reveals just how much more efficient the AI is which implements Alpha-Beta pruning, as it will still find the same optimal move as the Minimax AI but in a fraction of the time. It also illustrates how as the depth of the search tree is increased, the benefits gained from optimizing the Minimax Algorithm with Alpha-Beta pruning become increasingly larger in terms of time taken to evaluate the search tree.

The win-ratio of the Alpha-Beta AI was 50%, as during the tests the two AIs played as the 1st/2nd player 50% of the time. Although Alpha-Beta pruning reduces the time spent by the AI finding the optimal move for a given ply, the optimal move it returns is still exactly the same as a Minimax AI searching to the same ply. In other words, Alpha-Beta pruning only reduces the time taken to find the optimal move, as it rules out sub-optimal moves more quickly than pure Minimax search, but it doesn't change the optimal move found; the optimal move found using Alpha-Beta pruning is the same as that found by the pure Minimax algorithm. Therefore, if the Alpha-Beta AI is playing against the Minimax AI searching to the same depth of the search tree, the winner of the match is dependent on who plays 1st/2nd, as they both return the same optimal moves as each other. This was interesting as it illustrated that for 3 out of the 5 values of ply I investigated, it was advantageous to play 2nd (blue) rather than 1st (red).

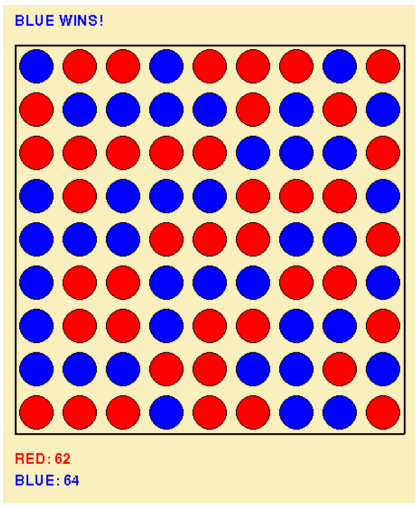
However, although both algorithms return the same optimal moves, the difference in their time taken to return these optimal moves would cause me to choose the Alpha-Beta AI in a tournament. This is because in a tournament I would want to use as high a value of ply as possible (although it wouldn't need to exceed 81 as there are only 81 places on the board so the AI wouldn't need to play out over 81 moves into the future), as using a higher value of ply enables the AI to look a greater number of moves into the future, enabling it to choose better moves based on longer-sighted predictions of how the game will be played if the opponent plays optimally. Using the Alpha-Beta AI, therefore, would vastly reduce the time spent by the AI in finding the optimal move when using a large value for the ply. In contrast, the pure Minimax AI would take too long to find the optimal move to play in a tournament, as for large values of ply it becomes increasingly less efficient than the Alpha-Beta AI. Tournaments often impose time limits for moves too, so using the Alpha-Beta AI would enable a greater search depth (ply) than the Minimax AI would achieve in the same time limit. For example, based on the results gathered by running the algorithms on my laptop, if the tournament imposed a time limit on the mean time spent by the AI per move of 1 second, the Minimax AI would have to use a ply of 2 whereas the Alpha-Beta AI would be able to use a ply of up to 4, enabling it to stand a stronger chance of winning its games.

It was interesting to see the optimal playing strategies for different values of ply, so below are some pictures of the final board states for different values of ply. Please note that for ply = 0 and ply = 2 the boards aren't full. This is because the rules of the game state that the winner of the game is the player who reaches a score of 100 or has the highest score once the board has been filled up, whichever comes first. Therefore, since for ply = 0 red has achieved a score of 135 before the board is full, the game is terminated at this point since the red player has won the game. Similarly for ply = 2 blue has achieved a score of 100 before the board is full, so the game is terminated at this point since the blue player has won the game.

