ANALYSIS OF THE ALPHA-BETA PRUNING ALGORITHM

S. H. Fuller, J. G. Gaschnig and J. J. Gillogly

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

July, 1973

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-73-C-0074) and is monitored by the Air Force Office of Scientific Research.

ABSTRACT

Many game-playing programs must search very large game trees. Use of the alpha-beta pruning algorithm instead of the simple minimax search reduces by a large factor the number of bottom positions which must be examined in the search. An analytical expression for the expected number of bottom positions examined in a game tree using alpha-beta pruning is derived, subject to the assumptions that the branching factor N and the depth D of the tree are arbitrary but fixed, and the bottom positions are a random permutation of N^D unique values. A simple approximation to the growth rate of the expected number of bottom positions examined is suggested, based on a Monte Carlo simulation for large values of N and D. The behavior of the model is compared with the behavior of the alpha-beta algorithm in a chess playing program and the effects of correlation and non-unique bottom position values in real game trees are examined.

TABLE OF CONTENTS

1. INTRODUCTION

Searching trees of possible alternatives is a task common to a wide range of programs. The efficiency with which these trees can be searched is of critical importance to such programs, since the trees are typically very large. This paper is concerned with measuring the efficiency of a particular tree-searching algorithm, the minimax search of a game tree with alpha-beta pruning.

The probabilistic model used in our study is presented in the next section and we derive an analytical expression for the expected number of bottom positions evaluated in the search of a game tree using alpha-beta pruning. A reasonably accurate simple approximation to the analytical result based upon an empirical analysis is suggested. Since our model incorporates several simplifying assumptions, the relevance of our model will be examined in Section 6 where we compare the behavior of our model with the observed behavior of the alpha-beta procedure as it is used in a non-trivial example, a chess playing program.

In this paper, the operation of the minimax search procedure and the alpha-beta pruning procedure are illustrated in the context of game playing programs. We give the name Max to the player whose turn it is to move and the name Min to his opponent. Max attempts to maximize the ultimate value of the game while Min attempts to minimize the value. A number of strategies exist to aid a player in determining his next move, but the minimax procedure has received the most attention in programs which play games of perfect information. The procedure is most easily illustrated

with the aid of the simple game tree of Figure 1.1. The nodes of the tree are interpreted as positions, and the arcs from each node are the legal moves from that position. The square nodes indicate it is Max's turn to move while the circles indicate it is Min's turn. The static values associated with each of the nine bottom positions are given independently of the application of any search procedure. Increasing values are interpreted as a measure of the "goodness" of a board position, i.e., the amount of advantage to player Max. In the minimax procedure the backed-up value of a Max position is the maximum of the values of its immediate successors and similarly, the backed-up value of a Min position is the minimum of the values of its immediate successors, i.e., at each node the player to move will choose the move which is most favorable to himself. The minimax procedure recursively applies these two rules until the static values at the leaf nodes have been used to generate a backed-up value for the root node. For example, in Figure 1.1 the backed-up values of p(1), p(2), and p(3) are 3, -2 and -10, respectively and the backed-up value of p, the root node, is 3. For a more complete discussion of the minimax procedure see Shannon [1950] or Nilsson [1970].

We will frequently use the game of chess in this paper to illustrate some of the practical implications and limitations of our analysis. The classic example of the limitation of the minimax procedure is its application to chess. Consider the game tree for chess where the position p is defined by the location and identity of each piece on the board, the identity of the player whose turn it is to move, and historical information relating to castling, en passant captures, and draws by repetition. Suppose we extend the chess game tree until every leaf node is a win, loss, or draw.

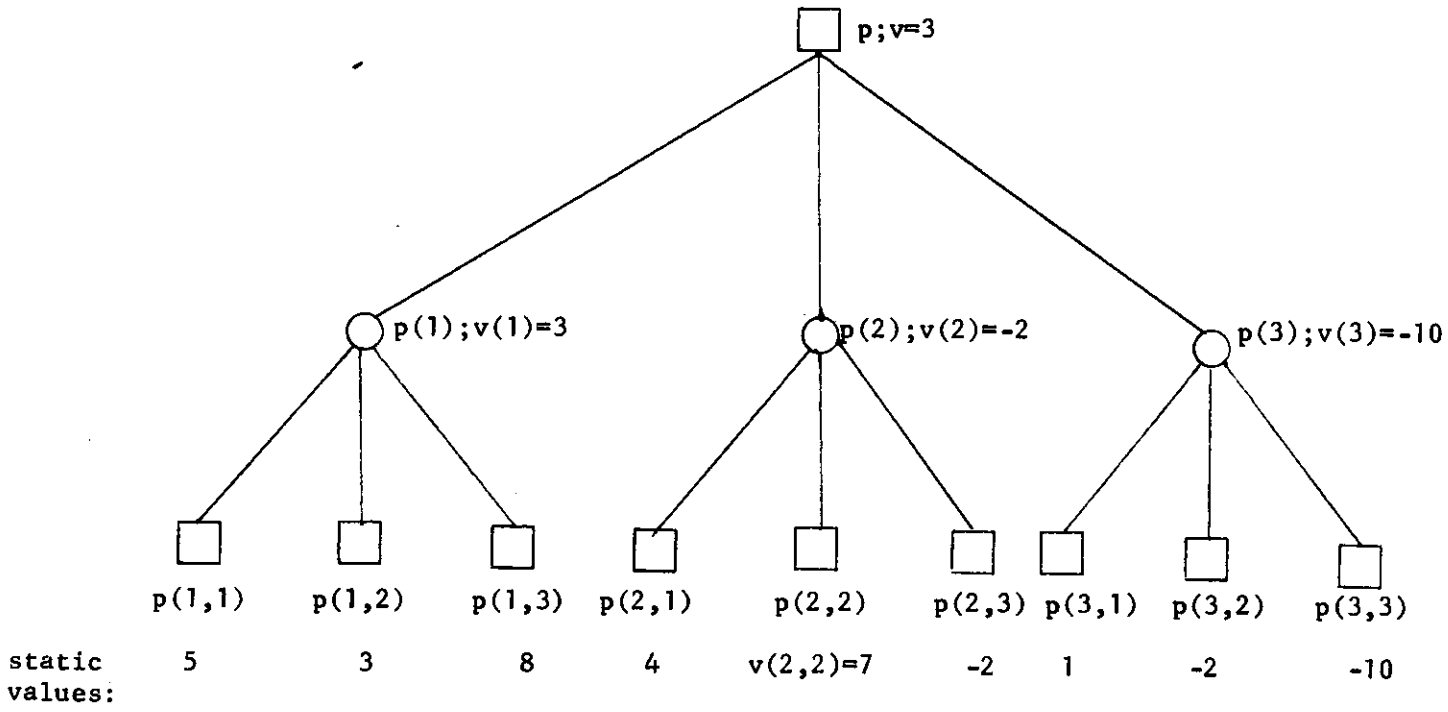


Figure 1.1. A game tree with branching factor 3 and depth 2.

Then the minimax procedure could be applied to this tree to find the optimal playing strategy. However, the exponential explosion of the "look-ahead" tree makes this impossible in practice. (It is estimated that there are about 10^{40} possible checkers games [Samuel, 1959] and about 10^{120} possible chess games [Shannon, 1950], but less than 10^{16} microseconds per century.) Therefore, the look-ahead process is typically continued down to some non-terminal (and possibly fixed) depth at which the position is evaluated with a less accurate evaluation function. If the branching factor, N, and the depth, D, are both fixed, then N^D bottom positions are generated in the minimax search. Even using incomplete (non-terminal) trees, the look-ahead trees for most game playing programs are still very large. In chess, for example, a typical value for the number of legal moves from a middle-game position is 35. If $\langle N,D\rangle = \langle 35,4\rangle$, then the number of bottom positions, N^D, which must be evaluated using simply minimax search is 1,500,625. For $\langle N,D\rangle = \langle 35,5\rangle$, $N^D = 42,521,875$. Chess playing programs are expected to satisfy the time constraints of tournament play: they are allowed two hours of computation time to make 40 moves. For a tree of size $\langle N,D\rangle = \langle 30,4\rangle$, this would mean that on the average about 220 microseconds would be available for evaluation of each bottom position if the minimax algorithm were used, including the tree-searching overhead involved in reaching that position. The need to effectively reduce the size of the tree to be searched is apparent.

In the remainder of this paper we will restrict our attention to Max-trees, i.e., game trees that maximize at the top level. We can do this without any loss in generality because of the obvious mappings that exist to

transform Min-trees to Max-trees. For example, consider the isomorphism: $\varphi(x) = -x$. Then by the definition of the min and max operators we see:

$$\begin{aligned} \max(x_1,x_2,\ldots,x_n) &= -\min(-x_1,-x_2,\ldots,-x_n) \\ &= \varphi(\min(\varphi(x_1),\varphi(x_2),\ldots,\varphi(x_n)) \end{aligned}$$

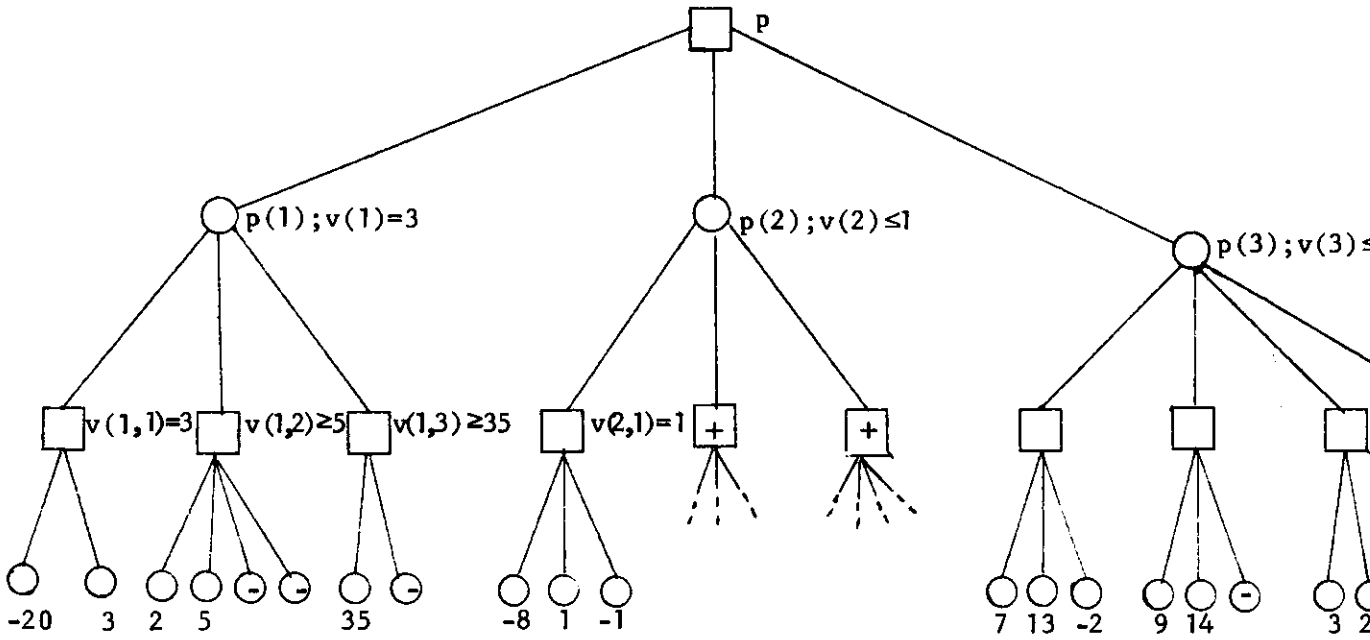
and

$$\begin{aligned} \min(x_1,x_2,\ldots,x_n) &= -\max(-x_1,-x_2,\ldots,-x_n) \\ &= \varphi(\max(\varphi(x_1),\varphi(x_2),\ldots,\varphi(x_n)) \end{aligned}$$

Since these identities can be applied recursively, an arbitrary Min-tree can be analyzed by analyzing the corresponding Max-tree created by complementing all the values in the Min-tree, replacing all min's by max's, and replacing all max's by min's. The only difference between a Min-tree and its associated Max-tree is that all backed-up (and static) values in the Max-tree will be the complement of the corresponding values in the Min-tree.

2. THE ALPHA-BETA PRUNING ALGORITHM

The alpha-beta algorithm is equivalent to the minimax algorithm in that they both find the same best move from position p and both will assign the same value of expected advantage to it. Alpha-beta is faster than minimax because it does not explore some branches of the tree that will not affect the backed-up value. The algorithm can be illustrated with the tree of depth three in Figure 2.1. Assuming that the searching proceeds in a depth-first fashion from left to right and that the root node is a Max node, the successors of Min node p(1) are first examined and the maximum value 3 is backed up to p(1,1). The value 3 now becomes an upper limit (beta value) for the backed-up value of node p(1). At this point the final value p(1) is unknown, but since p(1) is a Min node we do know that its value must be at most 3.



[+]: position not evaluated because of α cutoffs.

