An Analysis of Alpha-Beta Pruning[1]

Donald E. Knuth and Ronald W. Moore

Computer Science Department, Stanford University, Stanford, Calif. 94305, U.S.A.

Recommended by U. Montanari

ABSTRACT

The alpha-beta technique for searching game trees is analyzed, in an attempt to provide some insight into its behavior. The first portion of this paper is an expository presentation of the method together with a proof of its correctness and a historical discussion. The alpha-beta procedure is shown to be optimal in a certain sense, and bounds are obtained for its running time with various kinds of random data.

Put one pound of Alpha Beta Prunes
in a jar or dish that has a cover.
Pour one quart of boiling water over prunes.
The longer prunes soak, the plumper they get
Alpha Beta Acme Markets, Inc.,
La Habra, California

0. Introduction

Computer programs for playing games like chess typically choose their moves by searching a large tree of potential continuations. A technique called "alpha-beta pruning" is generally used to speed up such search processes without loss of information. The purpose of this paper is to analyze the alpha-beta procedure in order to obtain some quantitative estimates of its performance characteristics.

[1] This research was supported in part by the National Science Foundation under grant number GJ 36473X and by the Office of Naval Research under contract NR 044-402.

Artificial Intelligence 6 (1975), 293–326

Section 1 defines the basic concepts associated with game trees. Section 2 presents the alpha-beta method together with a related technique which is similar, but not as powerful, because it fails to make "deep cutoffs". The correctness of both methods is demonstrated, and Section 3 gives examples and further development of the algorithms. Several suggestions for applying the method in practice appear in Section 4, and the history of alpha-beta pruning is discussed in Section 5.

Section 6 begins the quantitative analysis, by deriving lower bounds on the amount of searching needed by alpha-beta and by *any* algorithm which solves the same general problem. Section 7 derives upper bounds, primarily by considering the case of random trees when no deep cutoffs are made. It is shown that the procedure is reasonably efficient even under these weak assumptions. Section 8 shows how to introduce some of the deep cutoffs into the analysis; and Section 9 shows that the efficiency improves when there are dependencies between successive moves. This paper is essentially self-contained, except for a few mathematical results quoted in the later sections.

1. Games and Position Values

The two-person games we are dealing with can be characterized by a set of "positions", and by a set of rules for moving from one position to another, the players moving alternately. We assume that no infinite sequence of positions is allowed by the rules,[2] and that there are only finitely many legal moves from every position. It follows from the "infinity lemma" (see [11, Section 2.3.4.3]) that for every position p there is a number $N(p)$ such that no game starting at p lasts longer than $N(p)$ moves.

If p is a position from which there are no legal moves, there is an integer-valued function $f(p)$ which represents the *value* of this position to the player whose turn it is to play from p; the value to the other player is assumed to be $-f(p)$.

If p is a position from which there are d legal moves $p_1, \ldots, p_d$, where $d > 1$, the problem is to choose the "best" move. We assume that the best move is one which achieves the greatest possible value when the game ends, if the opponent also chooses moves which are best for him. Let $F(p)$ be the greatest possible value achievable from position p against the optimal defensive strategy, from the standpoint of the player who is moving from that

[2] Strictly speaking, chess does not satisfy this condition, since its rules for repeated positions only give the players the *option* to request a draw, in certain circumstances; if neither player actually does ask for a draw, the game can go on forever. But this technicality is of no practical importance, since computer chess programs only look finitely many moves ahead. It is possible to deal with infinite games by assigning appropriate values to repeated positions, but such questions are beyond the scope of this paper.

position. Since the value (to this player) after moving to position p_i will be $-F(p_i)$, we have

$$F(p) = \begin{cases} f(p) & \text{if } d = 0, \\ \max(-F(p_1), \ldots, -F(p_d)) & \text{if } d > 0. \end{cases} \tag{1}$$

This formula serves to define $F(p)$ for all positions p, by induction on the length of the longest game playable from p.

In most discussions of game-playing, a slightly different formalism is used; the two players are named Max and Min, where all values are given from Max's viewpoint. Thus, if p is a terminal position with Max to move, its value is $f(p)$ as before, but if p is a terminal position with Min to move its value is

$$g(p) = -f(p). \tag{2}$$

Max will try to maximize the final value, and Min will try to minimize it. There are now two functions corresponding to (1), namely

$$F(p) \text{ V} = \begin{cases} f(p) & \text{if } d = 0, \\ \max(G(p_1), \ldots, G(p_d)) & \text{if } d > 0, \end{cases} \tag{3}$$

which is the best value Max can guarantee starting at position p, and

$$G(p) = \begin{cases} g(p) & \text{if } d = 0, \\ \min(F(p_1), \ldots, F(p_d)) & \text{if } d > 0, \end{cases} \tag{4}$$

which is the best that Min can be sure of achieving. As before, we assume that $p_1, \ldots, p_d$ are the legal moves from position p. It is easy to prove by induction that the two definitions of F in (1) and (3) are identical, and that

$$G(p) = -F(p) \tag{5}$$

for all p. Thus the two approaches are equivalent.

Sometimes it is easier to reason about game-playing by using the "minimax" framework of (3) and (4) instead of the "negmax" approach of eq. (1); the reason is that we are sometimes less confused if we consistently evaluate the game positions from one player's standpoint. On the other hand, formulation (1) is advantageous when we're trying to prove things about games, because we don't have to deal with two (or sometimes even four or eight) separate cases when we want to establish our results. Eq. (1) is analogous to the "NOR" operation which arises in circuit design; two levels of NOR logic are equivalent to a level of ANDs followed by a level of ORs.

The function $F(p)$ is the maximum final value that can be achieved if both players play optimally; but we should remark that this reflects a rather conservative strategy that won't always be best against poor players or against the nonoptimal players we encounter in the real world. For example, suppose that there are two moves, to positions p_1 and p_2, where p_1 assures a draw (value 0) but cannot possibly win, while p_2 give a chance of either victory or defeat depending on whether or not the opponent overlooks a

rather subtle winning move. We may be better off gambling on the move to p_2, which is our only chance to win, unless we are convinced of our opponent's competence. Indeed, humans seem to beat chess-playing programs by adopting such a strategy.

2. Development of the Algorithm

The following algorithm (expressed in an ad-hoc ALGOL-like language) clearly computes $F(p)$, by following definition (1):

```
integer procedure F (position p):
  begin integer m, i, t, d;
    determine the successor positions p_1, ..., p_d;
    if d = 0 then F := f(p) else
    begin m := -∞;
      for i := 1 step 1 until d do
        begin t := -F(p_i);
          if t > m then m := t;
        end;
      F := m;
    end;
  end.
```


Here ∞ denotes a value that is greater than or equal to $|f(p)|$ for all terminal positions of the game, hence $-\infty$ is less than or equal to $\pm F(p)$ for all p. This algorithm is a "brute force" search through all possible continuations; the infinity lemma assures us that the algorithm will terminate in finitely many steps.

It is possible to improve on the brute-force search by using a "branch-and-bound" technique [14], ignoring moves which are incapable of being better than moves which are already known. For example, if $F(p_1) = -10$, then $F(p) \geq 10$, and we don't have to know the exact value of $F(p_2)$ if we can deduce that $F(p_2) \geq -10$ (i.e., that $-F(p_2) \leq 10$). Thus if p_{21} is a legal move from p_2 such that $F(p_{21}) \leq 10$, we need not bother to explore any other moves from p_2. In game-playing terminology, a move to p_2 can be "refuted" (relative to the alternative move p_1) if the opposing player can make a reply to p_2 that is at least as good as his best reply to p_1. Once a move has been refuted, we need not search for the best possible refutation.

This line of reasoning leads to a computational technique that avoids much of the computation done by F. We shall define $F1$ as a procedure on two parameters p and *bound*, and our goal is to achieve the following conditions:

$$\begin{aligned} F1(p, bound) &= F(p) \quad \text{if } F(p) < bound, \\ F1(p, bound) &\geq bound \quad \text{if } F(p) \geq bound. \end{aligned} \tag{6}$$

These relations do not fully define $F1$, but they are sufficiently powerful to calculate $F(p)$ for any starting position p because they imply that

$$F1(p, \infty) = F(p). \tag{7}$$

The following algorithm corresponds to this branch-and-bound idea.

```
integer procedure F1 (position p, integer bound):
  begin integer m, i, t, d;
    determine the successor positions p_1, ..., p_d;
    if d = 0 then F1 := f(p) else
    begin m := -∞;
      for i := 1 step 1 until d do
        begin t := -F1(p_i, -m);
          if t > m then m := t;
          if m ≥ bound then go to done;
        end;
    done: F1 := m;
    end;
  end
```


We can prove that this procedure satisfies (6) by arguing as follows: At the beginning of the ith iteration of the **for** loop, we have the "invariant" condition

$$m = \max(-F(p_1), \ldots, -F(p_{i-1})) \tag{8}$$

just as in procedure F. (The max operation over an empty set is conventionally defined to be $-\infty$.) For if $-F(p_i)$ is $> m$, then $F1(p_i, -m) = F(p_i)$, by condition (6) and induction on the length of the game following p; therefore (8) will hold on the next iteration. And if $\max(-F(p_1), \ldots, -F(p_i)) \geqslant bound$ for any i, then $F(p) \geqslant bound$. It follows that condition (6) holds for all p.

The procedure can be improved further if we introduce both lower and upper bounds; this idea, which is called *alpha-beta pruning*, is a significant extension to the one-sided branch-and-bound method. (Unfortunately it doesn't apply to all branch-and-bound algorithms, it works only when a game tree is being explored.) We define a procedure $F2$ of three parameters p, *alpha*, and *beta*, for *alpha* $<$ *beta*, satisfying the following conditions analogous to (6):

$$\begin{aligned} &F2(p, alpha, beta) \leqslant alpha && \text{if } F(p) \leqslant alpha, \\ &F2(p, alpha, beta) = F(p) && \text{if } alpha < F(p) < beta, \\ &F2(p, alpha, beta) \geqslant beta && \text{if } F(p) \geqslant beta. \end{aligned} \tag{9}$$

Again, these conditions do not fully specify $F2$, but they imply that

$$F2(p, -\infty, \infty) = F(p). \tag{10}$$

It turns out that this improved algorithm looks only a little different from the others, when it is expressed in a programming language:

```
integer procedure F2 (position p, integer alpha, integer beta):
  begin integer m, i, t, d;
    determine the successor positions p_1, . . ., p_d;
    if d = 0 then F2 := f(p) else
    begin m := alpha;
      for i := 1 step 1 until d do
        begin t := −F2(p_i, −beta, −m);
          if t > m then m := t;
          if m ⩾ beta then go to done;
        end;
    done: F2 := m;
    end;
  end;
```

To prove the validity of $F2$, we proceed as we did with $F1$. The invariant relation analogous to (8) is now

$$m = \max(alpha, -F(p_1), \ldots, -F(p_{i-1})) \tag{11}$$

and $m < beta$. If $-F(p_i) \geqslant beta$, then $-F2(p_i, -beta, -m)$ will also be $\geqslant beta$, and if $m < -F(p_i) < beta$, then $-F2(p_i, -beta, -m) = -F(p_i)$; so the proof goes through as before, establishing (9) by induction.

Now that we have found two improvements of the minimax procedure, it is natural to ask whether still further improvement is possible. Is there an "alpha-beta-gamma" procedure $F3$, which makes use say of the second-largest value found so far, or some other gimmick? Section 6 below shows that the answer is no, or at least that there is a reasonable sense in which procedure $F2$ is optimum.

