Asymptotic Properties of Minimax Trees and Game-Searching Procedures*

Judea Pearl
Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, CA, U.S.A.

Recommended by Nils Nilsson

ABSTRACT

The model most frequently used for evaluating the behavior of game-searching methods consists of a uniform tree of height h and a branching degree d, where the terminal positions are assigned random, independent and identically distributed values. This paper highlights some curious properties of such trees when h is very large and examines their implications on the complexity of various game-searching methods.

If the terminal positions are assigned a WIN–LOSS *status with the probabilities P_0 and $1-P_0$, respectively, then the root node is almost a sure* WIN *or a sure* LOSS, *depending on whether P_0 is higher or lower than some fixed-point probability $P^*(d)$. When the terminal positions are assigned continuous real values, the minimax value of the root node converges rapidly to a unique predetermined value v^*, which is the $(1-P^*)$-fractile of the terminal distribution.*

Exploiting these properties we show that a game with WIN–LOSS *terminals can be solved by examining, on the average, $O[(d)^{h/2}]$ terminal positions if $P_0 \neq P^*$ and $O[(P^*/(1-P^*))^h]$ positions if $P_0 = P^*$, the former performance being optimal for all search algorithms. We further show that a game with continuous terminal values can be evaluated by examining an average of $O[(P^*/(1-P^*))^h]$ positions, and that this is a lower bound for all directional algorithms. Games with discrete terminal values can, in almost all cases, be evaluated by examining an average of $O[(d)^{h/2}]$ terminal positions. This performance is optimal and is also achieved by the* ALPHA–BETA *procedure.*

1. The probability of winning a standard *h*-level game tree with random WIN positions

We consider a class of two-person perfect information games represented by the tree of Fig. 1. Two players, called MAX and MIN, take alternate turns. In each turn a player may choose one out of d legal moves. The game lasts exactly

*This work was supported in part by the National Science Foundation Grants MCS 78-07468 and MCS 78-18924.

Copyright © 1980 by North-Holland Publishing Company

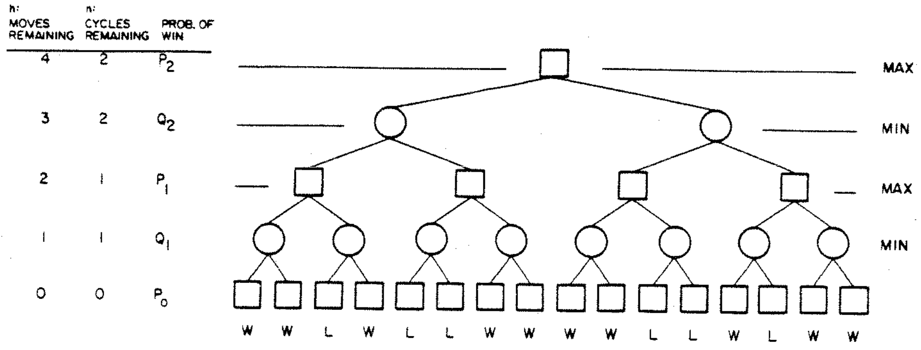


FIG. 1. A uniform binary game tree with two move-cycles, $h = 4$, $n = 2$, $d = 2$.

n move-cycles or $h = 2n$ moves, at which point a terminal position is reached. Each terminal position constitutes either a WIN or a LOSS for the first player. We assume that the assignment of labels to the d^h terminal positions is done at random, prior to the beginning of the game, and that each terminal position may receive a WIN with probability P_0 (and a LOSS with probability $1 - P_0$) independently of other assignments. We shall refer to such a tree as a (h, d, P_0)-tree.

The first quantity we wish to compute is P_n, the probability that MAX can force a WIN given that it is his turn to move and that exactly n move-cycles are left in the game. Similarly, we denote by Q_n the probability that MAX can force a WIN given that it is MIN's turn to move and there are a total of $2n - 1$ individual moves left in the game. Clearly, P_n and Q_n are calculated prior to examining the terminal positions. Once the WIN–LOSS assignment is known, each node of the tree can be unequivocally labeled either a WIN or a LOSS.