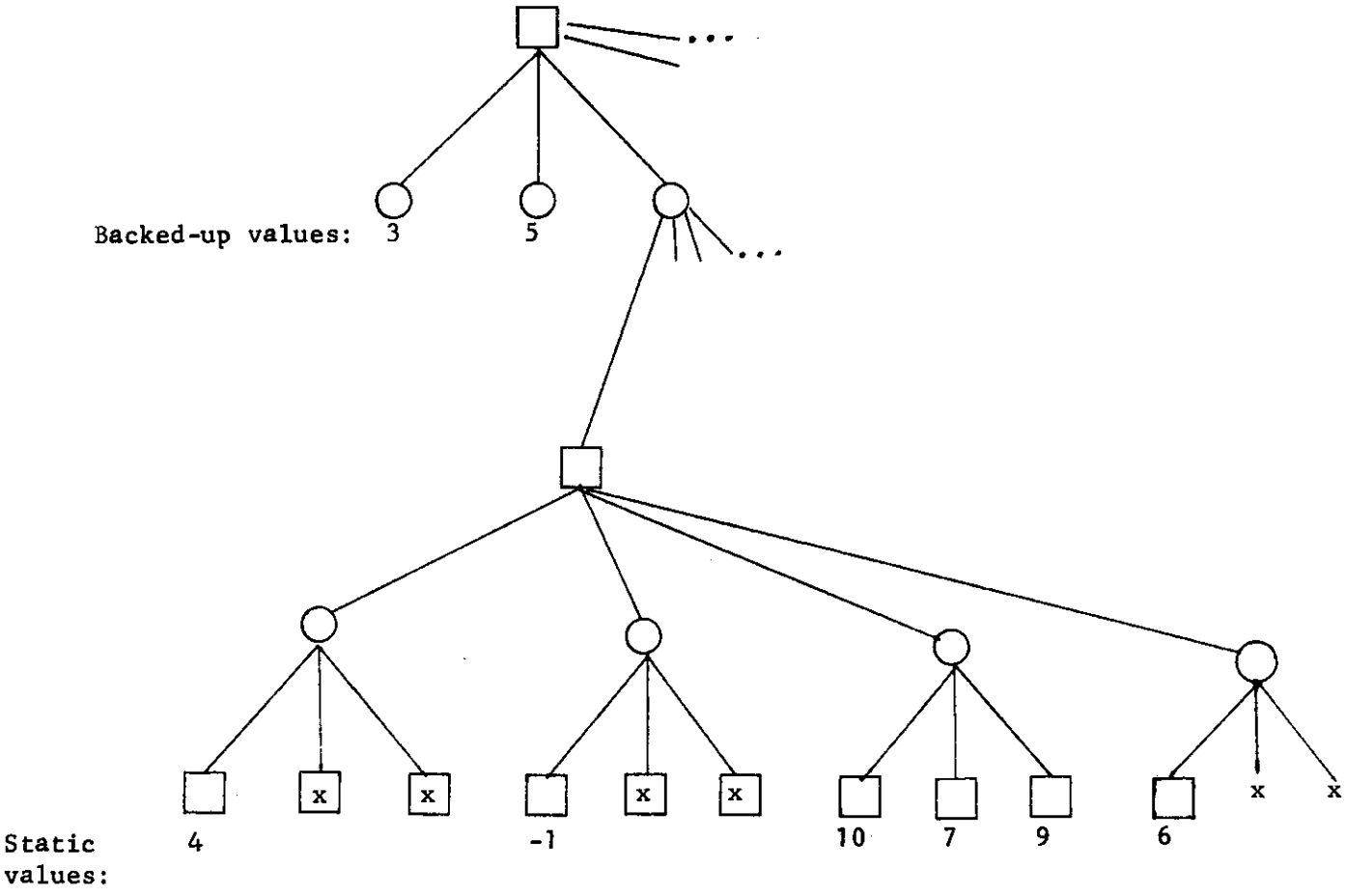
(-): position not evaluated because of β cutoffs.

Figure 2.1. Example of alpha and beta cutoffs.

Next the procedure begins to examine the successors of p(1,2). When p(1,2,2) is evaluated the lower limit (alpha value) for the backed-up value of the Max node p(1,2) becomes 5. Since the alpha value of p(1,2) is greater than the beta value of p(1) (=3), p(1,2) cannot be the lowest valued successor of p(1), and thus there is no need to evaluate the remaining successors of p(1,2). That is, Min will not select p(1,2) because Max can choose a branch leading to a higher value than Min knows can be achieved with p(1,1). Hence we have a beta cutoff at p(1,2,2). Additional beta cutoffs occur at p(1,3,1) and p(3,2,2).

After the beta prunes at p(1,2,2) and p(1,3,1) occur, the value 3 is backed-up to p(1) and becomes the lower limit (alpha value) for the backed-up value of node p. The procedure now begins to investigate the successors of p(2). On evaluation of p(2,1) the beta value of p(2) becomes 1. Since this is less than the alpha value (=3) of p, an alpha prune occurs at p(2,1). Because of alpha cutoffs, nodes p(2,2), p(2,3), and p(3,4), and their successors, are not evaluated. Note that only 15 bottom positions are evaluated by the alpha-beta procedure, whereas the minimax procedure would examine all 28.

In this example the alpha value used to obtain the alpha cutoffs was associated with the root node and the cutoffs occurred near the bottom level of the tree. Note that if the tree in the example were one of greater depth, the cutoffs at p(2,1) and p(3,3) would prune the potentially vast subtrees rooted at p(2,2), p(2,3), and p(3,4). Furthermore, an alpha or beta value may generate cutoffs at any node an even number of levels below it. These are called deep cutoffs and a deep alpha cutoff is illustrated in Figure 2.2.



[x] position not evaluated because of deep α cutoff

x position not evaluated because of shallow α cutoff

Figure 2.2. Example of deep alpha cutoffs.

Beta cutoffs are analogous to alpha cutoffs, with the roles of minimizing and maximizing reversed. The beta value specifies an upper limit for the backed-up value of a Min node and is used to generate cutoffs among the successors of Max nodes at any level deeper in the tree. Assuming that the root node (D=0) is a Max node, alpha cutoffs occur at even levels and beta cutoffs occur at odd levels.

In order to formally define the alpha-beta pruning algorithm described above, we introduce a few notational conveniences. Consider the partial game tree shown in Figure 2.3. We identify a node at depth $d \leq D$ in the tree as $p(\vec{i}_d)$, where $\vec{i}_d$, sometimes denoted $(i_1,i_2,\ldots,i_d)$, is a vector of length d whose components $i_1,i_2,\ldots,i_d$ identify the branch selected from the nodes at successive depths in the tree along the path from the root node to $p(\vec{i}_d)$. $v(\vec{i}_d)$ is the backed-up (for an intermediate node) or static (for a leaf node) value associated with node $p(\vec{i}_d)$.

To simplify subsequent subscripts and summation ranges, we introduce the notation

$$\lfloor k \rfloor_e = 2\lfloor \frac{k}{2} \rfloor = \begin{cases} k \text{ if } k \text{ is even} \\ k-1 \text{ if } k \text{ is odd} \end{cases} \tag{2.1}$$

$$\lfloor k \rfloor_o = 2\lfloor \frac{k-1}{2} \rfloor + 1 = \begin{cases} k \text{ if } k \text{ is odd} \\ k-1 \text{ if } k \text{ is even} \end{cases} \tag{2.2}$$

Consider the path from the root node to $p(\vec{i}_d)$. At level j, for j and d even and $0 \leq j < d$, a maximizing operation is in progress and we have a lower bound $a_j(\vec{i}_d)$ on $v(\vec{i}_j)$, denoted the <u>j-level alpha value</u>, where

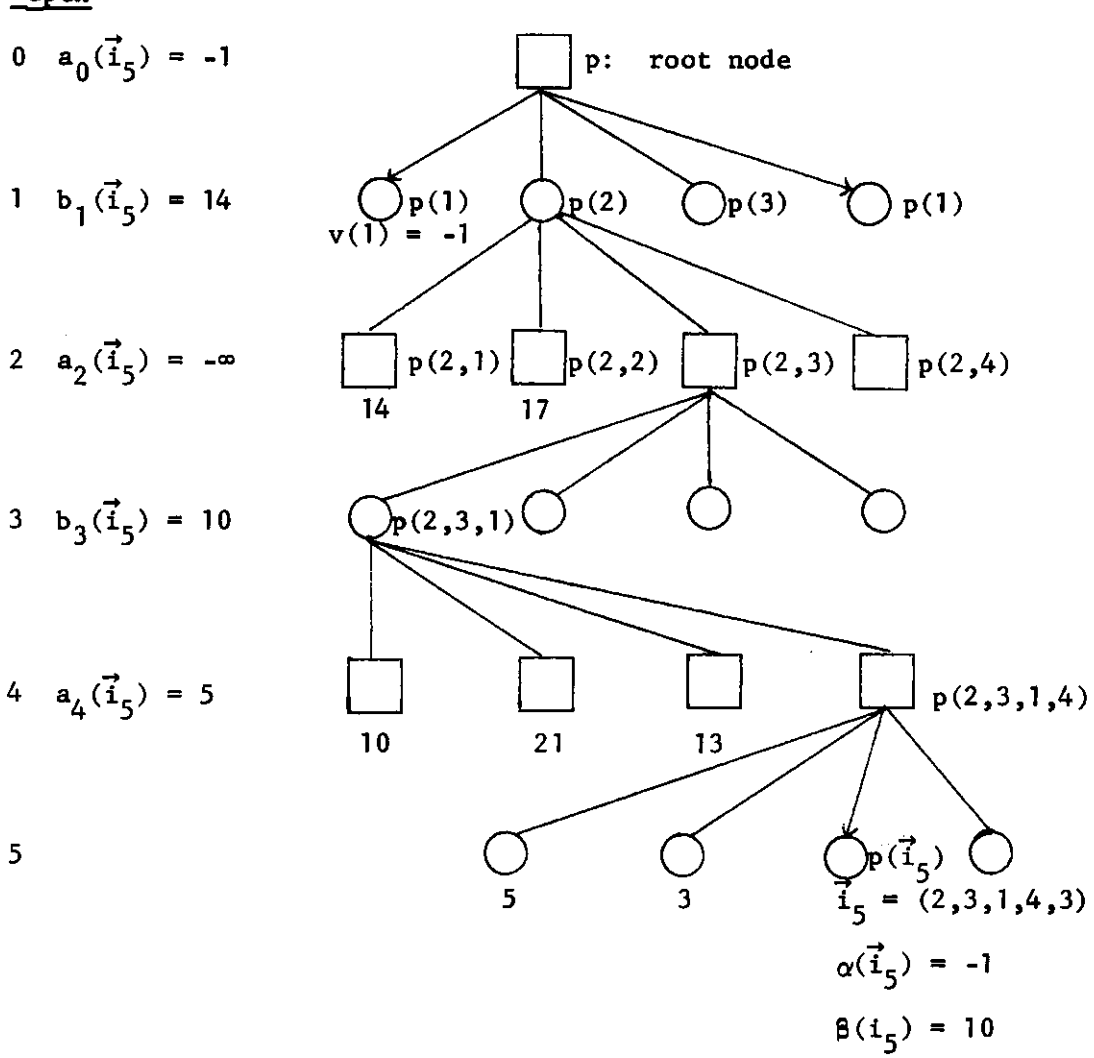


Figure 2.3. A game tree illustrating our notation for the alpha-beta algorithm.

$$a_j(\vec{i}_d) = \begin{cases} \max\{v(i_1,\ldots,i_j,1),\ v(i_1,\ldots,i_j,2),\ldots, \\ \qquad v(i_1,\ldots,i_j,i_{j+1}-1)\} & \text{for } i_{j+1} > 1 \\ -\infty & \text{for } i_{j+1} = 1 \end{cases} \tag{2.3}$$

Similarly, at level j, for j and d odd and $1 \le j < d$, a minimizing operation is in progress and we have an upper bound $b_j(\vec{i}_d)$ on $v(\vec{i}_j)$, denoted the <u>j-level beta value</u> where

$$b_j(\vec{i}_d) = \begin{cases} \min\{v(i_1,\ldots,i_j,1),\ v(i_1,\ldots,i_j,2),\ldots,v(i_1,\ldots,i_j,i_{j+1}-1) \\ \qquad \text{for } i_{j+1} > 1 \\ \infty \qquad \text{for } i_{j+1} = 1 \end{cases} \tag{2.4}$$

Finally, define the <u>greatest alpha value</u>, or simply <u>alpha value</u> as

$$\alpha(\vec{i}_d) = \max\{a_0(\vec{i}_d), a_2(\vec{i}_d),\ldots,a_{\lfloor d \rfloor_e}(\vec{i}_d)\} \tag{2.5}$$

and the <u>least beta value</u>, or simply <u>beta value</u>, as

$$\beta(\vec{i}_d) = \min\{b_1(\vec{i}_d), b_3(\vec{i}_d),\ldots,b_{\lfloor d \rfloor_o}(i_d)\} \tag{2.6}$$

If k is the level at which the maximum (minimum) of the $a_j(\vec{i}_d)$'s is attained, then the greatest alpha (least beta) value is a lower (upper) bound on the eventual backed-up value of the subtree rooted at $p(\vec{i}_k)$, and continuing to explore subtrees whose backed-up value cannot be greater than the alpha value (cannot be less than the beta value) is pointless.

<u>Definition of Alpha-Beta Pruning Algorithm</u>

The alpha-beta pruning algorithm is identical to the minimax algorithm except that whenever

$$v(\vec{i}_d) \le \alpha(\vec{i}_d) \text{ and d even}$$

an alpha cutoff occurs, and whenever

$$v(\vec{i}_d) \geq \beta(\vec{i}_d) \text{ and } d \text{ odd}$$

a beta cutoff occurs. A cutoff at node $p(\vec{i}_d)$ means that the remainder of the subtree rooted at $p(i_1,\ldots,i_{d-1})$, i.e., $p(\vec{i}_d)$'s parent node, is not examined in the minimax search.

