An Analysis of Forward Pruning *

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

Dana S. Nau
Department of Computer Science, and
Institute for Systems Research
University of Maryland
College Park, MD 20742
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 forward pruning, we have set up models of forward pruning on two different kinds of game trees, and used these models to investigate how forward pruning affects the probability of choosing the correct move. In our studies, forward pruning did better than minimaxing when there was 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 [4, 24]. 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 game trees, and we use this model to investigate how forward pruning affects the probability of choosing the correct move. 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*:

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

Due to the size of the game tree, computing a node's true minimax value is impractical for most games. For this 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 [16] 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

*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 NSFD CDR-88003012 and IRI-9306580.

[1] In particular, we are developing a forward-pruning search technique for the game of bridge [17, 18], by extending task-network planning techniques [22, 23, 13, 20] to represent multi-agency and uncertainty.

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 further decrease the number of nodes examined, game-tree-search procedures have been developed such as alpha-beta [5], B* [2], or SSS* [21]. These procedures will ignore any node v below u that they can prove will not affect u's minimax value $mm(u)$.

This approach has worked well in games such as chess [3, 7], checkers [15, 14], and othello [6]. 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, it is no longer widely used, because chess programs that used it did less well than those that did not [4, 24].

Our Model of Forward Pruning

In the game trees investigated in this paper, the value of each leaf node is either 1, representing a win for Max, or 0, representing a win for Min. Our model of a forward-pruning algorithm works as follows. At each node u where it is Max's move, u has three children, u_1, u_2, and u_3. The forward-pruning algorithm will choose exactly two of these three nodes to investigate further. Normally, it will make this choice by applying a static evaluation function *eval*(.) to the three nodes, and discarding the node having the lowest value—and this is what we do in the "Statistical Studies" section. For our mathematical derivations, we assume fixed probabilities for which nodes will be chosen and which node will be discarded, as described below. There are three possible cases:

1. Two of the nodes, say u_1 and u_2, have minimax values representing wins for the current player. One of the nodes, say u_3, has a minimax value representing a loss for the current player. Then the *correct* two children to investigate further are the ones whose minimax values are the same as the minimax value of u, in this case u_1 and u_2. Thus, for a Max node, the correct children have value 1; for a Min node, the correct children have value 0. If the algorithm does not choose both of the correct children, then the algorithm will search only one of u_1 and u_2. Thus, it will continue part of its search down an incorrect branch, in this case the branch leading to u_3. This may result in an error in the algorithm's computation of u's minimax value.

   In our mathematical derivations, we assume that the probability of choosing the correct two children is p, where p is constant throughout the tree. The algorithm's probability of choosing one correct child and the incorrect child is thus $(1-p)/2$ for each correct child.

2. One of the nodes, say u_1, has a minimax value representing a win for the current player. Two of the nodes, say u_2 and u_3, have minimax values representing losses for the current player. In this case, the correct child is u_1. If the algorithm chooses the two incorrect nodes, it will continue the rest of its search down incorrect branches, those leading to u_2 and u_3. This is likely to result in an error in the algorithm's computation of u's minimax value.

   In our mathematical derivations, we assume that the probability of choosing the two incorrect nodes is r, where r is constant throughout the tree. The algorithm's probability of choosing the correct child and one incorrect child is thus $(1-r)/2$ for each incorrect child. (For the rest of this paper, we will set $r = (1-p)^2$ and define $q = 1-(p+r)$.)

3. All of the nodes have the same minimax value. In this case, all children are equally correct; the algorithm's probability of choosing any given pair of branches is 1/3.

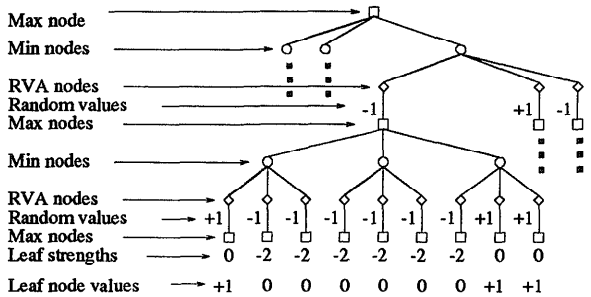

Figure 1: Example of an N-game-like tree.