For a MAX node (h even) to be a WIN, at least one of its d successors must be a WIN; therefore:

$$1 - P_n = (1 - Q_n)^d. \tag{1}$$

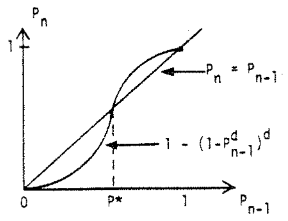
Also, for a MIN position (h odd) to be a WIN, all of its d successors must be a WIN; thus:

$$Q_n = P_{n-1}^d. \tag{2}$$

Combining (1) and (2) we obtain the recursive relationship:

$$P_n = 1 - (1 - P_{n-1}^d)^d. \tag{3}$$

The asymptotic behavior of P_n for large n can be inferred from the diagram below:



The curve $P_n = 1-(1-P_{n-1}^d)^d$ intersects the line $P_n = P_{n-1}$ in three points: two stable points $P_{n-1} = 0$ and $P_{n-1} = 1$, and one unstable point at $P_{n-1} = P^*$. P^* is the unique solution of the equation: $(1-x^d)^d-(1-x) = 0$ in the range $0 < x < 1$ or more conveniently, the positive root of the equation $x^d+x-1 = 0$. It can be easily ascertained that every root of the latter equation is also a root of the former.

The significance of the probability P^* lies in the fact that if the terminal positions are assigned a WIN with probability $P_0 = P^*$, then, prior to examining any of these positions, MAX is assured a probability P^* of winning the game from any of his moves, regardless of the height of the tree.

Most significantly, if P_0 is slightly different than P^*, we have:

$$\lim_{n\to\infty} P_n(P_0) = \begin{cases} 1 & \text{if } P_0 > P^*, \\ 0 & \text{if } P_0 < P^*. \end{cases} \tag{4}$$

This means that when $P_0 > P^*$, MAX is almost assured a WIN if n is large enough, whereas he faces an almost sure LOSS in the case where $P_0 < P^*$. To illustrate this phenomenon, consider a binary game ($d = 2$) with five move-cycles ($n = 5$). P^* is the solution to $x^2+x-1 = 0$, or $P^* = 1/2(\sqrt{5}-1) = 0.6180339$. If all we know about the terminal positions is that 61.80% of them are WIN's, then we also know that the first player to move has a 61.8% chance of being able to force a WIN. However, if only 50% of the terminal positions are winning, his chances to force a WIN drop to 1.95%, whereas when $P_0 = 70\%$, his chances increase to 98.5%. These numbers become much more dramatic in higher trees, as shown in Fig. 2.

It is simple to show that the slope at the transition region is increasing exponentially with n:

$$\frac{\mathrm{d}}{\mathrm{d}P_0}(P_n)\bigg|_{P_0=P^*} = \left[\frac{d(1-P^*)}{P^*}\right]^{2n} \quad \text{and} \quad \frac{d(1-P^*)}{P^*} > 1 \qquad \text{for } d > 1. \tag{5}$$

Also, a more detailed analysis shows that for sufficiently large n, P_n converges toward its asymptotic values at a super-exponential rate, i.e., for every $0 < \delta < 1$ we can find two integers n_1 and n_2, such that:

$$\begin{aligned} P_n &\leq (\delta)^{d^{(n-n_1)}} && \text{for all } n > n_1 \text{ and } P_0 < P^*, \\ 1-P_n &\leq (\delta)^{d^{(n-n_2)}} && \text{for all } n > n_2 \text{ and } P_0 > P^* \end{aligned} \tag{6}$$

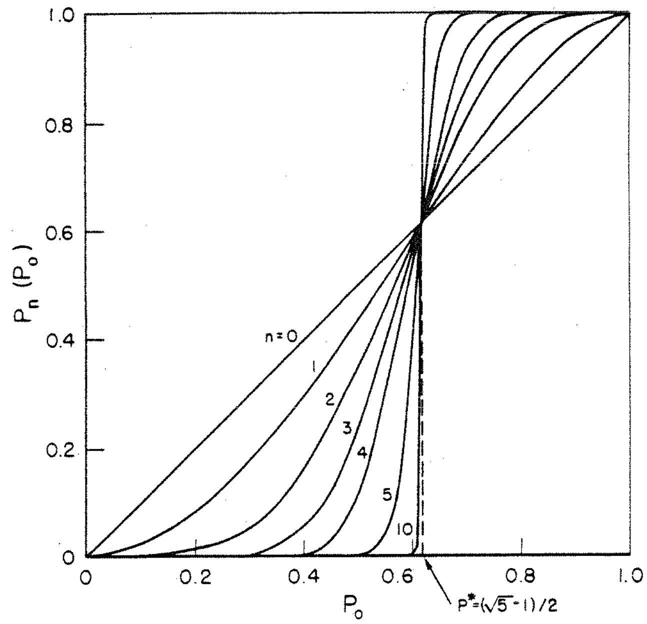


FIG. 2. The probability of winning a n-cycle game (P_n) versus the probability of winning a terminal position (P_0), for a binary ($d = 2$) tree.

where n_1 and n_2 are functions of δ and $P^* - P_0$. It is, thus, safe to conclude that for sufficiently large n, the function $P_n(P_0)$ resembles a step function with an extremely narrow transition region around P^*.