The above discussion of j-level alpha and beta values proves the following fundamental lemma.

Alpha-Beta Lemma. Let $v_{\alpha\beta}(\vec{i}_0)$ be the backed-up value of a game tree using the alpha-beta pruning algorithm and let $v_{mm}(\vec{i}_0)$ be the backed-up value of the same game tree using the min-max algorithm. Then

$$v_{\alpha\beta}(\vec{i}_0) = v_{m\,m}\ (\vec{i}_0).$$

It should be noted that there is at least one class of risk-free pruning algorithms that is not subsumed by the alpha-beta algorithm. For example, consider the case where a top level move is found to lead to a win. Using the alpha-beta algorithm the next branch would have to be explored to some extent before being pruned; but it is clear that all other branches at the top level could be pruned immediately. This could, of course, be applied at any point in the tree where a win for the player to move is found.*

The use of alpha-beta pruning in the minimax search reduces by a large factor the number of bottom positions which need to be examined, typically

*Some care must be taken in the implementation of this algorithm. In the Second Annual Computer Chess Championship (Chicago, 1971) a chess program using this algorithm discovered a mate in two moves and terminated its search. After the opponent moved, the program began the search again, discovering first a mate in three. It immediately pruned and made the first move of this sequence, missing the possible mate on the move. It continued finding mates in more than one move until due to another bug it finally lost the game.

by several orders of magnitude in many game playing programs. Previous results [Slagle and Dixon, 1969] have established the lower limit for the number of bottom positions examined. The lower limit will be achieved if the static values of the bottom positions are in "perfect order", i.e., ordered such that every possible alpha and beta cutoff occurs. It can be shown that if perfect order is achieved at every level, so that every possible alpha or beta cutoff occurs, then the number of positions at the bottom of the tree of depth D and constant branching factor N is:

$$NBP_{po} = 2N^{\frac{D}{2}} - 1 \qquad \text{for D even,}$$

$$NBP_{po} = N^{\frac{D+1}{2}} + N^{\frac{D-1}{2}} - 1 \qquad \text{for D odd.}$$

Thus for $\langle N,D\rangle = \langle 35,4\rangle$, $NBP_{po} = 2449$, which differs from $35^4 = 1{,}500{,}625$ by a factor of 612.

This very large ratio of extremes in performance has important implications for searching large game trees. The performance of the alpha-beta procedure may be further improved by the incorporation of heuristics which reorder the nodes of the tree into a "more perfect" arrangement. Various techniques of fixed and dynamic ordering of nodes at intermediate levels of the tree are available [e.g., Slagle, 1963]. The rationale for these types of heuristics is based on a correlation between the static values of nodes at intermediate levels of the tree and the final backed-up values obtained for these nodes. This means that the nodes may be reordered before evaluation of their subtrees to more closely approximate perfect ordering and thus obtain a higher rate of pruning. The evaluation of the expected gain

over the simple alpha-beta algorithm obtained by the use of such heuristics is complicated by the fact that, while the perfect ordering results provide a greatest lower bound for the number of bottom positions evaluated, the upper bound of N^D is unrealistic because it is greater, often by several orders of magnitude than the number of bottom positions evaluated with the unmodified alpha-beta algorithm.

Knowledge of the expected value of the number of bottom positions evaluated in a look-ahead tree using alpha-beta pruning should be useful because the expected value provides a much tighter upper bound for the average performance of the tree-searching procedures than does the upper bound given by the minimax algorithm. Thus, when evaluating the effectiveness of heuristics to be used in conjunction with the alpha-beta algorithm one might determine not only how closely the resulting performance approaches the limit under perfect ordering, but also how much better (or worse!) the resulting performance is compared with that of the unmodified alpha-beta algorithm.

3. A PROBABLISTIC MODEL OF GAME TREES AND SOME INITIAL OBSERVATIONS

In order to draw some quantitative conclusions about the performance of the alpha-beta procedure it is necessary to precisely model game trees. However, our purpose here is to keep the model sufficiently simple so that analytical techniques can be applied to our study of the performance of the alpha-beta procedure.

Our model includes three simplifying assumptions.

1. Let us assume our game trees are complete trees of depth D with constant branching factor N, e.g., Figure 2.3 where D = 5 and N = 4.

Note that there are always N^D bottom positions, and in general N^d nodes at depth d in the game tree.

2. To study the probabilistic properties of the game trees we must provide a model of the static values assigned to the bottom positions. A simple yet appealing assumption to make is that the values, $v(\vec{i}_D)$, of all N^D bottom positions are independent, identically distributed (iid) random variables with arbitrary distribution function $V_D(x)$.

3. The only requirement on $V_D(x)$, in addition to the standard properties of a cumulative distribution function [cf. Parzen, 1960], is that it be continuous. In other words, we require that the probability that the value of a leaf node is precisely x is vanishingly small, to eliminate the possibility of two or more nodes having the same value.

The second and third assumptions can be equivalently restated by modeling the leaf nodes as a random permutation of the ordered list of values; i.e., each of the $N^D!$ assignment of values to the nodes is equally likely. Note that the actual values of the N^D bottom positions is not of interest when studying the behavior of minimax searching, and the alpha-beta procedure in particular, but only their relative ordering. Our previous discussion of the transformation of Min-trees to Max-trees implies that the probability of examining a particular bottom position in a Min-tree with continuous distribution $V_D(x)$ is equal to the probability of examining the corresponding bottom position in the associated Max-tree with distribution $V_D(-x)$. Thus, since the behavior of the search is independent of the specific distribution (as long as it is continuous), each of the subsequent results about Max-trees will be true of Min-trees as well.

We can now make the obvious but important observation that the value of a node at any level in the game tree is independent of the values of the other nodes at the same level. In addition, since the leaf nodes are iid random variables, it follows from the structure of our game trees, i.e., uniform depth at all bottom positions and constant brancing factors, that all the nodes at <u>any</u> level in the tree are iid random variables. It is interesting to consider the actual distribution of the values of the nodes at an arbitrary level. It follows from first principles in order statistics that the distribution function of the maximum of n iid random variables with distribution function $F(x)$ is $[F(x)]^n$ and the distribution function of the minimum of n iid random variables with distribution function $F(x)$ is $1-[1-F(x)]^n$. Hence:

$$V_0(x) = [V_1(x)]^N,$$
$$V_1(x) = 1-[1-V_2(x)]^N,$$
$$\bar{V}_1(x) = [\bar{V}_2(x)]^N;$$

and in general:

$$V_k(x) = V_{k+1}^N(x), \text{ for } k=0,2,\ldots,\lfloor D-1 \rfloor_e \qquad (3.1)$$

$$\bar{V}_k(x) = \bar{V}_{k+1}^N(x), \text{ for } k=1,3,\ldots,\lfloor D-1 \rfloor_o \qquad (3.2)$$

where $\bar{F}(x)$ denotes the survivor function, i.e., $\bar{F}(x) = 1-F(x)$.

To illustrate the relation of the distribution of the nodes from one level to the next, the distribution function at all the levels in the game tree of Figure 2.3 are shown in Figure 3.1. The value of the leaf nodes are assumed to be uniformly distributed over the unit interval in Figure 3.1, but this is only for illustrative purposes; as stated before, $V_D(x)$ can be any continuous distribution function.

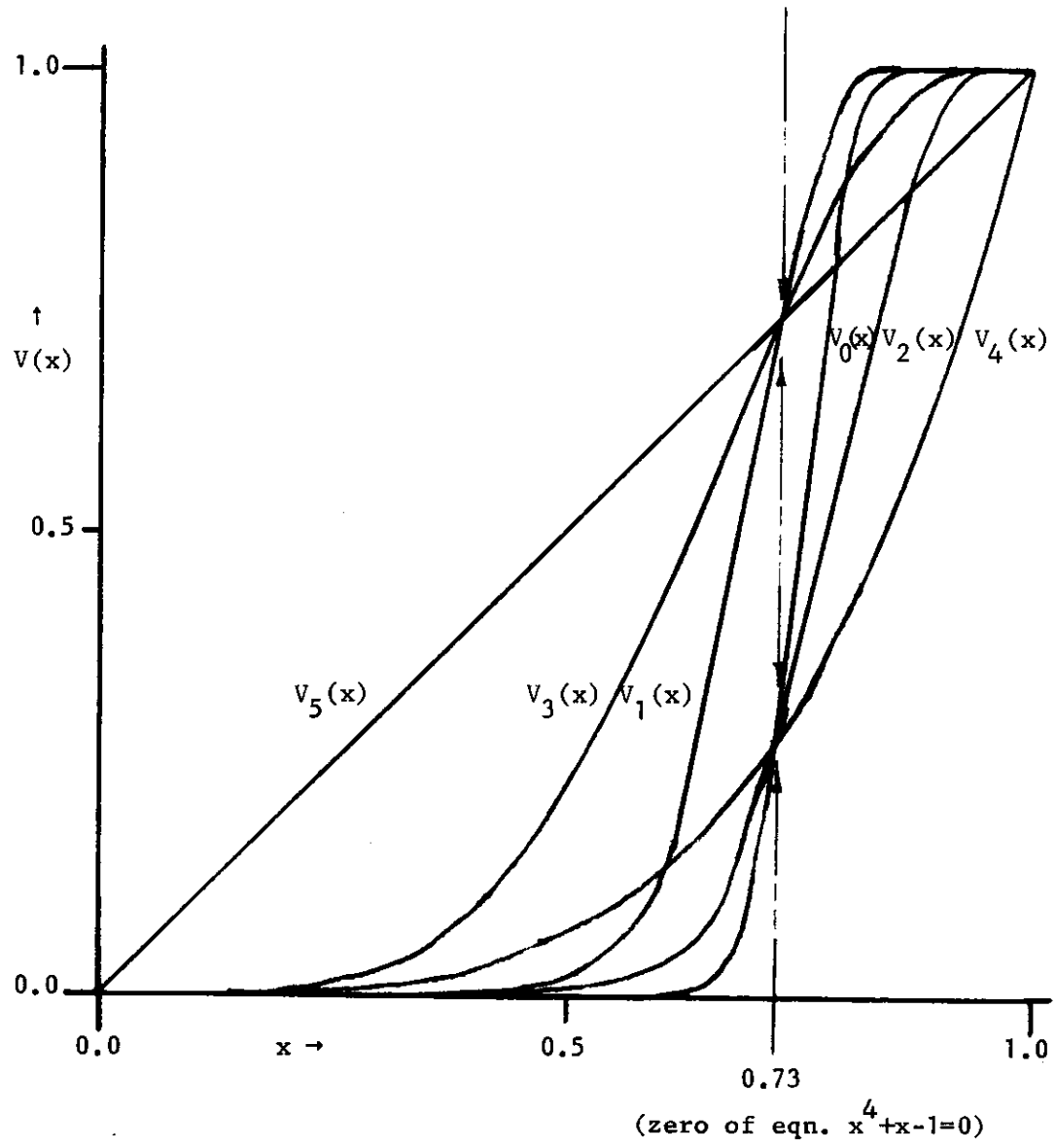


Figure 3. Cumulative distribution function of values of nodes in game tree with $\langle N,D \rangle = \langle 4,5 \rangle$.

4. THE PROBABILITY OF EVALUATING A NODE IN THE GAME TREE

We are interested in the statistics concerning the number of bottom positions evaluated in an alpha-beta search of a game tree. We will start by finding the probability that an arbitrary node with indices $\vec{i}_d$ is examined by the alpha-beta procedure; call this probability of examination $\Pr\{\vec{i}_d\}$.

To find $\Pr\{\vec{i}_d\}$ we first consider the path from the root node to $p(\vec{i}_d)$. At level j, for j a non-negative, even integer less than d, a maximizing operation is in progress, we have a lower bound on $v(\vec{i}_j)$, i.e., $a_j(\vec{i}_d)$, and the distribution function for the j-level alpha value is

$$A_{j,i_{j+1}}(x) = [V_{j+1}(x)]^{i_{j+1}-1}. \tag{4.1}$$

As i_{j+1} approaches N, the form of $A_{j,i_{j+1}}(x)$ approaches $V_j(x)$.

Similarly, at level j, for j a positive, odd integer less than D, a minimizing operation is in progress, we have an upper bound on $v(\vec{i}_j)$, i.e., $b_j(\vec{i}_d)$ and the survivor function for the j-level beta value is

$$\bar{B}_{j,i_{j+1}}(x) = [\bar{V}_{j+1}(x)]^{i_{j+1}-1}. \tag{4.2}$$

Note that the j-level alpha and beta values associated with $\vec{i}_d$ are <u>independent</u>, but not identically distributed random variables and the distribution function of $\alpha(\vec{i}_d)$ is

$$\underset{\sim}{A}_{\vec{i}_d}(x) = A_{0,i_1}(x)\, A_{2,i_3}(x) \ldots A_{\lfloor d-1 \rfloor_e, i_{\lfloor d \rfloor_o}}(x) \tag{4.3}$$

and similarly the survivor function of $\beta(\vec{i}_d)$ is

$$\underset{\sim}{B}_{\vec{i}_j}(x) = \bar{B}_{1,i_2}(x)\ \bar{B}_{3,i_4}(x)\ldots\bar{B}_{\lfloor d-1\rfloor_o,i_{\lfloor d\rfloor_e}}(x) \tag{4.4}$$

We can now prove several fundamental properties of the alpha-beta pruning algorithm.

