Toward an Analysis of Forward Pruning *

Stephen J. J. Smith
Computer Science Department
University of Maryland
College Park, MD 20742 USA
sjsmith@cs.umd.edu

Dana S. Nau
Computer Science Department and
Institute for Systems Research
University of Maryland
College Park, MD 20742 USA
nau@cs.umd.edu

Abstract

Several early game-playing computer programs used *forward pruning* (i.e., the practice of deliberately ignoring nodes that are believed unlikely to affect a game tree's minimax value), but this technique did not seem to result in good decision-making. The poor performance of forward pruning presents a major puzzle for AI research on game playing, because some version of forward pruning seems to be "what people do," and the best chess-playing programs still do not play as well as the best humans.

As a step toward deeper understanding of how forward pruning affects quality of play, in this paper we set up a model of forward pruning on two abstract classes of binary game trees, and we use this model to investigate how forward pruning affects the accuracy of the minimax values returned. The primary result of our study is that forward pruning does better when there is a high correlation among the minimax values of sibling nodes in a game tree.

This result suggests that forward pruning may possibly be a useful decision-making technique in certain kinds of games. In particular, we believe that bridge may be such a game.

Introduction

Much of the difficulty of game-playing is due to the large number of alternatives that must be examined and discarded. One method for reducing the number of nodes examined by a game tree search is *forward pruning*, in which at each node of the search tree, the search procedure may discard some of the node's children before searching below that node. On perfect-information zero-sum games such as chess, forward pruning has not worked as well as approaches that do not use forward pruning (Biermann, 1978; Truscott, 1981). This presents a major puzzle for AI research on game playing, because some version of forward pruning seems to be "what people do," and the best chess-playing programs still do not play as well as the best humans. Thus, it is important to try to understand why programs have been unable to utilize forward pruning as effectively as humans have done, and whether there are ways to utilize forward pruning more effectively.[1]

As a step toward deeper understanding of how forward pruning affects quality of play, in this paper we set up a model of forward pruning on two abstract classes of binary game trees, and we use this model to investigate how forward pruning affects the accuracy of the minimax values returned. Our results suggest that forward pruning works best in situations where there is a high correlation among the minimax values of sibling nodes. Since we believe that bridge has this characteristic, this encourages us to believe that forward pruning may work better in the game of bridge than it has worked in other games.

Forward-Pruning Models

Consider a zero-sum game between two players, Max and Min. If the game is a perfect-information game, then the "correct" value of each node u is normally taken to be the well known *minimax value*:

$$mm(u) = \begin{cases} \text{the payoff at } u \\ \qquad \text{if } u \text{ is a terminal node,} \\ \max\{mm(v) : v \text{ is a child of } u\} \\ \qquad \text{if it is Max's move at } u, \\ \min\{mm(v) : v \text{ is a child of } u\} \\ \qquad \text{if it is Min's move at } u. \end{cases}$$

Due to the size of the game tree, computing a node's true minimax value is impractical for most games. For

*This work supported in part by an AT&T Ph.D. scholarship to Stephen J. J. Smith, Maryland Industrial Partnerships (MIPS) grant 501.15, Great Game Products, and NSF grants IRI-8907890, NSFD CDR-88003012, and IRI-9306580.

[1] In particular, we are developing a forward-pruning search technique for the game of bridge (Smith *et al.*, 1992; Smith and Nau, 1993), by extending task-network planning techniques (Tate, 1976; Tate, 1977; Sacerdoti, 1977; Stefik, 1981) to represent multi-agency and uncertainty.