3. Examples and Refinements

As an example of these procedures, consider the tree in Fig. 1, which represents a position that has three successors, each of which has three successors, etc., until we get to $3^4 = 81$ positions possible after four moves; and these 81 positions have been assigned "random" f values according to the first 81 digits of π. Fig. 1 shows the F values computed from the f's; thus, the root node at the top of the tree has an effective value of 2 after best play by both sides.

Fig. 2 shows the same situation as it is evaluated by procedure $F1(p, \infty)$. Note that only 36 of the 81 terminal positions are examined, and that one of the nodes at level 2 now has the "approximate" value 3 instead of its true value 7; but this approximation does not of course affect the value at the top.

Fig. 3 shows the same situation as it is evaluated by the full alpha-beta pruning procedure. $F2(p, -\infty, +\infty)$ will always examine the same nodes as $F1(p, \infty)$ until the fourth level of lookahead is reached, in any game tree;

this is a consequence of the theory developed below. On levels 4, 5, . . ., however, procedure *F*2 is occasionally able to make "deep cutoffs" which *F*1 is incapable of finding. A comparison of Fig. 3 with Fig. 2 shows that there are five deep cutoffs in this example.

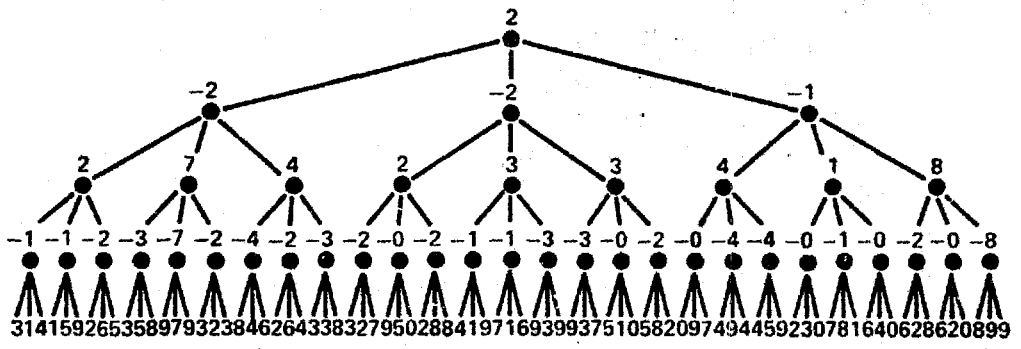

FIG. 1. Complete evaluation of a game tree.

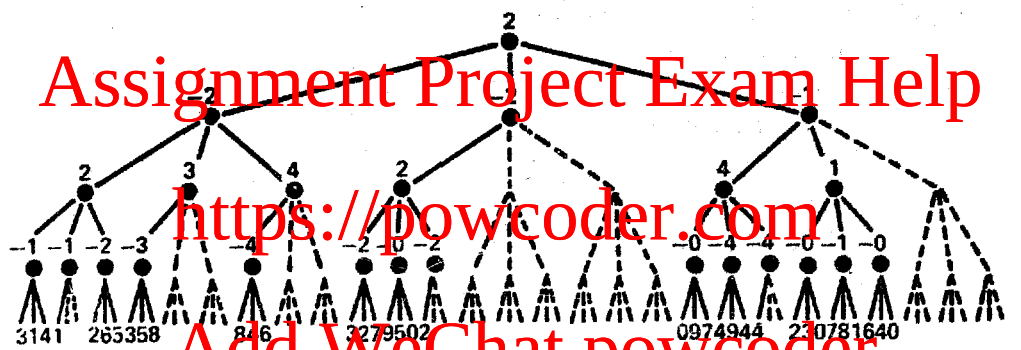

FIG. 2. The game tree of Fig. 1 evaluated with procedure *F*1 (branch-and-bound strategy).

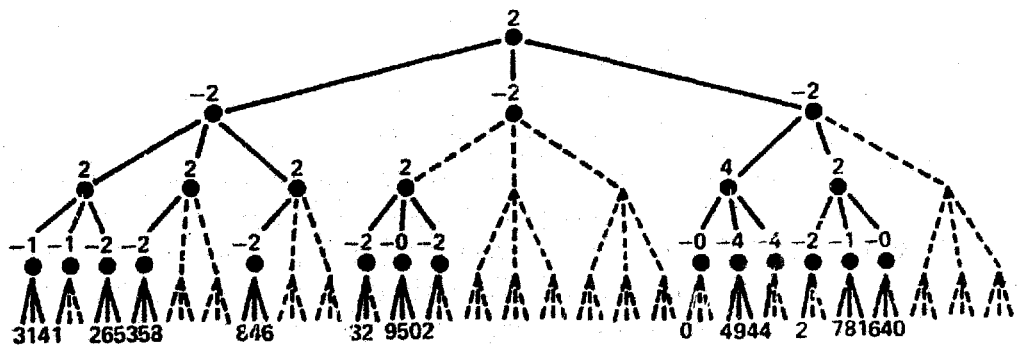

FIG. 3. The game tree of Fig. 1 evaluated with procedure *F*2 (alpha-beta strategy).

All of these illustrations present the results in terms of the "negamax" model of Section 1; if the reader prefers to see it in "minimax" terms, it is sufficient to ignore all the minus signs in Figs. 1–3. The procedures of Section 2 can readily be converted to the minimax conventions, for example by replacing *F*2 by the following two procedures:

```
integer procedure F2 (position p, integer alpha, integer beta):
  begin integer m, i, t, d;
    determine the successor positions p_1, . . ., p_d;
    if d = 0 then F2 := f(p) else
    begin m := alpha;
      for i := 1 step 1 until d do
        begin t := G2(p_i, m, beta);
          if t > m then m := t;
          if m ≥ beta then go to done;
        end;
    done: F2 := m;
    end;
  end;
integer procedure G2 (position p, integer alpha, integer beta);
  begin integer m, i, t, d;
    determine the successor positions p_1, . . ., p_d;
    if d = 0 then G2 := g(p) else
    begin m := beta;
      for i := 1 step 1 until d do
        begin t := F2(p_i, alpha, m);
          if t < m then m := t;
          if m ≤ alpha then go to done;
        end;
    done: F2 := m;
    end;
  end.
```


It is a simple but instructive exercise to prove that $G2(p, alpha, beta)$ always equals $-F2(p, -beta, -alpha)$.

The above procedures have made use of a magic routine that determines the successors $p_1, \ldots, p_d$ of a given position p. If we want to be more explicit about how positions are represented, it is natural to use the format of linked records: When p is a reference to a record denoting a position, let $first(p)$ be a reference to the first successor of that position, or Λ (a null reference) if the position is terminal. Similarly if q references a successor p_i of p, let $next(q)$ be a reference to the next successor p_{i+1}, or Λ if $i = d$. Finally let $generate(p)$ be a procedure that creates the records for $p_1, \ldots, p_d$, sets their *next* fields, and makes $first(p)$ point to p_1 (or to Λ if $d = 0$). Then the alpha-beta pruning method takes the following more explicit form.

```
integer procedure F2 (ref (position) p, integer alpha, integer beta):
  begin integer m, t; ref (position) q;
    generate(p);
    q := first(p);
```

```
    if q = Λ then F2 := f(p) else
    begin m := alpha;
      while q ≠ Λ and m < beta do
        begin t := −F2(q, −beta, −m);
          if t > m then m := t;
          q := next(q);
        end;
      F2 := m;
    end;
  end.
```

It is interesting to convert this recursive procedure to an iterative (non-recursive) form by a sequence of mechanical transformations, and to apply simple optimizations which preserve program correctness (see [13]). The resulting procedure is surprisingly simple, but not as easy to prove correct as the recursive form:

```
integer procedure alphabeta (ref (position) p);
  begin integer l; comment level of recursion;
    integer array a[−2:L]; comment stack for recursion, where
      a[l − 2], a[l − 1], a[l], a[l + 1] denote respectively
      alpha, −beta, m, −t in procedure F2;
    ref (position) array r[0:L + 1]; comment another stack for
      recursion, where r[l] and r[l + 1] denote respectively
      p and q in F2;
    l := 0; a[−2] := −∞; a[−1] := +∞; r[0] := p;
  F2: generate (r[l]);
    r[l + 1] := first(r[l]);
    if r[l + 1] = Λ then a[l] := f(r[l]) else
    begin a[l] := a[l − 2];
      loop: l := l + 1; go to F2;
      resume: if −a[l + 1] > a[l] then
          begin a[l] := −a[l + 1];
            if a[l + 1] ⩽ a[l − 1] then go to done;
          end;
        r[l + 1] := next(r[l + 1]);
        if r[l + 1] ≠ Λ then go to loop;
    end;
  done: l := l − 1; if l ⩾ 0 then go to resume;
  alphabeta := a[0];
end.
```


This procedure *alphabeta*(*p*) will compute the same value as $F2(p, -\infty, +\infty)$; we must choose L large enough so that the level of recursion never exceeds L.

4. Applications

When a computer is playing a complex game, it will rarely be able to search all possibilities until truly terminal positions are reached; even the alpha-beta technique won't be fast enough to solve the game of chess! But we can still use the above procedures, if the routine that generates all moves is modified so that sufficiently deep positions are considered to be terminal. For example, if we wish to look six moves ahead (three for each player), we can pretend that the positions reached at level 6 have no successors. To compute f at such artificially-terminal positions, we must of course use our best guess about the value, hoping that a sufficiently deep search will ameliorate the inaccuracy of our guess. (Most of the time will be spent in evaluating these guessed values for f, unless the determination of legal moves is especially difficult, so some quickly-computed estimate is needed.)

Instead of searching to a fixed depth, it is also possible to carry some lines further, e.g., to play out all sequences of captures. An interesting approach was suggested by Floyd in 1965 [6]), although it has apparently not yet been tried in large-scale experiments. Each move in Floyd's scheme is assigned a "likelihood" according to the following general plan: A forced move has "likelihood" of 1, while very implausible moves (like queen sacrifices in chess) get 0.01 or so. In chess a "recapture" has "likelihood" greater than $\frac{1}{2}$; and the best strategic choice out of 20 or 30 possibilities gets a "likelihood" of about 0.1, while the worst choices get say 0.02. When the product of all "likelihoods" leading to a position becomes less than a given threshold (say 10^{-8}), we consider that position to be terminal and estimate its value without further searching. Under this scheme, the "most likely" branches of the tree are given the most attention.

Whatever method is used to produce a tree of reasonable size, the alpha-beta procedure can be somewhat improved if we have an idea what the value of the initial position will be. Instead of calling $F2(p, -\infty, +\infty)$, we can try $F2(p, a, b)$ where we expect the value to be greater than a and less than b. For example, if $F2(p, 0, 4)$ is used instead of $F2(p, -10, +10)$ in Fig. 3, the rightmost "−4" on level 3, and the "4" below it, do not need to be considered. If our expectation is fulfilled, we may have pruned off more of the tree; on the other hand if the value turns out to be low, say $F2(p, a, b) = v$, where $v \leqslant a$, we can use $F2(p, -\infty, v)$ to deduce the correct value. This idea has been used in some versions of Greenblatt's chess program [8].

5. History

Before we begin to make quantitative analyses of alpha-beta's effectiveness, let us look briefly at its historical development. The early history is somewhat obscure, because it is based on undocumented recollections and because

some people have confused procedure *F*1 with the stronger procedure *F*2; therefore the following account is based on the best information now available to the authors.

