Error Minimizing Minimax: Avoiding Search Pathology in Game Trees

Brandon Wilson[1] and **Austin Parker**[2] and **Dana Nau**[1,3,2]
[1]Dept. of Computer Science, [2]Institute for Advanced Computer Studies, and [3]Institute for Systems Research
University of Maryland
College Park, Maryland 20742, USA
email: {bswilson,austinjp,nau}@cs.umd.edu

Abstract

Game-tree pathology is a phenomenon where deeper minimax search results in worse play. It was was discovered 30 years ago (Nau 1982) and shown to exist in a large class of games. Most games of interest are not pathological so there has been little research into searching pathological trees. In this paper we show that even in non-pathological games, there likely are pathological subtrees. Further, we introduce error minimizing minimax search, an adaptation of minimax that recognizes pathological subtrees in arbitrary games, and cuts off search accordingly (shallower search is more effective than deeper search in pathological subtrees). Finally, we present experimental studies of error minimizing minimax in two different games. In our experiments, error minimizing minimax outperformed minimax, sometimes substantially, and never exhibited pathological characteristics.

Introduction

Game-tree search pathology is a phenomenon where deeper minimax search leads to more erroneous decision-making. In a pathological game tree, an algorithm minimaxing to depth 9 will make worse decisions than an algorithm minimaxing to depth 5 (so long as the game ends in more than 9 moves). To those familiar with computational game playing, this is a strange result. Conventional wisdom states that deeper search produces better play – this has certainly been the case for chess. However, there is a large class of games that exhibit pathology, and a large body of work trying to understand why and what kinds of game trees are pathological (Nau 1982; Bratko and Gams 1982; Sadikov, Bratko, and Kononenko 2005; Beal 1980; Pearl 1984).

In this paper, we show how *local* pathologies can occur at certain kinds of subtrees of a game tree, and how to modify a minimax style search procedure to recognize local pathologies. Local pathologies are likely to occur in all interesting games, so we can hypothesize that even in games not known to be pathological, a search procedure that recognizes and accounts for such pathologies should produce better results.

We call the search procedure that recognizes and avoids local pathologies *error minimizing minimax* (EMM) search. EMM search works by tracking both the minimax value of a node and the error associated with the node. As the minimax value of a node is aggregated up the tree in a minimax fashion, the associated error is also aggregated up the tree in a fashion similar to the product rule from (Tzeng and Purdom 1983). At each node in the search, the static evaluation function is run and its associated error estimated. If the static evaluation's error happens to be less than the error associated with deeper search, then EMM search will throw away the results of the deeper search and instead use the statically evaluated results. If the static evaluation function's error is larger than that achieved by deeper search, then the deeper search results are preferred. In this way, those portions of the tree that are locally pathological (i.e. result in greater error) are not included in the search, while those portions of the tree that are non-pathological (i.e. result in smaller error) are sure to be included in the search. The key insight in EMM search is the recognition that the static evaluation function produces erroneous values, and that by tracking that error explicitly in the algorithm, one can produce better results.

The contributions of this paper are:

- An analysis of pathology showing that certain types of subtrees are pathological, and that such subtrees are likely to occur in all interesting games.
- Error minimizing minimax search: a minimax-style search procedure designed to recognize and account for errors caused by local pathologies.
- Experimental results showing error minimizing minimax search to exhibit no pathology even in situations where all other tested algorithms exhibited pathology.

Setup

In this paper, we are looking at two-player, perfect information, zero-sum games. We name the game tree G where each node n has a set of moves $m(n)$ for the player-to-move $p(n)$. The terminal nodes are assigned a utility $u(n)$, where 1 represents a win for player 1 and -1 represents a win for player 2. The standard minimax formula is well known (e.g., (Russell and Norvig 2003)):

$$minimax(n) = \begin{cases} u(n) & \text{if } n \text{ is terminal,} \\ \max_{n' \in m(n)} minimax(n') & \text{if it's player 1's move,} \\ \min_{n' \in m(n)} minimax(n') & \text{if it's player 2's move.} \end{cases}$$

