Received: 5 September 2016 | Revised: 30 October 2017 | Accepted: 15 January 2018

DOI: 10.1111/coin.12162

ORIGINAL ARTICLE

WILEY Computational Intelligence

Avoiding game-tree pathology in 2-player adversarial search

Inon Zuckerman[1] | Brandon Wilson[2] | Dana S. Nau[2]

[1]Department of Industrial Engineering and Management, Ariel University, Ariel, Israel

[2]Department of Computer Science, University of Maryland, College Park, MD, USA

Correspondence
Inon Zuckerman, Department of Industrial Engineering and Management, Ariel University, Ariel 40700, Israel.
Email: inonzu@ariel.ac.il

Abstract

Adversarial search, or *game-tree search*, is a technique for analyzing an adversarial game to determine what moves a player should make in order to win a game. Until recently, lookahead pathology (in which deeper game-tree search results in worse play) has been thought to be quite rare. We provide an analysis that shows that every game should have some sections that are *locally* pathological, assuming that both players can potentially win the game.

We also modify the minimax algorithm to recognize local pathologies in arbitrary games and cut off search accordingly (shallower search is more effective than deeper search when local pathologies occur). We show experimentally that our modified search procedure avoids local pathologies and consequently provides improved performance, in terms of decision accuracy, when compared with the minimax algorithm. In addition, we provide an experimental evaluation on the African game of Kalah, which shows the improved performances of our suggested error-minimizing minimax algorithm when there is a large degree of pathology.

KEYWORDS

adversarial search, game playing, game-tree search, 2-player games

1 | INTRODUCTION

For many years and across many cultures, games have been considered to be a model of real-life strategic situations, and skilled game-playing abilities are still regarded as a distinctive mark of human intelligence. Game playing is a perfect laboratory for studying complex problem-solving

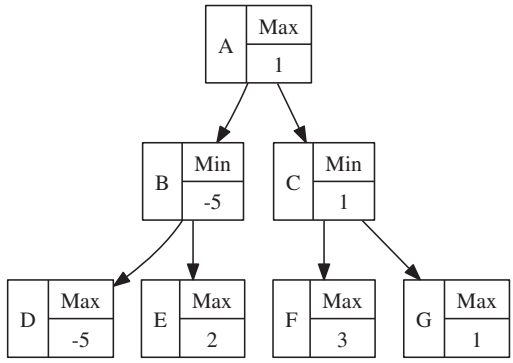

FIGURE 1 An example tree using minimax search

as they are easy to represent and pose a restricted environment in terms of the available actions as defined by the rules of the game.

Adversarial search, or *game-tree search*, is a technique for analyzing an adversarial game to determine what moves a player should make in order to win a game. In game trees, nodes represent world states (eg, an arrangement of pawns on the board game), and edges represent actions (eg, moving a pawn from one board position to another). One of the state-of-the-art adversarial search algorithms is based on the *minimax* theorem, which is known to return an optimal move in 2-player, perfect information, zero-sum games, when allowed to search the *entire game tree*.[1]

However, since many games have combinatorically large game trees that are far too large to permit exhaustive search in normal game play, implementations of the minimax algorithm, as proposed by Shannon,[2] generally involve searching to a limited depth d, applying a heuristic function called a *static evaluation function* that estimates the utility value of the nodes at that depth, and inserting these estimates into the recursive minimax formula in place of the nodes' true utility values. The values of nodes with depths of less than d can then be computed as normal, giving estimated utility values for each possible move.

For example, the game tree depicted in Figure 1 is limited to depth 3 ($d = 3$), where, in this bound, the max player heuristic values are written in the leaves. When these values are propagated upward in the tree, the min player on depth 2 will pick the node that minimizes the heuristic value (hence choosing node D over E and G over H). Consequently, the max player of depth 1 will be faced between choosing a game state with a heuristic value of -5 from node B or 1 from node C. Therefore, the move that takes him to node C would be the output of the minimax algorithm.

Various elaborations and enhancements of this formula have been used in computer programs that outperform humans in a large variety of board games, the most famous example being the Deep Blue program for the game of Chess.[3] There is a wide array of work that extends the algorithm in various directions. Some look to define more accurately the assumptions on the opponents' behavior.[4,5] Some examine and develop various pruning techniques.[6,7] Several works look at modeling and exploiting their opponents,[8,9] whereas others look at the search from the perspective of a single agent or the physical environment.[10,11] There are also game-specific techniques that enhance player ability by examining the tree structure and extensions to different types of environments (eg, partial information[12]).

Nevertheless, all of these adversarial search methods and their extensions are implicitly based on the following assumption: searching deeper **improves** the quality of decisions.

Although no theoretical model has supported this belief,[13] in practice, in all popular games (eg, Chess and Checkers), algorithms that searched deeper almost always resulted in better automated players.[14,15]

However, in the early 1980s, Nau[16] and Beal[17] independently discovered that there are infinitely many games that exhibit a phenomenon known as *game-tree pathology*, in which deeper minimax search results in worse performance. Several years later, Mutchler[18] proved that the Max^n algorithm also suffers from the same pathology. In other words, pathological situations are one in which searching deeper consistently *degrades* decision quality.

In spite of the fact that there have been several attempts to study this phenomenon,[13,17-25] it has generally been thought to be quite rare in real games and occur only in artificial game trees that do not model real games.

Nevertheless, recent work[26] has shown that pathological *situations* can occur in Chess (in 2 specific endgame positions), the African game of Kalah, and the single-player 8-puzzle. These new insights suggest that the widespread assumption that real games are *not* pathological should be revisited. In this research, we explore the problem both from a *theoretical* point of view and from an *algorithmic* one. Specifically, in this paper, we make the following contributions.

- We present a thorough review of the game-tree pathology problem.
- We analyze and detect *local* pathologies and show how they arise within a game tree and that local pathologies are likely to occur in all interesting games.
- We show how to modify the minimax search procedure to recognize and overcome local pathologies. Our modified search algorithm is called *error-minimizing minimax* (EMM), and it works by tracking both the minimax value of a node and the error associated with it. 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.
- We provide experimental results showing that the EMM provides improved decision accuracy compared to the minimax algorithm. We also show that EMM exhibits no pathology even in situations where minimax does exhibit pathology.
- We evaluate experimentally the EMM algorithm in the African game of Kalah and show a setting in which it performs better than minimax for all depths.

We start our presentation in the next section with a review of the game-tree pathology problem. We will present its background, causes, and various insights that were collected for almost 30 years of research. We then move to Section 3 with the setup and analysis of our theoretic model. Our EMM algorithm is presented in Section 4, and our experimental evaluation is presented in Section 5. We then conclude with a discussion (Section 6) and future directions.*

2 | BACKGROUND AND RELATED WORK

When an automated agent needs to make a decision about what course of action to take, it is believed that looking further ahead to predict the result of that action will lead to a better decision. For instance, when playing Chess, it is believed that a player who can visualize in his mind 6 moves ahead will be better than a player who can visualize 3 moves ahead. The intuition is that as we look further ahead, we will be getting closer to the end of the game and see events that we would have missed with a shallower observation.