McCarthy [15] thought of the method during the Dartmouth Summer Research Conference on Artificial Intelligence in 1956, when Bernstein described an early chess program [3] which didn't use any sort of alpha-beta. McCarthy "criticized it on the spot for this [reason], but Bernstein was not convinced. No formal specification of the algorithm was given at that time." It is plausible that McCarthy's remarks at that conference led to the use of alpha-beta pruning in game-playing programs of the late 1950s. Samuel has stated that the idea was present in his checker-playing programs, but he did not allude to it in his classic article [21] because he felt that the other aspects of his program were more significant.

The first published discussion of a method for game tree pruning appeared in Newell, Shaw and Simon's description [16] of their early chess program. However, they illustrate only the "one-sided" technique used in procedure *F*1 above, so it is not clear whether they made use of "deep cutoffs".

McCarthy coined the name "alpha-beta" when he first wrote a LISP program embodying the technique. His original approach was somewhat more elaborate than the method described above, since he assumed the existence of two functions "*optimistic value*(*p*)" and "*pessimistic value*(*p*)" which were to be upper and lower bounds on the value of a position. McCarthy's form of alpha-beta searching was equivalent to replacing the above body of procedure *F*2 by

if *optimistic value*(*p*) $\leqslant$ *alpha* **then** *F*2 := *alpha*
else if *pessimistic value*(*p*) $\geqslant$ *beta* **then** *F*2 := *beta*
else begin ⟨the above body of procedure *F*2⟩ **end**.

Because of this elaboration, he thought of alpha-beta as a (possibly inaccurate) heuristic device, not realizing that it would also produce the same value as full minimaxing in the special case that *optimistic value*(*p*) = $+\infty$ and *pessimistic value*(*p*) = $-\infty$ for all *p*. He credits the latter discovery to Hart and Edwards, who wrote a memorandum [10] on the subject in 1961. Their unpublished memorandum gives examples of the general method, including deep cutoffs; but (as usual in 1961) no attempt was made to indicate why the method worked, much less to demonstrate its validity.

The first published account of alpha-beta pruning actually appeared in Russia, quite independently of the American work. Brudno, who was one of the developers of an early Russian chess-playing program, described an algorithm identical to alpha-beta pruning, together with a rather complicated proof, in 1963 (see [4]).

The full alpha-beta pruning technique finally appeared in "Western"

computer-science literature in 1968, within an article on theorem-proving strategies by Slagle and Bursky [24], but their description was somewhat vague and they did not illustrate deep cutoffs. Thus we might say that the first real English descriptions of the method appeared in 1969, in articles by Slagle and Dixon [25] and by Samuel [22]; both of these articles clearly mention the possibility of deep cutoffs, and discuss the idea in some detail.

The alpha-beta technique seems to be quite difficult to communicate verbally, or in conventional mathematical language, and the authors of the papers cited above had to resort to rather complicated descriptions; furthermore, considerable thought seems to be required at first exposure to convince oneself that the method is correct, especially when it has been described in ordinary language and "deep cutoffs" must be justified. Perhaps this is why many years went by before the technique was published. However, we have seen in Section 2 that the method is easily understood and proved correct when it has been expressed in algorithmic language; this makes a good illustration of a case where a "dynamic" approach to process description is conceptually superior to the "static" approach of conventional mathematics.

Excellent presentations of the method appear in the recent textbooks by Nilsson [18, Section 4] and Slagle [23, pp. 16–24], but in prose style instead of the easier-to-understand algorithmic form. Alpha-beta pruning has become "well known"; yet to the authors' knowledge only two published descriptions have heretofore been expressed in an algorithmic language. In fact the first of these, by Wells [27, Section 4.3.3], isn't really the full alpha-beta procedure, it isn't even as strong as procedure *F*1. (Not only is his algorithm incapable of making deep cutoffs, it makes shallow cutoffs only on strict inequality.) The other published algorithm, by Dahl and Belsnes [5, Section 8.1], appears in a recent Norwegian-language textbook on data structures; however, the alpha-beta method is presented using label parameters, so the corresponding proof of correctness becomes somewhat difficult. Another recent textbook [17, Section 3.3.1] contains an informal description of what is called "alpha-beta pruning", but again only the method of procedure *F*1 is given; apparently many people are unaware that the alpha-beta procedure is capable of making deep cutoffs.[3] For these reasons, the authors of the present paper do not feel it redundant to present a new expository account of the method, even though alpha-beta pruning has been in use for more than 15 years.

[3] Indeed, one of the authors of the present paper (D.E.K.) did some of the research described in Section 7 approximately five years before he was aware that deep cutoffs were possible. It is easy to understand procedure *F*1 and to associate it with the term "alpha-beta pruning" your colleagues are talking about, without discovering *F*2.

6. Analysis of the Best Case

Now let us turn to a quantitative study of the algorithm. How much of the tree needs to be examined?

For this purpose it is convenient to assign coordinate numbers to the nodes of the tree as in the "Dewey decimal system" [11, p. 310]: Every position on level l is assigned a sequence of positive integers $a_1\, a_2 \ldots a_l$. The root node (the starting position) corresponds to the empty sequence, and the d successors of position $a_1 \ldots a_l$ are assigned the respective coordinates $a_1 \ldots a_l 1$, $\ldots$, $a_1 \ldots a_l d$. Thus, position 314 is reached after making the third possible move from the starting position, then the first move from that position, and then the fourth.

Let us call position $a_1 \ldots a_l$ *critical* if $a_i = 1$ for all even values of i or for all odd values of i. Thus, positions 21412, 131512, 11121113, and 11 are critical, and the root position is always critical; but 12112 is not, since it has non-1's in both even and odd positions. The relevance of this concept is due to the following theorem, which characterizes the action of alpha-beta pruning when we are lucky enough to consider the best move first from every position.

THEOREM 1. *Consider a game tree for which the value of the root position is not* $\pm\infty$, *and for which the first successor of every position is optimum; i.e.,*

$$F(a_1 \ldots a_l) = \begin{cases} f(a_1 \ldots a_l) & \text{if } a_1 \ldots a_l \text{ is terminal,} \\ -F(a_1 \ldots a_l 1) & \text{otherwise.} \end{cases} \tag{12}$$

The alpha-beta procedure F2 examines precisely the critical positions of this game tree.

Proof. Let us say that a critical position $a_1 \ldots a_l$ is of type 1 if all the a_i are 1; it is of type 2 if a_j is its first entry >1 and $l - j$ is even; otherwise (i.e., when $l - j$ is odd, hence $a_l = 1$) it is of type 3. It is easy to establish the following facts by induction on the computation, i.e., by showing that they are invariant assertions:

(1) A type 1 position p is examined by calling $F2(p, -\infty, +\infty)$. If it is not terminal, its successor position p_1 is of type 1, and $F(p) = -F(p_1) \neq \pm\infty$. The other successor positions $p_2, \ldots, p_d$ are of type 2, and they are all examined by calling $F2(p_i, -\infty, F(p_1))$.

(2) A type 2 position p is examined by calling $F2(p, -\infty, beta)$, where $-\infty < beta \leqslant F(p)$. If it is not terminal, its successor position p_1 is of type 3, and $F(p) = -F(p_1)$; hence, by the mechanism of procedure $F2$ as defined in Section 2, the other successors $p_2, \ldots, p_d$ are not examined.

(3) A type 3 position p is examined by calling $F2(p, alpha, +\infty)$ where $+\infty > alpha \geqslant F(p)$. If it is not terminal, each of its successor positions p_i is of type 2 and they are all examined by calling $F2(p_i, -\infty, -alpha)$.

It follows by induction on l that every critical position is examined.

COROLLARY 1. *If every position on levels* $0, 1, \ldots, l-1$ *of a game tree satisfying the conditions of Theorem* 1 *has exactly* d *successors, for some fixed constant* d, *then the alpha-beta procedure examines exactly*

$$d^{\lceil l/2 \rceil} + d^{\lfloor l/2 \rfloor} - 1 \tag{13}$$

positions on level l.

Proof. There are $d^{\lceil l/2 \rceil}$ sequences $a_1 \ldots a_l$, with $1 \leqslant a_i \leqslant d$ for all i, such that $a_i = 1$ for all odd values of i; there are $d^{\lfloor l/2 \rfloor}$ such sequences with $a_i = 1$ for all even values of i; and we subtract 1 for the sequence $1 \ldots 1$ which was counted twice.

This corollary was first derived by Levin in 1961, but no proof was apparently ever written down at the time. In fact, the informal memo [10] by Hart and Edwards justifies the result by saying: "For a convincing personal proof using the new heuristic hand waving technique, see the author of this theorem." A proof was later published by Slagle and Dixon [25]. However, none of these authors pointed out that the value of the root position must not equal $\pm\infty$. Although this is a rare occurrence in nontrivial games, since it means that the root position is a forced win or loss, it is a necessary hypothesis for both the theorem and the corollary, since the number of positions examined on level l will be $d^{\lceil l/2 \rceil}$ when the root value is $+\infty$, and it will be $d^{\lfloor l/2 \rfloor}$ when the root value is $-\infty$. Roughly speaking, we gain a factor of 2 when the root value is $\pm\infty$.

The characterization of perfect alpha-beta pruning in terms of critical positions allows us to extend Corollary 1 to a much more general class of game trees, having any desired probability distribution of legal moves on each level.

COROLLARY 2. *Let a random game tree be generated in such a way that each position on level* j *has probability* q_j *of being nonterminal, and has an average of* d_j *successors. Then the expected number of positions on level* l *is* $d_0 d_1 \ldots d_{l-1}$; *and the expected number of positions on level* l *examined by the alpha-beta technique under the assumptions of Theorem* 1 *is*

$$\begin{aligned} &d_0q_1d_2q_3 \ldots d_{l-2}q_{l-1} + q_0d_1q_2d_3 \ldots q_{l-2}d_{l-1} - q_0q_1 \ldots q_{l-1} \quad l \text{ even}; \\ &d_0q_1d_2q_3 \ldots q_{l-2}d_{l-1} + q_0d_1q_2d_3 \ldots d_{l-2}q_{l-1} - q_0q_1 \ldots q_{l-1} \quad l \text{ odd}. \end{aligned} \tag{14}$$

(More precisely, the assumptions underlying this random branching process are that level $j+1$ of the tree is formed from level j as follows: Each position p on level j is assigned a probability distribution $\langle r_0(p), r_1(p), \ldots \rangle$, where $r_d(p)$ is the probability that p will have d successors; these distributions may be different for different positions p, but each must satisfy $r_0(p) = 1 - q_j$, and each must have the mean value $r_1(p) + 2r_2(p) + \cdots$

$= d_j$. The number of successor positions for p is chosen at random from this distribution, independently of the number of successors of other positions on level j.)

Proof. If x is the expected number of positions of a certain type on level j, then xd_j is the expected number of successors of these positions, and xq_j is the expected number of "number 1" successors. It follows as in Corollary 1 that (14) is the expected number of critical positions on level l; for example, $q_0 q_1 \cdots q_{l-1}$ is the expected number of positions on level l whose identifying coordinates are all 1's.

Note that (14) reduces to (13) when $q_j = 1$ and $d_j = d$ for $0 \leqslant j < l$.