2. Game trees with an arbitrary distribution of terminal values

Consider a uniform tree (constant d) where the terminal nodes are assigned numerical values, $V_0(S_1)$, $V_0(S_2), \ldots, V_0(S_{d^h})$, and assume the latter to be independent identically distributed random variables, drawn from a common distribution function $F_{V_0}(v) = P(V_0 \leq v)$. We shall refer to a tree drawn from such an ensemble as a (h, d, F)-tree and calculate the distribution of the minimax value of the root node.

Denoting the minimax values of nodes at the nth cycles by $V_n(S)$ for MAX nodes and by $U_n(S)$ for MIN nodes, we have:

$$V_n(S) = \max[U_n(S_1), U_n(S_2), \ldots, U_n(S_d)],$$
$$U_n(S) = \min[V_{n-1}(S_1), V_{n-1}(S_2), \ldots, V_{n-1}(S_d)] \tag{7}$$

where $S_1, S_2, \ldots, S_d$ denote the d successors of S. The distribution of $V_n(S)$ is obtained by writing:

$$F_{V_n}(v) \triangleq P[V_n(S) \leq v] = \prod_{i=1}^{d} P[U_n(S_i) \leq v] = [F_{U_n}(v)]^d. \tag{8}$$

$$1 - F_{U_n}(v) \triangleq P[U_n(S) > v] = \prod_{i=1}^{d} P[V_{n-1}(S_i) > v] = [1 - F_{V_{n-1}}(v)]^d \qquad (9)$$

yielding the recursive relation:

$$F_{V_n}(v) = \{1 - [1 - F_{V_{n-1}}(v)]^d\}^d. \qquad (10)$$

Note that (8), (9), and (10) are identical to (1), (2), and (3), respectively, if one identifies $1 - F_{V_n}(v)$ with P_n and $1 - F_{U_n}(v)$ with Q_n. This is not surprising since for any fixed v, the propositions '$V(S_i) > v$' propagate by the same logic as the propositions 'S_i is a WIN'; MAX nodes function as OR gates and MIN nodes perform an AND logic.

From the fact that P_n converges to a step-function as $n \to \infty$ (see (4)), we must conclude that $F_{V_n}(v)$, likewise, satisfies:

$$\lim_{n\to\infty} F_{V_n}(v) = \begin{cases} 0 & F_{V_0}(v) < 1 - P^*, \\ 1 - P^* & F_{V_0}(v) = 1 - P^*, \\ 1 & F_{V_0}(v) > 1 - P^*. \end{cases} \qquad (11)$$

Assume, for the moment, that the terminal values V_0 are continuous random variables and that the distribution $F_{V_0}(v)$ is strictly increasing in the range $0 < F_{V_0} < 1$. In this case $F_{V_0}(v)$ has a unique inverse and the condition $F_{V_0}(v) = 1 - P^*$ is satisfied by one unique value of v which we call v^*:

$$v^* = F_{V_0}^{-1}(1 - P^*). \qquad (12)$$

(11) then implies that when the game tree is sufficiently tall, the cumulative distribution of the root-node value approaches a step function in v, and that the transition occurs at a unique value v^* which is the $(1 - P^*)$-fractile of the terminal distribution $F_{V_0}(\cdot)$. That implies that the density of $V_n(S)$, $f_{V_n}(v)$, becomes highly concentrated around v^* or, in other words, that the root-node value is almost certain to fall in the very close neighborhood of v^*. It appears that the repeated application of alternating MIN–MAX operations on the terminal values has the effect of filtering out their uncertainties until the result emerges at the high levels of the tree as an almost certain, predetermined, quantity.

This is a rather remarkable phenomenon which deserves to be decorated by a theorem.

THEOREM 1. *The root value of a (h, d, F)-tree with continuous strictly increasing terminal distribution* F *converges, as $h \to \infty$ (in probability) to the $(1 - P^*)$-fractile of* F, *where* P* *is the solution of $x^d + x - 1 = 0$.*

If the terminal values are discrete: $v_1 < v_2 < \cdots < v_M$, then the root value converges to a definite limit iff $1 - P^* \neq F_{V_0}(v_i)$ for all i, in which case the limit is the smallest v_i satisfying:

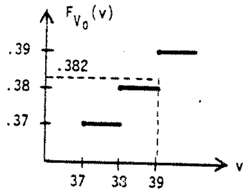
$$F_{V_0}(v_{i-1}) < 1 - P^* < F_{V_0}(v_i).$$

The second part of Theorem 1 becomes evident by writing:

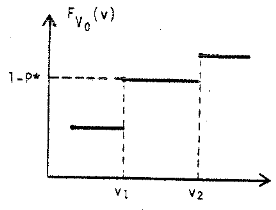
$$P[V_n(S) = v_i] = F_{V_n}(v_i) - F_{V_n}(v_{i-1}).$$

If $1-P^*$ can be 'sandwiched' between two successive levels of F in such a way that $F_{V_0}(v_{i-1}) < 1-P^* < F_{V_0}(v_i)$, then according to (11) $F_{V_n}(v_i) \to 1$, $F_{V_n}(v_{i-1}) \to 0$, and consequently $P[V_n(S) = v_i] \to 1$.