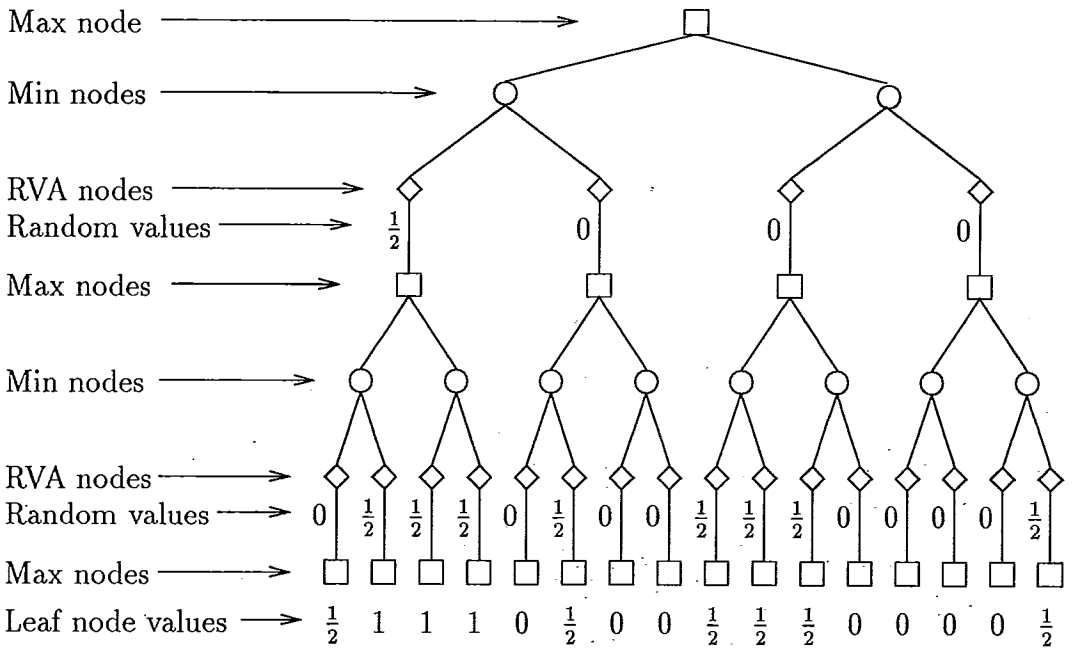

Figure 1: Example of an N-game-like tree.

this reason, game-playing programs usually mark some non-terminal nodes as terminal, and evaluate them using some static evaluation function $e(u)$. The simplest version of this approach is what Shannon (Shannon, 1950) called "Type A" pruning: choose some arbitrary cutoff depth d, and mark a non-terminal node u as terminal if and only if u's depth exceeds d. A more sophisticated version of this is *quiescence search*: mark a non-terminal node u as terminal if and only if u's depth exceeds d and u is "quiet" (i.e., there is reason to believe that $e(u)$ will be reasonably accurate at u).

To decrease the number of nodes examined even further, a number of game-tree-search procedures have been developed such as alpha-beta (Knuth and Moore, 1975), B* (Berliner, 1979), or SSS* (Stockman, 1979). These procedures will ignore any node v below u that they can prove will not affect u's minimax value $mm(u)$.

The above approach has worked well in games such as chess (Berliner *et al.*, 1990; Levy and Newborn, 1982), checkers (Samuel, 1967), and othello (Lee and Mahajan, 1990). A more aggressive approach is *forward pruning*, in which the procedure deliberately ignores v if it believes v is *unlikely* to affect $mm(u)$, even if there is no proof that v will not affect $mm(u)$. Although several early computer chess programs used forward pruning, this approach is no longer widely used, because chess programs that used it did less well than those that did not (Biermann, 1978; Truscott, 1981).

Our Model of Forward-Pruning. Our model of a forward-pruning algorithm works as follows. At each node u where it is Max's move or Min's move, u has two children, u_1 and u_2. The forward pruning algorithm will choose exactly one of these two nodes to investigate further. There are two possible cases:

1. u_1 and u_2 do not have the same minimax value. Then the *correct* child to investigate further is the one whose minimax value is the same as the minimax value of u. Thus, for a Max node, the correct child is the one with the higher value; for a Min node, the correct child is the one with the lower value. If the algorithm does not choose the correct child, it will continue its search down an incorrect branch, which is likely to result in an error in the algorithm's computation of u's minimax value. We assume that the algorithm's probability of choosing the correct child is p, where p is constant throughout the tree.
2. u_1 and u_2 have the same minimax value. In this case, we assume that the algorithm's probability of choosing u_1 is $\frac{1}{2}$, and likewise for u_2.

Game-Tree Models

In this section, we define two different classes of game trees. In later sections, we will investigate how forward pruning behaves on these trees.