Intuitively we might think that alpha-beta pruning would be most effective when perfect-ordering assumption (12) holds; i.e., when the first successor of every position is the best possible move. But this is not always the case: Fig. 4 shows two game trees which are identical except for the left-to-right ordering of successor positions; alpha-beta search will investigate more of the left-hand tree than the right-hand tree, although the left-hand tree has its positions perfectly ordered at every branch.

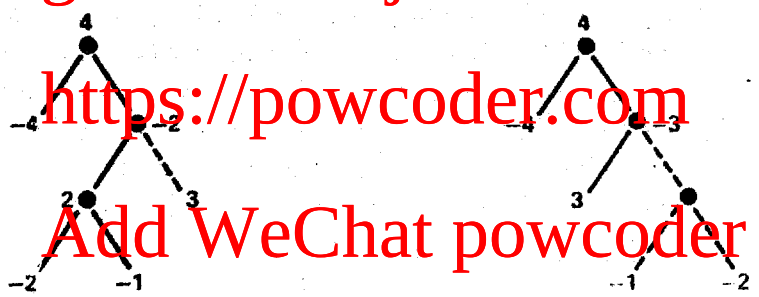

FIG. 4. Perfect ordering is not always best.

Thus the truly optimum order of game trees traversal isn't obvious. On the other hand it is possible to show that there always exists an order for processing the tree so that alpha-beta examines as few of the terminal positions as possible; no algorithm can do better. This can be demonstrated by strengthening the technique used to prove Theorem 1, as we shall see.

THEOREM 2. *Alpha-beta pruning is optimum in the following sense: Given any game tree and any algorithm which computes the value of the root position, there is a way to permute the tree (by reordering successor positions if necessary) so that every terminal position examined by the alpha-beta method under this permutation is examined by the given algorithm. Furthermore if the value of the root is not* $\pm\infty$, *the alpha-beta procedure examines precisely the positions which are critical under this permutation.*

(It is assumed that all terminal positions have independent values, or

equivalently that the algorithm has no knowledge about dependencies between the values of terminal positions.)

An equivalent result has been obtained by G. M. Adelson-Velskiy [1, Appendix 1]; a somewhat simpler proof will be presented here.

Proof. The following functions F_l and F_u define the best possible bounds on the value of any position p, based on the terminal positions examined by the given algorithm:

$$F_l(p) = \begin{cases} -\infty & \text{if } p \text{ is terminal and not examined,} \\ f(p) & \text{if } p \text{ is terminal and examined,} \\ \max(-F_u(p_1), \ldots, -F_u(p_d)) & \text{otherwise;} \end{cases} \tag{15}$$

$$F_u(p) = \begin{cases} +\infty & \text{if } p \text{ is terminal and not examined,} \\ f(p) & \text{if } p \text{ is terminal and examined,} \\ \max(-F_l(p), \ldots, -F_l(p_d)) & \text{otherwise.} \end{cases} \tag{16}$$

Note that $F_l(p) \leqslant F_u(p)$ for all p. By independently varying the values at unexamined terminal positions below p, we can make $F(p)$ assume any given value between $F_l(p)$ and $F_u(p)$, but we can never go beyond these limits. When p is the root position we must therefore have $F_l(p) = F_u(p) = F(p)$.

Assume that the root value is not $\pm\infty$. We will show how to permute the tree so that every critical terminal position (according to the new numbering of positions) is examined by the given algorithm, and that precisely the critical positions are examined by the alpha-beta procedure $F2$. The critical positions will be classified as type 1, 2, or 3 as in the proof of Theorem 1, the root being type 1. The following facts can be proved by induction:

(1) A type 1 position p has $F_l(p) = F_u(p) = F(p) \neq \pm\infty$, and it is examined during the alpha-beta procedure by calling $F2(p, -\infty, +\infty)$. If p is terminal, it must be examined by the given algorithm, since $F_l(p) \neq -\infty$. If it is not terminal, let j and k be such that $F_l(p) = -F_u(p_j)$ and $F_u(p) = -F_l(p_k)$. Then by (15) and (16) we have

$$F_l(p_k) \leqslant F_l(p_j) \leqslant F_u(p_j) = -F(p) = F_l(p_k),$$

hence $F_l(p_j) = F_l(p_k)$ and we may assume that $j = k$. By permuting the successor positions we may assume in fact that $j = k = 1$. Position p_1 (after permutation) is of type 1; the other successor positions $p_2, \ldots, p_d$ are of type 2, and they are all examined by calling $F2(p_1, -\infty, -F(p_1))$.

(2) A type 2 position p has $F_l(p) > -\infty$, and it is examined during the alpha-beta procedure by calling $F2(p, -\infty, beta)$, where $-\infty < beta \leqslant F_l(p)$. If p is terminal, it must be examined by the given algorithm. Otherwise let j be such that $F_l(p) = -F_u(p_j)$, and permute the successor positions if necessary so that $j = 1$. Position p_1 (after permutation) is of type 3 and is examined by calling $F2(p_1, -beta, +\infty)$. Since $F_u(p_1) = -F_l(p) \leqslant -beta$, this call returns a value $\leqslant -beta$; hence the other successors $p_2, \ldots, p_d$

(which are not critical positions) are not examined by the alpha-beta method, nor are their descendants.

(3) A type 3 position p has $F_u(p) < \infty$, and it is examined during the alpha-beta procedure by calling $F2(p, alpha, +\infty)$, where $F_u(p) \leqslant alpha < \infty$. If p is terminal, it must be examined by the given algorithm. Otherwise all its successor positions p_i are of type 2, and they are all examined by calling $F2(p_i, -\infty, -alpha)$. (There is no need to permute them, the ordering makes absolutely no difference here.)

A similar argument can be given when the root value is $+\infty$ (treating it as a type 2 position) or $-\infty$ (type 3).

A surprising corollary of this proof is that *the ordering of successors to type 3 positions in an optimally-ordered tree has absolutely no effect on the behavior of alpha-beta pruning*. Type 1 positions constitute the so-called "principal variation", corresponding to the best strategy by both players. The alternative responses to moves on the principal variation are of type 2. Type 3 positions occur when the best move is made from a type 2 position, and the successors of type 3 positions are again of type 2. Hence about half of the critical positions of a perfectly ordered game tree are of type 3, and current game-playing algorithms are probably wasting nearly half of the time they now spend trying to put successor moves in order.

Let us say that a game tree is *uniform* of degree d and height h if every position on levels $0, 1, \ldots, h-1$ has exactly d successors, and if every position on level h is terminal. For example, Fig. 1 is a uniform tree of height 4 and degree 3, but the trees of Fig. 4 are not uniform. Since all permutations of a uniform tree are uniform, Theorem 2 implies the following generalization of Corollary 1.

COROLLARY 3. *Any algorithm which evaluates a uniform game tree of height h and degree d must evaluate at least*

$$d^{\lceil h/2 \rceil} + d^{\lfloor h/2 \rfloor} - 1 \tag{17}$$

terminal positions. The alpha-beta procedure achieves this lower bound, if the best move is considered first at each position of types 1 *and* 2.

7. Uniform Trees Without Deep Cutoffs

Now that we have determined the best case of alpha-beta pruning, let's be more pessimistic and try to look at the worst that can happen. Given any finite tree, it is possible to find a sequence of values for the terminal positions so that the alpha-beta procedure will examine every node of the tree, without making any cutoffs unless the tree branches are permuted. (To see this, arrange the values so that whenever $F2(p, alpha, beta)$ is called, the condition $-alpha > F(p_1) > F(p_2) > \ldots > F(p_d) > -beta$ is satisfied.) On the other

hand, there are game trees with distinct terminal values for which the alpha-beta procedure will always find some cutoffs no matter how the branches are permuted, as shown in Fig. 5. (Procedure *F*1 does not enjoy this property.)

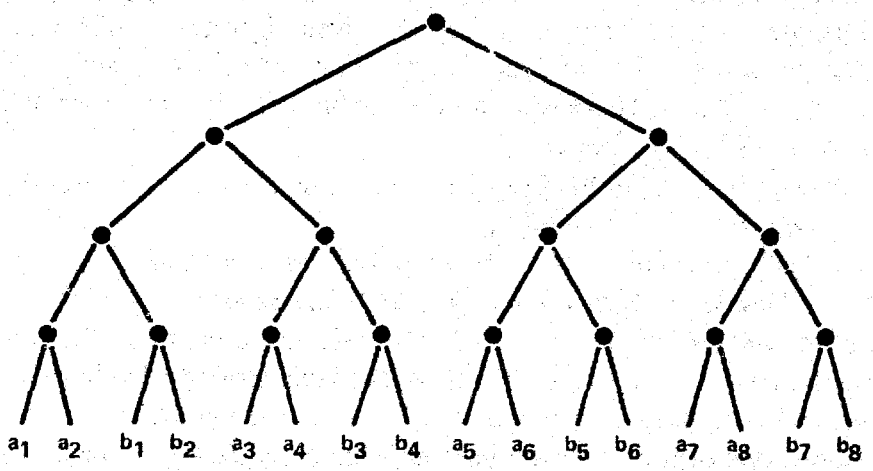

FIG. 5. If $\max(a_1, \ldots, a_8) < \min(b_1, \ldots, b_8)$, the alpha-beta procedure will always find at least two cutoffs, no matter how we permute the branches of this game tree.

Since game-playing programs usually use some sort of ordering strategy in connection with alpha-beta pruning, these facts about the worst case are of little or no practical significance. A more useful upper bound relevant to the behavior we may expect in practice can be based on the assumption of *random* data. Fuller, Gaschnig and Gillogly have recently undertaken a study [7] of the average number of terminal positions examined when the alpha-beta procedure is applied to a uniform tree of degree d and height h, giving independent random values to the terminal positions on level h. They have obtained formulas by which this average number can be computed, in roughly d^h steps, and their theoretically-predicted results were only slightly higher than empirically-observed data obtained from a modified chess-playing program. Unfortunately the formulas turn out to be extremely complicated, even for this reasonably simple theoretical model, so that the asymptotic behavior for large d and/or h seems to defy analysis.

Since we are looking for upper bounds anyway, it is natural to consider the behavior of the weaker procedure *F*1. This method is weaker since it doesn't find any "deep cutoffs"; but it is much better than complete minimaxing, and Figs. 1–3 indicate that deep cutoffs probably have only a second-order effect on the efficiency. Furthermore, procedure *F*1 has the great virtue that its analysis is much simpler than that of the full alpha-beta procedure *F*2.

On the other hand, the analysis of *F*1 is by no means as easy as it looks, and the mathematics turns out to be extremely interesting. In fact, the

authors' first analysis was found to be incorrect, although several competent people had checked it without seeing any mistakes. Since the error is quite instructive, we shall present our original (but fallacious) analysis here, challenging the reader to "find the bug"; then we shall study how to fix things up.

With this understanding, let us consider the following problem: A uniform game tree of degree d and height h is constructed with random values attached to its d^h terminal positions. What is the expected number of terminal positions examined when procedure $F1$ is applied to this tree? The answer to this problem will be denoted by $T(d, h)$.

Since the search procedure depends only on the relative order of the d^h terminal values, not on their magnitudes, and since there is zero probability that two different terminal positions get the same value, we may assume that the respective values assigned to the terminal positions are permutations of $\{1, 2, \ldots, d^h\}$, each permutation occurring with probability $1/(d^h)!$. From this observation it is clear that the d^l values of positions on each level l are also in random order, for $0 \leqslant l \leqslant h$. Although procedure $F1$ does not always compute the exact F values at every position, it is not difficult to verify that the decisions $F1$ makes about cutoffs depend entirely on the F values (not on the approximate values $F1(p)$); so we may conclude that the expected number of positions examined on level l is $T(d, l)$ for $0 \leqslant l \leqslant h$. This justifies restricting attention to a single level h when we count the number of positions examined.

