Experienced Search

I. Problem Statement

Depth-first search algorithms are characterized by $O(b^m)$ performance, depending on both the branching factor b and maximum depth of the search tree m. In the context of two-player zero-sum games, a complete minimax search is typically infeasible. For example, Chess has approximately 10^{43} possible game states [5]. Although game-specific heuristics can significantly prune the game tree, there are often knowledge gaps that lead to "bad" moves, resulting in larger-than-optimal trees. In our work, we enhance $\alpha\beta$ search with game-agnostic enhancements. We examine how the history heuristic, proposed by Schaeffer [4], leverages experience acquired during during $\alpha\beta$ search to order moves for evaluation, thereby pruning a game tree closer to the minimal $O(b^{m/2})$ size. We employ the history heuristic as a search optimization applicable to any two-player zero-sum game. Additionally, we posit that this type of general-purpose game optimization technique contributes to techniques related to general-purpose artificial intelligence, an important problem in the AI field [2].

II. Related Work

There have been many algorithms proposed by the research community to reduce the size of the game tree, including the Killer Heuristic [1] and transposition tables [4]. As noted by Schaeffer [4], the Killer Heuristic is a special case of the history heuristic, and moreover its effectiveness is unclear. Schaeffer notes that although transposition tables effectively prune the game tree, they are memory-intensive. As a result, transposition tables are unsuitable as a general-purpose game search improvement technique, particularly in light of the recent development of mobile games where resource constraints are once again critical. In our project, we study the reduction in game tree size resulting from Schaeffer's history heuristic, which adds a very small performance and memory overhead to standard $\alpha\beta$ search [4].

III. Approach

We implemented three different algorithms that return suggested moves for an *nxn* Tic-Tac-Toe game:

1. standard minimax search
2. standard $\alpha\beta$ search with unordered move evaluation
3. $\alpha\beta$ search with move ordering via the history heuristic

First, we developed and validated an *nxn* Tic-Tac-Toe game engine designed for two human players. Our definition of the rules for the *nxn* version specifies that a win is achieved for any board state containing n consecutive moves by any one player in a column, row, or diagonal. This is a natural extension of the standard *3x3* Tic-Tac-Toe.

Next, we developed an AI opponent for a human player that uses a pluggable search strategy, and implemented standard Minimax search as our first search strategy, as described in Russell and Norvig [3]. We subsequently implemented standard $\alpha\beta$ search. In our implementation, the default move order is from the upper left of the board to the bottom right. Lastly, we implemented Schaeffer's history heuristic as an extension to our $\alpha\beta$ pruning implementation. We enhanced our implementation of the history heuristic with two additional features: 1) depth-limited search, and 2) an evaluation function; that is, a function used to evaluate a non-terminal board position.

To obtain data on the algorithm performance, we modified our game implementation to use two AI agents. In addition, we added metadata to the search algorithms to count the total number of nodes expanded during search.

In the next section, we expand on our approach described above to provide a description of the history heuristic and the details of our implementation.

IV. The History Heuristic

The history heuristic is an enhancement that scores "sufficient" moves and subsequently uses move scores to sort the moves for evaluation, with the goal of achieving early $\alpha\beta$ cutoff, thereby significantly reducing the size of the "effective game tree" (i.e., the actual tree that is searched, as opposed to the full theoretical game tree). Schaeffer defines a "sufficient" move as a move at an interior node that "1) causes a cutoff, or 2) if no cutoff occurs, the one yielding the best minimax score" [4]. When a move is evaluated and deemed sufficient, the associated "history score" is increased. The new score is subsequently used to order moves in descending order, thus ensuring that the first moves evaluated are those corresponding to the "sufficiency" criteria.