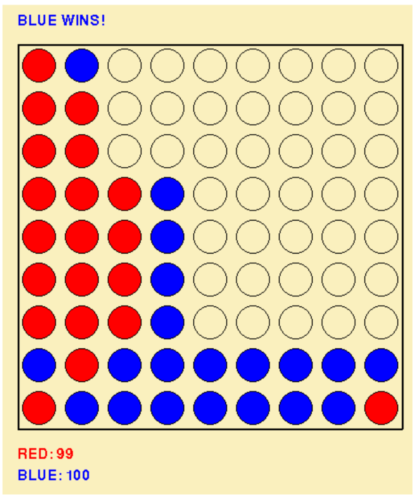


Ply: 0

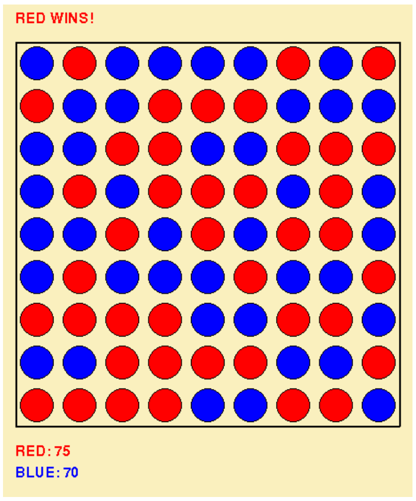
The Effects of Alpha-Beta Pruning on Minimax Search in Connect 4



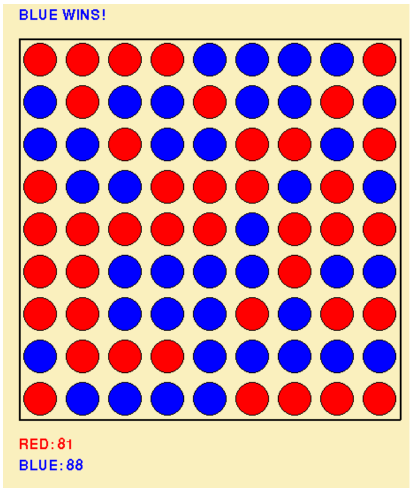
Ply:1



Ply: 2



Ply: 3



Ply: 4

Minimax Time Spent Per Move

A final interesting pattern in the data I found was how long the Minimax AI took to make each move, and how this changed throughout the game as the search tree got smaller. Interestingly, by plotting the length taken for each move against the number of the move for different values of ply, the time taken for each move decreases at different rates for different values of ply, as can be seen below.