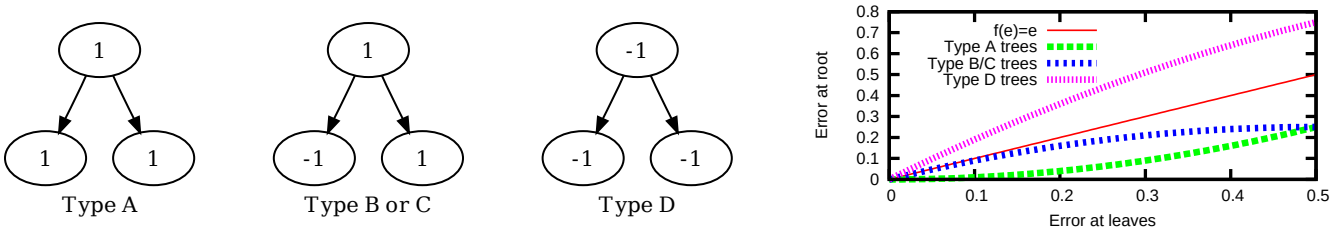

Figure 1: The different tree types available when searching forward one level. Trees B and C are mirror images of one another so are presented together. The graph shows the relationship between the error in the leaves and the error in the root after a minimax search on each tree. Notice that only type D trees increase the error.

To use minimax to determine which move is best, one need simply compute the minimax values for all states one can move to, then pick a state with a maximal minimax value (assuming the computation is done from player one's point of view). When ambiguous, we will use the term *correct minimax value* to refer to the value of a node n computed according to $minimax(n)$.

In games such as the ones investigated in this paper (two player, perfect information, zero-sum games), minimax is known to return an optimal move when allowed to search the entire game tree (Osborne and Rubinstein 1994). However, in many cases, the computation resources needed for such a search are not available – lots of interesting games have combinatorially large game trees that do not permit exhaustive search in normal game-play (i.e. chess, go, connect-6, kalah, checkers, othello). As such, implementations of minimax generally include a heuristic function called a *static evaluation function*. The static evaluation function quickly estimates the true minimax value of a given node.

With the static evaluation function, depth-limited minimax search is possible. The idea is simple: do minimax search to a depth d, then apply the static evaluation function to estimate the minimax values of these depth d nodes. The minimax values of nodes with depth less than d can then be computed as normal, giving estimated minimax values for each possible move. If $eval(n)$ estimates the value of $minimax(n)$, then

$$minimax_d(n) = \begin{cases} eval(n) & \text{if } d = 0 \\ u(n) & \text{if } n \text{ is terminal,} \\ \max_{n' \in m(n)} minimax_{d-1}(n') & \text{if it's player 1's move,} \\ \min_{n' \in m(n)} minimax_{d-1}(n') & \text{if it's player 2's move.} \end{cases}$$

It is generally accepted that deeper search results in better game-play, in most practical situations. However, in the early 1980s, one of the authors of this paper discovered that in a class of games known as *P-games*, deeper minimax search resulted in worse performance, a phenomenon that has come to be know as *game-tree pathology* (Nau 1982).

In pathological game trees, minimax with deeper search is less likely to produce a correct move; one whose correct minimax value is optimal. Most naturally occurring games such as chess, checkers, and the like have been observed to not be pathological: deeper searching minimax algorithms tend to result in better play. In the next section we will show that certain kinds of game trees are pathological while others are not.

Analysis

We now show a quick analysis of games with branching factor two[1] showing that pathological sub-trees are likely to occur in all interesting games. For this analysis, we will assume a static evaluation function that returns the correct minimax value on any given node with probability $1-e$ (i.e. $P(eval(n) = minimax(n)) = 1 - e$). We will be looking at the evaluation error in depth one trees. Evaluation error occurs when a node's minimax value is miscalculated by a depth limited minimax computation. At one extreme, we can imagine a depth 0 minimax computation wherein a static evaluation function is simply applied to the node. In this case, the evaluation error will simply be that of the static evaluation function, e. When deeper minimax searches occur, we have different evaluation errors for different types of trees. Here we examine only searches of depth one, as a search to depth d can be instead thought of as d depth-one searches.

In games with branching factor two, there are four possible depth one trees. These are shown in Figure 1 (trees B and C are symmetric and are therefore considered together). In each tree, it is player 1's move, so the minimax value of the root is the maximum of the minimax values of the children. These trees are partial trees experienced by a search: the leaf nodes are not terminal, but rather the search's horizon.