*The preliminary results of this paper were presented at the European Conference on Artificial Intelligence.[27]

This principle has been demonstrated many times with automated Chess and Checkers players who were using the minimax algorithm and showed tremendous improvement with an improvement in the processing power that allowed them to search deeper in the game tree.[14,15]

However, in the early 1980s, Nau[16] and Beal[17] independently found out that there is a class of games in which searching farther ahead consistently led to worse decisions rather than better ones. This property was denoted as *lookahead pathology* or *game-tree pathology*.

Over the years, there have been several attempts to explain the phenomena and describe factors that affect it. The first of the factors that has been shown to correlate with game-tree pathology is the game tree's **branching factor**. The branching factor is the number of successor nodes at each node in the tree. In his original publication, Nau presented a mathematical proof that there exists a (large) class of games in which game-tree pathology was inevitable if the branching factor was sufficiently large. This is so because of a tendency in the minimax algorithm to eliminate low values at Max's move and high values at Min's move. In later papers, Nau[19,28] experimented with Pearl's game and showed experimentally that game-tree pathology was indeed more likely with large branching factors.

In the early years of research on game-tree pathology, people were looking into the phenomena using only a 2-valued function: a win and a loss. Later on, the **granularity** of the heuristic function (that is, the number of different values the heuristic function can have) was thought to be one of the factors that causes game-tree pathology. Early on, Bratko and Gams[20] as well as Pearl[29] compared the granularity value of 2 to higher granularities and concluded that higher granularities do not prevent game-tree pathology.

However, later on, Scheucher and Kaindl[22] considered multivalued evaluation functions (ie, high granularities) and showed (using simulation studies) that the multivalued model did cause a sharp error reduction for deeper search using minimaxing. More recently, Luštrek et al[30] also showed that as the branching factor increases, the granularity needed to avoid game-tree pathology also increases. This is somewhat more intuitive than the above factor because a low-granularity heuristic function will not be able to distinguish among game states that are different from one another in a finer manner. Consequently, deeper search is more likely to return the same value for every child in the current node, thus reducing its accuracy.

Probably the most accepted cause for pathology is the notion of **local similarity**, that is, similarity among the utility values of nearby nodes in the game tree. Researchers were using different methods to control the local similarity feature and explore it in their model. Beal[31] included in his game trees a fraction of nodes with all the successors having the same utility. Nau[19,28] used a modified Pearl game to construct instances of the game while controlling the amount of dependencies between the nodes (this is further discussed in Section 5). Pearl[13] 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. Others[22,30] used an incremental approach that is also somewhat similar to Nau's model.

Sadikov et al[23] differentiated between 2 types of accuracy affecting pathology: evaluation accuracy and decision accuracy. Evaluation accuracy 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 deeper search compared to shallow search. 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.

All of the above methods for controlling local similarities showed that eliminating local similarity resulted in game-tree pathology. In other words, local similarities between the nodes, as often occurs in real games, reduce the phenomena of game-tree pathology in the game tree. Another factor that affects game-tree pathology is the **graph structure** of the game tree. Specifically, Nau[32] showed that a class of games normally pathological is shown to become nonpathological when the games are modified so that game positions can be reached by more than one path. Another factor is with respect to the **reliability of evaluated nodes**, that is, if sufficiently many nodes in a game tree are evaluated reliably compared to other nodes on their level, performing minimax to lower depths will reduce the heuristic error.[20] Nevertheless, in a recent paper, Nau et al[26] showed that both of those factors can be considered specific cases of the local similarity factor discussed above.

About 20 years after the first observation of the game-tree pathology phenomena, in a paper by Bulitko et al,[24] the authors showed that game-tree pathology can also be found in a **single-agent search**, that is, even when there is no opponent in the game. Single-agent search usually revolves around the A^*, IDA^*, RTA^*, or *LRTS* algorithms,[33-36] which attempt to estimate the cost from the current state to a goal state. Nowadays, *UTC* and *MCTS* are very popular game-tree techniques for the real-life application of game-tree search in various domains, for example, in general (video) game playing.[37-39] In several research papers described below, the above algorithms have been shown to behave pathologically in different problem instances.

In the work of Luštrek,[40] the author shed some light on the causes for game-tree pathology in single-agent searches. His experiments showed that the distribution of true values is just one cause for pathology, the other being the heuristic function itself. Luštrek concluded that pathology has been observed even for consistent and admissible heuristic functions, and the reasons are still unknown. We do know that pessimistic heuristic functions (functions that never *underestimate* the difficulty of the problem) were found to be less prone to game-tree pathology in synthetic game trees,[40] the 8-puzzle problem,[25,41] and path-finding problems[42] in which an empirical study showed a degree of pathology in over 90% of the problems considered.

Game-tree pathology has also been observed in **multiplayer game-tree search**, where the straightforward extension of the minimax algorithm, Max^n,[43] has been shown to also suffer from pathological behavior.[18] A recent attempt to cope with game-tree pathology in multiplayer games was described in the work of Shmueli and Zuckerman.[44]

Finally, a large experimental study was performed to examine the relationship between the degree of pathology in a game tree and the 3 prominent causes of pathology, namely, branching factor, local node similarity, and evaluation function granularity.[26] In that study, the authors defined the degree of pathology for a search of depth d as the fraction of correct decisions made by searching to depth d over the fraction of correct decisions made by searching one level deeper. Experimenting on synthetic trees, they discovered that, in general, pathology is more likely to occur, and has more severe effects, when searching with a higher branching factor, lower evaluation function granularity, and lower local node similarity. Expanding their study to include real games, they showed that endgame databases exhibit some degree of local pathology despite being, overall, nonpathological (5.5%-9.2% of positions for chess were pathological). They also showed that the African game of Kalah (for a sufficiently high branching factor) is the first real game to consistently exhibit pathology throughout the game. In addition, experiments on the single-agent 8-puzzle showed that 19.7% of positions exhibit pathology.

All of the work above either suggests potential sources of pathology or classifies a set of games as being pathological. Based on the literature review, 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, in

this work, 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. This will be explained in the following section with our theoretical analysis of "local game-tree pathology."

3 | THEORETICAL ANALYSIS

In this paper, we are looking at perfect information, zero-sum games for 2 players. 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. Utilities can then be propagated using the standard minimax formula.[2]

$$minimax_d(n) = \begin{cases} eval(n), & \text{if } d = 0 \\ u(n), & \text{if } n \text{ is terminal} \\ \max_{n' \in m(n)} minimax_{d-1}(n'), & \text{if p1's move} \\ \min_{n' \in m(n)} minimax_{d-1}(n'), & \text{if p2's move.} \end{cases}$$

To determine which move is best, one needs to simply compute the minimax values and then pick a state with a maximal minimax value. When ambiguous, we will use the term *correct minimax value* (or *correct maxn value*) to refer to the value of a node n computed according to $minimax(n)$. In the above formula, $u(n)$ represents the *real* utility values when reaching a terminal state, whereas $eval(n)$ is the *static evaluation function* that provides an estimate of the utility value of the nodes at the depth limit.

