Preliminary Results Regarding Quality of Play Versus Depth of Search in Game Playing *

Dana S. Nau
Computer Science Department, Duke University
Durham, NC 27706

ABSTRACT

Tree search is important in a number of areas, yet many of its properties are not well understood. Tree search tends to be time consuming, because the amount of time necessary to search a tree generally increases exponentially with the depth of the tree. Thus, tree search is usually used only in cases where the tree is known to be small, or where no better approach is known.

One important application of tree search is in "looking ahead" to try to predict the results of a decision. This is done, for example, in the field of decision analyisis, and in the playing of games such as chess, checkers, go-moku, othello, etc. For such games, there is a consensus that looking ahead improves play, but this agreement is based purely on empirical evidence.

The author has developed a mathematical theory modeling the effects of tree search on quality of play for games such as those mentioned above. This research yields the surprising result that <u>there is a large set of such games for which looking farther ahead consistently decreases the probability of correct play rather than increasing it</u>.

Research aimed at demonstrating the predictions of this theory on commonly played games is currently under way. In addition, preliminary studies indicate that similar types of results should hold for other kinds of decision making situations in which looking ahead is used.

* This work, which was supported in part by National Science Foundation and James B. Duke graduate fellowships, is discussed in detail in the author's Ph.D. dissertation [7].

INTRODUCTION

Tree searching is a technique which is quite important in a number of areas of computer science. Searching a tree of any size is usually a time-consuming task, because the amount of time necessary to search the branches of the tree generally increases exponentially with the depth of the tree. For this reason, tree search is generally used only in cases where the tree is known to be very small, or where no better approach is known. For example, one of the prime applications of tree search techniques is to solve instances of NP-hard problems [1, 3].

One type of tree which is important in a number of areas is the AND-OR tree. Its most well known areas of use are problem reduction search strategies [6], computational complexity [2, 8], and decision making [5, 6]. In decision-making situations where AND-OR trees are used, it is generally agreed that increasing the amount of look-ahead improves the quality of the decision. For example, there have been some rather dramatic demonstrations of this with game playing computer programs [9]. However, such results are purely empirical. The author knows of no previous theoretical work on the effects of search depth on the quality of a decision.

This paper is concerned with a mathematical theory modeling the effects of searching deeper on the quality of a decision. Since the methodology for such search has been best developed for the playing of games such as chess, checkers, go-moku, othello, three-dimensional tic-tac-toe, and others, the model is couched in terms of game playing. However, as discussed near the end of the paper, the results should extend to other areas as well. The major result of this study is that <u>there is a large class of games of the type mentioned above for which looking farther ahead will consistently decrease the probability of correct play rather than increase it</u>.

This idea may seem counter-intuitive. One might suppose that since searching deeper gives more information, it will always improve play. However, there are some games for which searching deeper causes the real information to be lost in the midst of misleading information and noise.

In this paper, the word "game" shall mean only games having the same general characteristics as the ones mentioned above. These games form a subset of the set of two person, perfect information games. Each of them is played by two players, with a strict alternation of the play between them, and each player has complete knowledge of what both players have done and can do in the game. In none of the games is the outcome determined even partially by chance (as occurs in dice games and most card games). However, the results stated in this paper could be generalized to remove one or more of these restrictions.

For the kind of game under consideration, a common way for a player to choose a move is to look ahead in an effort to gauge the consequences of each possible move. An analogous technique is used in computer programs which play such games. This technique shall be informally described here; for a more detailed description, the reader is referred to [6].

The set of all possible courses a game might take can be represented by a <u>game tree</u> in which the root node is the current position in the game, and the children of each node are the positions which can be reached in one move from that node. Looking ahead is done by searching the branches of the tree to some arbitrary depth, and applying a heuristic evaluation function to each node at this depth. This function returns a numerical value for each node, based on how good it estimates the game position to be. A high value indicates a position favorable to one player, and a low value indicates a position favorable to the other. The values returned by the evaluation function are propagated back to the children of the root using a technique called <u>minimaxing</u>. Readers should note that this term is used here in a considerably different sense than in the "minimax" theorem of game theory. Those unfamiliar with this usage are referred to [6].