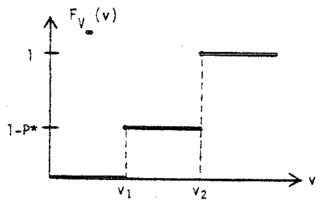
The remarkable feature of this phenomenon is that Theorem 1 holds for any arbitrary distribution of the terminal values. Thus, for example, the root value of a binary tree ($d = 2$) with terminal values uniformly distributed between 0 and 1 would converge to the value $1 - 1/2(\sqrt{5}-1) = 0.382 \therefore\ldots$ If the terminal values are integers, uniformly distributed between 1 and 100, then $F_{V_0}(38) < 1 - P^* = 0.382 < F_{V_0}(39)$.



Therefore, the root value will converge to the integer 39. Exceptions to the theorem would occur only in rare pathological cases where $1-P^*$ coincides exactly with one of the plateaus of $F_{V_0}(v)$, as shown in the diagram below.

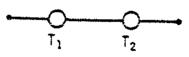


In such a case, the asymptotic distribution of the root node would go from 0 to 1 in two steps, one at v_1 and the other at v_2, as is shown below:

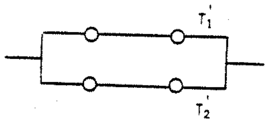


This implies that $V_n(S)$ does not converge to a single limit but may assume two possible values; in a fraction P^* of the instances it will be assigned the value v_2 and in the remaining instances the value v_1. In fact, Section 1 dealt with such a case where, if $P_0 = P^*$, the status of the root-node remains undetermined between WIN ($V_n = 1$) and LOSS ($V_n = 0$).

For the reader's amusement, another manifestation of Theorem 1 will be mentioned, unrelated to game trees. Consider a large collection of unreliable electrical components (e.g., light bulbs) whose times to failure are identically distributed random variables. Connect two of them in series:



The failure time of this series connection is given by $\min[T_1, T_2]$. Now connect two such circuits in parallel:



The failure time of the parallel circuit is equal to that of the longest surviving branch, i.e., $\max(T_1', T_2')$. Continue the process, alternatingly connecting duplicate circuits in series and in parallel, for n cycles. What can be said about the limiting distribution of the failure time, T_n, of the entire circuit? Clearly, T_n is equal to the minimax value of the root-node in an n-cycle binary tree with terminal values determined by the failure times of the individual components. According to Theorem 1, T_n converges to a definite value given by the $(1-P^*)$-fractile of the terminal distribution. Thus, assuming that n is sufficiently large, the entire circuit should fail at a predictable, precise time, which is quite remarkable considering the fact that the circuit is assembled from a host of independent, unreliable, and unpredictable components.

At this point a natural question to ask is how fast the density distribution of the root value contracts to its final value v^*. The answer is that the width of the density function decreases exponentially with n. The range of values W_ϵ (around v^*) which contains all but 2ϵ of the total area under the density function can be shown (for $d = 2$) to be proportional to $(\log 1/2\epsilon)^{0.5842^{-n}}$.

This finding raises some interesting questions regarding the advisability of searching deep uniform game trees. If the final values of these trees can be predetermined with virtual certainty, why spend the exponentially growing effort demanded by an exact evaluation? Instead of insisting on selecting the best first move, we might as well select just any move at random. The expected loss of opportunity induced by such selection is guaranteed not to exceed some predetermined limit which diminishes exponentially with the height of the remaining tree. It makes more sense to reserve one's computational powers for the end-game where the shallowness of the trees is accompanied by more widely varying node values.

These arguments touch on the more general question of how the willingness to act somewhat suboptimally can be converted into computational savings, a question which we hope to study more fully in future studies. At this point, it suffices to state that the uniform tree model with independent and identically distributed terminal values was not devised as a practical game playing tool but rather as a test bed for comparing the efficiencies of various exact-search methods. We shall pursue this plan in the remaining part of this report.