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 trees and N-Game-Like Trees

An *N-game-like tree* is a complete tree that contains the following types of nodes (for example, see Fig. 1):

1. *Max* nodes, where it is Max's move. Each Max node is either a leaf node or has three children, all of which are Min nodes.
2. *Min* nodes, where it is Min's move. Each Min node has three children, all of which are RVA nodes.
3. *RVA* (random-value addition) nodes, which have numeric values assigned to them at random. The numeric value of each RVA node is chosen independently from the set $\{-1, 1\}$ with probability p_N being the probability of choosing 1. (For the rest of this paper, we will set $p_N = 0.61803$, the golden ratio, so that in the limit, there is still a nonzero probability of each player having a forced win in the game tree.) Each RVA node has a single child, which is a Max node.

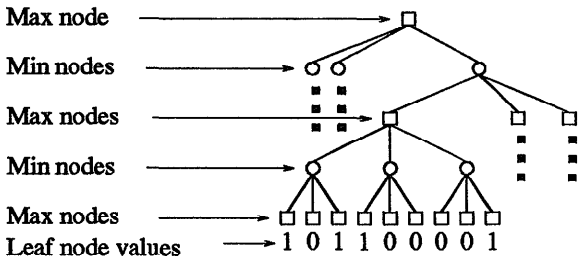

Figure 2: Example of a P-game tree.

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.[2] The *strength* of each leaf node u is the sum of the values of the RVA nodes on the path from the root to u. If the strength of a leaf node is nonnegative, it is classified as a win; otherwise, it is classified as a loss.

An N-game tree, as defined in [9, 10], is similar to the N-game-like trees defined above, except that N-game trees have no RVA nodes. Instead, a value of 1 or -1 is randomly assigned to each arc, with a probability of 0.5 of choosing 1. In this paper, we study N-game-like trees in the "Mathematical Derivations" section, and N-game trees in the "Statistical Studies" section.

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 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 strength of the leaves of T cannot differ from one another by any more than $2n$.

P-Game Trees

A *P-game tree* [9, 10, 12] 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 three children, all of which are Min nodes.
2. *Min* nodes, where it is Min's move. Each Min node has exactly three 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 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 the set $\{0, 1\}$, with probability p_P of choosing 1. (For the rest of this paper, we will set $p_P = 0.68233$, in order to guarantee that in the limit, there is still a nonzero probability that each player will have a forced win in the game tree [1, 11, 9].) 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 want to compute the probability that the forward-pruning algorithm estimates a value of s and the actual value is t for an N-game-like tree T whose Max-height is h. That is, we want Pr[estimated value s, actual value t | T's Max-height is h]. We can compute this from the node strengths, as follows. Let

$e_{h,x,y}$ = Pr[estimated strength x, actual strength y | Max-height h, root is a Max node];
$f_{h,x,y}$ = Pr[estimated strength x, actual strength y | Max-height h, root is an RVA node];
$g_{h,x,y}$ = Pr[estimated strength x, actual strength y | Max-height h, root is a Min node].

These probabilities depend on p and p_N (recall that $p_N = 0.61803$). The base case is $e_{0,x,y} = 1$ if $x = y = 0$, and $e_{0,x,y} = 0$ otherwise. The recurrence for $f_{h,x,y}$ is

$$f_{h,x,y} = p_N e_{h,x-1,y-1} + (1 - p_N) e_{h,x+1,y+1}.$$

The recurrences for $g_{h,x,y}$ and $e_{h+1,x,y}$ are too complicated to include here; see [19]. Now, let

$\bar{e}_{h,s,t}$ = Pr[estimated value s, actual value t | Max-height h, root is a Max node];
$\bar{f}_{h,s,t}$ = Pr[estimated value s, actual value t | Max-height h, root is an RVA node];
$\bar{g}_{h,s,t}$ = Pr[estimated value s, actual value t | Max-height h, root is a Min node].