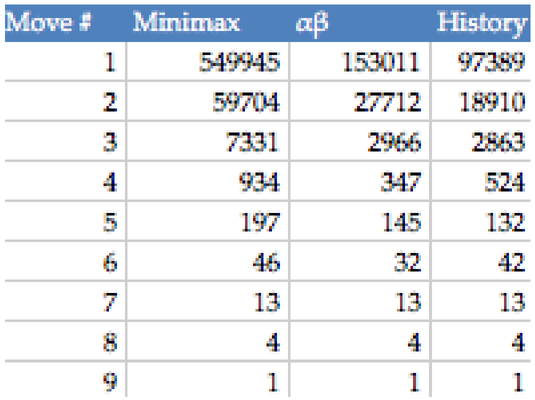
Implementation of the history heuristic is dependent upon two parameters: a *mapping* of moves to history scores and a *weight* to add to the score of a sufficient move. The concept of the mapping is that it should correspond to a set of legal moves for which scores can be determined, although the mapping doesn't need to include an entry for every possible individual move. For example, in *3x3* Tic-Tac-Toe, one could consider a mapping for squares by "type": corner square, side square, and center square.

Similarly, the concept behind the weight parameter is that earlier (shallower) moves should count for more (i.e., have a higher weight) because the scores for shallow moves will impact move ordering for a larger portion of the search tree, resulting in actual performance improvements due to pruning. Details are fully explained in Schaeffer's paper [4]. As in Schaeffer's work, we used a weight of 2^{depth} (note that $depth = 0$ for leaf nodes, and increases moving up levels of the search tree). The mapping in the history heuristic is game-specific. For *nxn* Tic-Tac-Toe, we used an dictionary mapping each square in the *nxn* board to a score. Note that memory usage for the history table is insignificant: even for an extraordinarily large *1000x1000* board, the memory usage is only approximately 3.8 MB.

In the next section, we provide performance data for our three search strategies.

V. Evaluation

As noted above, we recorded the number of moves (nodes) evaluated (expanded) during search of the game tree, resulting in metrics for the size of the effective game tree. We note that a Tic-Tac-Toe game tree has growth characteristics worse than exponential growth: for an *nxn* board, the size of the maximal game tree is $n!$. As a result, full minimax search is infeasible on a standard laptop computer even for a *4x4* board. Consequently, we configured all three algorithms to search to endgame on a *3x3* board. Because Tic-Tac-Toe is a fully-observable, deterministic game, two AI players with the ability to search to endgame will play the *same game every time*. Thus, we present our data for this case. Figure 1 shows the total number of nodes searched by each algorithm, and Figure 2 shows the data for $\alpha\beta$ and history heuristic as a percentage of nodes searched relative to the baseline Minimax algorithm (which is 100%).



Move #	Minimax	αβ	History
1	549945	153011	97389
2	59704	27712	18910
3	7331	2966	2863
4	934	347	524
5	197	145	132
6	46	32	42
7	13	13	13
8	4	4	4
9	1	1	1

Figure 1: *3x3 Board: Number of Nodes Searched by Minimax, $\alpha\beta$ and History Heuristic Search*

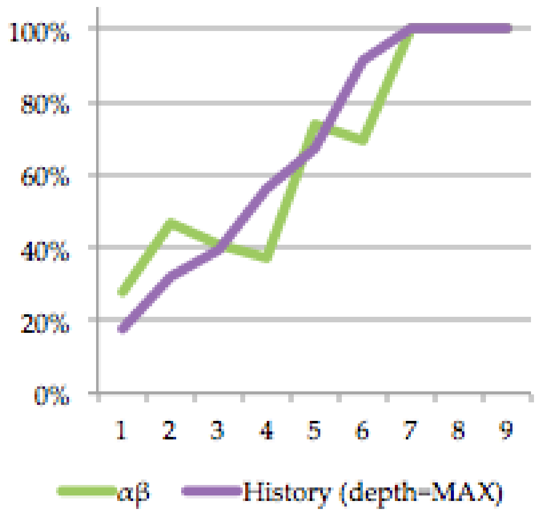


Figure 2: *3x3 Board: Number of Nodes Searched by $\alpha\beta$ and History Heuristic as a Percentage of Minimax Nodes*

As noted above, our history heuristic search strategy takes an input parameter of *depth* which can be used to perform depth-limited search. For the *3x3* results shown above, we set the depth equal to the maximum game tree depth (9) in order to search to end-game. Because searching to end-game is not feasible for a *4x4* game (due to the 16! nodes in the maximal game tree), we searched using the history heuristic to a specified search depth. Figure 3 shows results for the history heuristic on a *4x4* game with search depths of 5 and 6.