Theorem 1. Node $p(\vec{i}_d)$ is examined, i.e., not pruned, by the α-β pruning algorithm if and only if

$$\alpha(\vec{i}_d) < \beta(\vec{i}_d). \tag{4.5}$$

Proof. First, suppose $\alpha(\vec{i}_d)$, the current alpha value, is less than $\beta(\vec{i}_d)$, the current beta value. In a proof by contradiction we will show this requires $p(\vec{i}_d)$ to be examined.

Suppose $p(\vec{i}_d)$ is not examined. By the definition of the alpha-beta algorithm, this implies there exists a node $p(\vec{i}_j{}^*)$ such that

$$\vec{i}_j{}^* = i_1,\ldots,i_{j-1},i_j{}^*; \qquad i_j{}^* < i_j$$

where $i_1,\ldots,i_{j-1},i_j$ are elements of $\vec{i}_d$ and

$$v(\vec{i}_j{}^*) \le \alpha(\vec{i}_j{}^*) = \alpha(\vec{i}_{j-1}); \quad j=2,4,\ldots,\lfloor d\rfloor_e \tag{4.6}$$

or

$$v(\vec{i}_j{}^*) \ge \beta(\vec{i}_j{}^*) = \beta(\vec{i}_{j-1}); \quad j=1,3,\ldots,\lfloor d\rfloor_o \tag{4.7}$$

In other words, if $p(\vec{i}_d)$ is not examined, an alpha or beta cutoff has occurred; the candidates for $p(\vec{i}_j{}^*)$ are shown in Figure 2.4. If we consider the alpha cutoff case, Eqn. (4.6), we see

$$b_{j-1}(\vec{i}_d) \le v(\vec{i}_j{}^*) \tag{4.8}$$

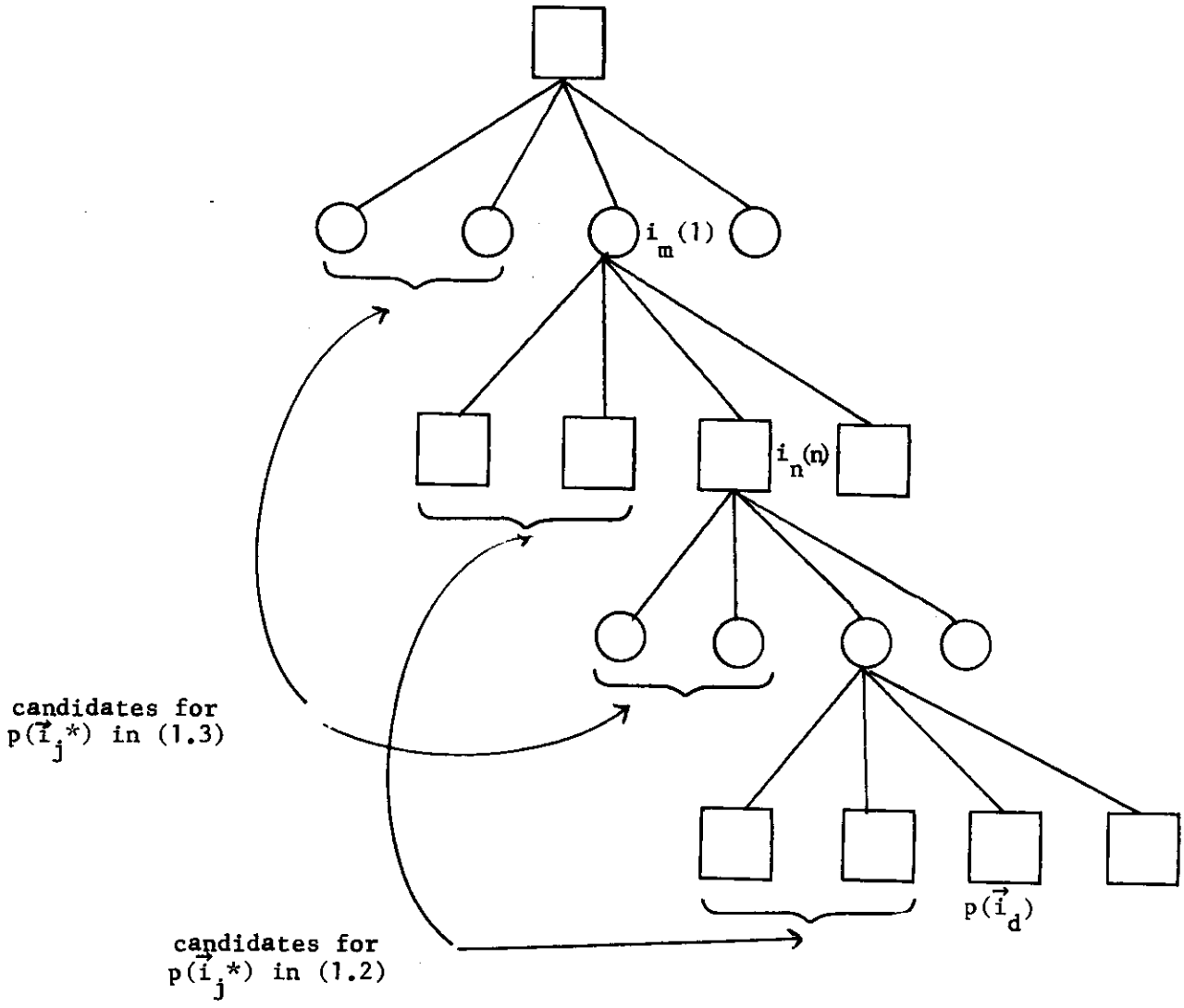

$i_m(1)$
$i_n(n)$
candidates for $p(\vec{i}_j^*)$ in (1.3)
candidates for $p(\vec{i}_j^*)$ in (1.2)
$p(\vec{i}_d)$

Figure 2.4

since $b_{j-1}(\vec{i}_d)$ is the minimum of the i_j-1 successors of $p(\vec{i}_{j-1})$. Clearly

$$\beta(\vec{i}_{j-1}) \leq b_{j-1}(\vec{i}_{j-1}) \tag{4.9}$$

by definition of the beta value, Eqn. 2.6. From Equations (4.6), (4.8), and (4.9) it follows that

$$\alpha(\vec{i}_{j-1}) \geq \beta(\vec{i}_{j-1}) \tag{4.10}$$

and from Eqns. (2.5) and (2.6)

$$\alpha(\vec{i}_d) \geq \beta(\vec{i}_d) \tag{4.11}$$

which contradicts Eqn. (4.5). By a precisely analogous argument our second case, Eqn. (4.7), also leads to Eqn. (4.11), and hence a contradiction.

Now it remains to be shown that if $p(\vec{i}_d)$ is examined, then Eqn. (4.5) must follow. Again proof by contradiction provides the simplest argument, i.e., suppose

$$\alpha(\vec{i}_d) \geq \beta(\vec{i}_d). \tag{4.12}$$

It follows from this inequality that there must exist a j and a k such that

$$b_j(\vec{i}_d) \geq a_k(\vec{i}_d). \tag{4.13}$$

Suppose $k > j$; then there exists a node $p(\vec{i}_{k+1}^*)$ such that

$$v(\vec{i}_{k+1}^*) = a_k(\vec{i}_d). \tag{4.14}$$

However, the above two equations guarantee a beta cutoff no later than $p(\vec{i}_{k+1}^*)$ and this contradicts the assumption of no cutoff.

If $k < j$, by a precisely analogous argument we get an alpha cutoff, again a contradiction. ■

Now that Theorem 1 has been formally presented it may be helpful to provide an intuitive description. Theorem 1 says that a node in a game tree is examined if and only if the associated upper bound (beta value) is greater than the associated lower bound (alpha value). Note that in this paper we have defined alpha and beta values for all nodes in the tree, not just those nodes examined by the alpha-beta procedure.

The next theorem is the central result of this section: an expression for the probability of evaluating an arbitrary node in the game tree.

Theorem 2. Let $\underset{\sim}{A}^{\rightarrow}_{i_j}(x)$ and $\underset{\sim}{B}^{\rightarrow}_{i_j}(x)$ be the distribution functions of the alpha and beta values, respectively for a node $p(\vec{i}_d)$ at depth d in a game tree. Then if $i_j > 1$ for some $j \in \{2,4,\ldots,\lfloor d \rfloor_e\}$:

$$\text{(a)} \quad \Pr\{\vec{i}_d\} = \int_{-\infty}^{\infty} \bar{\underset{\sim}{B}}^{\rightarrow}_{i_d}(z)\, d\underset{\sim}{A}^{\rightarrow}_{i_d}(z)$$

and if $i_j > 1$ for some $j \in \{1,3,\ldots,\lfloor d \rfloor_o\}$:

$$\text{(b)} \quad \Pr\{\vec{i}_d\} = \int_{-\infty}^{\infty} \underset{\sim}{A}^{\rightarrow}_{i_d}(z)\, d\bar{\underset{\sim}{B}}^{\rightarrow}_{i_d}(z)$$

and if $i_j = 1$ for all j:

$$\text{(c)} \quad \Pr\{\vec{i}_d\} = 1.$$

Proof. First, part (a). From Theorem 1 we know that the statement "position $p(\vec{i}_k)$ is not pruned" is equivalent to the statement "$\alpha(\vec{i}_n) < \beta(\vec{i}_n)$" and so:

$$\Pr\{\vec{i}_d\} = \Pr\{\alpha(\vec{i}_d) < \beta(i_d)\}$$

$$= \int_{-\infty}^{\infty} \Pr\{\alpha(\vec{i}_d) = z\} \Pr\{\alpha(\vec{i}_d) < \beta(\vec{i}_d) \mid \alpha(\vec{i}_d) = z\}\, dz$$

(Note: the condition $\alpha(\vec{i}_d) = z$ is defined only if some element of $\vec{i}_d$ with an even index is greater than 1.)

$$= \int_{-\infty}^{\infty} \frac{d}{dz} \underset{\sim}{A}_{\vec{i}_d}(z)\ \underset{\sim}{\bar{B}}_{\vec{i}_d}(z)\, dz$$

$$= \int_{-\infty}^{\infty} \underset{\sim}{\bar{B}}_{\vec{i}_d}(z)\, d\underset{\sim}{A}_{\vec{i}_d}(z)$$

The proof of part (b) is analogous to the above proof for part (a). Part (c) is obvious, since the first leaf node must always be evaluated. ■

5. THE EXPECTED NUMBER OF BOTTOM POSITIONS

In order to derive the expected number of bottom positions $E[NBP_{N,D}]$ evaluated in a tree of depth D and branching factor N which conforms to our model, we take advantage of the linearity of the expected value operator, i.e., $E[\Sigma x_i] = \Sigma E[x_i]$. Hence $E[NBP_{N,D}]$ is equal to the sum over the set of all bottom positions of the probability that the bottom position is evaluated, i.e.,

$$E[NBP_{N,D}] = \sum_{1 \le i_1 \le N} \sum_{1 \le i_2 \le N} \cdots \sum_{1 \le i_D \le N} \Pr\{\vec{i}_D\} \qquad (5.1)$$

and we may compute $\Pr\{\vec{i}_D\}$ using Theorem 2.

To illustrate the method we will first evaluate $E[NBP_{N,2}]$. First consider the case for $i_2 > 1$.

$$\Pr\{\vec{i}_2\} = -\int_{-\infty}^{\infty} \underset{\sim}{A}_{\vec{i}_2}(z)\, d\underset{\sim}{B}_{\vec{i}_2}(z) \qquad \text{(from Thm. 2b)}$$

$$= -\int_{-\infty}^{\infty} V_1^{i_1-1}(z)\, d\bar{V}_2^{i_2-1}(z) \qquad \text{(from Eqns. 4.1-4.4)}$$

$$= -\int_{-\infty}^{\infty} [1-\bar{V}_2^N(z)]^{i_1-1}\, d\bar{V}_2^{i_2-1}(z) \qquad \text{(from Eqn. 3.2)}$$

We may now perform the substitution $u = \bar{V}_2(z)$, eliminating the specific distribution of bottom positions.

$$\Pr\{\vec{i}_2\} = -\int_1^0 (1-u^N)^{i_1-1}\, d\, u^{i_2-1}$$

$$= \int_0^1 (1-x)^{i_1-1}\, d\, x^{\frac{i_2-1}{N}}$$

$$= \frac{i_2-1}{N}\int_0^1 (1-x)^{i_1-1} x^{\frac{i_2-1}{N}-1}\, dx$$

$$= \frac{i_2-1}{N}\,\beta\left(i_1, \frac{i_2-1}{N}\right) \qquad (i_2 > 1)$$

where $\beta(x,y) = \dfrac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$, the beta function.