Then

$$\bar{e}_{h,1,1} = \sum_{x:x\geq 0} \sum_{y:y\geq 0} e_{h,x,y};$$

$$\bar{e}_{h,1,0} = \sum_{x:x\geq 0} \sum_{y:y<0} e_{h,x,y};$$

$$\bar{e}_{h,0,1} = \sum_{x:x<0} \sum_{y:y\geq 0} e_{h,x,y};$$

$$\bar{e}_{h,0,0} = \sum_{x:x<0} \sum_{y:y<0} e_{h,x,y}.$$

For $\bar{f}$ and $\bar{g}$, the equations are similar.

[2] 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.

Forward Pruning on P-Game Trees

Since there are no strengths in P-game trees, we can compute the probabilities for the values directly. We define

$$e'_{h,x,y} = \Pr[\text{estimated value } x, \text{ actual value } y \mid \text{Max-height } h, \text{ root is a Max node}];$$
$$g'_{h,x,y} = \Pr[\text{estimated value } x, \text{ actual value } y \mid \text{Max-height } h, \text{ root is a Min node}].$$

The base case is $e_{0,x,y} = p_P$ if $x = 1$ and $y = 1$; $(1-p_P)$ if $x = 0$ and $y = 0$; and 0 otherwise. As shown in [19], the recurrence for $g'_{h,x,y}$ is identical to that for $f_{h,x,y}$, except that each occurrence of $e_{h,m,n}$ is replaced by $e'_{h,m,n}$. The recurrence for $e'_{h+1,x,y}$ is identical to that for $e_{h+1,x,y}$, except that each occurrence of $f_{h,m,n}$ is replaced by $f'_{h,m,n}$.

Probability of Correct Decision

We can use the above recurrences to measure the *probability of correct decision*. This is the probability that the forward-pruning algorithm, given a choice between two alternatives that have different minimax values, will choose the correct one.[3] In particular, consider an N-game-like tree T of Max-height h, whose root is a Max node u with children u_1 and u_2 such that the value of u_1 is greater than the value of u_2. Then

$$\begin{aligned} D_h &= \Pr[\text{estimated value of } u_1 > \text{estimated value of } u_2] + \frac{1}{2}\Pr[\text{estimated value of } u_1 = \text{estimated value of } u_2] \\ &= [\bar{g}_{h-1,1,1}\bar{g}_{h-1,0,0} + \bar{g}_{h-1,0,1}\bar{g}_{h-1,0,0}/2 + \bar{g}_{h-1,1,1}\bar{g}_{h-1,0,1}/2] \\ &\quad \overline{[(\bar{g}_{h-1,1,1} + \bar{g}_{h-1,0,1}) \times (\bar{g}_{h-1,1,0} + \bar{g}_{h-1,0,0})]}. \end{aligned}$$

Similarly, for P-game trees,

$$\begin{aligned} D'_h &= [g_{h-1,1,1}g_{h-1,0,0} + g_{h-1,0,1}g_{h-1,0,0}/2 + g_{h-1,1,1}g_{h-1,0,1}/2] \\ &\quad \overline{[(g_{h-1,1,1} + g_{h-1,0,1}) \times (g_{h-1,1,0} + g_{h-1,0,0})]}. \end{aligned}$$

Results and Interpretations

To derive closed-form solutions for the recurrences described in the "Mathematical Derivations" section would be very complicated. 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. We have computed D_h and D'_h for trees of height $h = 1, 2, \ldots, 15$. The results are shown in Fig. 3, along with the probability of correct decision by random guess, included for comparison purposes. Our interpretation of these results is as follows:[4]

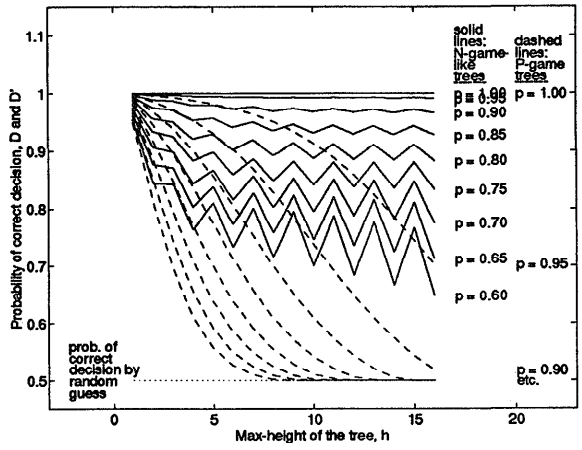

Figure 3: D_h and D'_h versus h for various values of p.