We still do not have a decision procedure to verify whether a certain game *is* pathological or not. In the following analysis, we show that the question is not a binary one; rather, we claim that *every* game has pathological situations. We call these pathological situations *local pathologies*. As a consequence (and in accordance with the work of Nau et al[26]), one can say that different games exhibit different degrees of local pathologies.

To simplify the presentation, we start with a quick analysis of a game with a branching factor of 2,[†] showing that local pathologies are likely to occur in all interesting games. For this analysis, we will assume a static evaluation function that returns the correct utility value on any given node with probability $1 - e$ (similar to the model used in the work of Delcher and Kasif[21]), which also means that incorrect values will be returned with probability e.

We will be looking at the evaluation error at nonterminal nodes. An 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 applied to the parent node. In this case, the evaluation error will simply be that of the static evaluation function, ie, e. When deeper minimax searches occur, we have different evaluation errors for different types of nodes. Here, we examine only searches of depth 1, as any search to depth d can be instead thought, for the sake of analysis, as many depth-1 searches.

In games with a branching factor of 2, there are 4 possible types of nonterminal nodes. These are shown in Figure 2 (nodes B and C are symmetric and are therefore considered together). At

[†]We hope it will be obvious how the analysis will extend to higher branching factors. The pseudocode of our algorithm is general to any branching factor.

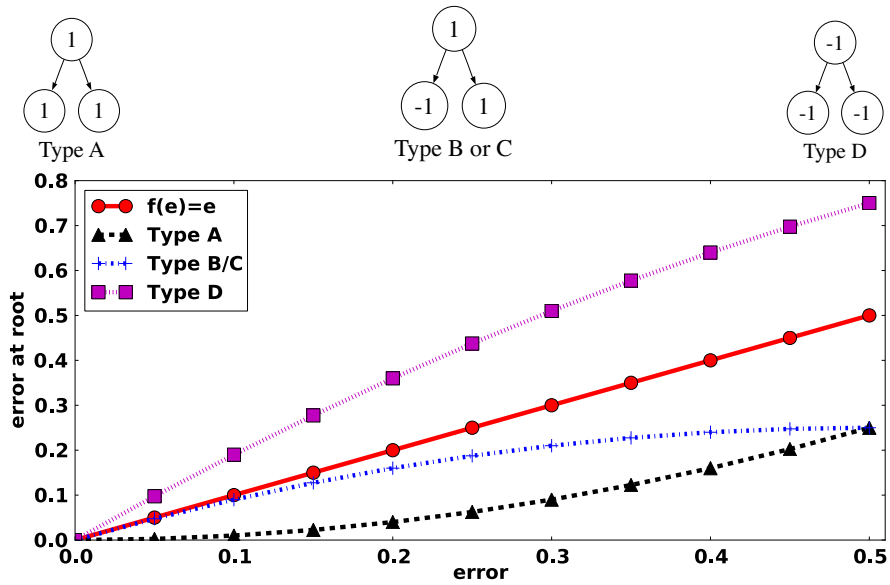

FIGURE 2 The different types of nonterminal nodes when searching forward one level of a minimax search. Types *B* and *C* are mirror images of one another, so are presented together. The graph shows the relationship between the errors in the nodes and their children after a minimax search. Notice that only type-*D* nodes increase the error [Color figure can be viewed at wileyonlinelibrary.com]

each node, it is player 1's move; thus, the node's minimax value is the maximum of the minimax values of its children (which are not terminal, but rather the search's horizon).

Using an evaluation function with error e, we can calculate the probability that a depth-1 minimax search will return the wrong value for the root node in each type of node, ie,

$$\begin{aligned} error(A) &= e^2 \\ error(B) &= e(1-e) \\ error(C) &= error(B) \\ error(D) &= 1-(1-e)^2. \end{aligned}$$

In trees of type A, the root player sees that there is a forced "win" for him. Therefore, in order for him to "loss," or in other words, in order for this "win" label to be false and changed to a "loss" label, it must be the case that both his children are mislabeled. This is because if only one is incorrectly labeled, he will still win from taking the other action (the one that is still marked as "win"). Thus, the "win" label at the root will need to be changed to a "loss" only if *both* children will be mislabeled. Mislabeling a child occurs at the probability of e (as this is the error of the evaluation function); therefore, as we need both to be mislabeled, we arrive at e^2.

In trees of types B and C, the root player sees that there is a win for him at one of his children (the other child is a loss). Therefore, in order for him to "loss," or in order for this "win" label to be false and changed to a "loss" label, it must be the case that both his "win" children are wrong (probability of e) and the "loss" child stays true (probability of $1-e$); therefore, we arrive at $e(1-e)$.

Last, for trees of type D, the root player sees a forced "loss," and in order for it to be really a "win," it can be due to 3 possibilities: both children can be mislabeled and changed to a "win," but even if only one of them turns out to be mislabeled, it is enough, as the root player would

choose it and not the "loss" node. We simply compute as *all possibilities minus the possibility that both children are labeled correctly with a "loss."* Therefore, we arrive at $1-(1-e)(1-e)$.

When comparing these functions by simply applying the static evaluation function with error e to the root node, we get

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

for any error $e \in [0, 0.5]$.[‡] That is, the error resulting from searching below type-D nodes exceeds the error resulting from simply applying the static evaluation directly, whereas for nodes of types A, B, and C, the error for depth-1 search is less than that for simply applying the evaluation function. Figure 2 shows this relationship in a graph, where we plot the value of e against the error present at each type of node for simply evaluating the node ($f(e) = e$) and for searching below it.

Only in type-D nodes is the error at the root greater than the error at the leaves, and, since any depth-d search can be seen as a combination of d depth-1 searches, we can conclude that type-D nodes are the source of search pathology. This is not to say that any time one reaches a type-D node, a shallower search should be preferred—it may be that each child of a type-D node is a type-A node, in which case the error at the root will be $1-(1-e)(1-e)$, which is less than e. However, if the entire tree consisted of nodes of type A, B, or C, then there *could not be evaluation pathology*.[§]

We expect all interesting games to contain nodes of type D. This is especially true for zero-sum games as they are not interesting if 1 player always wins, and without type-D nodes, it would be impossible for another player to win! (having nodes of types C and B will always result in the root player winning, unless he decides to choose the losing action on purpose).

4 | THE EMM ALGORITHM

Our search algorithm is based on the minimax algorithm but also tracks the error associated with the node value. Its input is a game state and outputs an action to take. The search computes the static evaluation function at any given node. If the static evaluation allows a tighter error bound than the propagated value, then that value and error bound are substituted in the final return statement.