Using an evaluation function with error e, we can calculate fraction of the time a depth one minimax search will return the wrong value for the root node in each type of tree:

$$error(A) = e^2 \quad (1)$$
$$error(B) = e(1-e) \quad (2)$$
$$error(C) = error(B) \quad (3)$$
$$error(D) = 1 - (1-e)^2. \quad (4)$$

Comparing these functions to simply applying the static

[1] We hope it will be obvious how the analysis will extend to higher branching factors.

evaluation function with error e to the root node, we get that

$$error(D) \geq e \geq error(B) \geq error(A)$$

for any error $e \in [0, 0.5]$. That is, the error resulting from searching type D trees exceeds the error resulting from simply applying the static evaluation at the root node, while for types A, B, and C trees, the error for depth one search is less than that of simply applying the evaluation function. Figure 1 shows this relationship in a graph, where we plot the value of e against the error present at the root for simply evaluating the root ($f(e) = e$) and for searching each tree type.

Type D trees are the only ones in which the error at the root is greater than the error at the leaves, and, since any depth d search can be seen as a combination of d depth one searches, we can conclude that type D trees are the source of all pathology in two-move games. This is not to say that anytime one reaches a type D tree, a shallower search should be preferred – it may be that each child of a type D tree roots a type A tree, in which case the error at the root will be $1 - (1 - e^2)^2$, which is less than e. But simply to say that if the entire tree were of type A, B, or C, then there *could not be evaluation pathology*[2].

We expect all interesting games contain type D trees: as games are not interesting if player one always wins, and without type D trees, it would be impossible for player two to win!

Error Minimizing Minimax Search

Error minimizing minimax (hereafter EMM) search is an algorithm that computes the minimax value of a node while keeping track of the error associated with that minimax value. The algorithm also computes the static evaluation function at any given node. If the static evaluation allows a tighter error bound than the minimax search, then that error bound is substituted in the final return statement. EMM search is detailed in Algorithm 1.

By keeping track of both the error from searching and the error from evaluating, the algorithm naturally distinguishes between pathological trees (type D above) and non-pathological trees (types A, B, and C above). Further, notice that the algorithm is not limited to branching factor two.

We now detail a short example of how EMM might traverse a given tree, shown in Figure 2. This tree shows a depth two search – the leaf nodes are non-terminal, but are instead evaluated with a static evaluation function with 10% error. Thus the evaluations of nodes D, E, F, and G are all given with 10% error. When processing node B, in which it is player 2's move, we see that both children of B are evaluated as a loss (value 1) for player 2, and therefore that the node is a loss for player 2. However, since this value is in error if either of the static evaluations for nodes D or E is in error, we have a 19% chance that the evaluation at node B is in error. Since a static evaluation of the same node gives the same value (1 – loss for player 2), but with only 10% error, EMM uses the statically evaluated value and those error guarantees for that node. For node C, the opposite occurs. In node C EMM concludes that the node is a win for player 2 with 9% chance of error, as node F would have to have been evaluated correctly (90% chance) and node G incorrectly (10% chance). Thus the error resulting from the search ducks the 10% error resulting from the static evaluation function and error minimizing minimax assigns a loss

[2]So long as the static evaluation function mislabels each node with independent probability e.

Algorithm 1 $\mathsf{EMM}(s, eval, d)$: Error minimizing minimax search. For game state s, evaluation function $eval$ (returning an evaluation of a board from the perspective of the player-to-move) with error e, and search depth d, returns a pair (a, e) where a is the valuation of the state s and e is the error associated with that valuation. $\gamma(s, m)$ denotes the state transition function that returns the new state after making move m in state s.

Let $curVal = eval(s)$, and $curErr = e$.
If d is 0, **return** $(curVal, e)$
// Determine values v_i and errors er_i for children nodes.
Let $mv_1, \ldots, mv_n$ be set of moves at s.
For $i = 1, \ldots, n$, let $(vTmp_i, er_i) = \mathsf{EMM}(\gamma(s, mv_i), eval, d-1)$
Let $v_i = -vTmp_i$.
Let $val = \max_i(v_i)$. // This node's value.
// Determine error for this node $aggErr$.
if val is a loss **then**
  // All children are losses, if any of them are wrong, this node is in error.
  $aggErr = 1 - \prod_i (1 - er_i)$
else
  // There is at least one win child. Error occurs if winning children are wrong and losing children are right.
  Let $aggErr = 1$
  for each (v_i, er_i) **do**
    if v_i is a win **then**
      $aggErr = aggErr \times er_i$
    else
      $aggErr = aggErr \times (1 - er_i)$
    end if
  end for
end if
// Flip values if $aggErr$ is too big.
if $aggErr > 0.5$, $(val, aggErr) = (-val, 1 - aggErr)$.
// Check if static evaluation matches minimax value.
if $curVal = val$ **then**
  // Return the result with the stronger error guarantee.
  return $(curVal, \min(curErr, aggErr))$.
else if $curErr \geq aggErr$ **then**
  // Non-pathological case: statically evaluated error is greater than search's error. Use minimax results.
  return $(value, aggErr))$.
else
  // Pathological case: the statically evaluated error is less than the search's error. Use static results.
  return $(curVal, curErr)$
end if