3. The mean complexity of solving (h, d, P_0)-game

Solving a game tree means deciding whether the root-node is a WIN or a LOSS. An absolute lower bound on the number of terminal node examinations needed for establishing the status of the root-node is given by the following argument. If the root node is a WIN, then there exists a subtree (called a solution tree) consisting of one branch emanating from each MAX node and all branches emanating from each MIN node, terminating at a set of WIN terminal positions. Similarly, if the game is a LOSS, such a solution tree exists for the opponent, terminating at all LOSS nodes. In either case, the number of terminal positions in a solution tree is d^n (representing a branching factor d in each move-cycle) or $d^{h/2}$ where h is the number of individual moves. The number of terminal node examinations required to solve the game must exceed $d^{h/2}$ since, regardless of how the solution tree was discovered, one must still ascertain that all its $d^{h/2}$ terminal nodes are WIN in order to defend the proposition 'root is a WIN'. Thus, $d^{h/2}$ represents the number of terminal nodes inspected by a non-deterministic algorithm which solves the (h, d, P_0)-game and is, therefore, a lower bound for all deterministic algorithms.

It is not hard to show that any algorithm solving the (h, d, P_0)-game would, in the worst case, inspect all d^h terminal positions. This can be done by cleverly arranging the terminal assignments in such a way that a decision could not be reached until the last node is inspected. Since the difference between $d^{h/2}$ and d^h may be quite substantial, it is interesting to evaluate the expected number of terminal examinations where the expectation is taken with respect to all possible WIN–LOSS assignments to the terminal nodes.

DEFINITION. Let A be a deterministic algorithm which solves the (h, d, P_0)-game and let $I_A(h, d, P_0)$ denote the expected number of terminal positions examined by A. The quantity:

$$r_A(d, P_0) = \lim_{h\to\infty} [I_A(h, d, P_0)]^{1/h}$$

is called the *branching factor* corresponding to the algorithm A.

DEFINITION. Let C be a class of algorithms capable of solving a general (h, d, P_0)-tree. An algorithm A is said to be *asymptotically optimal over C* if for some P_0 and all d:

$$r_A(d, P_0) \leq r_B(d, P_0) \qquad \forall B \in C.$$

DEFINITION. An algorithm A is said to be *directional* if for some linear arrangement of the terminal nodes it never selects for examination a node situated to the left of a previously examined node.

Simply stated, an algorithm is directional if it always examines nodes from left to right, regardless of the content of the nodes examined.

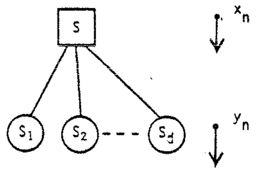
We now compute the branching factor of a simple directional algorithm, called SOLVE, given by the following informal description[1]:

ALGORITHM SOLVE(S): To solve S, start solving its successors from left to right.

If S is MAX, return a WIN as soon as one successor is found to be a WIN; return a LOSS if all successors of S are found to be a LOSS.

If S is MIN, return a LOSS as soon as one successor is found to be a LOSS; return a WIN if all successors of S are found to be a WIN.

To compute $I_{\text{SOLVE}}(h, d, P_0)$ we consider the nth cycle preceding the terminal positions. Let x_n stand for the expected number of terminal nodes inspected in solving the root S of an n-cycle tree, and y_n the expected number of inspections used for solving any of the successors of S.

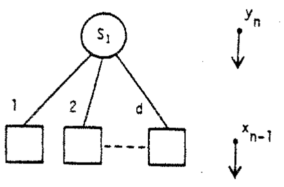


The probability of issuing a WIN after solving the kth successor is $(1 - Q_n)^{k-1}Q_n$. Such an event requires an average of $(k-1)y_n^- + y_n^+$ terminal inspections, where y_n^- and y_n^+ stand for the mean number of inspections required for establishing a LOSS or a WIN, respectively. Also, the event of issuing a LOSS for S carries a probability $(1 - Q_n)^d$ and a mean expenditure of dy_n^- inspections. Therefore:

$$\begin{aligned}
x_n &= \sum_{k=1}^{d} Q_n(1-Q_n)^{k-1}[(k-1)y_n^- + y_n^+] + d(1-Q_n)^d y_n^- \\
&= y_n^+ Q_n \frac{1-(1-Q_n)^d}{Q_n} + y_n^-(1-Q_n)\frac{1-(1-Q_n)^d}{Q_n} \\
&= [y_n^+ Q_n + y_n^-(1-Q_n)]\frac{[1-(1-Q_n)^d]}{Q_n} \qquad \text{(using (1) and (2))} \\
&= y_n \frac{P_n}{P_{n-1}^d}. \qquad (13)
\end{aligned}$$

[1]A more formal definition is given by the flow-chart of Fig. 5, with the few simple modifications discussed at the head of Section 4.

Now examine the solution of any successor of S, say S_1.