A MATHEMATICAL MODEL

In this section, a mathematical model is developed to investigate when deeper search improves and degrades play. All games considered shall be games without draws, and in which the play strictly alternates between one player and the other. Unless otherwise stated, the player who has the first move shall always be called Max, and his opponent shall be called Min. Removal of these restrictions would not present any particular difficulty with respect to the results discussed in this paper.

Let G be a game, and suppose that at every move after some position P in G, both Max and Min play perfectly. Since draws are not allowed, someone (say, Max) must win. Then we say that Max has a <u>forced win</u> at

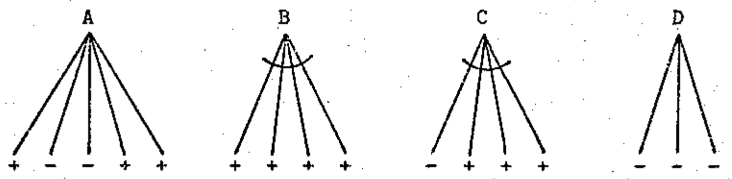

FIGURE 1.--Examples of A, B, C, and D nodes. The arcs on the B and C nodes indicate Min's move.

position P, for if Max plays perfectly from that point on, then Max will win regardless of how well Min plays. Obviously, every position in every game we shall consider will be a forced win for someone. Forced wins for Max and Min shall be labeled "+" and "-", respectively. Each game tree node now falls into one of four equivalence classes (see Figure 1):

A. It is Max's move, and at least one of his possible moves leads to a "+" node. Then Max has a forced win.

B. It is Min's move, and all of his possible moves lead to "+" nodes. Then Max again has a forced win.

C. It is Min's move, and at least one of his possible moves leads to a "-" node. Then Min has a forced win. This kind of node is to Min what an A node is to Max.

D. It is Max's move, and all of his possible moves lead to "-" nodes. Then Min has a forced win. This kind of node is to Max what an A node is to Min.

Given strict alternation between moves by Max and moves by Min, it follows that

1. the successors of an A node are always B and C nodes,

2. the successors of a B node are always A nodes,

3. the successors of a C node are always A and D nodes, and

4. the successors of a D node are always C nodes.

A and C nodes shall be called <u>critical</u> nodes, for it is only at such nodes that it makes a difference which move is chosen.

Let f be an evaluation function (as described in the introduction). If f is implemented on a real computer, then f returns one of a finite (although perhaps very large) number of values. For convenience, we shall consider the range of f to be the set {0, 1, ..., n}, for some nonnegative integer n.

If f is a good evaluation function, then f(t) will generally be high if t is a "+" node and low if t is a "-" node. However, f will sometimes make mistakes; i.e., there will be some "-" nodes to which f gives high evaluations and some "+" nodes to which f gives low evaluations. Otherwise, tree search would not be necessary, for choosing as a next move the one of highest value would yield perfect play.

For almost any universal property one might want to state about the behavior of evaluation functions in terms of increasing search depth, a function can be constructed which fails to have that property, simply by increasing or decreasing the accuracy of the function on nodes near the end of the tree. A convenient way to avoid this difficulty is to set up a model which makes probabilistic rather than universal predictions about the behavior of evaluation functions.

When setting up a probabilistic model of a class of deterministic objects, the usual procedure is to abstract the "important" features of the objects being modeled, and to consider each object X as a random element from the set of all objects having the same features as X. In the case of evaluation functions, the main feature which we shall choose to consider important is their average performance.

One way to do this is to compute <u>frequency counts</u> for the values returned by an evaluation function. For an evaluation function f, we can let q(i) be the number of nodes t such that f(t) = i, and then consider f to be a randomly chosen element of the set of all evaluation functions yielding the same function q. The problem with this idea is that both good and bad evaluation functions will be members of the same equivalence class. A function that gives "+" nodes high values and "-" nodes low values will generally yield the same q as many other functions that give "+" nodes low values and "-" nodes high values. A measure of the average performance of f should in some sense be a measure of the goodness of f: each good evaluation function should be in an equivalence class containing other good evaluation functions, and each bad evaluation function should be in an equivalence class containing other bad evaluation functions.