Similarly we can find the value of $\Pr\{\vec{i}_2\}$ for $i_2 = 1$ and $i_1 > 1$ from Theorem 2a.

$$\Pr\{\vec{i}_2\} = \int_{-\infty}^{\infty} \underset{\sim}{B}_{\vec{i}_2}(z)\, d\underset{\sim}{A}_{\vec{i}_2}(z)$$

$$= \int_{-\infty}^{\infty} \bar{V}_2^{i_2-1}(z)\, d\, V_1^{i_1-1}(z)$$

For $i_2 = 1$ we have

$$\Pr\{\vec{i}_2\} = \int_{-\infty}^{\infty} d\, V_1^{i_1-1}(z)$$

$$= V_1^{i_1-1}\Big|_{-\infty}^{\infty}$$

by the fundamental theorem of calculus. Therefore

$$\Pr\{\vec{i}_2\} = 1 \qquad (i_2 > 1,\ i_2 = 1)$$

By Theorem 2c, $\Pr\{(1,1)\} = 1$.

Thus

$$E[NBP_{N,2}] = 1 + \sum_{i_1=2}^{N} 1 + \sum_{i_1=1}^{N} \sum_{i_2=2}^{N} \frac{i_2-1}{N}\, \beta(i_1, \frac{i_2-1}{N})$$

$$= N + \frac{1}{N} \sum_{i=1}^{N-1} i \sum_{j=1}^{N} \beta(j, \frac{i}{N})$$

$$= N + \frac{1}{N} \sum_{i=1}^{N-1} i \sum_{j=1}^{N} \int_0^1 u^{j-1}(1-u)^{\frac{i}{N}-1}\, du$$

$$= N + \frac{1}{N} \sum_{i=1}^{N-1} i \int_0^1 (1-u)^{\frac{i}{N}-1} \left(\sum_{j=1}^{N} u^{j-1} \right) du$$

$$= N + \frac{1}{N} \sum_{i=1}^{N-1} i \int_0^1 (1-u)^{\frac{i}{N}-2} (1-u^N)\, du$$

$$= N + \frac{1}{N} \sum_{i=1}^{N-1} i \left[\frac{N}{i-N} - \beta(\tfrac{i}{N}-1,\ N+1) \right]$$

$$= N + \frac{1}{N} \sum_{i=1}^{N-1} i \left[\frac{N}{i-N} - \frac{N^2}{i-N}\, \beta(\tfrac{i}{N}, N) \right]$$

$$E[NBP_{N,2}] = N + \sum_{i=1}^{N-1} \frac{i}{N-i} \left[N\beta(\tfrac{i}{N}, N) - 1 \right]$$

This form is quite adequate for computing the expected value over the range of branching factors useful in game playing programs. For small values of N, $E[NBP_{N,2}]$ was computed exactly using MACSYMA, a symbolic manipulation program developed at MIT [Bogen, et al. 1972]. These values are presented in Table 5.1.

Next we will evaluate $Pr\{\vec{i}_D\}$ for arbitrary depth. A few preliminary definitions and lemmas will supply the necessary foundations.

First we define the operator T(f,k) for a function f and non-negative integer k as follows:

$$T(f,k) = \begin{cases} f & \text{if } k = 0 \\ 1-[T(f,k-1)]^N & \text{if } k > 0 \end{cases}$$

For example, $T(V_3(x), 2) = 1-[T(V_3(x), 1)]^N$

$$= 1-[1-[T(V_3(x), 0)]^N]^N$$

$$= 1-[1-[V_3(x)]^N]^N$$

Lemma 5.1a: $V_{D-k}(x) = T(\bar{V}_D(x), k)$, D even, $k = 1,3,5,\ldots,D-1$

Lemma 5.1b: $V_{D-k}(x) = T(V_D(x), k)$, D odd, $k = 0,2,4,\ldots,D-1$

Lemma 5.1c: $\bar{V}_{D-k}(x) = T(\bar{V}_D(x), k)$, D even, $k = 0,2,4,\ldots,D$

Lemma 5.1d: $\bar{V}_{D-k}(x) = T(V_D(x), k)$, D odd, $k = 1,3,5,\ldots,D$

Proof: We will prove 5.1a by induction on k. The other proofs are the same.

For k = 1, D even, we have from Eqn. 3.2

$$\bar{V}_{D-1}(x) = \bar{V}_D^N(x)$$

$$V_{D-1}(x) = 1-\bar{V}_D^N(x)$$

$$V_{D-1}(x) = T(\bar{V}_D(x), 1)$$

$$E[NBP_{2,2}] = NBP2(2) = \frac{11}{3}$$

$$NBP2(3) = \frac{521}{70}$$

$$NBP2(4) = \frac{547033}{45045}$$

$$NBP2(5) = \frac{1720291169}{97349616}$$

$$NBP2(6) = \frac{14786001017771}{617109200400}$$

$$NBP2(7) = \frac{1483634273431527617}{47913489552349980}$$

$$NBP2(8) = \frac{125261514563804380970672 69}{32408631691415084087 4825}$$

$$NBP2(9) = \frac{606301942492917251264911019249 77}{129028361597620968737807901920 0}$$

$$NBP2(10) = \frac{3901522625927798419681780997160622175 1809}{6972037522971247716453380893531230355 68}$$

$$NBP2(11) = \frac{2019654264238087656238385656645961132348930138 3119}{3081521767653863505846263824165958587352116488 00}$$

$$NBP2(12) = \frac{2004150344234999412358773732137332143080908520205944926339}{2646891734883767626538481525642032211958379067379061835 0}$$

Table 5.1. $E[NBP_{N,2}]$

Assume $V_{D-k^*}(x) = T(\bar{V}_D(x), k^*)$ for a particular odd k^*. Then

$$
\begin{aligned}
V_{D-k^*-2}(x) &= 1-\bar{V}^N_{D-k^*-1}(x) && \text{(from Eqn. 3.2)} \\
&= 1-(1-V_{D-k^*-1}(x))^N \\
&= 1-(1-V^N_{D-k^*}(x))^N && \text{(from Eqn. 3.2)} \\
&= 1-[1-T^N(\bar{V}_D(x), k^*)]^N \\
&= 1-T^N(\bar{V}_D(x), k^*+1) \\
&= T(\bar{V}_D(x), k^*+2), \text{ proving the induction step. } \blacksquare
\end{aligned}
$$

We now observe that if $i_m = 1$ for $m = 2,4,\ldots,\lfloor D \rfloor_e$, then Theorem 2a may be applied directly.

$$
\begin{aligned}
\Pr\{\vec{i}_D\} &= \int_{-\infty}^{\infty} \underset{\sim}{\bar{B}}_{\vec{i}_D}(z)\, d\underset{\sim}{A}_{\vec{i}_D}(z) \\
&= \int_{-\infty}^{\infty} \prod_{\ell=2,4,\ldots,\lfloor D \rfloor_e} \bar{B}_{\ell-1}(z)\, d\Big(\prod_{k=1,3,\ldots,\lfloor D \rfloor_o} A_{k-1}(z) \Big) \\
&= \int_{-\infty}^{\infty} \prod_{\ell=2,4,\ldots,\lfloor D \rfloor_o} \bar{V}^{i_\ell - 1}(z)\, d\Big(\prod_{k=1,3,\ldots,\lfloor D \rfloor_o} V_k^{i_k-1}(z) \Big) \\
&= \int_{-\infty}^{\infty} d\Big(\prod_{k=1,3,\ldots,\lfloor D \rfloor_o} V_k^{i_k-1}(z) \Big), \text{ since } i_\ell = 1 \text{ for } \ell \text{ even} \\
&= \prod_{k=1,3,\ldots,\lfloor D \rfloor_o} V_k^{i_k-1}(z) \Big|_{-\infty}^{\infty} \text{ by the Fundamental Theorem of Calculus.}
\end{aligned}
$$

$\therefore \Pr\{\vec{i}_D\} = 1$ for $i_2 = i_4 = \ldots = i_{\lfloor D \rfloor_e} = 1$ and $i_m \neq 1$ for some odd m.

$\Pr\{\vec{i}_D\} = 1$ for $i_1 = i_2 = \ldots = i_D = 1$. (Thm. 2c)

For the rest of the development we will assume $i_m > 1$ for some even m.

We are now ready to consider $\Pr\{\vec{i}_D\}$ for arbitrary depth D. From Theorem 2b we have

$$\Pr\{\vec{i}_D\} = -\int_{-\infty}^{\infty} \underset{\sim}{\vec{A}}_{\vec{i}_D}(z)\ d\underset{\sim}{\vec{\bar{B}}}_{\vec{i}_D}(z)$$

$$= -\int_{-\infty}^{\infty} \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} A_{k-1}(z)\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} \bar{B}_{\ell-1}(z)\Big)$$

$$= -\int_{-\infty}^{\infty} \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} V_k^{i_k-1}(z)\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} \bar{V}_\ell^{i_\ell-1}(z)\Big)$$

As in the case for D = 2, we wish to perform a substitution which will eliminate the underlying distribution. For D odd we can apply lemmas 5.1b and 5.1d:

$$\Pr\{\vec{i}_D\} = -\int_{-\infty}^{\infty} \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} [T(V_D(z),\ D-k)]^{i_k-1}\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} [T(V_D(z),\ D-\ell)]^{i_\ell-1}\Big)$$

Substituting $u = V_D(z)$, we obtain

$$\Pr\{\vec{i}_D\} = -\int_0^1 \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} [T(u,D-k)]^{i_k-1}\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} [T(u,D-\ell)]^{i_\ell-1}\Big) \quad (5.3)$$

for D odd.

Similarly, for D even we apply lemmas 5.1a and 5.1c:

$$\Pr\{\vec{i}_D\} = -\int_{-\infty}^{\infty} \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} [T(\bar{V}_D(z),\ D-k)]^{i_k-1}\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} [T(\bar{V}_D(z),\ D-\ell)]^{i_\ell-1}\Big)$$

(5.4)

$$\Pr\{\vec{i}_D\} = +\int_0^1 \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} [T(u,D-k)]^{i_k-1}\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} [T(u,\ D-\ell)]^{i_\ell-1}\Big),\ D \text{ even.}$$

Combining equations (5.3) and (5.4),

$$\Pr\{\vec{i}_D\} = (-1)^D \int_0^1 \prod_{k=1,3,\ldots,\lfloor D\rfloor_o} [T(u,D-k)]^{i_k-1}\ d\Big(\prod_{\ell=2,4,\ldots,\lfloor D\rfloor_e} [T(u,D-\ell)]^{i_\ell-1}\Big)$$

Differentiating the second product and observing that the term for which $i_\ell = 1$ disappear, we have

$$\Pr\{\vec{i}_D\} = (-1)^D \int_0^1 \prod_{k=1,3,\ldots,\lfloor D \rfloor_o} T^{i_k-1}(u,D-k)) \sum_{\substack{m=2,4,\ldots,\lfloor D \rfloor_e \\ i_m \neq 1}} [(\prod_{\substack{\ell=2,4,\ldots,\lfloor D \rfloor_e \\ \ell \neq m}} T^{i_\ell-1}(u,D-\ell))\, dT^{i_m-1}(u,D-m)] \tag{5.5}$$

We evaluate the derivative of Eqn. 5.5 with the following lemma.

Lemma 5.2: $\frac{d}{du} T(u,k) = \begin{cases} 1 \text{ if } k = 0 \\ (-N)^k \prod_{n=0}^{k-1} T^{N-1}(u,n) \text{ if } k > 0 \end{cases}$

Proof (by induction on k):

$$\frac{d}{du} T(u,0) = 1$$

$$\frac{d}{du} T(u,1) = \frac{d}{du} (1-u^N) = -NT^{N-1}(u,0).$$

Now assume $\frac{d}{du} T(u,k^*-1) = (-N)^{k^*-1} \prod_{n=0}^{k^*-2} T^{N-1}(u,n)$ for some fixed k^*. Then

$$\frac{d}{du} T(u,k^*) = \frac{d}{du}(1-T^N(u,k^*-1))$$

$$= -N\, T^{N-1}(u,k^*-1)\, \frac{d}{du} T(u,k^*-1)$$

$$= (-N)^{k^*} \prod_{n=0}^{k^*-1} T^{N-1}(u,n) \quad \blacksquare$$

Applying this lemma to Eqn. 5.5 we obtain

$$\Pr\{\vec{i}_D\} = (-1)^D \int_0^1 (\prod_{k=1,3,\ldots,\lfloor D \rfloor_o} T^{i_k-1}(u,D-k)) \sum_{\substack{m=2,4,\ldots,\lfloor D \rfloor_e \\ i_m \neq 1}} [(\prod_{\substack{\ell=2,4,\ldots,\lfloor D \rfloor_e \\ \ell \neq m}} T^{i_\ell-1}(u,D-\ell))$$

$$(i_m-1) T^{i_m-2}(u,D-m)(-N)^{D-m} \prod_{n=0}^{D-m-1} T^{N-1}(u,n))]\, du$$

Moving the summation to the front, we have

$$\Pr\{\vec{i}_D\} =$$

$$\sum_{\substack{m=2,4,\ldots,\lfloor D\rfloor_e \\ i_m \neq 1}} (-1)^D (i_m-1)(-N)^{D-m} \int_0^1 \Big(\prod_{\substack{k=1 \\ k\neq m}}^{D} T^{i_k-1}(u,D-k) \Big) T^{i_m-2}(u,D-m) \prod_{n=0}^{D-m-1} T^{N-1}(u,n)\, du$$

Collecting terms and noticing that $(-1)^{2D-m} = 1$, we finally have

$$\Pr\{\vec{i}_D\} = \sum_{m=2,4,\ldots,\lfloor D\rfloor_e} (i_m-1) N^{D-m} \int_0^1 \prod_{k=0}^{D-1} T^{P_k(m)}(u,k)\, du \qquad (5.6)$$

where $P_k(m) = \begin{cases} 0 \text{ if } i_m = 1 \\ i_{D-k}+N-2 \quad \text{if } k < D-m \\ i_{D-k}-2 \quad \text{if } k = D-m \\ i_{D-k}-1 \quad \text{if } k > D-m \end{cases}$

$$\lfloor D\rfloor_e = \begin{cases} D \text{ if } D \text{ is even} \\ D-1 \text{ if } D \text{ is odd} \end{cases}$$

and $i_m > 1$ for some even m.