In order to simplify the notation, let us consider first the case of ternary trees, $d = 3$; the general case will follow easily once this one is understood. Our first step is to classify the positions of the tree into types A, B, C as follows:

The root position is type A.
The first successor of every nonterminal position is type A.
The second successor of every nonterminal position is type B.
The third successor of every nonterminal position is type C.

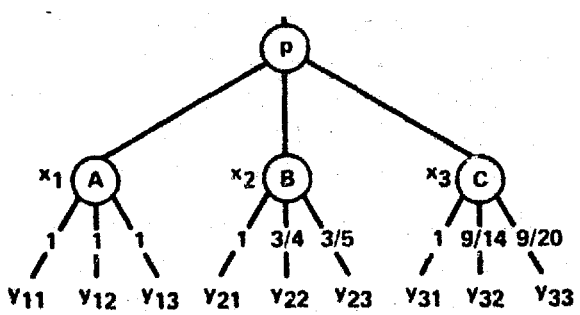

FIG. 6. Part of a uniform ternary tree.

Fig. 6 shows the local "environment" of typical A, B, C positions, as they appear below a nonterminal position p which may be of any type. The F-values of these three positions are x_1, x_2, x_3, respectively, and their descendants have respective F-values $y_{11}, \ldots, y_{33}$. Our assumptions guarantee that $y_{11}, \ldots, y_{33}$ are in random order, no matter what level of the tree we are studying; hence the values

$$x_1 = \max(-y_{11}, -y_{12}, -y_{13}), \ldots, x_3 = \max(-y_{31}, -y_{32}, -y_{33})$$

are also in random order.

If position p is examined by calling $F1(p, \mathit{bound})$, then position A will be examined by the subsequent call $F1(\mathrm{A}, +\infty)$, by definition of $F1$ (see Section 2). Eventually the value x_1 will be returned; and if $-x_1 < \mathit{bound}$, position B will be examined by calling $F1(\mathrm{B}, x_1)$. Eventually the value x_2 will be returned; or, if $x_2 \geqslant x_1$, any value $x_2' \geqslant x_1$ may be returned. If $\max(-x_1, -x_2') < \mathit{bound}$, position C will be examined by calling $F1(\mathrm{C}, \min(x_1, x_2))$. Note that $-\max(-x_1, -x_2') = \min(x_1, x_2)$; the precise value of x_2' is not involved when C is called.

This argument shows that all three successors of an A position are always examined (since the corresponding *bound* is $+\infty$). Each B position will examine its first successor, but (since its *bound* is $x_1 = -\min(y_{11}, y_{12}, y_{13})$) it will examine the second successor if and only if $-y_{21} < -\min(y_{11}, y_{12}, y_{13})$, i.e., if and only if the values satisfy $\min(y_{11}, y_{12}, y_{13}) < y_{21}$. This happens with probability $\frac{3}{4}$, since the y's are randomly ordered and since the relation $\min(y_{11}, y_{12}, y_{13}) > y_{21}$ obviously holds with probability $\frac{1}{4}$. Similarly the third successor of a B position is evaluated if and only if the values satisfy $\min(y_{11}, y_{12}, y_{13}) < \min(y_{21}, y_{22})$, and this has probability $\frac{3}{5}$. The probability that the second successor of a C position is evaluated is the probability that $\max(\min(y_{11}, y_{12}, y_{13}), \min(y_{21}, y_{22}, y_{23})) < y_{31}$, and this occurs $\frac{9}{14}$ of the time; the third successor is examined with probability $\frac{9}{20}$. (A general formula for these probabilities is derived below.)

Let A_n, B_n, C_n be the expected number of positions examined n levels below an A, B, or C position examined by procedure $F1$ in a random game tree. Our discussion proves that

$$\begin{aligned} \mathrm{A}_0 &= \mathrm{B}_0 = \mathrm{C}_0 = 1; \\ \mathrm{A}_{n+1} &= \mathrm{A}_n + \mathrm{B}_n + \mathrm{C}_n; \\ \mathrm{B}_{n+1} &= \mathrm{A}_n + \tfrac{3}{4}\mathrm{B}_n + \tfrac{3}{5}\mathrm{C}_n; \\ \mathrm{C}_{n+1} &= \mathrm{A}_n + \tfrac{9}{14}\mathrm{B}_n + \tfrac{9}{20}\mathrm{C}_n; \end{aligned} \tag{18}$$

and $T(3, h) = \mathrm{A}_h$ is the answer to our problem when $d = 3$.

The solution to these simultaneous linear recurrences can be studied in many ways, and for our purposes the use of generating functions is most convenient. Let

$$A(z) = \sum_{n\geq 0} A_n z^n, \qquad B(z) = \sum_{n\geq 0} B_n z^n, \qquad C(z) = \sum_{n\geq 0} C_n z^n,$$

so that (18) is equivalent to

$$\begin{aligned} A(z) - 1 &= zA(z) + zB(z) + zC(z), \\ B(z) - 1 &= zA(z) + \tfrac{3}{4}zB(z) + \tfrac{3}{5}zC(z), \\ C(z) - 1 &= zA(z) + \tfrac{9}{14}zB(z) + \tfrac{9}{20}zC(z). \end{aligned} \tag{19}$$

By Cramer's rule, $A(z) = U(z)/V(z)$, where

$$\begin{aligned} U(z) &= \det\begin{pmatrix} -1 & z & z \\ -1 & \frac{3}{4}z-1 & \frac{3}{5}z \\ -1 & \frac{9}{14}z & \frac{9}{20}z-1 \end{pmatrix}, \\ V(z) &= \det\begin{pmatrix} z-1 & z & z \\ z & \frac{3}{4}z-1 & \frac{3}{5}z \\ z & \frac{9}{14}z & \frac{9}{20}z-1 \end{pmatrix} \end{aligned} \tag{20}$$

are polynomials in z. If the equation $z^3V(1/z) = 0$ has distinct roots r_1, r_2, r_3, there will be a partial fraction expansion of the form

$$A(z) = \frac{c_1}{1-r_1z} + \frac{c_2}{1-r_2z} + \frac{c_3}{1-r_3z} \tag{21}$$

where

$$c_j = -r_jU(1/r_j)/V'(1/r_j). \tag{22}$$

Consequently $A(z) = \sum_{n\geq 0}(c_1(r_1z)^n + c_2(r_2z)^n + c_3(r_3z)^n)$, and we have

$$A_n = c_1r_1^n + c_2r_2^n + c_3r_3^n$$

by equating coefficients of z^n. If we number the roots so that $|r_1| \geq |r_2| \geq |r_3|$ (and the theorem of Perron [17] assures us that this can be done), we have asymptotically

$$A_n \sim c_1r_1^n. \tag{23}$$

Numerical calculation gives $r_1 = 2.533911$, $c_1 = 1.162125$; thus, the alpha-beta procedure without deep cutoffs in a random ternary tree will examine about as many nodes as in a tree of the same height with average degree 2.534 instead of 3. (It is worthwhile to note that (23) predicts about 48 positions to be examined on the fourth level, while only 35 occurred in Fig. 2; the reason for this discrepancy is chiefly that the one-digit values in Fig. 2 are nonrandom because of frequent equalities.)

Elementary manipulation of determinants shows that the equation $z^3V(1/z) = 0$ is the same as

$$\det\begin{pmatrix} 1-z & 1 & 1 \\ 1 & \frac{3}{4}-z & \frac{3}{5} \\ 1 & \frac{9}{14} & \frac{9}{20}-z \end{pmatrix} = 0;$$

hence r_1 is the *largest eigenvalue* of the matrix

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & \frac{3}{4} & \frac{3}{5} \\ 1 & \frac{9}{14} & \frac{9}{20} \end{pmatrix}.$$

We might have deduced this directly from eq. (18), if we had known enough matrix theory to calculate the constant c_1 by matrix-theoretic means instead of function-theoretic means.

This solves the case $d = 3$. For general d we find similarly that the expected number of terminal positions examined by the alpha-beta procedure without deep cutoffs, in a random uniform game tree of degree d and height h, is asymptotically

$$T(d, h) \sim c_0(d)\, r_0(d)^h \tag{24}$$

for fixed d as $h \to \infty$, where $r_0(d)$ is the largest eigenvalue of a certain $d \times d$ matrix

$$M_d = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1d} \\ p_{21} & p_{22} & \cdots & p_{2d} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ p_{d1} & p_{d2} & \cdots & p_{dd} \end{pmatrix} \tag{25}$$

and where $c_0(d)$ is an appropriate constant. The general matrix element p_{ij} in (25) is the probability that

$$\max_{1 \le k < i} (\min(Y_{k1}, \ldots, Y_{kd})) < \min_{1 \le k < j} Y_{lk} \tag{26}$$

in a sequence of $(i-1)d + (j-1)$ independent identically distributed random variables $Y_{11}, \ldots, Y_{i-1,j}$.

When $i = 1$ or $j = 1$, the probability in (26) is 1, since the min over an empty set is $+\infty$ and the max is $-\infty$. When $i, j > 1$ we can evaluate the probability in several ways, of which the simplest seems to be combinatorial: For (26) to hold, the minimum of all the Y's must be $Y_{k_1 t_1}$ for some $k_1 < i$, and this occurs with probability $(i-1)d/((i-1)d + j - 1)$; removing $Y_{k_1 1}, \ldots, Y_{k_1 d}$ from consideration, the minimum of the remaining Y's must be $Y_{k_2 t_2}$ for some $k_2 < i$, and this occurs with probability $(i-2)d/((i-2)d + j - 1)$; and so on. Therefore (26) occurs with probability

$$p_{ij} = \frac{(i-1)d}{(i-1)d + j - 1} \cdot \frac{(i-2)d}{(i-2)d + j - 1} \cdot \cdots \cdot \frac{d}{d + j - 1}$$
$$= 1 \Big/ \binom{i - 1 + (j-1)/d}{i-1}. \tag{27}$$

This explicit formula allows us to calculate $r_0(d)$ numerically for small d without much difficulty, and to calculate $c_0(d)$ for small d with somewhat more difficulty using (22).

The form of (27) isn't very convenient for asymptotic calculations; there is a much simpler expression which yields an excellent approximation:

LEMMA 1. *When* $0 \leqslant x \leqslant 1$ *and* k *is a positive integer,*

$$k^x \leqslant \binom{k-1+x}{k-1} \leqslant k^x/\Gamma(1+x). \tag{28}$$

(Note that $0.885603 < \Gamma(1+x) \leqslant 1$ for $0 \leqslant x \leqslant 1$, with the minimum value occurring at $x = 0.461632$; hence the simple formula k^x is always within about 11% of the exact value of the binomial coefficient.)