If f is a good evaluation function, then f(t) will usually be high if t is a "+" node and low if t is a "-" node. Let f' be defined by

$$f'(t) = f(t) \text{ if } t \text{ is a "+" node}$$
$$= n-f(t) \text{ if } t \text{ is a "-" node.}$$

If f is good, then f'(t) will usually be high, regardless of whether t is a "+" node or a "-" node. conversely, if f is bad, f'(t) will not usually be high. Therefore, let us put p(i) = the number of nodes t such that f'(t) = i. If f is good, then p(i) will usually be small if i is small and large if i is large, and vice versa if f is bad. Thus p is a reasonable measure of goodness for f.

Let us now put f into an equivalence class with all other functions g yielding the same function p. If we now consider f to be drawn at random from this class, then p determines the probability density function (p.d.f.) for the values returned by f. In particular, it can be shown that

$$p(i) = \Pr[f(t) = i \mid t \text{ is a "+" node}]$$
$$= \Pr[f(t) = n-i \mid t \text{ is a "-" node}].$$

Since p(i) is defined only for i = 0,1,...,n, p can be represented by a vector P = (p(0), p(1), ..., p(n)). This vector we shall call the <u>probability vector</u> for f.

The only nodes for which it matters which move is chosen are the critical nodes; i.e., the A and C nodes. If P is indeed a useful measure of goodness for f, then from P it should be possible to calculate the probability of making a correct decision at any critical node in the tree for a given search depth. Indeed this can be done. Although a discussion of how it is done is beyond the scope of this paper,* we note that if p(0) = p(1) = ... = p([n/2]) = 0), then f will always evaluate a "+" node higher than it will evaluate a "-" node, and thus the probability of making a correct decision will

* Both this mathematical development and the proofs of the theorems presented later are omitted. This is for brevity--were they included, the paper would be several times its current length! Readers interested in seeing a mathematically rigorous presentation are referred to the author's Ph.D. dissertation [7].

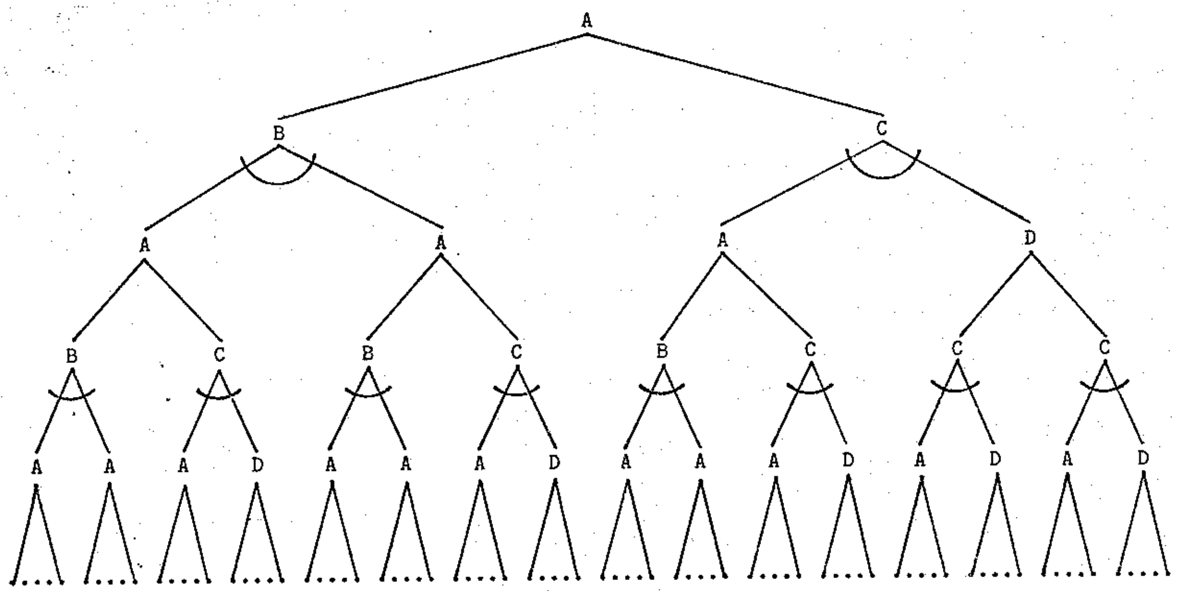

FIGURE 2.--The tree T(1,1).

be 1, regardless of the search depth. In this case, we say that both f and P are perfect.