We now detail a short example of how EMM might traverse a given tree, shown in Figure 3. This tree shows a depth-2 search—the leaf nodes are nonterminal 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 node 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 a 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 avoids the 10% error resulting from the static evaluation function, and EMM assigns a loss with 9% error for node C. We can now conclude that node A is a win, with a 9.1% error rate. Only if node B turns out to be incorrect (10%) and

[‡]The error is limited to 0.5 as larger values mean that the evaluation function is *worse* than strict random selection.
[§]So long as the static evaluation function mislabels each node with independent probability e.

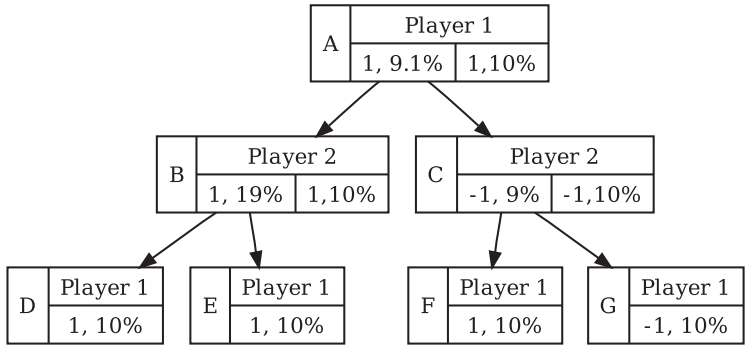

FIGURE 3 An example tree using error-minimizing minimax search

node *C* stays correct (91%) will node *A* be incorrectly labeled. 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%.

Figure 4 details the EMM algorithm. By keeping track of both the error from searching and the error from evaluating, the algorithm naturally distinguishes between pathological nodes (type *D*) and nonpathological nodes (types *A*, *B*, and *C*). However, it is important to note that as the true node type is not explicitly known to the algorithm (just an estimation that is based on the propagated values), the algorithm might often misestimate the node type and propagate according to incorrect rules. Further, notice that the algorithm is not limited to a branching factor of 2.

5 | EXPERIMENTAL EVALUATION

In order to evaluate the algorithm, we have conducted an extensive set of experiments on 2 different games, the first being the board-splitting game.[13] This game will provide us a clean environment in which we can easily control the depth of the tree and the degree of pathology, and having a branching factor of 2, it will allow us to correctly evaluate the percentages of "correct" decisions of both algorithms. However, as the board-splitting game is not a very popular game, we have conducted an additional set of experiments on the African game of Kalah (also called Mancala), which will allow us to evaluate the EMM algorithm's ability to perform well in a real game.

5.1 | The board-splitting game

The board-splitting game was developed by Pearl.[13] It is a perfect information game in which 2 players take turns dividing a 2-dimensional board, consisting of 1's and 0's, into *b* equal pieces and discarding all but one piece. Player 1 splits the board vertically and decides which half of the board to keep, then player 2 splits the board horizontally and decides which half to keep, and the other way around. The game is over when only 1 square remains. If this square is a 1, then the last player to move is declared the winner; otherwise, the other player wins. A running example of an 8×8 game instance is depicted in Figure 5.

We focus on 2 versions of the game that differ only in the construction of the initial board. The first version is referred to as a P-game (short for the Pearl game). The initial board for each P-game is generated so that each square is randomly and independently assigned a value of 1 with

Algorithm 1 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_s, 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. $\gamma(s, mv)$ is the state-transition function, returning the new state after making move mv from state s.

```
Let curVal = eval(s), and curErr = e_s.
if d is 0, return (curVal, e)
{Determine values v_i and errors er_i for children nodes.}
Let mv_1, ..., mv_n be the moves from s.
for i = 1, ..., n do
   (vTmp_i, er_i) = EMM(γ(s, mv_i), eval, d − 1)
end for
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 − ∏_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 × er_i
      else
         aggErr = aggErr × (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 ≥ aggErr then
   {Non-pathological case: statically evaluated error is greater than search's error. Use
   minimax results.}
   return (val, aggErr))
else
   {Pathological case: the statically evaluated error is less than the search's error. Use
   static results.}
   return (curVal, curErr)
end if
```

FIGURE 4 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_s, and search depth d, returns a pair (a, e), where a is the valuation of state s and e is the error associated with that valuation. $\gamma(s, mv)$ is the state-transition function, returning the new state after making move mv from state s

probability p and a 0 with probability $1 - p$. The board size itself is $b^{\lfloor \frac{d}{2} \rfloor} \times b^{\lceil \frac{d}{2} \rceil}$, 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 an N-game. This construction was introduced by Nau[19] to emulate the dependence of heuristic values among siblings, in order to create nonpathological instances of the game. For an initial board of size $b^{\lfloor \frac{d}{2} \rfloor} \times b^{\lceil \frac{d}{2} \rceil}$, 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 up the edge values from the root to that leaf, giving the leaf a value of 1 if the sum is positive and 0 otherwise.

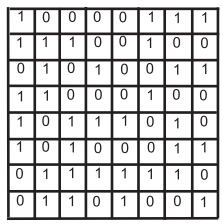

fig1: Initial board

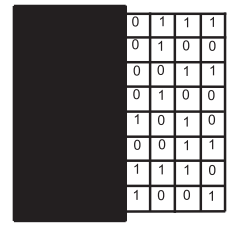

fig2: player 1-1st move

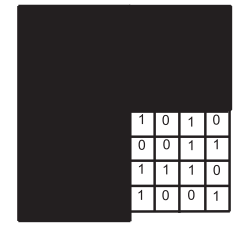

fig3: player 2-1st move

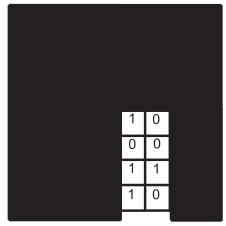

fig4: player 1-2nd move

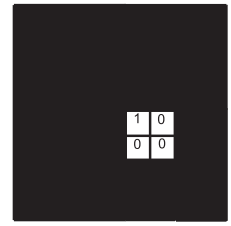

fig5: player 2-2nd move

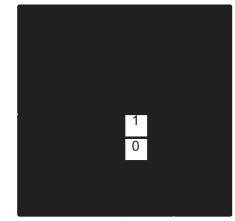

fig6: player 1-3rd move

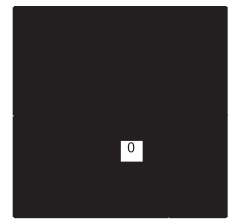

fig7: player 1 last move

FIGURE 5 An example of an 8×8 board-splitting game

Since these 2 versions of the game are considered to be on opposite ends of a spectrum (in terms of degree of pathology), we also experiment on games that fall in between, where we suspect that the game tree might be more similar to that of a real game. These games are constructed and classified by an additional parameter we refer to as the mixing factor, ie, $m \in [0.0, 1.0]$. After constructing a standard N-game, there is a probability m that each square is randomly perturbed and assigned a new value according to the P-game construction method. A game with a value of $m = 0.0$ is a pure N-game, and similarly, a game constructed with a value of $m = 1.0$ is a pure P-game.