The event of issuing a LOSS after solving its kth successor has a probability $P_{n-1}^{k-1}(1-P_{n-1})$ and carries a mean expenditure of $(k-1)x_{n-1}^{+}+x_{n}^{-}$ inspections. The event of exiting with a WIN involves solving all d successors and, therefore, occurs with probability P_{n-1}^{d} and costs an average of dx_{n-1}^{+} inspections. Thus:

$$\begin{aligned} y_n &= \sum_{k=1}^{d} P_{n-1}^{k-1}(1-P_{n-1})[(k-1)x_{n-1}^{+}+x_{n-1}^{-}] + \mathrm{d}P_{n-1}^{d}x_{n-1}^{+} \\ &= [x_{n-1}^{+}P_{n-1}+x_{n-1}^{-}(1-P_{n-1})]\frac{(1-P_{n-1}^{d})}{1-P_{n-1}} \\ &= x_{n-1}\frac{1-P_{n-1}^{d}}{1-P_{n-1}}. \end{aligned} \tag{14}$$

Combining (13) and (14) we obtain:

$$\frac{x_n}{x_{n-1}} = \frac{1-Q_n}{Q_n}\cdot\frac{P_n}{1-P_{n-1}} = \frac{P_n\cdot(1-P_{n-1}^{d})}{P_{n-1}^{d}\cdot(1-P_{n-1})}. \tag{15}$$

Since x_n is equivalent to $I_{\text{SOLVE}}(2n, d, P_0)$ and $x_0 = 1$, we can state:

THEOREM 2. *The expected number of terminal position in a (h, d, P_0)-tree examined by the* SOLVE *algorithm is given by*:

$$I_{\text{SOLVE}}(h, d, P_0) = \prod_{i=1}^{h/2}\frac{P_i(1-P_{i-1}^{d})}{P_{i-1}^{d}(1-P_{i-1})} \tag{16}$$

where P_i, $i = 1, \ldots, \frac{1}{2}h$, is related to P_0 by (3).

Theorem 2 permits an easy calculation of $I_{\text{SOLVE}}(h, d, P_0)$ for wide ranges of d and h, as shown in Fig. 3. In the special case where $P_0 = P^*$ all terms in the product of (16) are equal and, using $P^{*d} = 1 - P^*$, (16) reduces to:

$$I_{\text{SOLVE}}(h, d, P^*) = \left(\frac{P^*}{1-P^*}\right)^h. \tag{17}$$

Note that

$$\lim_{P_{n-1}\to 0}\frac{x_n}{x_{n-1}} = \lim_{P_{n-1}\to 1}\frac{x_n}{x_{n-1}} = d$$