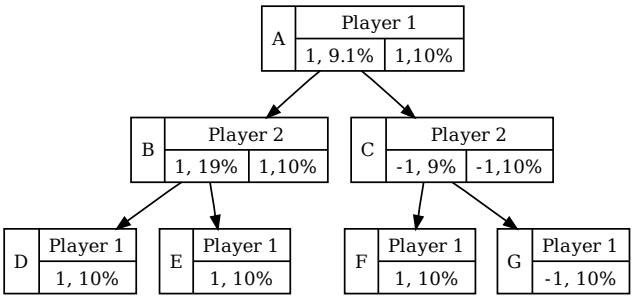

Figure 2: An example tree using EMM search.

with error 9% for node C. We can now conclude that node A is a win, with a 9.1% error rate: only if node B is incorrect (10%) and node C is correct (91%) is node A incorrect. This could be compared to when we did not prefer the statically evaluated error rates, in which case node A would be incorrect with a probability of 17.29%.

Discussion Error minimax bears some resemblance to the product rule (Tzeng and Purdom 1983). The product rule computes the probability that a given node is a win for player one, then aggregates those probabilities up the tree in a method similar to the one used by EMM. The major difference between EMM and product rule search is in the shortcutting of the aggregation up the tree when the static evaluation function is less erroneous than the minimaxed value. This limits the search of subtrees with pathological characteristics: when searching a subtree deeper produces more erroneous values, then the error associated with that search will be higher and the subtree search will be more likely to be thrown away. In this fashion, EMM can be said to "recognize" the pathological portions of a game tree, avoiding them, while doing full-depth search on non-pathological portions of the tree.

Despite this, there are several potential weaknesses present in EMM. The first is the assumption of a particular form of static evaluation function. Generally, if one finds a static evaluation function that is wrong 10% of the time, those errors do not occur independently at random (as is implicitly assumed by error minimizing minimax). Instead, for many natural static evaluation functions, when they are wrong about one game state, they are likely to also be wrong about children of that game state. Further, there is no obvious way to calculate the error characteristics for natural static evaluation functions. Finally, alpha-beta pruning presents a challenge for EMM. We see no obvious analog pruning technique for EMM, which must search the whole tree to calculate the errors. As such, EMM will search a shallower game tree than minimax combined with alpha-beta pruning in the same amount of computation time. If the game tree is not pathological, players playing according to EMM will be at a serious disadvantage because they are not able to search as much of the tree as minimax with alpha-beta. However, if the game tree is pathological, the deeper searches performed by minimax with alpha-beta actually degrades performance!

From our analysis of the algorithm, we concluded that it had promise as a potential alternative to minimax search, particularly in games known to be pathological. Therefore, we did experiments to determine the quality of EMM.

Experiments

Our experiments are performed on a board-splitting game developed by Judea Pearl. In this game two players take turns dividing a 2-D board, consisting of 1's and 0's, into b equal pieces and discarding all but one piece. Player one splits the board vertically while player two splits horizontally. The game is over when only one square remains. If this square is a 1 then the last player to move is declared the winner, otherwise the other player wins. This is a two player perfect information zero-sum game.

We focus on two versions of the game that differ only in the construction of the initial board. The first version is referred to as a P-game[3]. The initial board for each P-game is generated so that each square is randomly and independently assigned a value of 1 with probability p and a 0 with probability $1-p$. The board size itself is $b^{\lfloor \frac{d}{2} \rfloor}$-by-$b^{\lceil \frac{d}{2} \rceil}$ where b and d are the desired branching factor and depth of the game tree respectively. Minimax has been shown to be pathological on P-games using a natural evaluator.