The mixing factor is similar to the *local similarity* parameter that Nau et al[26] used to generate synthetic game trees with varying local similarity. They showed that this similarity measure is inversely correlated with the degree of pathology. Therefore, we expect that our analogous game construction will generate games with a greater amount of local pathology as the mixing factor varies from 0.0 to 1.0.

Our experiments compare the performance of minimax and EMM. We also use 2 different static evaluation functions.

1. An **artificial** static evaluation function. This is a binary function that returns 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.

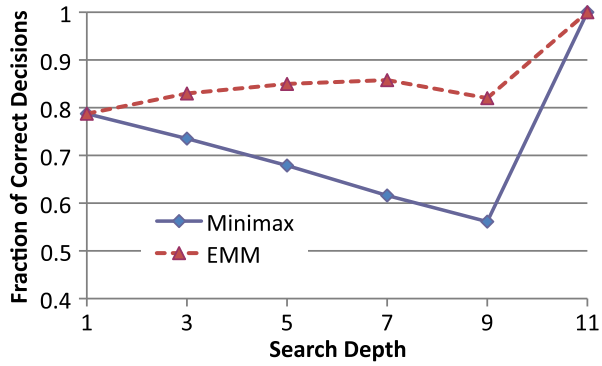

FIGURE 6 Fraction of correct decisions using the artificial evaluator ($e = 0.2$) in a 2-player P-game with $b = 2$. EMM, error-minimizing minimax [Color figure can be viewed at wileyonlinelibrary.com]

2. A **natural** static evaluation function based on the percentage of winning squares on the remaining board. The player runs many fast simulations of random moves in both parts of the board to find a winning percentage. To make this a binary evaluation (required by EMM), a state with more wins than losses is evaluated as a win, whereas one containing more losses than wins is a loss.

For the natural evaluation function, we estimated the associated error (used by the error-minimizing search) as the fraction of the board that is *not* associated with the estimated winner. For example, in a board of size 16, having 12 winning boxes will classify this board as a win, with an estimated error of $\frac{4}{16} = 0.25$.

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 other words, if a winning branch is available, a correct move will be one that directs the player toward that branch. Scenarios with different branching factors produced similar results.

Figure 6 shows the fraction of correct decisions made by each algorithm using the artificial evaluator ($e = 0.2$) on 5000 nontrivial P-games with 11 turns (ie, full game tree of height 11) and a branching factor of 2. EMM clearly outperforms minimax as the search depth increases. Both games achieve a perfect decision rate of 1.0 at search depth 11 since this equates to searching the complete game tree. We can also see that EMM does not exhibit pathological characteristics, whereas minimax does.[¶] In fact, at a search depth of 7, EMM is making over 20% more correct decisions than minimax ($P < 0.05$ in a t test for depths 5, 7, and 9).

Figure 7 shows the performance of the algorithms, but this time, the natural evaluator is used. Here, EMM is still nonpathological, whereas minimax search is pathological and loses approximately 10% accuracy by searching ahead just 3 moves to depth 7. Here, we can see that even using a more realistic evaluation function, with an estimated error, EMM still outperforms minimax with increasing search depth (in depth 7, $P < 0.01$ in a t test).

[¶]The slight drop from depth 7 to depth 9 is due to the fact that EMM does not always identify the correct node type to work on. In another series of experiments with the **true node types**, we obtained better results, and the slight drop had vanished.

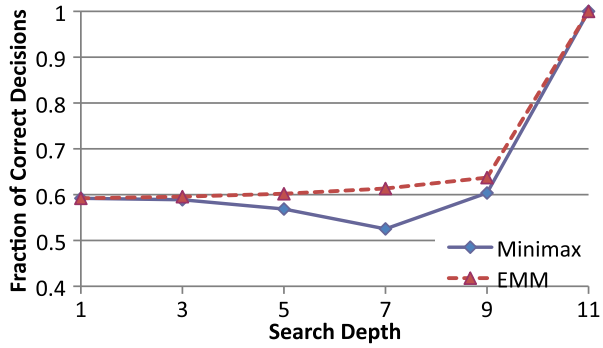

FIGURE 7 Fraction of correct decisions using the natural evaluator in a 2-player P-game with $b = 2$. EMM, error-minimizing minimax [Color figure can be viewed at wileyonlinelibrary.com]

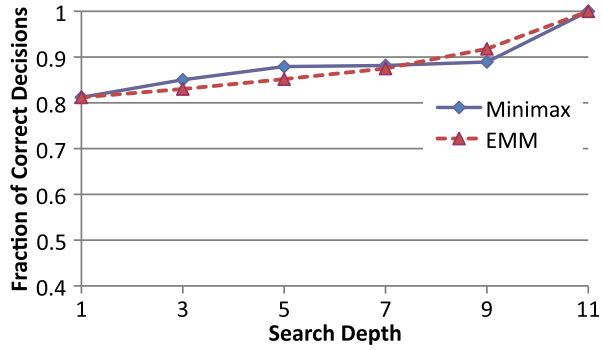

FIGURE 8 Fraction of correct decisions using the natural evaluator in a 2-player N-game with $b = 2$. EMM, error-minimizing minimax [Color figure can be viewed at wileyonlinelibrary.com]

Figure 8 shows that EMM also performs well on N-games (versions of the game that were constructed to be completely nonpathological). EMM performs comparably with minimax, which, given our independence assumption and the strong dependence among N-game siblings, is very promising.

For Figures 9 and 10, we fix the depth of the search to 5 ($e = 0.2$ as before) and observe the ratio of correct decisions made by EMM and minimax in games with a varying degree of local pathology; a number greater than 1.0 indicates that EMM is making more correct decisions than minimax. For the artificial evaluator (Figure 9), EMM always outperforms minimax, even in N-games, but it performs better as the games shift more toward P-games ($m = 1.0$) where it makes 26% more correct decisions. With respect to the natural evaluator (Figure 10), we see that around $m = 0.5$ and higher is where EMM begins to outperform minimax. This indicates that EMM is better not only in strongly pathological games (P-games) but also in games with smaller degrees of pathology.