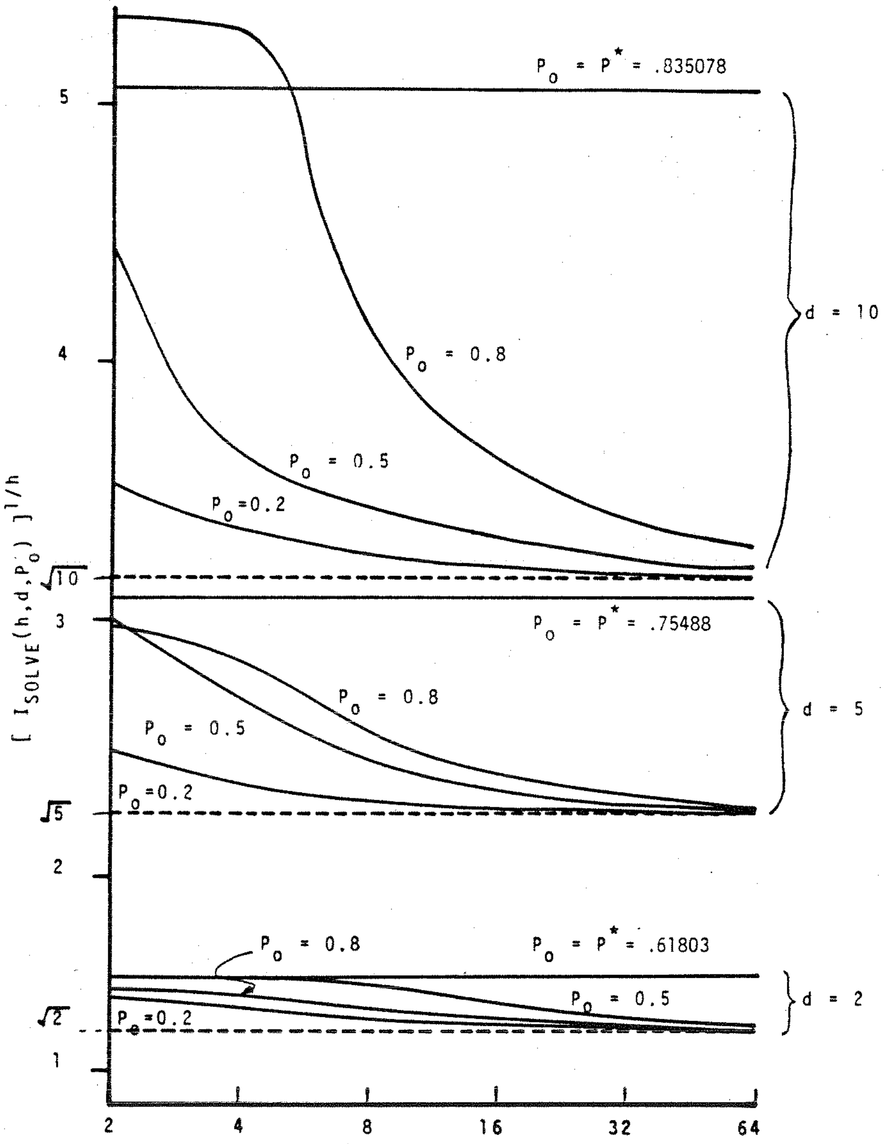


FIG. 3. The expected number of terminal nodes examined by SOLVE (normalized by $(\cdot)^{1/h}$ to represent an effective branching factor).

which implies:

$$\lim_{n\to\infty} \frac{x_n}{x_{n-1}} = \begin{cases} d & P_0 \neq P^*, \\ \left(\dfrac{P^*}{1-P^*}\right)^2 & P_0 = P^*. \end{cases} \tag{18}$$

This limit, combined with the very rapid convergence of P_n (see (6)), leads directly to the asymptotic branching factor of SOLVE:

COROLLARY 1. *The branching factor of the* SOLVE *algorithm is given by*:

$$r_{\text{SOLVE}}(d, P_0) = \begin{cases} d^{1/2} & P_0 \neq P^*, \\ \dfrac{P^*}{1-P^*} & P_0 = P^* \end{cases} \tag{19}$$

where P^* is the positive solution of $x^d + x - 1 = 0$.

Recalling that $d^{1/2}$ is an absolute lower bound for the branching factor of any tree solving algorithm, we conclude:

COROLLARY 2. SOLVE *is asymptotically optimal for $P_0 \neq P^*$.*

For finite values of h or for $P_0 = P^*$ we have no guarantee that SOLVE is optimal. Non-directional algorithms, such as that proposed by Stockman [7] may outperform SOLVE. However, Corollary 2 states that for very deep trees the savings could not be substantial in all cases where $P_0 \neq P^*$.

Any directional algorithm which is governed by a successor-ordering scheme identical to that of SOLVE must examine all the nodes examined by SOLVE. This is so because if some left-to-right algorithm B skips a node visited by SOLVE, a WIN–LOSS assignment can be found which would render the conclusion of SOLVE contrary to that of B. Thus B could not be a general algorithm for solving all (h, d, P_0)-trees. Now, since $I_{\text{SOLVE}}(h, d, P_0)$ is independent on the particular choice of ordering scheme, we may conclude that SOLVE is optimal over the class of directional game-solving algorithms. This leads to:

COROLLARY 3. *The optimal branching factor of any directional algorithm which solves a general (h, d, P_0)-tree is given by* (19).

The case $P_0 = P^*$ deserves a special attention. Although it is not very likely to occur in practical WIN–LOSS games, it plays an important role in the analysis of the α–β procedure. We conclude this section by examining the behavior of $r_{\text{SOLVE}}(d, P^*) = P^*/(1-P^*)$ for large values of d. Writing:

$$q(d) = 1 - P^*(d) \tag{20}$$

the defining equation for $q(d)$ becomes:

$$q(d) = [1 - q(d)]^d \tag{21}$$

which can be satisfied only when:

$$\lim_{d\to\infty} q(d) = 0. \tag{22}$$

Taking log on both sides of (21), gives:

$$\log q(d) = d \log[1-q(d)] = -d[q(d)+O(q^2)] \tag{23}$$

or:

$$q(d) = (1/d)\log 1/q(d) \tag{24}$$

By repeated iteration, the solution of (24) can be written:

$$q(d) = 1/d[\log d - \log\log d + \log\log\log d - \cdots] \tag{25}$$

from which we see that for large d:

$$q(d) = \frac{\log d}{d} + O\left(\frac{\log\log d}{d}\right). \tag{26}$$

This result was also shown by Baudet [1] using a slightly different method. Substituting (26) in (19), the asymptotic behavior of $r_{\text{SOLVE}}(d, P^*)$ becomes:

$$r_{\text{SOLVE}}(d, P^*) = \frac{d}{\log d}\left[1 + O\left(\frac{\log\log d}{\log d}\right)\right]. \tag{27}$$

the log–log graph of Fig. 4 depicts $r_{\text{SOLVE}}(d, P^*)$ for the range $2 \leq d \leq 10{,}000$. It is shown to be in remarkable agreement with the formula $(0.925)d^{0.74741}$, while the asymptotic expression $d/\log d$ becomes a better approximation only for $d > 5000$.