A PATHOLOGICAL TREE

A perfect probability vector P will yield perfect play regardless of the search depth. However, the set of perfect probability vectors has measure (and hence probability) 0. As discussed in the introduction, empirical experience suggests that for imperfect P, deeper search will increase the probability of correct decision. We shall now construct a game tree for which deeper search will decrease the probability of correct decision.

One way to try to do this is to try to make every "+" node look bad and every "-" node look good. Intuitively, this should occur if each node has many children as possible of opposite sign. If, for simplicity, the branching factor is restricted to 2, this yields the tree T(1,1) of Figure 2.

THEOREM 1. Suppose P is a probability vector which is not perfect. Then for T(1,1),

$$\lim_{d \to \infty} \Pr[\text{correct decision} \mid \text{depth } d \text{ search}] = 1/2.$$

One-half is what the correct choice would be on T(1,1) if one were choosing moves purely at random. Thus, given any imperfect evaluation function for T(1,1), no matter how bad or good, the farther one looks ahead, the more one will play as if one were choosing moves at random.

For example, one evaluation function for T(1,1) gave the following performance. The probability of correct choice at a search depth of 1 was 0.99. At search depth 2, it was 0.985. At search depth 8, it was less than 0.9. At search depth 16, it was less than 0.6. At search depth 24, it was 0.500, correct to 3 decimal places.

Intuitively, what happens on T(1,1) is a "manic-depressive" kind of behavior. If a player looks ahead an odd number of moves, each of his choices tends to look like a forced win. If he looks ahead an even number of moves, each of his choices tends to look like a forced loss. This tendency has been

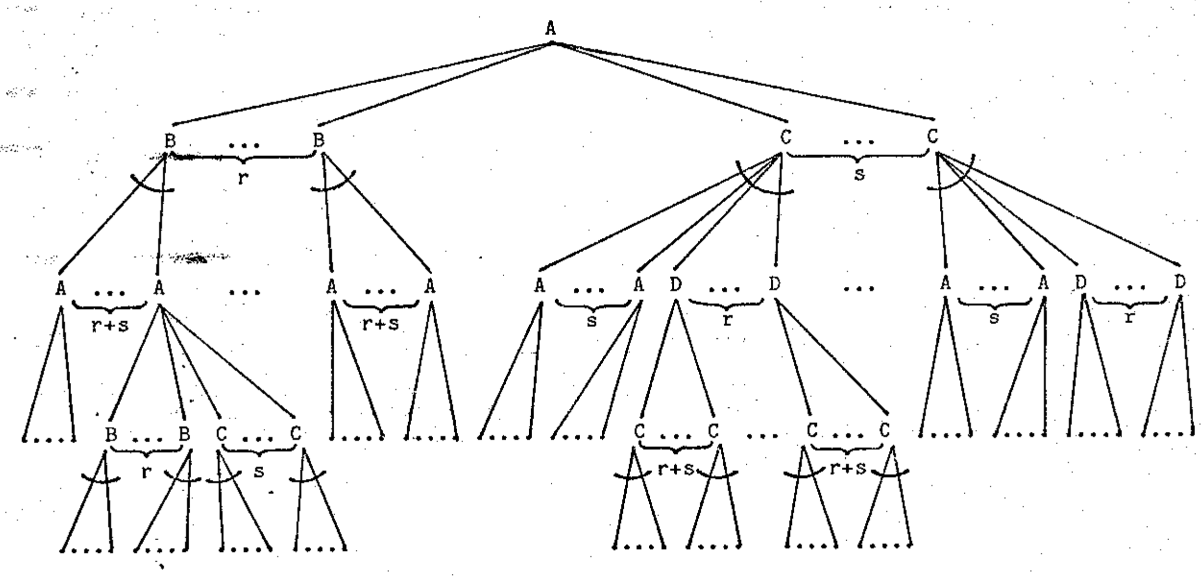

FIGURE 3.--The tree T(r,s).

noted by writers of computer programs to play games such as chess and checkers [9]; the difference for T(1,1) is that the tendency becomes worse as the search depth increases. As the search depth increases, more and more of the choices receive the same minimax values, and so the choice of what move to make becomes more and more random.