N-Game-Like Trees. In this section, we define an class of abstract game trees that are similar to the N-games described in (Nau, 1982; Nau, 1983). For this

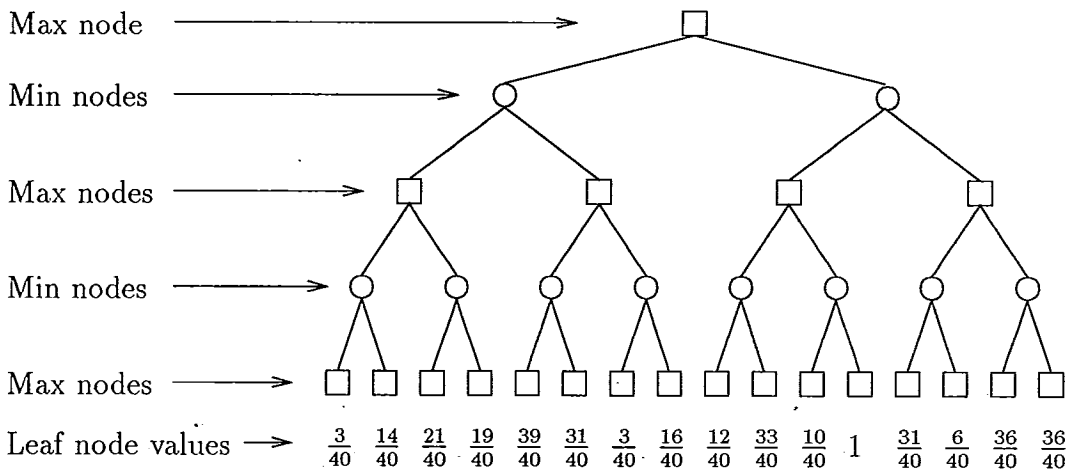

Figure 2: Example of a P-game-like tree, with $m = 40$.

reason, we call them *N-game-like trees.*[2] Each N-game-like tree is a complete tree that contains the following types of nodes (an example appears in Fig. 1):

1. *Max* nodes, where it is Max's move. Each Max node is either a leaf node or has exactly two children, both of which are Min nodes.
2. *Min* nodes, where it is Min's move. Each Min node has exactly two children, both of which are RVA nodes.
3. *RVA* (random-value addition) nodes, which have numeric values assigned to them at random, to represent the outcome of the trick. The numeric value of each RVA node is chosen independently from the set $\{0, 1/h\}$, where h is as defined below. Each RVA node has a single child, which is a Max node.

The tree's *Max-height*, h, is one less than the number of Max nodes on any path from the root to a leaf node.[3] Since the tree is complete, each leaf node has the same height, and thus the same Max-height. The value of each leaf node u is the sum of the values of the RVA nodes on the path from the root to u. Thus, the value of each leaf node falls in the range $\{0, 1/h, \ldots, 1\}$.

[2]The primary difference is that N-games have no RVA nodes. Instead, a value of 1 or -1 is randomly assigned to each arc, and the *strength* of each leaf node u is taken to be the sum of the arc values on the path from the root to u. Thus if u is a leaf node, its strength is between $-k$ and k (inclusive), where k is the height of the tree. A leaf node is called a win or loss, depending on whether or not its strength is nonnegative.

[3]This is analogous to the height of a complete tree (which is one less than the number of nodes on any path from the root to a leaf node), except that here we only count Max nodes.

Comparison with Bridge. In the game of bridge, the basic unit of play is the trick. After each side has made a move, one side or the other wins the trick. At each point in a bridge hand, the *trick score* for each side is the number of tricks that side has scored so far. The outcome of the hand depends on each side's trick score at the end of the hand.

This trick-scoring method gives bridge a superficial resemblance to the the N-game-like trees defined above. To see this, consider a node v in a bridge game tree, and suppose that v represents a bridge deal in which n tricks are left to be played. If T is the subtree rooted at v, then the trick scores of the leaves of T cannot differ from one another by any more than n. A similar situation occurs in an N-game-like tree of height h: if a Max node v has a Max-height of n, and T is the subtree rooted at v, then the value of the leaves of T cannot differ from one another by any more than n/h.