Figure 1

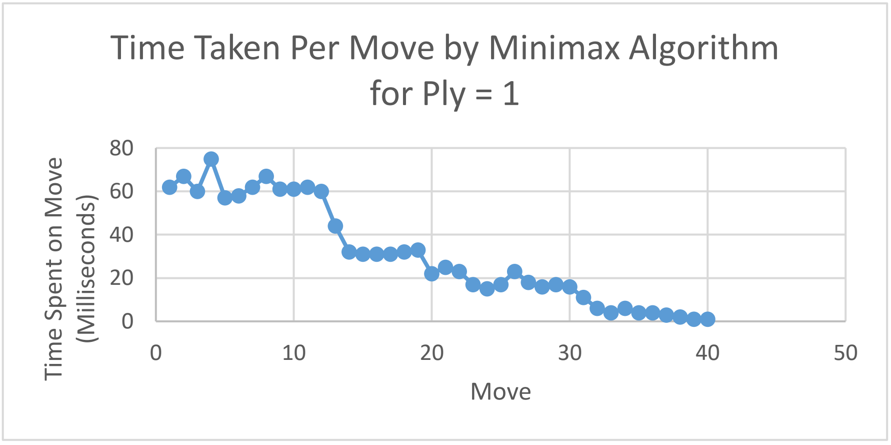


Figure 2

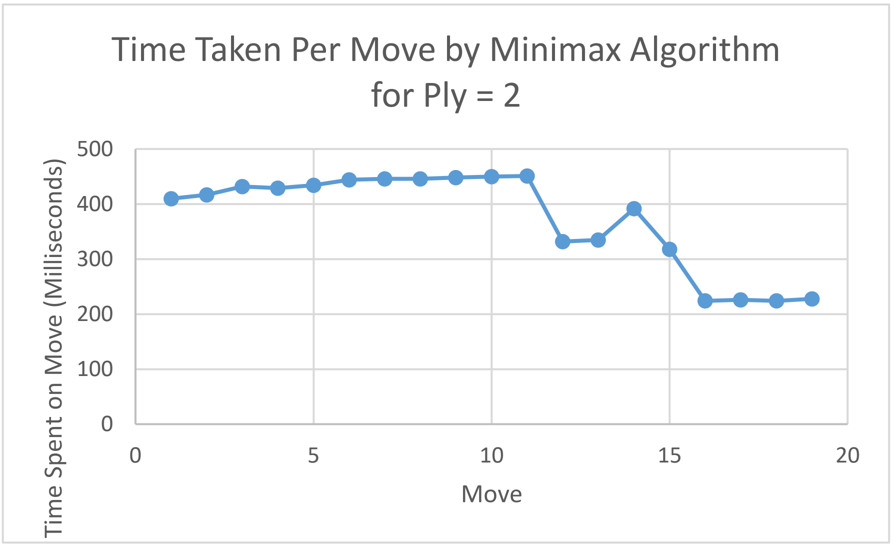
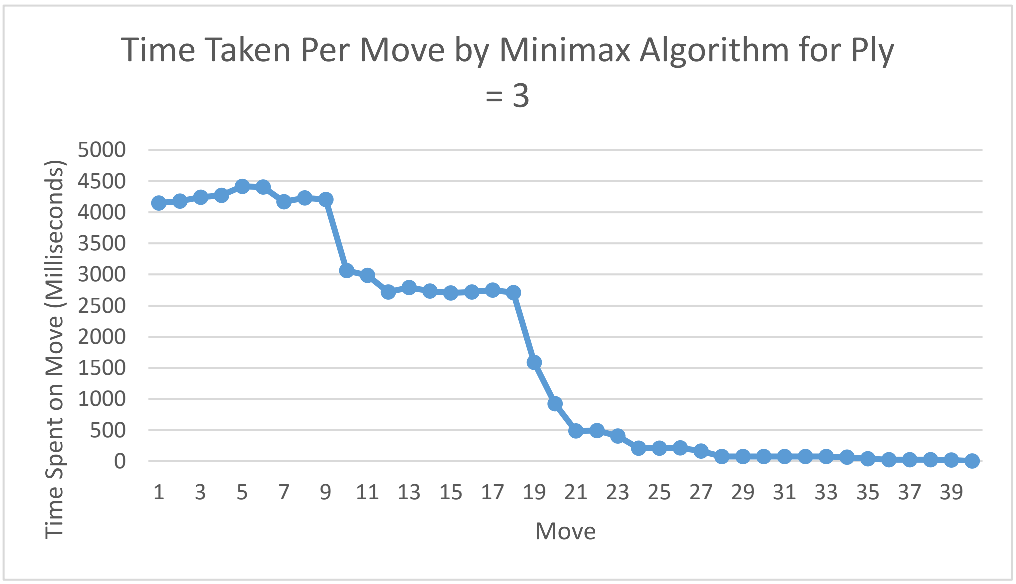
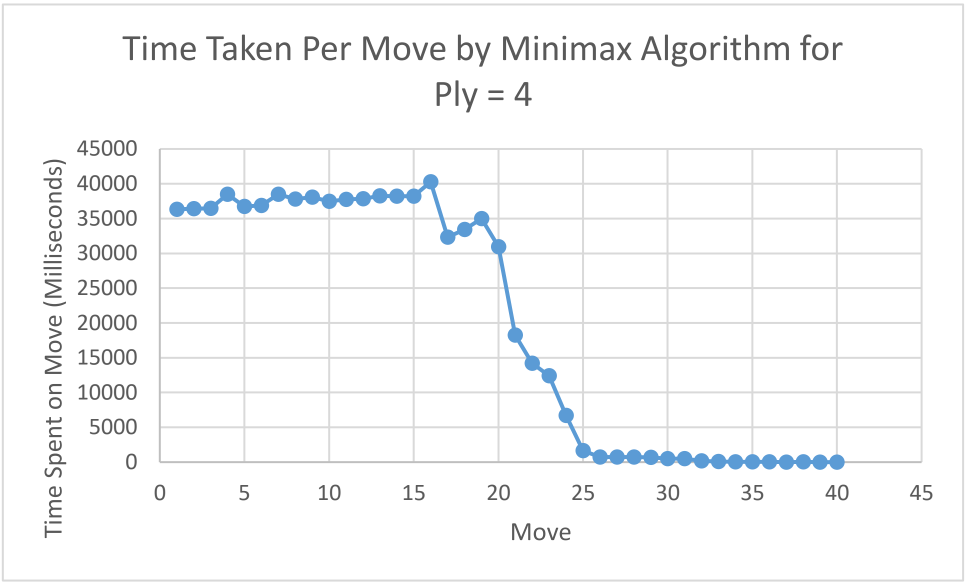