5.2 | Product rule

EMM bears some resemblance to the product rule.[45] The product rule computes the probability that a given node is a win for player 1 and 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

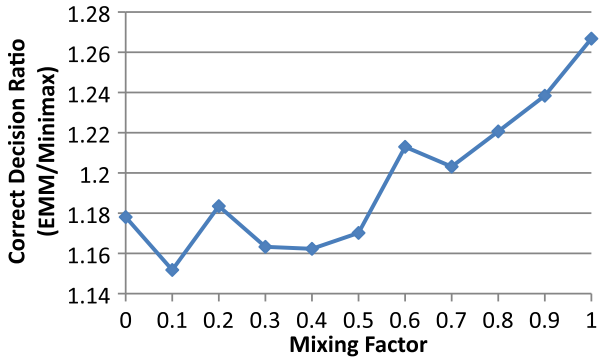

FIGURE 9 Ratio of correct decisions made by error-minimizing minimax (EMM) to the number of correct decisions made by minimax using the artificial evaluator in a 2-player board-splitting game with $b = 2$ and a varying degree of pathology [Color figure can be viewed at wileyonlinelibrary.com]

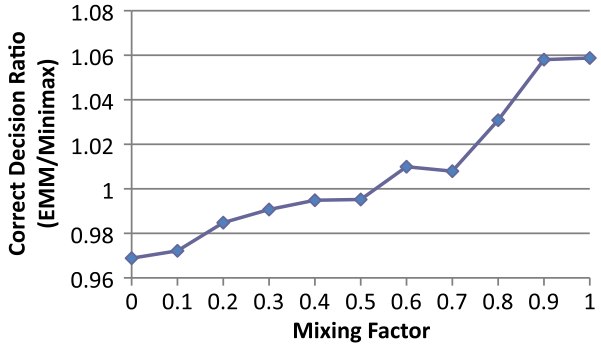

FIGURE 10 Ratio of correct decisions made by error-minimizing minimax (EMM) to the number of correct decisions made by minimax using the natural evaluator in a 2-player board-splitting game with $b = 2$ and a varying degree of pathology [Color figure can be viewed at wileyonlinelibrary.com]

search is in the short-cutting of the aggregation up the tree when the static evaluation function is less erroneous than the minimaxed value. This limits the search below nodes with pathological characteristics: when searching below a node produces more erroneous values, the error associated with that search will be higher and the results of the 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 nonpathological portions of the tree.

A product rule (PD) search to depth d can be written recursively as follows:

$$product_d(n) = \begin{cases} eval(n), & \text{if } d = 0, \\ u(n), & \text{if } n \text{ is terminal}, \\ 1 - \prod_{n' \in m(n)} product_{d-1}(n'), & \text{if p1's move}, \\ \prod_{n' \in m(n)} product_{d-1}(n'), & \text{if p2's move}. \end{cases}$$

We conducted similar experiments on the board-splitting games to evaluate the EMM algorithm against minimax and the product rule both in branching factors 2 and 3. Figure 11 shows the fraction of correct decisions made by each algorithm using the artificial evaluator on 5000 nontrivial P-games with varying depths. EMM clearly outperforms the other algorithms as the search depth

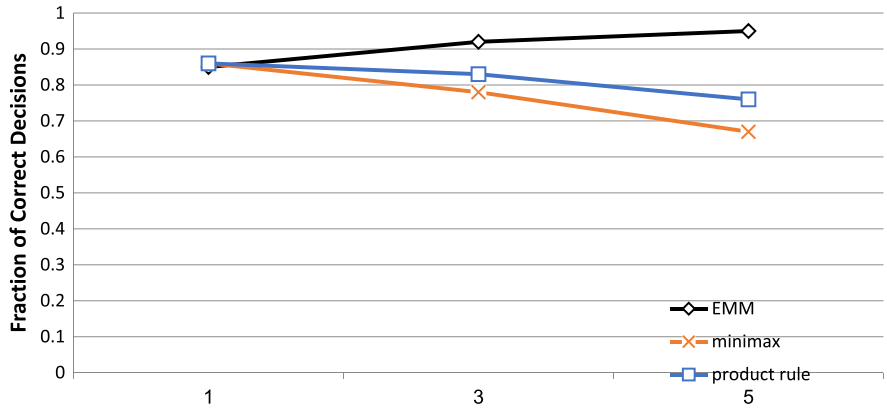

FIGURE 11 Fraction of correct decisions using the artificial evaluator on P-games. EMM, error-minimizing minimax [Color figure can be viewed at wileyonlinelibrary.com]

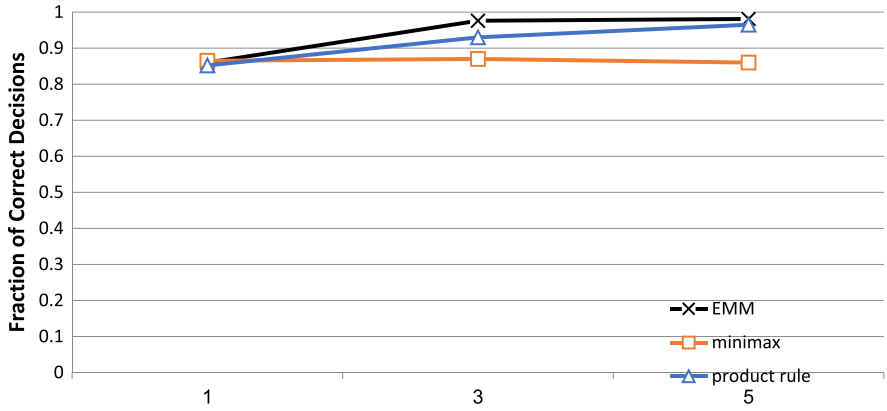

FIGURE 12 Fraction of correct decisions using the artificial evaluator on N-games. EMM, error-minimizing minimax [Color figure can be viewed at wileyonlinelibrary.com]

increases (in depth 5, $P < 0.01$ in a t test against both algorithms). In this set of experiments, EMM is the only algorithm that does not exhibit pathological characteristics. The results of applying EMM to N-games are also promising. Figure 12 shows that EMM significantly outperforms both of the other algorithms and is not pathological. The results show that EMM and the product rule both perform substantially better than minimax with increasing search depth (in depth 5, $P < 0.05$ in a t test).

5.3 | The Kalah game

The Kalah game is an ancient African game[46] (also called Mancala). A Kalah board contains a number of pits, each containing a number of seeds, in which the objective is to acquire more seeds than the opponent, either by moving them to a special pit (called a kalah) or by capturing them from the opponent's pits. The game is played by 2 players that take turns "sowing seeds." A player sows seeds by choosing a pit, scooping up all the seeds in that pit, and moving counterclockwise, dropping 1 seed in each of the pits immediately adjacent to the starting pit. If the last seed in his hand goes in his storage pit, he gets another turn. Otherwise, his turn ends. The game ends when

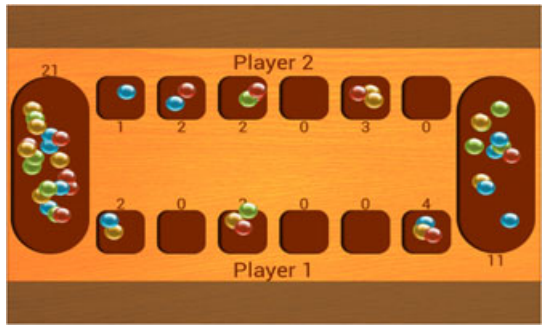

FIGURE 13 The Kalah game board [Color figure can be viewed at wileyonlinelibrary.com]

a player runs out of seeds on his side of the board, and the player with the most captured seeds wins. Figure 13 shows a diagram of the game board.