MORE GENERAL RESULTS

Suppose a player chooses a move by searching a game tree T to depth d, using an evaluation function f having probability vector P. If, as d approaches infinity, his probability of correct choice approaches what it would be if he were choosing a move at random, then we say that T is P-pathological.

Theorem 1 says that T(1,1) is P-pathological for every imperfect P. It is reasonable to suspect that similar results hold for other trees as well. We would like to have a theorem saying "given an arbitrary game tree, it is pathological if and only if ...".

Unfortunately, such a theorem would be very difficult to prove. The proof of Theorem 1 involves proving that certain recursively defined sequences of vectors converge to desired limits. The sequences can be defined recursively only because T(1,1) is "homogeneous" in the sense that all A nodes look alike, all B nodes look alike, and so forth. In an arbitrary tree, the recursive definition could not be set up. Thus we shall not generalize Theorem 1 to all trees, but instead to a smaller set of trees which we hope will be broad enough to suggest how the spirit of these results might carry over to game trees in general.

Let r and s be positive integers. We shall generalize T(1,1) by constructing a tree T(r,s) of constant branching factor r+s, in which every critical node has r children of the same sign and s children of opposite sign. In particular, every A node shall have r B children and s C children, every B node shall have r+s A children, every C node shall have r D children and s A children, and every D node shall have r+s C children. As with T(1,1), the root of T(r,s) shall be an A node. Using these rules gives the tree of Figure 3.

THEOREM 2. Suppose that P is not semiperfect (this word shall be discussed

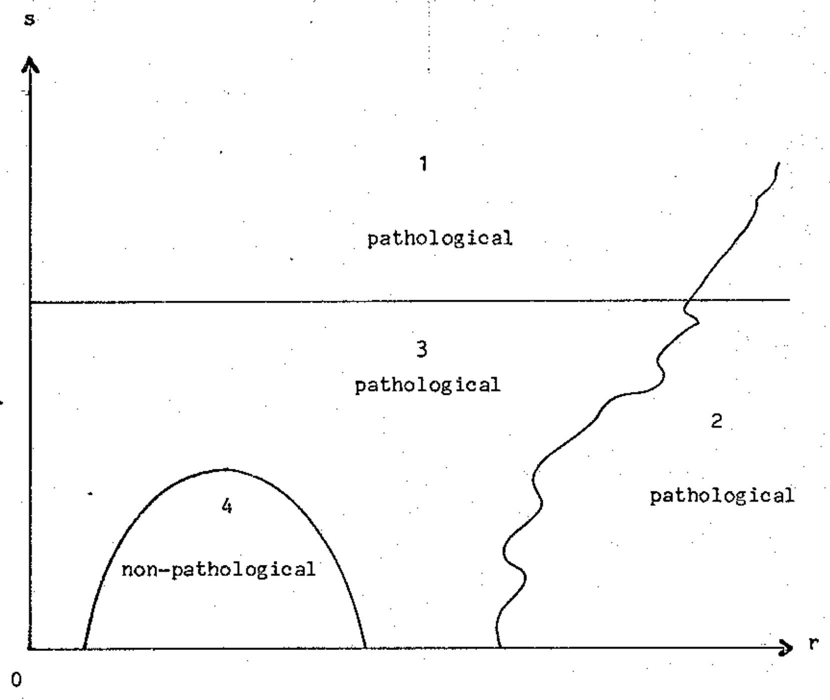

FIGURE 4.--Pathological and non-pathological behavior for a non-semiperfect probability vector P, as a function of r and s. The numbered areas are for reference in the text.

later). Then there is an integer S such that whenever $s > S$, T(r,s) is P-pathological for every r.