Figure 3



The Effects of Alpha-Beta Pruning on Minimax Search in Connect 4

Figure 4



Data

Ply: 0

Repeat	Red Player	Red Total Time (Secs)	Red Mean Time (Secs)	Blue Total Time (Secs)	Blue Mean Time (Secs)	Winner
1	Minimax	0.062	0.0062	0.067	0.0067	Red
2	Minimax	0.064	0.0064	0.07	0.007	Red
3	Minimax	0.067	0.0067	0.067	0.0067	Red
4	Alpha-Beta	0.069	0.0069	0.067	0.0067	Red
5	Alpha-Beta	0.074	0.0074	0.073	0.0073	Red
6	Alpha-Beta	0.068	0.0068	0.068	0.0068	Red

Ply: 1

Repeat	Red Player	Red Total Time (Secs)	Red Mean Time (Secs)	Blue Total Time (Secs)	Blue Mean Time (Secs)	Winner
1	Minimax	1.213	0.03032	0.472	0.0118	Blue
2	Minimax	1.205	0.03012	0.48	0.012	Blue
3	Minimax	1.211	0.03027	0.479	0.01197	Blue
4	Alpha-Beta	0.523	0.01307	1.274	0.03185	Blue
5	Alpha-Beta	0.524	0.0131	1.263	0.03157	Blue
6	Alpha-Beta	0.507	0.01268	1.237	0.03093	Blue

Ply: 2

Repeat	Red Player	Red Total Time (Secs)	Red Mean Time (Secs)	Blue Total Time (Secs)	Blue Mean Time (Secs)	Winner
1	Minimax	8.714	0.4357	2.322	0.12221	Blue
2	Minimax	7.202	0.3601	1.918	0.10095	Blue

3	Minimax	7.469	0.37345	1.987	0.10458	Blue
4	Alpha-Beta	1.594	0.0797	7.086	0.37295	Blue
5	Alpha-Beta	1.621	0.08105	7.166	0.37716	Blue
6	Alpha-Beta	1.599	0.07995	7.086	0.37295	Blue

Ply: 3

Repeat	Red Player	Red Total Time (Secs)	Red Mean Time (Secs)	Blue Total Time (Secs)	Blue Mean Time (Secs)	Winner
1	Minimax	64.889	1.62222	4.997	0.12492	Red
2	Minimax	65.764	1.6441	5.11	0.12775	Red
3	Minimax	66.208	1.6552	5.207	0.13018	Red
4	Alpha-Beta	4.574	0.11435	66.998	1.67495	Red
5	Alpha-Beta	4.623	0.1157	67.513	1.68783	Red
6	Alpha-Beta	4.719	0.11797	68.799	1.71997	Red

Ply: 4

Repeat	Red Player	Red Total Time (Secs)	Red Mean Time (Secs)	Blue Total Time (Secs)	Blue Mean Time (Secs)	Winner
1	Minimax	792.732	19.8183	37.865	0.94663	Blue
2	Minimax	734.839	18.37097	35.946	0.89865	Blue
3	Minimax	745.534	18.63835	36.336	0.9084	Blue
4	Alpha-Beta	37.939	0.94848	725.537	18.13842	Blue
5	Alpha-Beta	44.615	1.11538	845.261	21.13152	Blue
6	Alpha-Beta	40.935	1.02337	792.895	19.82238	Blue

Please note the times below are in milliseconds. The first value in each array is the time taken to compute the first move, then the second value is the time taken to compute the second move and so on.

Figure 1:

Move Times = ['62', '67', '60', '75', '57', '58', '62', '67', '61', '61', '62', '60', '44', '32', '31', '31', '31', '32', '33', '22', '25', '23', '17', '15', '17', '23', '18', '16', '17', '16', '11', '6', '4', '6', '4', '4', '3', '2', '1', '1']

Figure 2:

Move Times = ['410', '417', '432', '429', '434', '444', '446', '446', '448', '450', '451', '332', '335', '392', '318', '224', '226', '224', '228']

Figure 3:

Move Times = ['4149', '4181', '4243', '4272', '4417', '4404', '4168', '4232', '4205', '3065', '2989', '2719', '2793', '2734', '2706', '2719', '2748', '2711', '1590', '924', '488', '491', '403', '208', '208', '215', '162', '74', '75', '76', '75', '76', '75', '68', '39', '26', '24', '27', '18', '2']

Figure 4:

The Effects of Alpha-Beta Pruning on Minimax Search in Connect 4

Move Times = ['36322', '36428', '36471', '38478', '36725', '36859', '38506', '37795', '38054', '37499', '37749', '37861', '38249', '38212', '38230', '40279', '32307', '33434', '34999', '30963', '18252', '14199', '12393', '6715', '1635', '739', '736', '742', '669', '482', '505', '183', '93', '30', '31', '31', '17', '18', '4', '1']