Evaluation of Eqn. 5.6 requires integration of functions of the form

$$I_k(j_1,j_2,\ldots,j_k) = \int_0^1 \prod_{n=1}^{k} T^{j_n}(u,k-n+1).$$

For example, if $i_2 \neq 1$

$$\Pr\{\vec{i}_3\} = (i_2-1)\, N \int_0^1 T^{i_3+N-2}(u,0)\, T^{i_2-2}(u,1)\, T^{i_1-1}(u,2)\, du$$

$$= (i_2-1)\, N \int_0^1 u^{i_3+N-2} (1-u^N)^{i_2-2} (1-(1-u^N)^N)^{i_1-1}\, du$$

$$= (i_2-1)\, N\, I_3(i_1-1, i_2-2, i_3+N-2).$$

These integrals may be evaluated exactly using the recurrence relations of the following lemma.

Lemma 5.3:

$$I_k(j_1,\ldots,j_k) = \begin{cases} \dfrac{1}{j_1+1} & \text{if } k = 1 \\ \dfrac{1}{N}\beta(j_1+1,\dfrac{j_2+1}{N}) & \text{if } k = 2 \\ \displaystyle\sum_{\ell=0}^{j_1} (-1)^{\ell}\binom{j_1}{\ell} I_{k-1}(j_2+\ell N, j_3,\ldots,j_k) & \text{if } k > 1. \end{cases}$$

Proof: $I_1(j_1) = \int_0^1 u^{j_1}\,du = \dfrac{1}{j_1+1}$.

$$I_2(j_1,j_2) = \int_0^1 (1-u^N)^{j_1} u^{j_2}\,du$$

$$= \frac{1}{N}\int_0^1 (1-v)^{j_1} v^{\frac{j_2+1}{N}-1}\,dv$$

$$= \frac{1}{N}\beta(j_1+1,\ \frac{j_2+1}{N}).$$

$$I_k(j_1,\ldots,j_k) = \int_0^1 \prod_{n=1}^{k} T^{j_k}(u,k-n+1)\,du$$

$$= \int_0^1 T^{j_1}(u,k) \prod_{n=2}^{k} T^{j_k}(u,k-n+1)\,du$$

$$= \int_0^1 (1-T^N(u,k-1))^{j_1} \prod_{n=2}^{k} T^{j_k}(u,k-n+1)\,du$$

$$= \int_0^1 \sum_{\ell=0}^{j_1} (-1)^{\ell}\binom{j_1}{\ell} [T^N(u,k-1)]^{\ell} \prod_{n=2}^{k} T^{j_k}(u,k-n+1)\,du$$

(from the Binomial Theorem)

$$= \sum_{\ell=0}^{j_1} (-1)^\ell \binom{j_1}{\ell} \int_0^1 T^{\ell N}(u,k-1) \prod_{n=2}^{k} T^{j_k}(u,k-n+1)\, du$$

$$= \sum_{\ell=0}^{j_1} (-1)^\ell \binom{j_1}{\ell} I_{k-1}(j_2+\ell N, j_3, \ldots, j_k)$$

■

As another example of the method, we present the different cases for D = 4 and arbitrary N.

$\Pr\{\vec{i}_4\} = 1$ for $i_1=i_2=i_3=i_4=1$ from Thm. 2c.

$\Pr\{\vec{i}_4\} = 1$ for $i_2=i_4=1$ from Eqn. 5.2.

$$\Pr\{\vec{i}_4\} = (i_2-1)N^2 \int_0^1 T^{i_1-1}(u,3)\, T^{i_2-2}(u,2)\, T^{i_3+N-2}(u,1)\, du$$

$$= (i_2-1)N^2\, I_4(i_1-1, i_2-2, i_3+N-2, 0) \quad \text{if } i_4=1 \text{ and } i_2 \neq 1$$

$$\Pr\{\vec{i}_4\} = (i_4-1)\, I_4(i_1-1, 0, i_3-1, i_4-2) \quad \text{if } i_2=1 \text{ and } i_4 \neq 1$$

$$\Pr\{\vec{i}_4\} = (i_2-1)N^2\, I_4(i_1-1, i_2-2, i_3+N-2, i_4+N-2) +$$
$$+ (i_4-1)\, I_4(i_1-1, i_2-1, i_3-1, i_4-2) \quad \text{if } i_2 \neq 1 \text{ and } i_4 \neq 1.$$

Then $E[NBP_{N,4}] = \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \sum_{i_3=1}^{N} \sum_{i_4=1}^{N} \Pr\{\vec{i}_4\}$,

and $I_4(j_1,j_2,j_3,j_4)$ is evaluated with Lemma 5.3.

It is possible to eliminate one summation from this method of evaluating $E[NBP_{N,D}]$. We observe that

$$\Pr\{\vec{i}_D\} = \sum_{m=2,4,\ldots,\lfloor D \rfloor_e} (i_m-1)\, I_D(i_1+t_1, \ldots, i_D+t_D),$$

where $i_m > 1$ for some even m, $t_k = P_{D-k}(m)-i_k$ (Eqn. 5.6), and I_D is computed with a D-2-fold summation. However, the summation on the index i_1 may be combined with the first reduction of Lemma 5.3.

$$\sum_{i_1=1}^{N} \Pr\{\vec{i}_D\} = \sum_{i_1=1}^{N} \sum_{m=2,4,\ldots,\lfloor D \rfloor_e} (i_m - 1)\, I_D(i_1+t_1,\ldots,i_D+t_D)$$

$$= \sum_{m=2,4,\ldots,\lfloor D \rfloor_e} (i_m - 1) \sum_{i_1=1}^{N} \sum_{\ell=0}^{i_1+t_1} (-1)^{\ell} \binom{i_1+t_1}{\ell} I_{D-1}(i_2+t_2+N\ell, i_3+t_3,\ldots, i_D+t_D)$$

Reordering the terms of the last two summations, we obtain

$$\sum_{i_1=1}^{N} \Pr\{\vec{i}_D\} = \sum_{m=2,4,\ldots,\lfloor D \rfloor_e} (i_m - 1) \sum_{g=0}^{i_1+t_1} (-1)^{g} I_{D-1}(i_2+t_2+gN, i_3+t_3,\ldots,i_D+t_D) \sum_{h=g}^{N} \binom{h+t_1}{g}$$

$$= \sum_{m=2,4,\ldots,\lfloor D \rfloor_e} (i_m - 1) \sum_{g=0}^{i_1+t_1} (-1)^{g} \binom{N+t_1+1}{g+1} I_{D-1}(i_2+t_2+gN, i_3+t_3,\ldots,i_D+t_D),$$

reducing by one summation the complexity of evaluation.

For depth 3 and 4 the MACSYMA system was again used to obtain exact values of $E[NBP_{N,D}]$ for small values of N. These results are shown in Table 5.2. Because of the complexity of this formulation (a 2D-2-fold nested summation) it was not possible to evaluate Eqn. (5.6) exactly for large D and N. In addition, the nested alternating signs of Lemma 5.3 lead to a rapid loss of significance. Figure 5.1 therefore uses data from a Monte Carlo simulation using the same model as well as from evaluation of Eqn.(5.6). The standard deviations are also from the simulation.

For some game-playing programs the important measure of effort is not the number of bottom positions evaluated, but rather the number of legal move generations. This is the case for programs which use simple evaluation functions, but for which there is no simple way to generate legal moves. Because the number of bottom positions is independent of the distribution from which

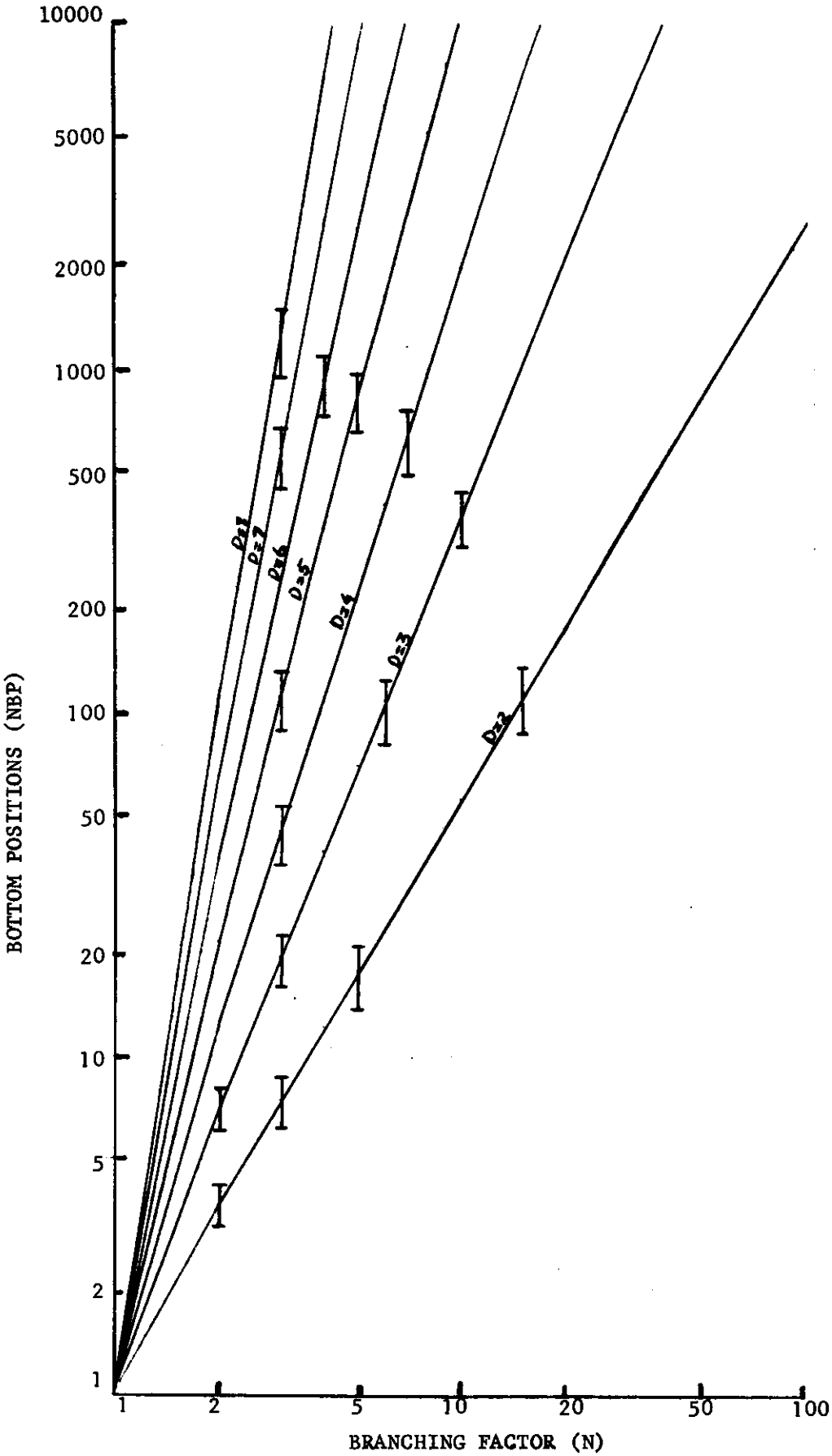


FIGURE 5.1: $E[NBP_{N,D}]$ vs. N for D = 2,3,..., 8

$$E[NBP_{2,3}] = NBP3(2) = \frac{719}{105}$$

$$NBP3(3) = \frac{57879389791}{2974571600}$$

$$NBP3(4) = \frac{10586121348272080231668796 7}{263898858058665684712357 5}$$

$$NBP3(5) = \frac{12165787617245775614978876851269378787623875904 17}{17413602576224093163828296974817345906119971648}$$

$$E[NBP_{2,4}] = NBP4(2) = \frac{77503}{6435}$$

$$NBP4(3) = \frac{37971767498059857058330872657008 3}{84003779473695889436992417264 00}$$

Table 5.2. Exact values for $E[NBP_{N,D}]$ for D = 3,4.

the values are drawn, we see that the expected number of move generation for a depth D search is simply the sum of the expected number of non-terminal positions at each level, i.e.,

$$E[NMG_{N,D}] = 1 + \sum_{i=2}^{D-1} E[NBP_{N,i}]$$

6. APPLICATION OF THE GAME TREE MODEL TO CHESS

Several assumptions have been made to simplify the analysis of the model which do not conform to the properties of game trees in general. First, the model assumes a fixed branching factor and fixed depth. Many game playing programs (though not all) use searches of variable depth and breadth. Second, the values of the bottom positions have been assumed to be independent; in practice there are strong clustering effects. For example, in a chess program with an evaluation function which depends strongly on material, a subtree whose parent move is a queen capture will have more bottom positions in the range corresponding to the loss (or win) of a queen than will subtrees whose parent move is a non-capture. The final assumption is that the probability that two bottom positions in the tree will have the same value is zero (continuity assumption). In practice, game programs select the value of the terminal position from a finite (and sometimes small) set of values.