P-Game-Like Trees. In this section, we define a class of abstract game trees called *P-game-like trees*, which are similar to the P-games described in (Nau, 1982; Nau, 1983; Pearl, 1984).[4] Each P-game-like tree is a complete tree that contains the following types of nodes (an example appears in Fig. 2):

1. *Max* nodes, where it is Max's move. Each Max node is either a leaf node or has exactly two children, both of which are Min nodes.
2. *Min* nodes, where it is Min's move. Each Min node has exactly two children, which are both Max nodes.

As before, the tree's *Max-height*, h, is one less than the number of Max nodes on any path from the root to

[4]The primary difference is that in a P-game, the leaf node values are chosen from the set {win, loss} rather than the set $\{0, 1/m, \ldots, 1\}$.

a leaf node. Since the tree is complete, each leaf node has the same height, and thus the same Max-height. The value of each leaf node u is randomly, independently chosen from a uniform distribution over the set $\{0, 1/m, \ldots, 1\}$, where m is an arbitrary constant. Because u's value does not depend on the path from the root to u, there is no need for RVA nodes.

Mathematical Derivations

Forward Pruning on N-Game-Like Trees. We are interested in computing the probability that the forward-pruning algorithm estimates a value of x and the actual value is y for an N-game-like tree T whose Max-height is h. That is, we want

$$\Pr[\text{estimated } x, \text{ actual } y \mid T\text{'s Max-height is } h].$$

More specifically, we write

$$e_{h,x,y} = \Pr\left[\begin{array}{l|l}\text{estimated } x, & T\text{'s Max-height is}\\ \text{actual } y & h \text{ and its root is a}\\ & \text{Max node}\end{array}\right];$$

$$f_{h,x,y} = \Pr\left[\begin{array}{l|l}\text{estimated } x, & T\text{'s Max-height is}\\ \text{actual } y & h \text{ and its root is an}\\ & \text{RVA node}\end{array}\right];$$

$$g_{h,x,y} = \Pr\left[\begin{array}{l|l}\text{estimated } x, & T\text{'s Max-height is}\\ \text{actual } y & h \text{ and its root is a}\\ & \text{Min node}\end{array}\right].$$

These probabilities depend on p. Since all leaf nodes are Max nodes, we can write

$$e_{0,j,k} = \begin{cases} 1 & \text{if } j = 0 \text{ and } k = 0,\\ 0 & \text{otherwise.}\end{cases}$$

It is clear that $f_{h,x,y}$ depends on $e_{h,x-\frac{1}{k},y-\frac{1}{k}}$ and $e_{h,x,y}$, that is,

$$f_{h,x,y} = \frac{1}{2}e_{h,x,y} + \frac{1}{2}e_{h,x-\frac{1}{k},y-\frac{1}{k}}.$$

There are three cases for $g_{h,x,y}$:

1. The forward-pruning algorithm chooses the correct branch, i.e., the one that gives the correct minimax value. In this case, $g_{h,x,y}$ is based on $f_{h,x,y}$ and $f_{h,i,j}$ for every i and some $j > y$.
2. The forward-pruning algorithm does not choose the correct branch. In this case, $g_{h,x,y}$ is based on $f_{h,x,j}$ and $f_{h,i,y}$ for every i and some $j > y$.
3. The choice does not matter, i.e., both branches yield the same minimax value. In this case, $g_{h,x,y}$ is based on $f_{h,x,y}$ and $f_{h,i,y}$ for every i.

Thus,

$$\begin{aligned} g_{h,x,y} &= 2p\sum_i \sum_{j:j>y} (f_{h,x,y} \cdot f_{h,i,j}) \\ &\quad + 2(1-p)\sum_i \sum_{j:j>y} (f_{h,x,j} \cdot f_{h,i,y}) \\ &\quad + \sum_i (f_{h,x,y} \cdot f_{h,i,y}). \end{aligned}$$

For $e_{h+1,x,y}$, the cases are similar, yielding

$$\begin{aligned} e_{h+1,x,y} &= 2p\sum_i \sum_{j:j<y} (g_{h,x,y} \cdot g_{h,i,j}) \\ &\quad + 2(1-p)\sum_i \sum_{j:j<y} (g_{h,x,j} \cdot g_{h,i,y}) \\ &\quad + \sum_i (g_{h,x,y} \cdot g_{h,i,y}). \end{aligned}$$