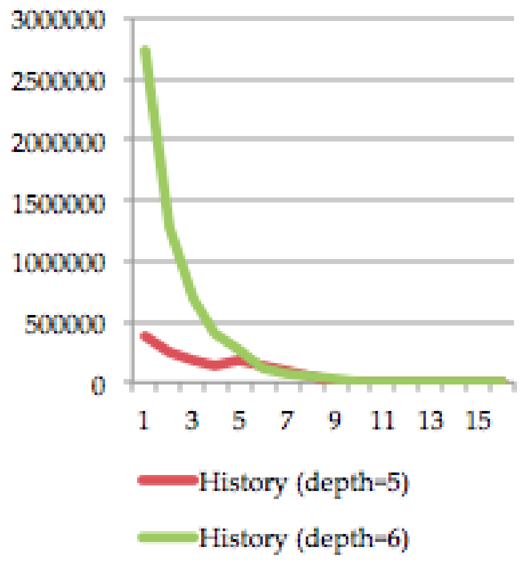


Figure 3: *4x4 Board: History Heuristic Tree Size*

In the next section, we analyze the results detailed above in order to effectively compare the three search strategies and determine if the history heuristic is a game-agnostic $\alpha\beta$ search enhancement.

VI. Discussion

Figure 2 clearly indicates that $\alpha\beta$ significantly prunes the maximal game tree searched by Minimax. The history heuristic, in turn, provides a significant improvement over an unordered $\alpha\beta$ search. When searching for the first move in a game, $\alpha\beta$ searches only 28% of the nodes that Minimax searches, and the history heuristic searches only 64% of the nodes that $\alpha\beta$ searches.

It is interesting to note that there are cases where the history heuristic actually searches more nodes than $\alpha\beta$. For example, when searching for a move on turns 4 and 6, the history heuristic evaluates 524 moves and 42 moves, as compared to $\alpha\beta$ which searches only 347 and 32 moves, respectively. Thus, although is it generally true that history heuristic is a significant performance improvement over $\alpha\beta$, that is, that

$$moves_{history} \leq moves_{\alpha\beta} \leq moves_{minimax}$$

However, there are cases where $\alpha\beta$ is simply "lucky" with its move ordering and searches fewer total nodes.

Regarding Figure 3, due to the growth characteristics of a Tic-Tac-Toe game tree, we could only run the top-performing algorithm (history heuristic) on a board size larger than *3x3*. Even then, we needed to limit the depth in order to manage the combinatorial explosion of the number of nodes. We found that depths of 5 and 6 produced bearable, although far from real-time, move generation times.

Lastly, we compare our results shown in Figure 1 with Schaeffer's, where he applied the history heuristic to Chess. In Schaeffer's performance results, he found that the history heuristic alone resulted in a $65-90\%$ reduction in game tree size [4]. In our results, we found a $0-82\%$ reduction across all moves of the game. Small reductions in the game tree size occur when searching near end-game. For example, the search for moves 7, 8, and 9 evaluates the same number of nodes among all search strategies. Thus, pruning very small search trees is ineffective. Comparing with Schaeffer's results, he sees a minimum 65% reduction in tree size because he doesn't search Chess endgame. More significantly, when searching large game trees, for example during search for moves 1 and 2 of the game, our history heuristic implementation provided a $68-82\%$ reduction in game tree size, which is comparable to Schaeffer's results.

Our work provided us with a thorough understanding of Minimax and $\alpha\beta$ as effective game search strategies, and most importantly made the idea of a "combinatorial explosion" of a search space extremely concrete! We determined that the history heuristic is applicable when searching during any two-player zero-sum game. Because research on two-player zero-sum games is mostly "complete" [4], an interesting line of future work would be to investigate optimizations of search over belief states for partially observable, deterministic two-player zero-sum games, although we note that a key consideration for that work would be to manage the size of the search space effectively.

VII. References

[1] Selim G. Akl and Monroe M. Newborn. The principal continuation and the killer heuristic. In *Proceedings of the 1977 annual conference*, ACM '77, pages 466–473, New York, NY, USA, 1977. ACM.

[2] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the AAAI competition. *AI magazine*, 26(2):62, 2005.

[3] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

[4] Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:1203–1212, 1989.

[5] C. E. Shannon. Computer chess compendium. chapter Programming a computer for playing chess, pages 2–13. Springer-Verlag New York, Inc., New York, NY, USA, 1988.