Proof. When $0 \leqslant x \leqslant 1$ and $t > -1$ we have

$$(1+t)^x \leqslant 1 + tx, \tag{29}$$

since the function $f(x) = (1+t)^x/(1+tx)$ satisfies $f(0) = f(1) = 1$, and since

$$f''(x) = ((\ln(1+t) - t/(1+tx))^2 + t^2/(1+tx)^2)f(x) > 0.$$

Using (29) for $t = 1, \frac{1}{2}, \frac{1}{3}, \ldots$ yields

$$1 \leqslant \frac{1+x}{1^x} \leqslant \frac{1+x}{2^x} \leqslant \frac{1+\frac{1}{2}x}{(\frac{3}{2})^x} \leqslant \cdots$$

$$\leqslant \lim_{m\to\infty} \frac{(1+x)}{1}\frac{(2+x)}{2}\cdots\frac{(m+x)}{m}\frac{1}{(m+1)^x} = \frac{1}{\Gamma(1+x)}$$

and the kth term of this series of inequalities is $\binom{k-1+x}{k-1}\Big/k^x$.

For trees of height 2, deep cutoffs are impossible, and procedures $F1$ and $F2$ have an identical effect. How many of the d^2 positions at level 2 are examined? Our analysis gives an exact answer for this case, and Lemma 1 can be used to give a good approximate result which we may state as a theorem.

THEOREM 3. *The expected number of terminal positions examined by the alpha-beta procedure on level 2 of a random uniform game tree of degree* d *is*

$$T(d,2) = \sum_{1\leqslant i,j\leqslant d} p_{ij}, \tag{30}$$

where the p_{ij} *are defined in* (27). *We have*

$$C_1\frac{d^2}{\log d} \leqslant T(d,2) \leqslant C_2\frac{d^2}{\log d} \tag{31}$$

for certain positive constants C_1 *and* C_2.

Proof. Eq. (30) follows from our previous remarks, and from Lemma 1 we know that

$$C\,S(d) \leqslant T(d,2) \leqslant S(d),$$

where $C = 0.885603 = \inf_{0\leqslant x\leqslant 1}\Gamma(1+x)$ and

$$S(d) = \sum_{1 \le i,j \le d} i^{-(j-1)/d}$$
$$= d + \sum_{k=2}^{d} \sum_{j=0}^{d-1} k^{-j/d}$$
$$= d + \sum_{k=2}^{d} \left(\frac{1 - k^{-1}}{1 - k^{-1/d}} \right).$$

Now for $k = d^t$ we have $k^{-1/d} = \exp(-t \ln d/d) = 1 - t \ln d/d + O((\log d/d)^2)$, hence for $\sqrt{d} \le k \le d$, $(1 - k^{-1})/(1 - k^{-1/d})$ lies between $d/\ln d$ and $2d/\ln d$ times $1 + O(\log d/d)$. The bounds in (31) now follow easily.

When the values of $r_0(d)$ for $d \le 30$ are plotted on log log paper, they seem to be approaching a straight line, suggesting that $r_0(d)$ is approximately of order $d^{0.75}$. In fact, a least-squares fit for $10 \le d \le 30$ yielded $d^{0.76}$ as an approximate order of growth; this can be compared to the lower bound $2d^{0.5}$ of an optimum alpha-beta search, or to the upper bound d of a full minimax search, or to the estimate $d^{0.72}$ obtained by Fuller et al. [7] for random alpha-beta pruning when deep cutoffs are included. However, we shall see that the true order of growth of $r_0(d)$ as $d \to \infty$ is really $d/\log d$.

There is a moral to this story: If we didn't know the theoretical asymptotic growth, we would be quite content to think of it as $d^{0.76}$ when d is in a practical range. The formula $d/\log d$ seems much worse than $d^{0.76}$, until we realize the magnitude of $\log d$ in the range of interest. (A similar phenomenon occurs with respect to Shell's sorting method, see [12, pp. 93–95].) On the basis of this theory we may well regard the approximation $d^{0.72}$ in [7] with some suspicion.

But as mentioned above, there is a much more significant moral to this story. *Formula* (24) *is incorrect* because the proof overlooked what appears to be a rather subtle question of conditional probabilities. Did the reader spot a fallacy? The authors found it only by comparing their results to those of [7] in the case $h = 3$, $d = 2$, since procedures $F1$ and $F2$ are equivalent for heights ≤ 3. According to the analysis above, the alpha-beta procedure will examine an average of $6\frac{7}{9}$ nodes on level 3 of a random binary game tree, but according to [7] the number is $6\frac{89}{105}$. After the authors of [7] were politely informed that they must have erred, since we had proved that $6\frac{7}{9}$ was correct, they politely replied that simulation results (including a test on all 8! permutations) had confirmed that the correct answer *is* $6\frac{89}{105}$.

A careful scrutiny of the situation explains what is going on. Theorem 3 is correct, since it deals only with level 2, but trouble occurs at level 3. Our theory predicts a cutoff on the right subtree of every B node with probability $\frac{2}{3}$, so that the terminal values $(f_1, \ldots, f_8)$ in Fig. 7 will be examined with respective probabilities $(1, 1, 1, \frac{2}{3}, 1, 1, \frac{2}{3}, \frac{4}{9})$. Actually f_8 is examined with probability $\frac{18}{35}$ instead of $\frac{4}{9}$; for f_8 is examined if and only if

$$f_7 > \min(f_5, f_6),$$
$$\min(f_5, f_6) < \max(\min(f_1, f_2), \min(f_3, f_4)). \tag{32}$$

Each of these two events has probability $\frac{2}{3}$, but they are not independent.

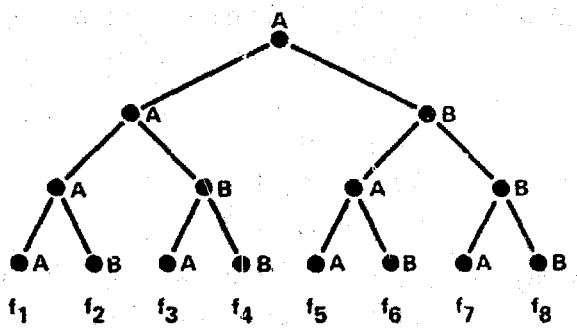

FIG. 7. A tree which reveals the fallacious reasoning.

When the fallacy is stated in these terms, the error is quite plain, but the dependence was much harder to see in the diagrams we had been drawing for ourselves. For example, when we argued using Fig. 6 that the second successor of a B position is examined with probability $\frac{3}{4}$, we neglected to consider that, when p is itself of type B or C, the B node in Fig. 6 is entered only when $\min(y_{11}, y_{12}, y_{13})$ is less than the bound at p; so $\min(y_{11}, y_{12}, y_{13})$ is somewhat smaller than a random value would be. What we should have computed is the probability that $y_{21} > \min(y_{11}, y_{12}, y_{13})$ *given that* position B is not cut off. And unfortunately this can depend in a very complicated way on the ancestors of p.

To make matters worse, our error is in the wrong direction, it doesn't even provide an upper bound for alpha-beta searching; it yields only a lower bound on an upper bound (i.e., nothing). In order to get information relevant to the behavior of procedure $F2$ on random data, we need at least an upper bound on the behavior of procedure $F1$.

A correct analysis of the binary case ($d = 2$) involves the solution of recurrences

$$\begin{aligned} A_{n+1} &= A_n + B_n^{(0)}, \\ B_{n+1}^{(k)} &= A_n + p_k B_n^{(k+1)} \quad \text{for } k \geqslant 0, \\ A_0 &= B_0^{(0)} = B_0^{(1)} = B_0^{(2)} = \cdots = 1, \end{aligned} \tag{33}$$

where the p_k are appropriate probabilities. For example, $p_0 = \frac{2}{3}$; $p_0 p_0$ is the probability that (32) holds; and $p_0 p_1 p_2$ is the probability that fifteen independent random variables satisfy

$$\begin{aligned} f_{15} &> f_{13} \wedge f_{14}, \\ f_{13} \wedge f_{14} &< (f_9 \wedge f_{10}) \vee (f_{11} \wedge f_{12}), \\ (f_9 \wedge f_{10}) \vee (f_{11} \wedge f_{12}) &> ((f_1 \wedge f_2) \vee (f_3 \wedge f_4)) \wedge ((f_5 \wedge f_6) \vee (f_7 \wedge f_8)), \end{aligned} \tag{34}$$

writing $\vee$ for max and $\wedge$ for min. These probabilities can be computed exactly by evaluating appropriate integrals, but the formulas are complicated and it is easier to look for upper bounds. We can at least show easily that the probability in (34) is $\leqslant \frac{4}{9}$, since the first and third conditions *are* independent, and they each hold with probability $\frac{2}{3}$. Thus we obtain an upper bound if we set $p_0 = p_2 = p_4 = \ldots = \frac{2}{3}$ and $p_1 = p_3 = \ldots = 1$; this is equivalent to the recurrence

$$\begin{aligned} A_0 &= B_0 = 1, \\ A_{n+1} &= A_n + B_n, \\ B_{n+1} &= A_n + \tfrac{2}{3}A_n. \end{aligned} \tag{35}$$

Similarly in the case of degree 3, we obtain an upper bound on the average number of nodes examined without deep cutoffs by solving the recurrence

$$\begin{aligned} A_0 &= B_0 = C_0 = 1, \\ A_{n+1} &= A_n + B_n + C_n, \\ B_{n+1} &= A_n + \tfrac{3}{4}A_n + \tfrac{3}{5}A_n, \\ C_{n+1} &= A_n + \tfrac{9}{14}A_n + \tfrac{9}{20}A_n, \end{aligned} \tag{36}$$

in place of (18). This is equivalent to

$$A_{n+1} = A_n + (1 + \tfrac{3}{4} + \tfrac{3}{5} + 1 + \tfrac{9}{14} + \tfrac{9}{20})A_{n-1}$$

and for general degree d we get the recurrence

$$A_{n+1} = A_n + S_d A_{n-1}, \tag{37}$$

where $A_0 = 1$, $A_1 = d$, and

$$S_d = \sum_{\substack{2 \leqslant i \leqslant d \\ 1 \leqslant j \leqslant d}} p_{ij}. \tag{38}$$

This gives a valid upper bound on the behavior of procedure *F*1, because it is equivalent to setting *bound* $\leftarrow +\infty$ at certain positions (and this operation never decreases the number of positions examined). Furthermore we can solve (37) explicitly, to obtain an asymptotic upper bound on $T(d, h)$ of the form $c_1(d)\, r_1(d)^h$, where the growth ratio is

$$r_1(d) = \sqrt{(S_d + \tfrac{1}{4})} + \tfrac{1}{2}. \tag{39}$$

Unfortunately it turns out that S_d is of order $d^2/\log d$, by Theorem 3; so (39) is of order $d/\sqrt{\log d}$, while an upper bound of order $d/\log d$ is desired.

Another way to get an upper bound relies on a more detailed analysis of the structural behavior of procedure *F*1, as in the following theorem.

THEOREM 4. *The expected number of terminal positions examined by the alpha-beta procedure without deep cutoffs, in a random uniform game tree of degree d and height h, satisfies*

$$T(d, h) < c^*(d)\, r^*(d)^h, \tag{40}$$

where $r^*(d)$ is the largest eigenvalue of the matrix

$$M_d^* = \begin{pmatrix} \sqrt{p_{11}} & \sqrt{p_{12}} & \cdots & \sqrt{p_{1d}} \\ \sqrt{p_{21}} & \sqrt{p_{22}} & \cdots & \sqrt{p_{2d}} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \sqrt{p_{d1}} & \sqrt{p_{d2}} & \cdots & \sqrt{p_{dd}} \end{pmatrix}, \tag{41}$$

and $c^*(d)$ is an appropriate constant.