Forward Pruning on P-Game-Like Trees. For a P-game-like tree T, we will define $e'_{h,m,x,y}$ and $g'_{h,m,x,y}$ in a way similar to the way we defined $e_{h,x,y}$ and $g_{h,x,y}$ for N-game-like trees:

$$e'_{h,m,x,y} = \Pr\left[\begin{array}{l|l}\text{esti-} & T\text{'s Max-height is } h, \text{ its}\\ \text{mated} & \text{root is a Max node, and}\\ x, \text{ ac-} & \text{its leaf node values are}\\ \text{tual } y & \text{in the set } \{0, \frac{1}{m}, \ldots, 1\}\end{array}\right];$$

$$g'_{h,m,x,y} = \Pr\left[\begin{array}{l|l}\text{esti-} & T\text{'s Max-height is } h, \text{ its}\\ \text{mated} & \text{root is a Min node, and}\\ x, \text{ ac-} & \text{its leaf node values are}\\ \text{tual } y & \text{in the set } \{0, \frac{1}{m}, \ldots, 1\}\end{array}\right].$$

In P-game-like trees, the base case for the recurrences is

$$e'_{0,m,x,y} = \begin{cases} \frac{1}{m+1} & \text{if } 0 \le x \le m \text{ and } x = y,\\ 0 & \text{otherwise.}\end{cases}$$

The recurrences themselves are almost identical to those for N-game-like trees:

$$\begin{aligned} g'_{h,m,x,y} &= 2p\sum_i \sum_{j:j>y} (e'_{h,m,x,y} \cdot e'_{h,m,i,j}) \\ &\quad + 2(1-p)\sum_i \sum_{j:j>y} (e'_{h,m,x,j} \cdot e'_{h,m,i,y}) \\ &\quad + \sum_i (e'_{h,m,x,y} \cdot e'_{h,m,i,y}); \\ e'_{h+1,m,x,y} &= 2p\sum_i \sum_{j:j<y} (g'_{h,m,x,y} \cdot g'_{h,m,i,j}) \\ &\quad + 2(1-p)\sum_i \sum_{j:j<y} (g'_{h,m,x,j} \cdot g'_{h,m,i,y}) \\ &\quad + \sum_i (g'_{h,m,x,y} \cdot g'_{h,m,i,y}). \end{aligned}$$

Expected Absolute Value of Difference. We can use the above recurrences to measure the accuracy of the forward-pruning algorithm's estimate of the value of the game tree. In particular, the expected value of the absolute value of the difference between the true minimax value and the value computed by the forward-pruning algorithm is

$$|x - y| \cdot \Pr\left[\begin{array}{l|l}\text{estimated } x, & T\text{'s Max-height is } h\\ \text{actual } y & \end{array}\right].$$