No attempt has been made to model modifications to the basic alpha-beta search, such as fixed ordering, dynamic ordering, or the use of aspiration levels. This model should be viewed as an upper bound in the sense that any program can perform this well if the moves are no worse than randomly ordered.

In this section we will investigate the effects of these assumptions by comparing the predicted average search effort with the observed searches of a chess program. Since several existing chess programs use a fixed branching factor (selecting the N best moves according to a static evaluation function) and fixed depth (attempting to resolve issues of quiescence with a static analysis), we will concentrate on the independence assumption and the continuity assumption.

In order to assess the expected effect of the continuity constraint, we relax it in the model so that the values for the evaluation function are chosen from R equally likely distinct values. If R = 1, we have, in effect, perfect ordering, since equal values will produce a cutoff. As R approaches infinity the expected number of bottom positions is predicted analytically by this paper. The variation of the number of bottom positions with R is shown in Figure 6.1 for D = 3; these curves were generated using a Monte Carlo simulation.

The chess program used for comparisons was a modification of the Technology Chess Program [Gillogly, 1972]. The Technology Program (Tech) is a "brute force" program which investigates all legal moves to a fixed depth; all chains of captures from these bottom positions are explored. The terminal positions are evaluated only with respect to material, where a pawn is considered to be worth 100 points, knight and bishop 330, rook 500, and queen 900. A number of positional heuristics are applied statically at the top level, and various modifications are made to the basic tree search.

Tech was modified for this analysis to search trees of fixed depth and branching factor (the branches to be examined selected randomly), and

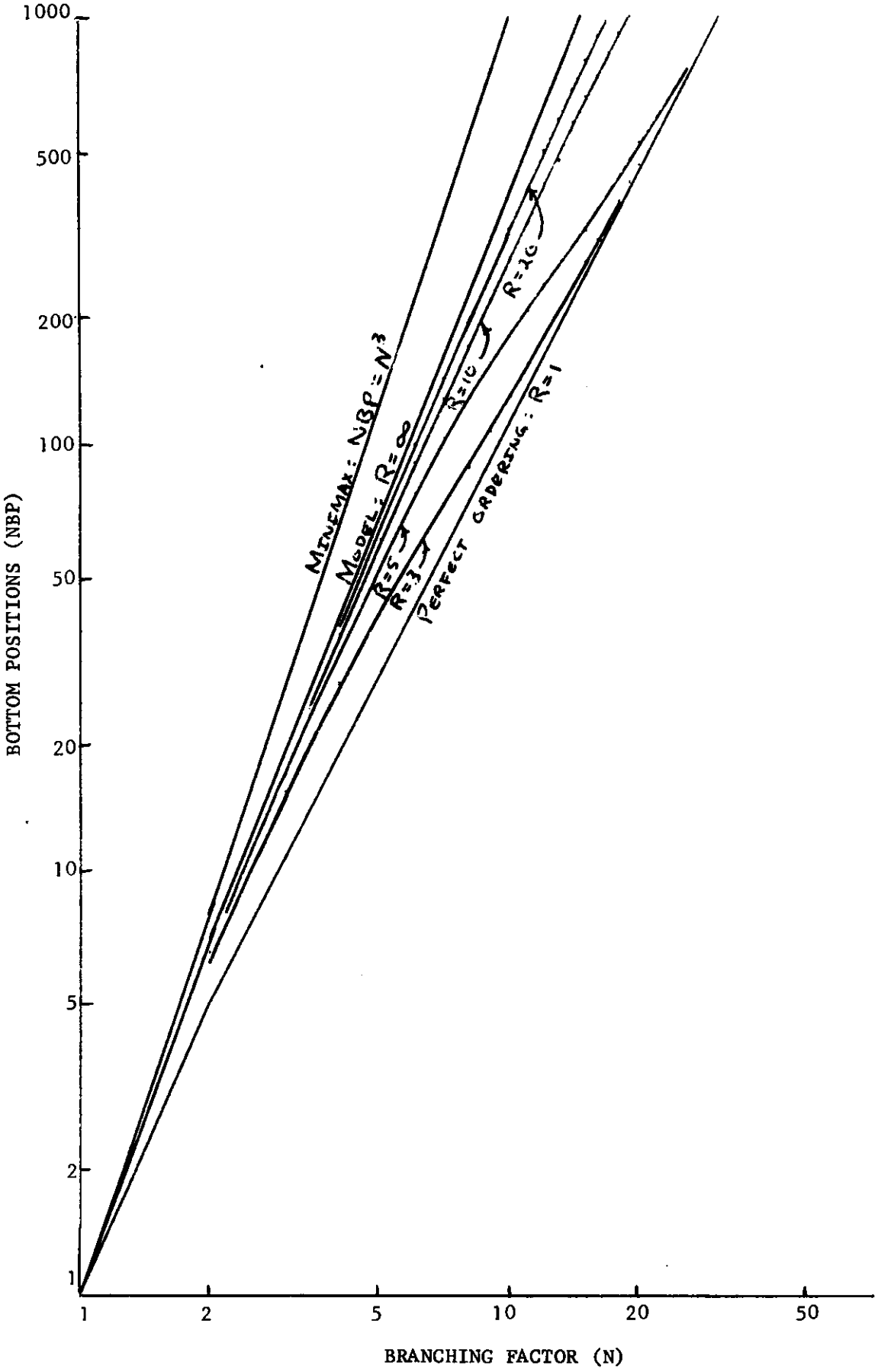


FIGURE 6.1 - Depth 3: EFFECT OF RANGE (R) OF EVALUATION FUNCTION ON NBP

the tree search modifications were deleted. In addition to Tech's standard evaluation function, an optional evaluation term for mobility was programmed. This term is useful for the analysis, since it increases the effective range of evaluation and decreases the degree of correlation in subtrees. Most chess programs have evaluation functions which are considerably more complex than Tech's.

In order to ensure a reasonable mix of opening, middle and endgame positions a complete game was analyzed, consisting of 80 positions (Spassky-Fischer, Reykjavik 1972, game 21). Each of these positions was analyzed by the modified Tech programs for D = 2, 3, and 4 over the effective range of branching factors. Figure 6.2 shows the analytic results for these parameters with the empirical values obtained using Tech's standard evaluation function. At a typical point ($\langle N,D \rangle = \langle 10,3 \rangle$), the observed range (R) of distinct bottom position values in the trees varied between 1 and 9, with median 5. This agrees well with Figure 6.1. To demonstrate the effect of the independence assumption, these points were re-run with the program modified so that a value which would result in a prune by equality was randomly perturbed up or down. This simulated an evaluation function that assigns unique values for all the bottom positions. The pertubations changed the value of the position by at most two points, since there are at most two alphas or two betas being kept at any given time in a depth 4 tree. Since two points is small compared to the value of a pawn (100 points), the tie-breaking procedure does not significantly affect the correlation among positions in a subtree. The systematic discrepancy between these points and the analytic curve must then be due to the assumption of independence.

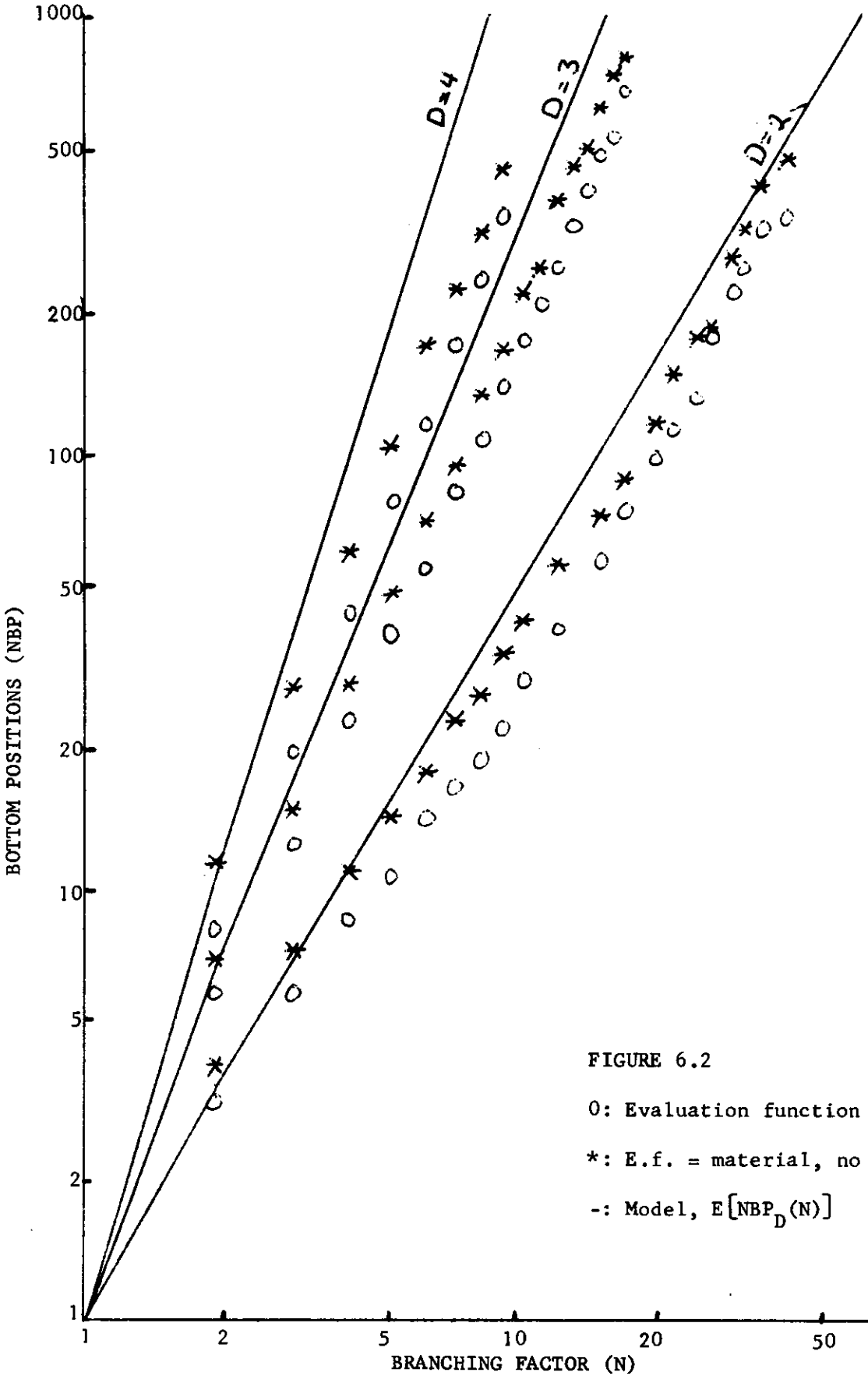


FIGURE 6.2

0: Evaluation function = material

*: E.f. = material, no equal values

-: Model, $E[NBP_D(N)]$

In Figure 6.3 we plot the analytic curves with the data from the evaluation function which includes mobility as well as material. The range is much higher for this evaluation function, so that the number of bottom positions is considerably higher than the unmodified version. The range (R) at $\langle N,D \rangle = \langle 10,3 \rangle$ for this version varied between 12 and 82, with median 43. Since the tiebroken points are only slightly higher, it is clear that the range is close to the effective maximum range for that evaluation function. It is of interest to note that the tiebroken points lie almost on the analytic line, indicating that the independence assumption is more nearly correct for this evaluation function.

Comparison of these graphs indicates two ways in which the evaluation function affects the number of bottom positions evaluated: (1) as the range of the evaluation function increases, the number of bottom positions increases; (2) as the correlation among values in the same subtree increases, the number of bottom positions decreases.