In order to cope with the complexity of the game, in our experimental study, we used the same limitations that were proposed in the work of Nau et al.[26] First, instead of ending the game when 1 player runs out of seeds, we will limit the game to a predefined number of moves. Second, in order to create a uniform branching factor, we allow making a move from an "empty" pit; such move will not have any effect on the board and it is basically a way of not taking any action. Last, we did not provide a second move when the last seed goes in the storage pit.

We used a real heuristic function that provides solid performance in real games. The function is composed of various features that were collected from the literature[47]: we looked at whether the last seed falls in the player or the opponent's kalah and whether the following opponent's move will increase the number of seeds in my side, and we aim to maximize the number of beneficial moves the player might take.

In the experiment, we generated random initial boards by distributing seeds across the available pits. Both algorithms were using the same search depth and the same heuristic function and played each random board twice: one in which the first player is minimax and the second player is EMM, and the other way around. The values of the heuristic function were normalized to the range between 0 and 1 and were mapped to the extreme values for the EMM case. We used a static error value of 0.2 that was found using trial and error. Each data point in Figure 14 is thus

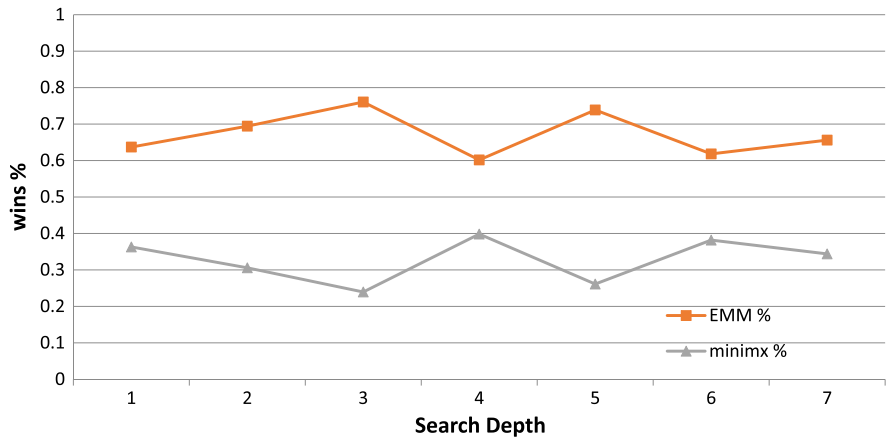

FIGURE 14 Win percentage for each algorithm as a function of the search depth. EMM, error-minimizing minimax [Color figure can be viewed at wileyonlinelibrary.com]

the result of 10 000 randomly generated games. The results show quite convincingly that in this setting, EMM has the upper hand across all depths.

6 | DISCUSSION AND FUTURE WORK

Despite the positive results we have seen, there are several potential weaknesses present in EMM that need to be addressed in future research. 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 we assume in our error propagation equations). Instead, for many natural static evaluation functions, when they are wrong about 1 game state, they are likely to also be wrong about children of that game state. Incorporating the dependence among sibling nodes is an important next step as that is the primary difference between the performance of the artificial and natural evaluators. However, even with an independent assumption among nodes, we saw that EMM performed better than minimax in games where node values were not completely independent (ie, games with degrees of pathology between P-games and N-games).

Second, it is not clear that estimating error characteristics for natural static evaluation functions the way we did for the board-splitting game (ie, scaling the evaluation function to a range of [0.0, 1.0] and treating them as probabilities) is the best approach. Understanding how the error characteristics are affected by parameters of the search, such as depth and branching factor, is another key to making EMM effective in a larger set of games. In a recent extension to this work,[44] we exemplify how the static evaluation error can be better estimated and utilized using learning techniques.

The algorithm is also limited to 2-player games. We plan to extend the work to multiplayer domains by building upon the multiplayer extension of minimax, the Max^n algorithm, where pathology has also been shown to exist.[18] We already have preliminary results in this area that look promising,[44] although the mathematical equations of the node type analysis are much more complex.

Finally, alpha-beta pruning presents a challenge for EMM, because EMM cannot calculate the errors unless it visits the nodes alpha-beta would prune. Consequently, EMM will be at a serious disadvantage if a game tree does not contain pathological nodes—but if it does contain pathological nodes, then the deeper searches performed by minimax with alpha-beta can actually degrade performance! A pruning procedure should look both at the heuristic value and the propagated error and try to approximate when to prune. A good starting point should be somewhat similar to the algorithms found in the work of Rivest.[48]

In recent years, we have seen a shift in adversarial search research toward Monte Carlo tree–based searches. In these algorithms, the decision at the root is taken based on a large number of random simulations of the interaction (see the work of Browne et al[49] for a good survey). Monte Carlo tree search showed promising results in difficult games such as Go, Chess, and others; however, to the best of our knowledge, game-tree pathology has not yet been researched with respect this technique, and it will be worthwhile exploring this point in future research.

7 | CONCLUSIONS

We have shown that, of the 4 possible types of nodes, only 1 kind of node (ie, type-*D* nodes) increases the evaluation error and, therefore, causes local pathologies in game trees. We also

present a probabilistic approach to propagating the evaluation error based on the type of node. Using these rules, we have argued that such nodes exist in all interesting games, even those not known to be pathological.

We have presented a new algorithm, based on minimax, that propagates both heuristic values *and* error estimates on those values. The algorithm uses the error estimates to recognize and avoid searching pathological portions of a game tree, while still searching nonpathological portions of the tree. In this way, the algorithm can adapt to the individual game tree and the degree of local pathology present.

In experiments performed on a board-splitting game and the Kalah game, the algorithm performed well: it always performed best or nearly identical to minimax. The results show that the performance of EMM varies as the degree of local pathology in the game changes. This leads us to conclude that EMM will be most beneficial when used in games with a medium-to-high degree of local pathology, not just purely pathological games, such as P-games.

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 in situations where such search previously performed badly. 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.