(The p_{ij} in (41) are the same as in (25).)

Proof. Assign coordinates $a_1 \ldots a_l$ to the positions of the tree as in Section 6. For $l \geqslant 1$, it is easy to prove by induction that position $a_1 \ldots a_l$ has *bound* $= \min\{F(a_1 \ldots a_{l-1}k) \mid 1 \leqslant k < a_l\}$ when it is examined by procedure $F1$; hence it is examined if and only if $a_1 \ldots a_{l-1}$ is examined and

$$-\min_{1 \leqslant k < a_l} F(a_1 \ldots a_{l-1}k) < \min_{1 \leqslant k < a_{l-1}} F(a_1 \ldots a_{l-2}k) \quad \text{or} \quad l = 1. \tag{42}$$

It follows that a terminal position $a_1 \ldots a_h$ is examined by $F1$ if and only if (42) holds for $1 \leqslant l \leqslant h$. Let us abbreviate (42) by P_l, so that $a_1 \ldots a_h$ holds if and only if P_1 and ... and P_h. Condition P_l by itself for $l \geqslant 2$ holds with probability p_{ij}, where $i = a_{l-1}$ and $j = a_l$, because of definition (25). Hence if the P_l were independent we would have $a_1 \ldots a_h$ examined with probability $p_{a_1a_2}\, p_{a_2a_3} \cdots p_{a_{h-1}a_h}$, and this is precisely equivalent to the analysis leading to (24). However, the P_l aren't independent, as we have observed in (32) and (34).

Condition P_l is a function of the terminal values

$$f(a_1 \ldots a_{l-2}\, j\, k\, a_{l+1} \ldots a_h),$$

where $j < a_{l-1}$ or $j = a_{l-1}$ and $k < a_l$. Hence P_l is *inaependent* of $P_1, P_2, \ldots, P_{l-2}$. (This generalizes an observation we made about (34).) Let x be the probability that position $a_1 \ldots a_h$ is examined, and assume for convenience in notation that h is odd. Then by the partial independence of the P_l's, we have

$$x < p_{a_1a_2}\, p_{a_3a_4} \cdots p_{a_{h-2}a_{h-1}},$$
$$x < p_{a_2a_3}\, p_{a_4a_5} \cdots p_{a_{h-1}a_h};$$

hence

$$x < \sqrt{p_{a_1a_2}\, p_{a_2a_3} \cdots p_{a_{h-1}a_h}}$$

and the theorem follows by choosing $c^*(d)$ large enough.

Now we are ready to establish the correct asymptotic growth rate of the branching factor for procedure $F1$.

THEOREM 5. *The expected number $T(d, h)$ of terminal positions examined by the alpha-beta procedure without deep cutoffs, in a random uniform game tree of degree d and height h, has a branching factor*

$$\lim_{h \to \infty} T(d, h)^{1/h} = r(d) \tag{43}$$

which satisfies

$$C_3 \frac{d}{\log d} \leqslant r(d) \leqslant C_4 \frac{d}{\log d} \tag{44}$$

for certain positive constants C_3 and C_4.

Proof. We have

$$T(d, h_1 + h_2) \leqslant T(d, h_1)\, T(d, h_2), \tag{45}$$

since the right-hand side of (45) is the number of positions that would be examined by *F*1 if *bound* were set to $+\infty$ for all positions at height h_1. Furthermore the arguments above prove that

$$\liminf_{h\to\infty} T(d, h) \geqslant r_0(d), \qquad \limsup_{h\to\infty} T(d, h) \leqslant r_1(d), r^*(d).$$

By a standard argument about subadditive functions (see, e.g., [20, Problem I.98]) it follows that the limit (43) exists.

To prove the lower bound in (44) we shall show that $r_0(d) \geqslant C_3\, d/\log d$. The largest eigenvalue of a matrix with positive entries p_{ij} is known to be $\geqslant \min_i(\sum_j p_{ij})$, according to the theory of Perron [19]; see [26, Section 2.1] for a modern account of this theory.[4] Therefore by Lemma 1,

$$r_0(d) \geqslant C \min_{1\leqslant i\leqslant d}\Bigg(\sum_{1\leqslant j\leqslant d} i^{-(j-1)/d}\Bigg)$$

$$= C \min_{2\leqslant i\leqslant d}\left(\frac{1 - i^{-1}}{1 - i^{-1/d}}\right)$$

$$> C \frac{1 - d^{-1}}{1 - d^{-1/d}} > C \frac{d - 1}{\ln d},$$

where $C = 0.885603 = \inf_{0\leqslant x\leqslant 1} \Gamma(1 + x)$, since $d^{-1/d} = \exp(-\ln d/d) > 1 - \ln d/d$.

To get the upper bound in (44), we shall prove that $r^*(d) < C_4\, d/\log d$, using a rather curious matrix norm. If s and t are positive real numbers with

$$\frac{1}{s} + \frac{1}{t} = 1, \tag{46}$$

then all eigenvalues λ of a matrix A with entries a_{ij} satisfy

$$|\lambda| \leqslant \Big(\sum_i \Big(\sum_j |a_{ij}|^s\Big)^{s/t}\Big)^{1/s}. \tag{47}$$

To prove this, let $Ax = \lambda x$, where x is a nonzero vector; by Hölder's inequality [9, Section 2.7],

$$|\lambda|\Big(\sum_i |x_i|^s\Big)^{1/s} = \Big(\sum_i \Big|\sum_j a_{ij}x_j\Big|^s\Big)^{1/s}$$

$$\leqslant \Big(\sum_i \Big(\sum_j |a_{ij}|\Big)^{s/t}\Big(\sum_j |x_j|^s\Big)\Big)^{1/s}$$

[4] We are indebted to Dr J. H. Wilkinson for suggesting this proof of the lower bound.

$$= \left(\sum_i \left(\sum_j |a_{ij}^t|\right)^{s/t}\right)^{1/s} \left(\sum_i |x_i^s|\right)^{1/s}$$

and (47) follows.

If we let $s = t = 2$, inequality (47) yields $r^*(d) = O(d/\sqrt{\log d})$, while if s or $t \to \infty$ the upper bound is merely $O(d)$. Therefore some care is necessary in selecting the best s and t; for our purposes we choose $s = f(d)$ and $t = f(d)/(f(d) - 1)$, where $f(d) = \frac{1}{2} \ln d / \ln \ln d$. Then

$$r^*(d) \leqslant \left(\sum_{1 \leqslant i \leqslant d} \left(\sum_{1 \leqslant j \leqslant d} i^{-t(j-1)/2d}\right)^{s/t}\right)^{1/s}$$

$$< \left(\sqrt{d}\, d^{s/t} + (d - \sqrt{d}) \left(\sum_{j \geqslant 1} \sqrt{d}^{-t(j-1)/2d}\right)^{s/t}\right)^{1/s}. \qquad (48)$$

The inner sum is $g(d) = 1/(1 - d^{-t/4d}) = (4d/\ln d)(1 + O(\ln \ln d/\ln d))$, so

$$d\, g(d)^{s/t} = d^{f(d)-1/2} \exp(\tfrac{1}{2} \ln 4 \ln d/\ln \ln d + \ln \ln d + O(1)).$$

Hence the right-hand side of (48) is

$$\exp(\ln d - \ln \ln d + \ln 4 + O((\ln \ln d)^2/\ln d));$$

we have proved that

$$r^*(d) \leqslant (4d/\ln d)(1 + O((\ln \ln d)^2/\ln d)) \quad \text{as } d \to \infty.$$

TABLE 1. Bounds for the branching factor in a random tree when no deep cutoffs are performed

d	$r_0(d)$	$r_1(d)$	$r^*(d)$	d	$r_0(d)$	$r_1(d)$	$r^*(d)$
2	1.847	1.884	1.912	17	8.976	11.678	11.470
3	2.534	2.666	2.722	18	9.358	11.938	12.021
4	3.142	3.397	3.473	19	9.734	12.494	12.567
5	3.701	4.095	4.186	20	10.106	13.045	13.108
6	4.226	4.767	4.871	21	10.473	13.593	13.644
7	4.724	5.421	5.532	22	10.836	14.137	14.176
8	5.203	6.059	6.176	23	11.194	14.678	14.704
9	5.664	6.684	6.805	24	11.550	15.215	15.228
10	6.112	7.298	7.420	25	11.901	15.750	15.748
11	6.547	7.902	8.024	26	12.250	16.282	16.265
12	6.972	8.498	8.618	27	12.595	16.811	16.778
13	7.388	9.086	9.203	28	12.937	17.337	17.288
14	7.795	9.668	9.781	29	13.277	17.861	17.796
15	8.195	10.243	10.350	30	13.614	18.383	18.300
16	8.589	10.813	10.913	31	13.948	18.903	18.802

Table 1 shows the various bounds we have obtained on $r(d)$, namely the lower bound $r_0(d)$ and the upper bounds $r_1(d)$ and $r^*(d)$. We have proved that $r_0(d)$ and $r^*(d)$ grow as $d/\log d$, and that $r_1(d)$ grows as $d/\sqrt{\log d}$; but the table shows that $r_1(d)$ is actually a better bound for $d \leqslant 24$.

8. Discussion of the Model

The theoretical model we have studied gives us an upper bound on the actual behavior obtained in practice. It is an upper bound for four separate reasons:

(a) the deep cutoffs are not considered;

(b) the ordering of successor positions is random;

(c) the terminal positions are assumed to have distinct values;

(d) the terminal values are assumed to be independent of each other.

Each of these conditions makes our model pessimistic; for example, it is usually possible in practice to make plausible guesses that some moves will be better than others. Furthermore, the large number of equal terminal values in typical games helps to provide additional cutoffs. The effect of assumption (d) is less clear, and it will be studied in Section 9.

In spite of all these pessimistic assumptions, the results of our calculations show that alpha-beta pruning will be reasonably efficient.

Let us now try to estimate the effect of deep cutoffs vs no deep cutoffs. One way to study this is in terms of the best case: Under ideal ordering of successor positions, what is the analogue for procedure F1 of the theory developed in Section 6? It is not difficult to see that the positions $a_1 \ldots a_l$ examined by $F1$ in the best case are precisely those with no two non-1's in a row, i.e., those for which $a_j > 1$ implies $a_{j+1} = 1$.

In the ternary case under best ordering, we obtain the recurrence

$$\begin{aligned} A_0 &= B_0 = C_0 = 1, \\ A_{n+1} &= A_n + B_n + C_n, \\ B_{n+1} &= A_n, \\ C_{n+1} &= A_n, \end{aligned} \tag{49}$$

hence $A_{n+1} = A_n + 2A_{n-1}$. For general d the corresponding recurrence is

$$A_0 = 1, \quad A_1 = d, \quad A_{n+2} = A_{n+1} + (d-1)A_n. \tag{50}$$

The solution to this recurrence is

$$A_n = \frac{1}{\sqrt{(4d-3)}}((\sqrt{(d-\tfrac{3}{4})} + \tfrac{1}{2})^{n+2} - (-\sqrt{(d-\tfrac{3}{4})} + \tfrac{1}{2})^{n+2}); \tag{51}$$

so the growth rate or effective branching factor is $\sqrt{(d-\frac{3}{4})} + \frac{1}{2}$, not much higher than the value $\sqrt{d}$ obtained for the full method including deep cutoffs. This result tends to support the contention that deep cutoffs have only a second-order effect, although we must admit that poor ordering of successor moves will make deep cutoffs increasingly valuable.