The second version is referred to as a N-game. This construction was introduced by Nau (Nau 1982) to emulate the dependence of heuristic values among siblings, a common feature of real games like checkers and chess. For an initial board of size, $b^{\lfloor \frac{d}{2} \rfloor}$-by-$b^{\lceil \frac{d}{2} \rceil}$, a value of 1 is assigned to each edge of the game tree with probability p and -1 with probability $1-p$. Each leaf of the game tree represents a single square on the board and its value is determined by summing the edge values from the root to that leaf, giving the leaf a value of 1 if the sum is positive and 0 otherwise.

Our experiments compare the performance of three propagation algorithms: minimax, product rule, and EMM. A product rule search to depth d can be defined recursivey similar to minimax:

$$product_d(n) = \begin{cases} u(n) & \text{if } n \text{ is terminal,} \\ eval(n) & \text{if } d = 0 \\ 1 - \prod_{n' \in m(n)} product_{d-1}(n') & \text{if it's player 1's move,} \\ \prod_{n' \in m(n)} product_{d-1}(n') & \text{if it's player 2's move.} \end{cases}$$

We also use two different static evaluation functions:

1. **Artificial**: An "artificial" static evaluation function is a binary function, returning the true minimax value of a state with probability $(1-e)$ and the incorrect value with probability e, where e is a predetermined error rate.
2. **Natural**: A natural static evaluation function for P-games is the one that returns the percent of the remaining board with winning squares (from the perspective of the player

[3]"P-game" is short for Pearl-game, as these games were first introduced by Judea Pearl (Pearl 1984).

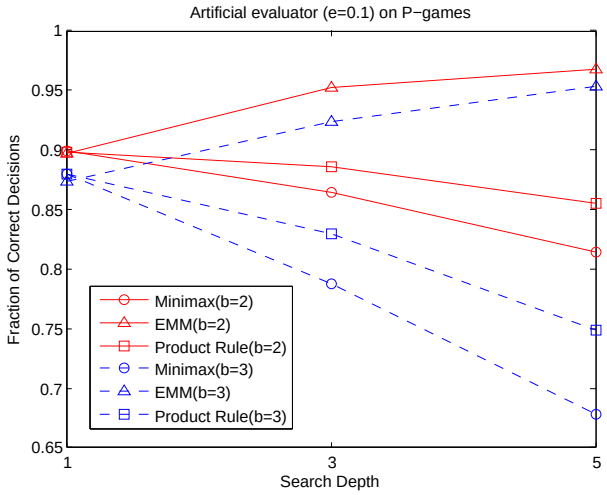

Figure 3: Fraction of correct decisions using the artificial evaluator ($e = 0.1$) on P-games.

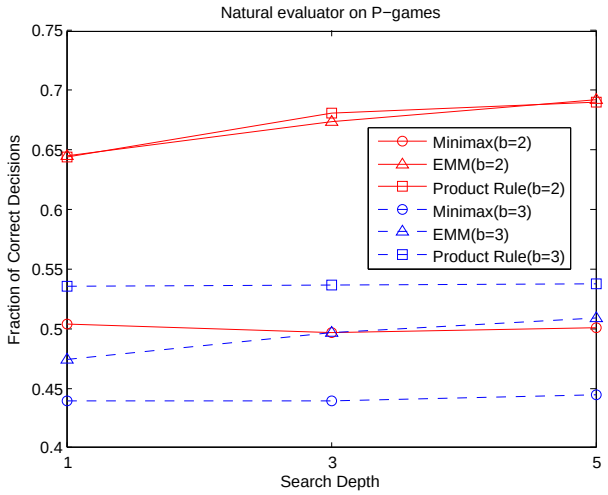

Figure 4: Fraction of correct decisions using the natural evaluator on P-games.

to move). To make this a binary evaluation (required by EMM), a state with more wins than losses is evaluated as a win, while one containing more losses than wins is a loss. For this evaluation function, we estimated the associated error (used by EMM) by generating many completely solved game trees and recording the percent occurrence along with the true minimax value of the node. In our experiments, the product rule still uses the unaltered natural evaluation.

A pathology is characterized by a decrease in correct decisions with an increase in search depth. Therefore, we measure performance in terms of the fraction of correct decisions made at the root node, where a returned move is "correct" when its true minimax value is maximal among moves at that node. In each trial a game tree is generated, solved, and searched to several depths with minimax, EMM, and the product rule.