ORCID

Inon Zuckerman http://orcid.org/0000-0002-9999-1750

REFERENCES

1. Osborne MJ, Rubinstein A. *A Course in Game Theory*. Cambridge, MA: MIT Press; 1994.
2. Shannon CE. XXII. Programming a computer for playing chess. *Philos Mag*. 1950;41(314):256-275.
3. Hsu FH. Chess hardware in Deep Blue. *Comput Sci Eng*. 2006;8(1):50-60.
4. Zuckerman I, Felner A. The MP-mix algorithm: dynamic search strategy selection in multiplayer adversarial search. *IEEE Trans Comput Intell AI Game*. 2011;3(4):316-331.
5. Wilson B, Zuckerman I, Nau D. Modeling social preferences in multi-player games. Paper presented at: The 10th International Conference on Autonomous Agents and Multiagent Systems - 2011; Taipei, Taiwan.
6. Sturtevant NR, Korf RE. On pruning techniques for multi-player games. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and the Twelfth Annual Conference on Innovative Applications of Artificial Intelligence. Palo Alto, CA: AAAI Press; 2000.
7. Knuth D, Moore R. An analysis of alpha-beta pruning. *Artif Intell*. 1975;6(4):293-326.
8. Sturtevant NR, Zinkevich M, Bowling MH. Prob-max^n: playing N-player games with opponent models. In: Proceedings of the 21st National Conference on Artificial Intelligence; 2006; Boston, MA.
9. Markovitch S, Reger R. Learning and exploiting relative weaknesses of opponent agents. *Auton Agent Multi-Agent Syst*. 2005;10(2):103-130.
10. Sharon G, Stern R, Felner A, Sturtevant NR. Conflict-based search for optimal multi-agent pathfinding. *Artif Intell*. 2015;219: 40-66.
11. Felner A, Stern R, Kraus S. PHA*: performing A* in unknown physical environments. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems; 2002; Bologna, Italy.
12. Parker A, Nau DS, Subrahmanian VS. Overconfidence or paranoia? Search in imperfect-information games. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence; 2006; Boston, MA.
13. Pearl J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA: Addison-Wesley Longman; 1984.
14. Campbell M, Hoane AJ Jr, Hsu FH. Deep Blue. *Artif Intell*. 2002;134(1-2):57-83.
15. Schaeffer J, Burch N, Björnsson Y, et al. Checkers is solved. *Science*. 2007;317(5844):1518-1522.

16. Nau DS. Quality of Decision Versus Depth of Search on Game Trees [PhD Dissertation]. Duke University: Durham, NC; 1979.
17. Beal DF. An analysis of minimax. In: Clarke MR, ed. *Advances in Computer Chess 2*. Edinburgh, UK: Edinburgh University Press; 1980:103-109.
18. Mutchler D. The multi-player version of minimax displays game-tree pathology. *Artif Intell*. 1993;64(2):323-336.
19. Nau DS. An investigation of the causes of pathology in games. *Artif Intell*. 1982;19(3):257-278.
20. Bratko I, Gams M. Error analysis of the minimax principle. *Advances in Computer Chess*. Oxford, UK: Pergamon Press; 1982.
21. Delcher AL, Kasif S. Improved decision-making in game trees: recovering from pathology. In: Proceedings of the Tenth National Conference on Artificial Intelligence; 1992; San Jose, CA.
22. Scheucher A, Kaindl H. Benefits of using multivalued functions for minimaxing. *Artif Intell*. 1998;99(2):187-208.
23. Sadikov A, Bratko I, Kononenko I. Bias and pathology in minimax search. *Theor Comput Sci*. 2005;349(2):268-281.
24. Bulitko V, Li L, Greiner R, Levner I. Lookahead pathologies for single agent search. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence; 2003; Acapulco, Mexico.
25. Piltaver R, Luštrek M, Gams M. Search pathology of 8-puzzle. In: Proceedings of the 10th International Multiconference Information Society; 2007; Ljubljana, Slovenia.
26. Nau DS, Luštrek M, Parker A, Bratko I, Gams M. When is it better not to look ahead? *Artif Intell*. 2010;174(16-17):1323-1338.
27. Wilson B, Zuckerman I, Parker A, Nau DS. Improving local decisions in adversarial search. Paper presented at: 20th European Conference on Artificial Intelligence including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track. Montpellier, France: IOS Press; 2012.
28. Nau D. Search and heuristics pathology on game trees revisited, and an alternative to minimaxing. *Artif Intell*. 1983;21(1-2):221-244.
29. Pearl J. On the nature of pathology in game searching. *Artif Intell*. 1983;20(4):427-453.
30. Luštrek M, Gams M, Bratko I. Is real-valued minimax pathological? *Artif Intell*. 2006;170(6-7):620-642.
31. Beal DF. Benefits of minimax search. In: Clarke MR, ed. *Advances in Computer Chess 3*. Oxford, UK: Pergamon Press; 1982:17-24.
32. Nau DS. On game graph structure and its influence on pathology. *Int J Comput Inf Sci*. 1983;12(6):367-383.
33. Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans Syst Sci Cybern*. 1968;4(2):100-107.
34. Korf RE. Depth-first iterative-deepening: an optimal admissible tree search. *Artif Intell*. 1985;27(1):97-109.
35. Korf RE. Real-time heuristic search. *Artif Intell*. 1990;42(2-3):189-211.
36. Bulitko V, Lee G. Learning in real-time search: a unifying framework. *J Artif Intell Res*. 2006;25:119-157.
37. Genesereth M, Love N, Pell B. General game playing: overview of the AAAI competition. *AI Mag*. 2005;26(2):62-73.
38. Perez-Liebana D, Samothrakis S, Togelius J, et al. The 2014 general video game playing competition. *IEEE Trans Comput Intell AI Game*. 2016;8(3):229-243.
39. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature*. 2015;518(7540):529-533.
40. Luštrek M. Pathology in single-agent search. In: Proceedings of the 8th International Multiconference Information Society; 2005; Ljubljana, Slovenia.
41. Sadikov A, Bratko I. Pessimistic heuristics beat optimistic ones in real-time search. In: Proceedings of the 17th International Conference on Artificial Intelligence (ECAI 2006). Amsterdam, the Netherlands: IOS Press; 2006.
42. Luštrek M, Bulitko V. Thinking too much: pathology in pathfinding. In: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008). Amsterdam, the Netherlands: IOS Press; 2008.
43. Luckhart C, Irani KB. An algorithmic solution of N-person games. In: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence; 1986; Philadelphia, PA.

44. Shmueli T, Zuckerman I. Avoiding game-tree pathology in multi-player games. Paper presented at: 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT); 2015; Singapore.
45. Tzeng CH, Purdom PW. A theory of game trees. In: Proceedings of the Eighth International Joint Conference on Artificial Intelligence; 1983; Karlsruhe, Germany.
46. Murray HJR. *A History of Board-Games other Than Chess*. Oxford, UK: Clarendon Press; 1952.
47. Irving G, Donkers J, Uiterwijk J. Solving kalah. *Int Comput Game Assoc J*. 2000;23(3):139-148.
48. Rivest R. Game tree searching by min/max approximation. *Artif Intell*. 1987;34(1):77-96.
49. Browne CB, Powley E, Whitehouse D, et al. A survey of Monte Carlo tree search methods. *IEEE Trans Comput Intell AI Game*. 2012;4(1):1-43.

How to cite this article: Zuckerman I, Wilson B, Nau DS. Avoiding game-tree pathology in 2-player adversarial search. *Computational Intelligence*. 2018;34:542–561. https://doi.org/10.1111/coin.12162