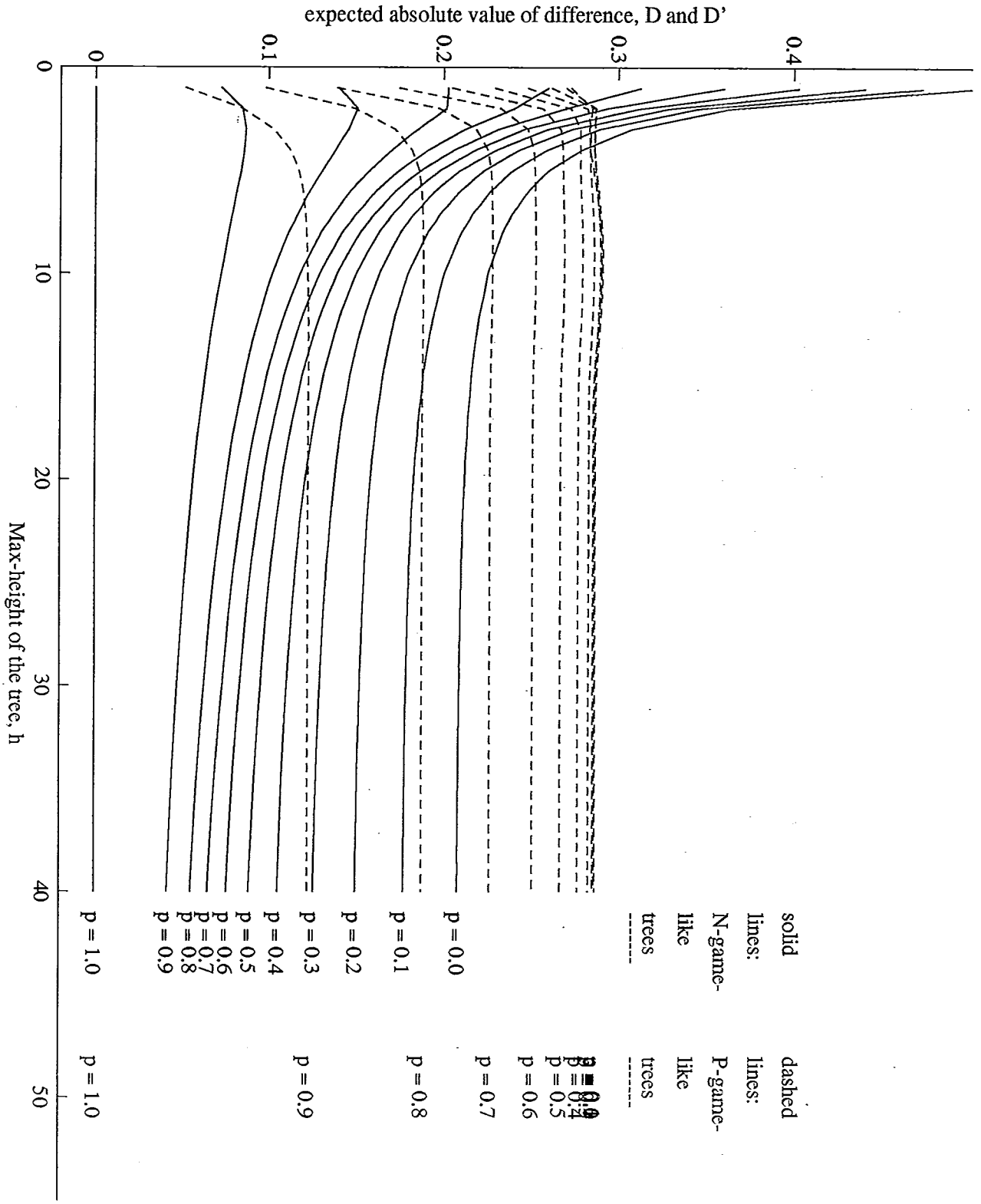

Figure 3: D_h and $D'_{h,m}$ versus h for various values of p, using $m = 40$.

For an N-game-like tree whose root is a Max node, this is $D_h = \sum_x \sum_y (|x - y| \cdot e_{h,x,y})$.

For a P-game-like tree whose root is a Max node, it is $D'_{h,m} = \sum_x \sum_y (|x - y| \cdot e'_{h,m,x,y})$.

Results and Interpretations

To derive closed-form solutions for the recurrences described in the "Mathematical Derivations" section would be very complicated, so we do not attempt it here. However, since we do have exact statements of the base cases and recurrences, we can compute any desired value of $e_{h,x,y}$ or $e'_{h,m,x,y}$, and thus any desired value of D_h or $D'_{h,m}$.

We have computed D_h for trees of height $h = 1, 2, \ldots, 40$; and we have computed $D'_{h,m}$ for trees of height $h = 1, 2, \ldots, 40$, with m set at 40. The results are shown in Fig. 3. Our interpretation of these results is as follows:

1. The higher the value of p is, the more likely it is that the forward-pruning algorithm will choose the correct node at each level of the tree, and thus the more likely it is that the algorithm will return a good approximation of the tree's minimax value. As shown in Fig. 3, this behavior occurs in both P-game-like trees and N-game-like trees.
2. In N-game-like trees, there is much stronger correlation among the values of sibling nodes than there is in P-game-like trees. Therefore, in N-game-like trees, even if the forward-pruning algorithm chooses the wrong node, the minimax value of this node is not too far from the minimax value we would compute anyway. Thus, as shown in Fig. 3, for each value of p, the forward-pruning algorithm returns more accurate values in N-game-like trees than in P-game-like trees.
3. In N-game-like trees, the returned values generally get more accurate as the game tree's height increases—but in P-game-like trees, the returned values generally get less accurate as the game tree's height increases. We believe this behavior is related to the fact that P-games are pathological and N-games are not (Nau, 1982; Nau, 1983).