Figure 3 shows the fraction of correct decisions made by each algorithm using the artificial evaluator ($e = 0.1$) on 10,000 non-trivial P-games with $d = 10$ and branching factors of 2 and 3. EMM clearly outperforms the other algorithms as the search depth increases. In this set of experiments, EMM is the only algorithm that does not exhibit pathological characteristics. Figure 4 once again shows the performance of the three algorithms, but this time the natural evaluator is used. Here EMM is slightly outperformed by the product rule, which is unsurprising given that the product rule was developed specifically for use in P-games with the natural static evaluation function (Tzeng and Purdom 1983). However, EMM is non-pathological and significantly outperforms minimax search.

The results of applying EMM to N-games are also promising. Figure 5 shows that EMM significantly outperforms both of the other algorithms and is not pathological. Also, similar to P-games, when using the natural evaluator, EMM is slightly outperformed by the product rule, but EMM performs substantially better than minimax with increasing search depth.

Related Work

Since the discovery of minimax pathology thirty years ago (Nau 1979), several explanations have been proposed for why it does not occur in most real-world games. Probably the most widely accepted explanation is that pathology is inhibited by similarity among different parts of the search tree (Nau 1982; Bratko and Gams 1982; Lustrek, Gams, and Bratko 2006),

Pearl (Pearl 1984) claimed that although the sibling node dependence eliminates pathology, it is not sufficient to explain the lack of pathology in real games such as chess and checkers. The level of dependence required to combat pathology does not occur in real games. Pearl suggested *traps* as an alternative explanation. Traps are moves that cause the game to end abruptly, introducing very accurate, if not perfect, heuristic values at some shallow nodes in the game tree. This in turn produces more accurate propagated values and eliminates pathology.

More recently, Lustrek et al. (Lustrek, Gams, and Bratko 2006) cited granularity of the evaluation function as a source of pathology. The granularity is the number of possible values that an evaluation function can return with a non-zero probability. They discovered that the required granularity is closely related to the branching factor. In the dependent scenario, where sibling values were normally distributed around the parent, the required granularity to avoid pathology is rather low. On the other hand, in an independent scenrio, where sibling values are independent, the required granularity grows exponentially with the branching factor. They suggest that since most chess programs use an evaluation function with thousands of possible values then this is a likely reason that chess and other real games are not pathological.

Sadikov et al. (Sadikov, Bratko, and Kononenko 2005) differentiated between two types of accuracy affecting pathology: evaluation and decision accuracy. Evaluation ac-

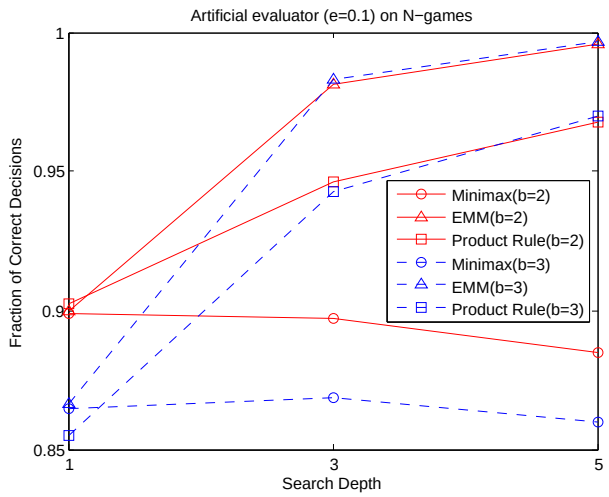

Figure 5: Fraction of correct decisions using the artificial evaluator ($e = 0.1$) on N-games.

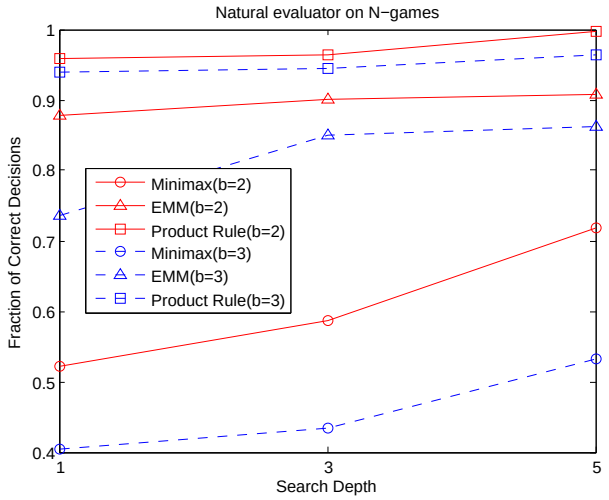

Figure 6: Fraction of correct decisions using the natural evaluator on N-games.