THEOREM 3. Suppose that P is not semiperfect. Then for every integer s there is an integer R such that for every $r > R$, T(r,s) is P-pathological.

A rigorous definition of "semiperfect" is beyond the scope of this paper. Intuitively, it denotes a class of "very good" evaluation functions. What is important about this concept is that considered as a subset of the set of all probability vectors P = (p(0), p(1), ..., p(n)), the set of probability vectors which are semiperfect is an n-1 dimensional subset of an n dimensional set, which hence has measure 0. Thus the probability of randomly choosing a probability vector to which Theorems 2 and 3 do not apply is 0.

Taken together, Theorems 2 and 3 state that for each non-semiperfect P there are only finitely many non-pathological trees T(r,s). However, the bounds they predict are not tight. Experimental computer simulation shows that pathology occurs under other circumstances as well. This is illustrated in Figure 4.

Areas 1 and 2 of Figure 4 are the areas of pathology predicted by Theorems 2 and 3, respectively. Area 3 is an additional region where experimentation has shown that pathology occurs. Trees with non-pathological behavior are found only in area 4.

DISCUSSION

The significance of the results discussed above is that for most classes of evaluation functions, all but a finite number of the T(r,s) will be pathological. It is the author's belief that this indicates an underlying pathological tendency present in almost all game trees. However, in most games this tendency is overridden by other factors. In games such as chess, for example, there are "good" positions and "bad" ones. Good positions are often characterized in terms of such things as the number of pieces, their mobility, how well the king is protected, etc., but a good position is generally one from which a player has a variety of good moves, and a bad one is generally one from which a player has few (if any) good moves. This clustering of good positions and bad positions should almost always override any pathological tendency which may be present.

Despite such considerations, it might be possible to get pathological behavior for limited periods of time even in games such as chess or checkers, given the appropriate conditions. In a part of the game where most moves look fairly similar and the players are fairly equally matched, the game tree might look "homogeneous" enough that pathology could be observed for a few moves.

Experimental results indicate that given a game with a small branching factor, pathological behavior is more likely to occur if the evaluation function has a small "vocabulary"; i.e., if its range is small. This suggests the possibility of observing pathological behavior in limited portions of a commonly played game by compressing the output of the evaluation function into a small set (for example, by mapping negative values into -1 and positive values into +1). Research is currently underway to see if pathology can be observed in this manner in such games.

Although the results discussed in this paper are couched in terms of game playing, there is no particular reason why they need be limited only to that area. Indeed, any sort of decision making situation which involves tree search using an evaluation function which sometimes makes mistakes is a logical area in which to try to develop pathology theorems. For example, preliminary investigations indicate that theorems similar to those of this paper should be obtainable for decision trees in decision analysis, certain decision problems arising in program synthesis, and possibly in other areas. This provides a topic for future research.

REFERENCES

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms. Reading, Mass.: Addison-Wesley, 1975.

2. A.K. Chandra and L.J. Stockmeyer, "Alternation," 17th Annual IEEE Symposium on Foundations of Computer Science, pp. 98-108, 1976.

3. R.M. Karp, "The probabilistic analysis of some combinatorial search algorithms," in Algorithms and Complexity, J.F. Traub, Ed. New York: Academic Press, 1976, pp. 1-19.

4. D.E. Knuth and R.W. Moore, "An analysis of alpha-beta pruning," Artificial Intelligence, vol. 6, pp. 293-326, 1975.

5. I.H. LaValle, Fundamentals of Decision Analysis. New York: Holt, Rinehart, and Winston, 1978.

6. N.J. Nilsson, Problem Solving Methods in Artificial Intelligence, New York: McGraw-Hill, 1971.

7. D.S. Nau, Quality of Play versus Depth of Search in Game Playing, Ph.D dissertation, Duke University, 1979.

8. L.J. Stockmeyer, "The polynomial-time hierarchy," Theoretical Computer Science, vol. 3, pp. 1-22, 1977.

9. T.R. Truscott, personal communication, 1978.