Conclusion

In this paper, we have set up a model of forward pruning on two classes of game trees: binary N-game-like game trees, and binary P-game-like game trees. Based on this, we have computed the expected error in the minimax values that would result from forward pruning in these game trees.

As discussed in the "Results and Interpretations" section, our results suggest that forward pruning does better when there is a high correlation among the minimax values of sibling nodes in a game tree. Thus, forward pruning may possibly be a viable decision-making technique on game trees having the following characteristics:

first characteristic: there is generally a high correlation among sibling nodes;

second characteristic: when there are exceptions to the first characteristic, one can accurately identify them.

Two straightforward ways to extend our research results are by studying what happens on game trees of branching factor larger than two, and by examining how the expected error in the minimax values affects the probability of making a correct decision. We intend to finish these extensions in the very near future. As longer-term goals, we hope to examine how well forward-pruning performs when playing entire games against an ordinary minimax strategy, and to examine game trees other than N-game-like and P-game-like trees.

In addition to the above work, we intend to do an empirical study of forward pruning on the game of bridge. We are interested in bridge for the following reasons:

- Bridge is an imperfect-information game, because no player knows exactly what moves the other players are capable of making. Because of this, the game tree for bridge has a large branching factor, resulting in a game tree containing approximately 6.01×10^{44} nodes in the worst case. Ordinary minimax search techniques do not do well in bridge, because they have no chance of searching any significant portion of the game tree.
- Our preliminary studies on the game of bridge show that by using forward-pruning techniques based on task-network planning, we can produce search trees of only about 1300 nodes in the worst case (Smith *et al.*, 1992). Thus, forward pruning will allow us to search all the way to the end of the game. Thus, we will not need to use a static evaluation function, and thus will not have to deal with the inaccuracies produced by such functions.
- We believe that bridge has the two characteristics described above, primarily because of the trick-scoring method used in bridge (this is discussed in more detail in the "Forward Pruning Models" section). Thus, we believe that forward pruning techniques may produce reasonably accurate results in bridge.

References

Berliner, H. J.; Goetsch, G.; Campbell, M. S.; and Ebeling, C. 1990. Measuring the performance potential of chess programs. *Artificial Intelligence* 43:7–20.

Berliner, H. J. 1979. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence* 12:23–40.

Biermann, A. W. 1978. Theoretical issues related to computer game playing programs. *Personal Computing* 86–88.

Knuth, D. E. and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artificial Intelligence* 6:293–326.

Lee, K.-F. and Mahajan, S. 1990. The development of a world class othello program. *Artificial Intelligence* 43:21–36.

Levy, D. and Newborn, M. 1982. *All About Chess and Computers.* Computer Science Press.

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

Nau, D. S. 1983. Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence* 21(1, 2):221–244.

Pearl, J. 1984. *Heuristics.* Addison-Wesley, Reading, MA.

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior.* American Elsevier Publishing Company.

Samuel, A. L. 1967. Some studies in machine learning using the game of checkers. ii–recent progress. *IBM Journal of Research and Development* 2:601–617.

Shannon, C. 1950. Programming a computer for playing chess. *Philosophical Magazine* 7(14):256–275.

Smith, S. J. J. and Nau, D. S. 1993. Strategic planning for imperfect-information games. In *AAAI Fall Symposium on Games: Planning and Learning.*

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1992. A hierarchical approach to strategic planning with non-cooperating agents under conditions of uncertainty. In *Proc. First Internat. Conf. AI Planning Systems.* 299–300.

Stefik, M. 1981. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence* 16:111–140.

Stockman, G. C. 1979. A minimax algorithm better than alpha-beta? *Artificial Intelligence* 12:179–196.

Tate, A. 1976. Project planning using a hierarchic non-linear planner. Technical Report 25, Department of Artificial Intelligence, University of Edinburgh.

Tate, A. 1977. Generating project networks. In *Proc. 5th International Joint Conf. Artificial Intelligence.*

Truscott, T. R. 1981. Techniques used in minimax game-playing programs. Master's thesis, Duke University, Durham, NC.