curacy refers to the difference between heuristic values and the backed up values. On the other hand, decision accuracy is a measure of how many correct decisions are made by a deeper search compared to a shallow one. Their experimental results on the King-Rook-King chess endgame show that although a heuristic evaluation may be increasingly inaccurate with deeper search, the decision accuracy may actually improve. The explanation for this unexpected result is that heuristic evaluators, by nature, introduce a bias into the evaluation values. The bias is similar among all nodes on the search frontier so the relative ordering among nodes is preserved. It is for this reason that we focus on decision accuracy as our measure of performance in our experiments.

Tzeng and Purdom show, in (Tzeng and Purdom 1983) that using a product rule rather than minimax for backup in P-games results in an algorithm that has no pathological tendencies and provide theorems to that effect. Unfortunately, their analysis is limited to "natural" evaluation function in this paper[4], and while their backup rule is shown to be non-pathological, there are no guarantees that more accurate techniques may exist. The search procedure in this paper differs from the product rule they propose by being more generally applicable and by allowing for the search to perform cutoff, avoiding pathological sections of the game tree.

All of the work above either suggests potential sources of pathology or classifies a set of games as being pathological. Based on that work, it is clear that identifying a single or even a handful of sources of pathology is a difficult task. Instead of isolating the cause of pathology, we propose to detect when it begins to manifest itself during the propagation process and truncate the pathological portions of search at a shallower depth.

[4]According to our experiments, the product rule does exhibit pathology when used with an artificial evaluation function.

Conclusion

We have presented an analysis of pathology in game trees showing certain kinds of subtrees (i.e. type D trees) to increase evaluation error and therefore to be the cause of pathology in game tree search. Further, we have argued that such trees exist in all interesting games, even those not known to be pathological. With that in mind, we present a new minimax-based algorithm for determining the best move in arbitrary game trees. The algorithm is designed to recognize and avoid searching pathological portions of a game tree, while still searching non-pathological portions of the tree. In experimentation, the algorithm performed well: it never exhibited pathology and always performed best or nearly best among the algorithms tested.

Future work on this algorithm includes theoretical analysis of its ability to recognize the best move, as well as looking into adapting something akin to alpha-beta pruning. Also, an examination of this algorithm in a real game, such as chess or checkers, is needed to establish the practicality of our model – it may turn out to be quite difficult to estimate the error rates of static evaluation functions for real games. Finally, the algorithm should be adapted to handle non-binary static evaluation functions.

In conclusion, we can say that by incorporating the error of the static evaluation function in the search, we were able to improve upon the abilities of minimax search. We think this may be a generally applicable lesson: when heuristic values exist in an algorithm, it may be advantageous to treat those values as probabilistically valid rather than blithely assuming them accurate.

Acknowledgments. This work was supported in part by AFOSR grant FA95500610405, NAVAIR contract N6133906C0149, DARPA's Transfer Learning and Integrated Learning Program, and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

References

Beal, D. F. 1980. An analysis of minimax. In M.R.B., C., ed., *Advances in Computer Chess 2*, 103–109. Edinburgh University Press.

Bratko, I., and Gams, M. 1982. Error analysis of the minimax principle. *Advances in computer chess* 3:1–15.

Lustrek, M.; Gams, M.; and Bratko, I. 2006. Is real-valued minimax pathological? *Artificial Intelligence* 170(6-7):620–642.

Nau, D. S. 1979. *Quality of Decision Versus Depth of Search on Game Trees*. Ph.D. dissertation, Duke University.

Nau, D. S. 1982. An investigation of the causes of pathology in games. *Artificial Intelligence* 19(3):257–278.

Osborne, M. J., and Rubinstein, A. 1994. *A Course In Game Theory*. The MIT Press.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

Sadikov, A.; Bratko, I.; and Kononenko, I. 2005. Bias and pathology in minimax search. *Theoretical Computer Science* 349(2):268–281.

Tzeng, C. H., and Purdom, P. W. 1983. A theory of game trees. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (Karlsrahe, West Germany, Aug. 8-12) Morgan Kaufmann, Los Altos, Calif*, 416–419.