1. The higher the value of p, the more likely it is that the forward-pruning algorithm will choose the correct two nodes to investigate 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 occurs in both P-game 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 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 trees.

Statistical Studies

The results in the "Mathematical Derivations" section suggest that minimax with forward pruning does better when there is a high correlation among the minimax values of sibling nodes in a game tree. Previous studies [9, 10] have shown that ordinary minimaxing also does better when there is a high correlation among the minimax values of sibling nodes in a game tree. Thus, the next question is whether minimax with forward

[3] We have also investigated the probability of correct decision among three alternatives; the formulas [19] are too complicated to present here, but the results are similar.

[4] The probability of correct decision for N-game-like trees exhibits a "manic-depressive" behavior similar to that observed in [8], that is, it is higher for odd Max-heights than it is for even Max-heights. We believe this is because our RVA nodes are only put below Min nodes. Standard N-game trees have the equivalent of our RVA nodes below both Min and Max nodes.

pruning would do better than ordinary minimaxing—for otherwise, it wouldn't make sense to use forward pruning for actual game playing.

To answer this question, we computed the probabilities of correct decision at various search depths on P-game trees and N-game trees, for minimax with and without forward pruning. For this study, we wanted to use a real evaluation function rather than a mathematical model of one. This made it impossible to do an analysis similar to the one in the "Mathmetical Derivations" section, so instead we did a statistical study.

For $h = 2, \ldots, 6$, we generated 5000 ternary N-game trees and P-game trees of Max-height h. The trees were generated at random, except that if a tree's root did not have at least one forced-win child c_{win} and one forced-loss child c_{loss}, we discarded the tree and generated another. For each tree T, we did a depth d minimax search of T,[5] using the same evaluation function used in [9, 10]:

$$eval(u) = \frac{\text{winning leaf-descendants of } u}{\text{all leaf-descendants of } u}.$$

We did this for $d = 1, \ldots, 2h-2$.[6] To get a statistical approximation of the probability of correct decision, we averaged the following over all 5000 N-game trees or P-game trees:

$$\text{quantity averaged} = \begin{cases} 1 & \text{if } mm(c_{win}, d-1) > mm(c_{loss}, d-1), \\ 1/2 & \text{if } mm(c_{win}, d-1) = mm(c_{loss}, d-1), \\ 0 & \text{otherwise.} \end{cases}$$

We then repeated the same experiment, using minimax with forward pruning.

The results are shown in Figures 4 and 5, which graph the probability of correct decision for minimaxing both with and without forward pruning. To indicate how good a decision each approach could produce given the same amount of search time, these figures graph the probability of correct decision as a function of the number of nodes generated by the search.[7] As can be seen, minimaxing with forward pruning generally does better than ordinary minimaxing on N-games, and slightly worse than ordinary minimaxing on P-games.

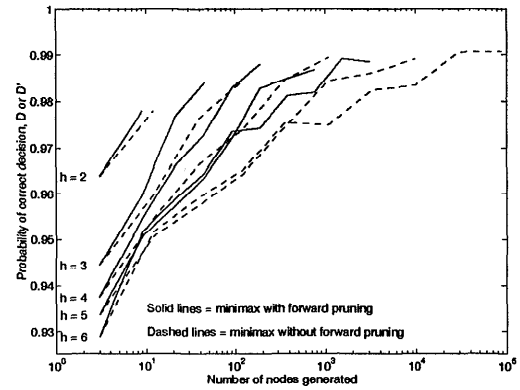

Figure 4: Probability of correct decision on N-games, versus number of nodes generated, for minimax with and without forward pruning. The data is averaged over 5000 game trees.