7. EMPRICAL OBSERVATIONS

For large values of N and D, the analytic result presented in Section 5 is in its present form computationally infeasible. The growth of computation time is governed by two factors. First, the probability that a bottom position is evaluated must be calculated for each bottom position in the complete tree, a D-fold nested summation for $D > 2$. Second, the expression for this probability is in the form of a recurrence relation (Lemma 5.3) which reduces the integral to a D-2-fold nested summation of evaluations of the beta function and binomial coefficients. The number of nested summations

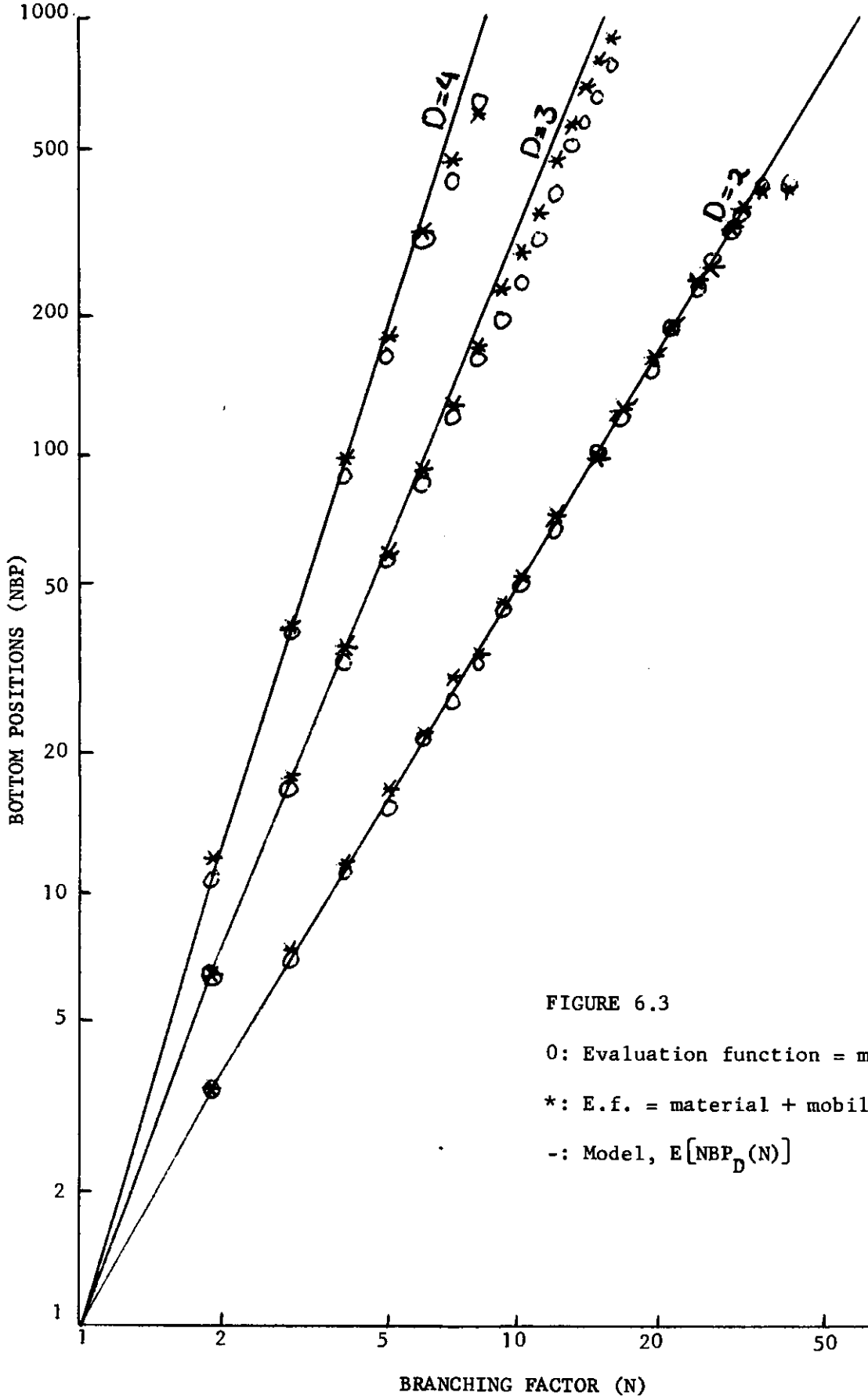


FIGURE 6.3

0: Evaluation function = material + mobility

*: E.f. = material + mobility, no equal values

-: Model, $E[NBP_D(N)]$

for the complete tree is thus 2D-2. The simplification of the analytical result to a form for which the computation effort grows at a significantly slower rate remains an open problem.

In the hope of providing some results of practical value, we present the empirical observations which follow. The set of data points upon which the empirical observations are based were obtained in part from the analytical formulation and in part from a Monte Carlo simulation of the alpha-beta algorithm. In the simulation the static values for the leaf nodes of the trees were drawn from a uniform random probability distribution of integers in the interval $(0,2^{35}-1)$. The sample mean and standard deviation were obtained from the results of 1000 iterations of the alpha-beta algorithm for each size of the tree. The sample mean thus lies within the interval $\bar{x}_{N,D} \pm R \cdot \bar{x}_{N,D}$ at the 95% confidence level, where $\bar{x}_{N,D}$ is the true mean and $0.008 < R < .013$ for the points collected in the simulation. The ratio of sample standard deviation to sample mean was computed for 173 points (the points derived from the analytic formulation were also simulated to obtain their sample standard deviations), and all of these ratios lie in the interval (.120, .215).

The values of $\log E[NBP_{N,D}]$ as a function of $\log N$ for fixed D are observed to be approximately a straight line (Figure 5.1). A least squares fit of a straight line to these data gives an approximation whose root mean square relative error is about .004 for the values examined. The corresponding approximation to the expected number of bottom positions takes the form $E[NBP_{N,D}] \approx K_D \cdot N^{L_D}$, with an RMS relative error of approximately .015. Furthermore, the values L_D are observed to be approximately linear with respect to D. A least squares fit of a straight line to the

values L_D yields the approximation $L_D = .720D + .227$, with an RMS relative error of approximately .007.

Table 7.1: K_D

Depth	K_D
1	1.00
2	1.12
3	1.30
4	1.39
5	1.51
6	1.59
7	1.67
8	1.71
9	1.73

It is of interest to note that the growth rate for the alpha-beta algorithm (.720D) is about midway between that of perfect ordering (0.5D) and minimax search (D). An accurate simple approximation to the values K_D is not readily apparent. The values of K_D for D=1,2,...,9 are found in Table 7.1. The values lie in the range [1,2].

This empirical analysis of the data available to us suggests an approximation of the form $E[NBP_{N,D}] \approx K_D \cdot N^{.720D + .277}$. The limited number and arbitrary selection of the data points used in this analysis prevent us from attaching high confidence to this approximation for the general case, although for the points from which it is derived the approximation is excellent. We do note that the approximation agrees with the boundary case D = 1, for which $E[NBP_{N,1}] = N \approx N^{.720D + .277} = N^{.997}$, and agrees fairly well with

the boundary case N = 1 for which $E[NBP_{1,D}] = K_D$.

Slagle and Dixon [1969] define a measure of the relative efficiency of a tree-searching algorithm:

$$DR_x \equiv \frac{\log \overline{NBP}_x}{\log \overline{NBP}_{MM}},$$

where $\overline{NBP}_{MM}$ is the expected number of bottom positions examined in a minimax search and $\overline{NBP}_x$ is the expected number of bottom positions examined by algorithm x.

The value of DR lies between 0 and 1. As an example of the interpretation of DR, DR = .6 indicates that the algorithm under consideration can be used to search a tree of depth 5 with about the same effort as that required by the minimax algorithm to search a tree of depth 3. Using the empirically derived approximation,

$$DR_{\alpha\text{-}\beta} \approx .720 + \frac{.277}{D} + \frac{\log K_D}{D \log N}$$

For large N, $DR_{\alpha\text{-}\beta} \approx .720 + \frac{.277}{D}$.

For the search with perfect ordering [Slagle and Dixon, 1969],

$$DR_{PO} \approx \frac{1}{2} + \frac{\log 2}{D \log N}.$$

For large N, $DR_{PO} \approx \frac{1}{2}$.

The depth ratio DR is useful as a measure of the efficiency of a tree searching algorithm relative to the minimax algorithm. However, the expected number of bottom positions examined by the alpha-beta algorithm provides a much tighter upper bound for the expected performance of a good tree-searching algorithm than does the number of bottom positions examined by the

minimax algorithm. Hence, it would be more desirable to redefine DR such that the performance of the alpha-beta algorithm rather than the minimax algorithm is used as the standard of comparison, i.e.,

$$DR^{*}_{x} \equiv \frac{\log \overline{NBP}_{x}}{\log \overline{NBP}_{\alpha\text{-}\beta}}$$

8. CONCLUSIONS

We have proven that the probability of examing a node $p(\vec{i}_k)$ in a game tree with α-β pruning is

$$Pr\{\vec{i}_k\} = -\int_{-\infty}^{\infty} A_{\underset{\sim}{i}_k}(z)\, d\bar{B}_{\underset{\sim}{i}_k}(z) = \int_{-\infty}^{\infty} \bar{B}_{\underset{\sim}{i}_k}(z)\, dA_{\underset{\sim}{i}_k}(z),$$

where $A_{\underset{\sim}{i}_k}(z)$ and $B_{\underset{\sim}{i}_k}(z)$ are the distribution functions of $\alpha(\vec{i}_k)$ and $\beta(\vec{i}_k)$ respectively. The integral may be evaluated exactly using Eqn. 5.6 and Lemma 5.3. This formula is used to compute the expected value of the number of bottom positions to be evaluated in a tree of arbitrary (but fixed) depth and branching factor.

For large values of D and N Lemma 5.3 is not computationally feasible and so we empirically fit the following curve to a set of simulation points:

$$E[NBP_{N,D}] \approx K_D N^{.720D+.277},$$

where K_D is a coefficient in the interval [1,2] depending on depth. This shows that the depth ratio as defined by Slagle and Dixon [1969] for α-β is about

$$DR_{\alpha\text{-}\beta} \approx .720 + \frac{.277}{D} \text{ for large N.}$$

We investigate the deficiencies of the model with respect to correlation and possible equality of bottom position values and demonstrate that higher correlation reduces the number of bottom positions, as does equality of bottom position values (Figures 6.2 and 6.3).

A number of interesting problems remain to be solved in the analysis of the alpha-beta algorithm: The simplification of the analytical expression to a form with a smaller computational growth rate than indicated by Lemma 5.3 is an open question. It would also be useful to find an analytical expression for the variance of the number of bottom positions.

Extension of the model to more general game trees would be an interesting goal. The range constraint may be relaxed, as shown in Section 5. Analysis of this model is considerably more complicated than the model we chose.

The present model is inadequate for modelling variations of the search strategy such as fixed ordering, dynamic ordering, and aspiration levels, because the model assumes independence of the underlying random variables. One possible extension of the model to encompass these requirements might be to assign a random number to each branch throughout the tree, and let the final evaluation of the bottom positions be the sum of the values of its predecessor. This would lead to an intuitively appealing static evaluation function that would introduce a measure of correlation into the tree.

APPENDIX: NOTATION

$a_k(\vec{i}_d)$	kth-level alpha value, where $k=0,2,\ldots,\lfloor d \rfloor_e$
$A_{k,i_{k+1}}(x)$	distribution function of $a_k(\vec{i}_d)$; $d > k$
$\underset{\sim}{A}_{\vec{i}_d}(x)$	distribution function of $\alpha(\vec{i}_d)$
$\alpha(\vec{i}_d)$	$\max\{a_0(\vec{i}_d), a_2(\vec{i}_d), \ldots, a(\vec{i}_{\lfloor d \rfloor_e})\}$
$b_k(\vec{i}_d)$	kth-level beta value, where $k=1,3,\ldots,\lfloor d \rfloor_o$
$B_{k,i_{k+1}}(x)$	distribution function of $b_k(\vec{i}_d)$; $d > k$
$\underset{\sim}{B}_{\vec{i}_d}(x)$	distribution function of $\beta(\vec{i}_d)$
$\beta(\vec{i}_d)$	$\min\{b_1(\vec{i}_d), b_3(\vec{i}_d), \ldots, b(\vec{i}_{\lfloor d \rfloor_e})\}$
$\beta(a,b)$	$\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$, the Beta function
D	depth of game tree
$\bar{F}(x)$	the survivor function, i.e., $\bar{F}(x) = 1-F(x)$
$\vec{i}_d$	$i_1, i_2, \ldots, i_d$. Let $\vec{i}_0$ be the empty vector.
$I_k(j_1, j_2, \ldots, j_k)$	$\int_0^1 \prod_{n=1}^{k} T^{j_n}(u, k-n+1)$
$\lfloor k \rfloor_e$	$2\left\lfloor \frac{k}{2} \right\rfloor$
$\lfloor k \rfloor_o$	$2\left\lfloor \frac{k-1}{2} \right\rfloor + 1$
N	branching factor of game tree
$p(\vec{i}_d)$	a node at level d in the game tree with index $\vec{i}_d$. Number of elements in vector indicates depth of node in the tree. For example, $p(\vec{i}_D)$ is a leaf node.

$P_k(m)$

$$\begin{cases} 0 & \text{if } i_m = 1 \\ i_{D-k} + N-2 & \text{if } k < D-m \\ i_{D-k} - 2 & \text{if } k = D-m \\ i_{D-k} - 1 & \text{if } k > D-m \end{cases}$$

$T(f,k)$

$$\begin{cases} f & \text{if } k = 0 \\ 1 - [T(f,k-1)]^N & \text{if } k > 0 \end{cases}$$

$v(\overleftarrow{i}_d)$ value of $p(\vec{i}_d)$. This is the backed-up value unless $d = D$, in which case $p(\vec{i}_d)$ is a leaf node and we use the leaf node's static value.

$V_d(x)$ cumulative distribution function of $v(\vec{i}_d)$

References

Bogen, R. A. et al. MACSYMA Users Manual, Project MAC, MIT (September 1972).

Gillogly, J. J., "The Technology Chess Program," Artificial Intelligence 3 (1972), 145-163.

Nilsson, N. J., Problem Solving Methods in Artificial Intelligence, McGraw-Hill (1971).

Parzen, E., Modern Probability Theory and Its Applications, Wiley and Sons, Inc., N. Y. (1960).

Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," IBM Journal 3, (1959), 211-229.

Shannon, C. E., "Programming a Computer for Playing Chess," Phil. Mag. 7th series, V. 41, N. 314 (1950), 256-275.

Slagle, J. R., "Game Trees, m and n Minimaxing, and the m and n Alpha-Beta Procedure," AI Group Rep. No. 3, UCRL-4671, Lawrence Radiation Laboratory, University of California (November, 1963).

Slagle, J. R. and J. K. Dixon, "Experiments with Some Programs that Search Game Trees," J.ACM 16,2 (April, 1969), 189-207.