9. Dependent Terminal Values

Our model gives independent values to all the terminal positions, but such independence doesn't happen very often in real games. For example, if $f(p)$

is based on the piece count in a chess game, all the positions following a blunder will tend to have low scores for the player who loses his men.

In this section we shall try to account for such dependencies by considering a *total dependency* model, which has the following property for all nonterminal positions p: For each i and j, all of the terminal successors of p_i either have greater value than all terminal successors of p_j, or they all have lesser value. This model is equivalent to assigning a permutation of $\{0, 1, \ldots, d-1\}$ to the moves at every position, and then using the concatenation of all move numbers leading to a terminal position as that position's value, considered as a radix-d number. For example, Fig. 8 shows a uniform ternary game tree of height 3 constructed in this way.

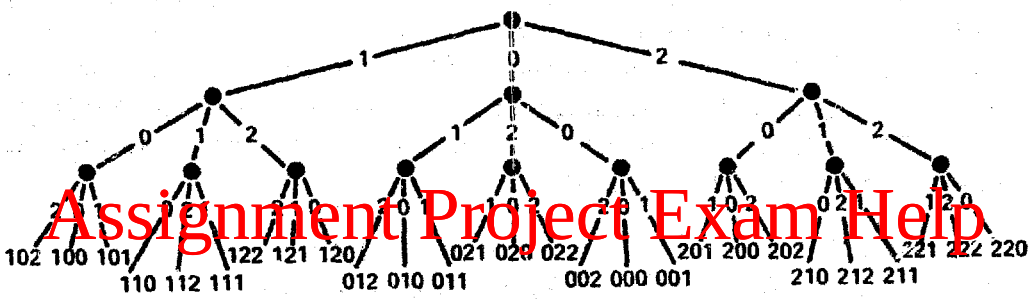

FIG. 8. A tree with "totally dependent" values.

Another way to look at this model is to imagine assigning the values $0, 1, \ldots, d^h - 1$ in d-ary notation to the terminal positions, and then to apply a random permutation to the branches emanating from every nonterminal position. It follows that the F value at the root of a ternary tree is always $-(0202\ldots20)_3$ if h is odd, $+(2020\ldots20)_3$ if h is even.

THEOREM 6. *The expected number of terminal positions examined by the alpha-beta procedure, in a random totally dependent uniform game tree of degree d and height h, is*

$$\frac{d - H_d}{d - H_d^2}(d^{\lceil h/2 \rceil} + H_d\, d^{\lfloor h/2 \rfloor} - H_d^{h+1} - H_d^h) + H_d^h, \tag{52}$$

where $H_d = 1 + \frac{1}{2} + \ldots + 1/d$.

Proof. As in our other proofs, we divide the positions of the tree into a finite number of classes or types for which recurrence relations can be given. In this case we use three types, somewhat as in our proof of Theorems 1 and 2.

A type 1 position p is examined by calling *F2(p, alpha, beta)* where all terminal descendants q of p have $alpha < \pm f(q) < beta$; here the + or − sign is used according as p is an even or an odd number of levels from the bottom of the tree. If p is nonterminal, its successors are assigned a definite ranking; let us say that p_i is *relevant* if $F(p_i) < F(p_j)$ for all $1 \leqslant j < i$. Then

all of the relevant successors of p are examined by calling $F2(p_i, -beta, -m)$ where $F(p_i)$ lies between $-beta$ and $-m$, hence the relevant p_i are again of type 1. The irrelevant p_i are examined by calling $F2(p_i, -beta, -m)$ where $F(p_i) > -m$, and we shall call them type 2.

A type 2 position p is examined by calling $F2(p, alpha, beta)$ where all terminal descendants q of p have $\pm f(q) > beta$. If p is nonterminal, its first successor p_1 is classified as type 3, and it is examined by calling $F2(p_1, -beta, -alpha)$. This procedure call eventually returns a value $\leqslant -beta$, causing an immediate cutoff.

A type 3 position p is examined by calling $F2(p, alpha, beta)$ where all terminal descendants q of p have $\pm f(q) < alpha$. If p is nonterminal, all its successors are classified type 2, and they are examined by calling $F2(p_i, -beta, -alpha)$; they all return values $\geqslant -alpha$.

Let A_n, B_n, C_n be the expected number of terminal positions examined in a random totally dependent uniform tree of degree d and height h, when the root is of type 1, 2, or 3 respectively. The above argument shows that the following recurrence relations hold:

$$\begin{aligned} A_0 = B_0 = C_0 &= 1, \\ A_{n+1} &= A_n + (\tfrac{1}{2}A_n + \tfrac{1}{2}B_n) + (\tfrac{1}{3}A_n + \tfrac{2}{3}B_n) + \ldots \\ &\qquad + ((1/d)A_n + ((d-1)/d)B_n) \\ &= H_d A_n + (d - H_d)B_n, \\ B_{n+1} &= C_n, \\ C_{n+1} &= dB_n. \end{aligned} \tag{53}$$

Consequently $B_n = d^{\lfloor n/2 \rfloor}$, and A_h has the value stated in (52).

COROLLARY 4. *When $d \geqslant 3$, the average number of positions examined by alpha-beta search under the assumption of totally dependent terminal values is bounded by a constant*[5] *times the optimum number of positions specified in Corollary* 3.

Proof. The growth of (52) as $h \to \infty$ is order $d^{h/2}$. The stated constant is approximately

$$(d - H_d)(1 + H_d)/2(d - H_d^2).$$

(When $d = 2$ the growth rate of (52) is order $(\frac{3}{2})^h$ instead of $\sqrt{2}^h$.)

Incidentally, we can also analyze procedure $F1$ under the same assumptions; the restriction of deep cutoffs leads to the recurrence

$$A_0 = 1, \quad A_1 = 1, \quad A_{n+2} = H_d A_{n+1} + (d - H_d)A_n, \tag{54}$$

and the corresponding growth rate is of order $(\sqrt{(d - H_d + \frac{1}{4}H_d^2)} + \frac{1}{2}H_d)^h$. So again the branching factor is approximately $\sqrt{d}$ for large d.

[5] This "constant" depends on the degree d, but not on the height h.

The authors of [7] have suggested another model to account for dependencies between positions: Each *branch* (i.e., each arc) of the uniform game tree is assigned a random number between 0 and 1, and the values of terminal positions are taken to be the sums of all values on the branches above. If we apply the naïve approach of Section 7 to the analysis of this model without deep cutoffs, the probability needed in place of eq. (26) is the probability that

$$\max_{1 \leq k < i} (X_k + \min(Y_{k1}, \ldots, Y_{kd})) < X_i + \min_{1 \leq k < j} Y_{ik}, \tag{55}$$

where as before the Y's are independent and identically distributed random variables, and where $X_1, \ldots, X_i$ are independent uniform random variables in [0, 1]. Balkema [2] has shown that (55) never occurs with greater probability than the value p_{ij} derived in Section 7, regardless of the distribution of the Y's (as long as it is continuous). Therefore we have good grounds to believe that dependencies between position values tend to make alpha-beta pruning more efficient than it would be if all terminal positions had independent values.

ACKNOWLEDGMENTS

We wish to thank J. R. Slagle, whose lecture at Caltech in 1967 originally stimulated some of the research reported here, and we also wish to thank Forest Baskett, Robert W. Floyd, John Gaschnig, James Gillogly, John McCarthy, and James H. Wilkinson for discussions which contributed significantly to our work. Computer time for our numerical experiments was supported in part by ARPA and in part by IBM Corporation.

REFERENCES

1. Adelson-Velskiy, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovskii, A. A. and Uskov, A. V. Programming a computer to play chess. *Uspehi Mat. Nauk* **25** (2) (March–April 1970), 221–260 (in Russian).
2. Balkema, G. Personal communication, July 19, 1974.
3. Bernstein, A., Roberts, M. De V., Arbuckle, T. and Belsky, M. A. A chess-playing program for the IBM 704 computer. *Proc. Western Joint Computer Conference* **13** (1958), 157–159.
4. Brudno, A. L. Bounds and valuations for shortening the scanning of variations. *Problemy Kibernet.* **10** (1963), 141–150 (in Russian).
5. Dahl, O.-J. and Belsnes, D. *Algoritmer og Datastrukturer*. Studentlitteratur, Lund (1973).
6. Floyd, R. W. Personal communication, January 18, 1965.
7. Fuller, S. H., Gaschnig, J. G. and Gillogly, J. J. Analysis of the alpha-beta pruning algorithm. Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa. (July 1973), 51 pp.
8. Greenblatt, R. D., Eastlake, D. E. and Crocker, S. D. The Greenblatt chess program. *Proc. AFIPS Fall Joint Computer Conference* **31** (1967), 801–810.
9. Hardy, G. H., Littlewood, J. E. and Pólya, G. *Inequalities*. Cambridge Univ. Press, London (1934).

10. Hart, T. P. and Edwards, D. J. The tree prune (TP) algorithm. M.I.T. Artificial Intelligence Project Memo #30, R.L.E. and Computation Center, Massachusetts Institute of Technology, Cambridge, Mass. (December 4, 1961), 6 pp; revised form: The α-β heuristic, D. J. Edwards and T. P. Hart (October 28, 1963), 4 pp.
11. Knuth, D. E. *Fundamental Algorithms: The Art of Computer Programming* **1**. Addison-Wesley, Reading, Mass. (1968/1973).
12. Knuth, D. E. *Sorting and Searching: The Art of Computer Programming* **3**. Addison-Wesley, Reading, Mass. (1973).
13. Knuth, D. E. Structured programming with go to statements. *Comput. Surveys* **6** (1974), 261–301.
14. Lawler, E. L. and Wood, D. E. Branch-and-bound methods: A survey. *Operations Res.* **14** (1966), 699–719.
15. McCarthy, J. Personal communication, December 1, 1973.
16. Newell, A., Shaw, J. C. and Simon, H. A. Chess-playing programs and the problem of complexity. *IBM J. Res. and Develop.* **2** (1958), 320–355; reprinted with minor corrections in *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York (1963), 109–133.
17. Nievergelt, J., Farrar, J. C. and Reingold, E. M. *Computer Approaches to Mathematical Problems*. Prentice-Hall, Englewood Cliffs, N.J. (1974).
18. Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York (1971).
19. Perron, O. Zur Theorie der Matrizen. *Math. Ann.* **64** (1907), 248–263.
20. Pólya, G. and Szegö, G. *Aufgaben und Lehrsätze aus der Analysis* **1**. Springer, Berlin (1925).
21. Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM J. Res. and Develop.* **3** (1959), 211–229; reprinted with minor additions and corrections in *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York (1963), 71–105.
22. Samuel, A. L. Some studies in machine learning using the game of checkers. II—Recent progress. *IBM J. Res. and Develop.* **11** (1967), 601–617.
23. Slagle, J. R. *Artificial Intelligence: The Heuristic Programming Approach*. McGraw-Hill, New York (1971).
24. Slagle, J. R. and Bursky, P. Experiments with a multipurpose, theorem-proving heuristic program. *J.ACM* **15** (1968), 85–99.
25. Slagle, J. R. and Dixon, J. K. Experiments with some programs that search game trees. *J.ACM* **16** (1969), 189–207.
26. Varga, R. S. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, N.J. (1962).
27. Wells, M. B. *Elements of Combinatorial Computing*. Pergamon, Oxford (1971).

Received 3 September 1974