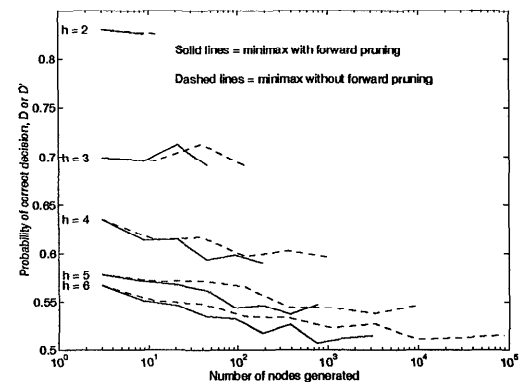

Figure 5: Probability of correct decision on P-games, versus number of nodes generated, for minimax with and without forward pruning. The data is averaged over 5000 game trees.

Conclusion

In this paper, we set up models of forward pruning on ternary N-game-like game trees, and ternary P-game trees. We used these models to compute the probabil-

[5] The depth-d minimax value of a node is

$$mm(u,d) = \begin{cases} eval(u) \text{ (the payoff at } u\text{)} \\ \quad \text{if } d = 0 \text{ or } u \text{ is a terminal node,} \\ \max\{mm(v, d-1) : v \text{ is a child of } u\} \\ \quad \text{if it is Max's move at } u, \\ \min\{mm(v, d-1) : v \text{ is a child of } u\} \\ \quad \text{if it is Min's move at } u. \end{cases}$$

where $eval(u)$ is u's evaluation function value. A depth d minimax search from a node u means computing the depth $d-1$ minimax values of u's children.

[6] We did not search to depths $2h-1$ and $2h$ because the comparison would not have been fair. At these depths, ordinary minimaxing applies $eval(u)$ only to nodes within one move of the end of the game. For such nodes, $eval(u)$ produces perfect results, hence so does ordinary minimaxing.

[7] For ternary game trees, the number of nodes generated by a ordinary minimax search is $3^1 + \ldots + 3^n = (3^{n+1} - 3)/2$. The number of nodes generated with forward pruning is $3(2^0 + \ldots + 2^{n-1}) = 3(2^n - 1)$. This is without alpha-beta pruning. With alpha-beta pruning, there would have been a different number of nodes generated in each game tree, making it difficult to obtain meaningful averages over our 5000 games.

ity of correct decision produced by minimax with and without forward pruning.

In our studies, minimax with forward pruning did better than ordinary minimaxing in cases where there was 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.

To extend our 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 [17]. 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. Thus, we believe that forward pruning techniques may produce reasonably accurate results in bridge.

References

[1] Baudet, G. M. 1978. On the branching factor of the alpha-beta pruning algorithm. *Artif. Intel.* 10:173–199.

[2] Berliner, H. J. 1979. The B* tree search algorithm: A best-first proof procedure. *Artif. Intel.* 12:23–40.

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

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

[5] Knuth, D. E. and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artif. Intel.* 6:293–326.

[6] Lee, K.-F. and Mahajan, S. 1990. The development of a world class othello program. *Artif. Intel.* 43:21–36.

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

[8] Nau, D. S. 1982. The last player theorem. *Artif. Intel.*, 18:53–65.

[9] Nau, D. S. 1982. An investigation of the causes of pathology in games. *Artif. Intel.* 19:257–278.

[10] Nau, D. S. 1983. Pathology on game trees revisited, and an alternative to minimaxing. *Artif. Intel.* 21(1, 2):221–244.

[11] Pearl, J. 1980. Asymptotic properties of minimax trees and game-searching procedures. *Artif. Intel.* 14:113–138.

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

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

[14] 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.

[15] Schaeffer, J.; Culberson, J.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1992. A world championship caliber checkers program. *Artif. Intel.* 53:273–290.

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

[17] 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.

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

[19] Smith, S. J. J.; Nau, D. S. Formal and statistical analysis of forward pruning. Forthcoming.

[20] Stefik, M. 1981. Planning with constraints (MOLGEN: Part 1). *Artif. Intel.* 16:111–140.

[21] Stockman, G. C. 1979. A minimax algorithm better than alpha-beta? *Artif. Intel.* 12:179–196.

[22] Tate, A. 1976. Project planning using a hierarchic non-linear planner. Technical Report 25, Department of Artif. Intel., University of Edinburgh.

[23] Tate, A. 1977. Generating project networks. In *Proc. 5th IJCAI.*

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