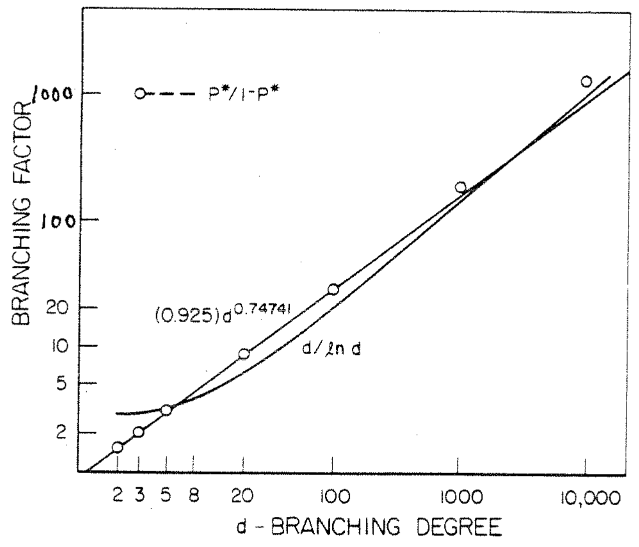


FIG. 4. Worst case branching-factor for the SOLVE algorithm

$$\left[r_{\text{SOLVE}}(d, P^*) = \frac{P^*(d)}{1-P^*(d)}\right].$$

4. Solving, testing, and evaluating game trees

When the terminal positions are assigned real values, the root-node must be *evaluated* rather than *solved*. The SOLVE algorithm discussed in Section 3 is insufficient to fully evaluate a (h, d, F_{V_0})-game tree because it produces a binary WIN–LOSS outcome rather than the (real) minimax value $V(S)$ of the root-node. It can, however, be used to test the proposition '$V(S) > v$', where v is any fixed reference value chosen for the test. We simply interpret any terminal node t for which $V_0(t) > v$ as a WIN position (otherwise it is a LOSS), and apply SOLVE directly. If it issues a WIN, the proposition '$V(S) > v$' is proven, otherwise we deduce '$V(S) \leqslant v$'. This procedure, which we call TEST$(S, v, >)$, is described in algorithmic details in Fig. 5. An almost identical algorithm, TEST$(S, v, \geqslant)$, could be used to test whether $V(S) \geqslant v$ by simply permitting equality in all the comparison tests of Fig. 5.

From the structural identity of SOLVE and TEST, it is clear that the expected number of nodes inspected by TEST, $I_{\text{TEST}}(h, d, F_{V_0}, v)$, is equal to that inspected by SOLVE if the terminal WIN labels are assigned with probability $P_0 = P[V_0(t) > v] = 1 - F_{V_0}(v)$. Therefore:

$$I_{\text{TEST}}(h, d, F_{V_0}, v) = I_{\text{SOLVE}}(h, d, 1 - F_{V_0}(v)). \tag{28}$$

(28), combined with (19), yields:

THEOREM 3. *The expected number of terminal positions examined by the* TEST *algorithm in testing the proposition '$V(S) > v$' for the root of a (h, d, F_{V_0})-tree, has a branching-factor*:

$$r_{\text{TEST}}(d, F_{V_0}, v) = \begin{cases} d^{1/2} & \text{if } v \neq v^*, \\ \dfrac{P^*}{1 - P^*} & \text{if } v = v^* \end{cases} \tag{29}$$

where v^* satisfies $F_{V_0}(v^*) = 1 - P^*$.

From the fact that TEST is directional and SOLVE is optimal we can also conclude:

COROLLARY 4. *The optimal branching factor of any directional algorithm which tests whether the root node of a (h, d, F_{V_0})-tree exceeds a specified reference v is given by $r_{\text{TEST}}(d, F_{V_0}, v)$ in (29).*

Note that when the terminal values are continuous (and $F_{V_0}(v)$ strictly increasing) Theorem 1 states that $V(S)$ converges to v^* for very large h. Thus, although testing the proposition '$V(S) > v$' is easier for $v \neq v^*$, the outcomes of such tests are almost trivial. The most informative test is that which verifies whether $V(S) > v^*$, and such a test, according to (29) is indeed the hardest.

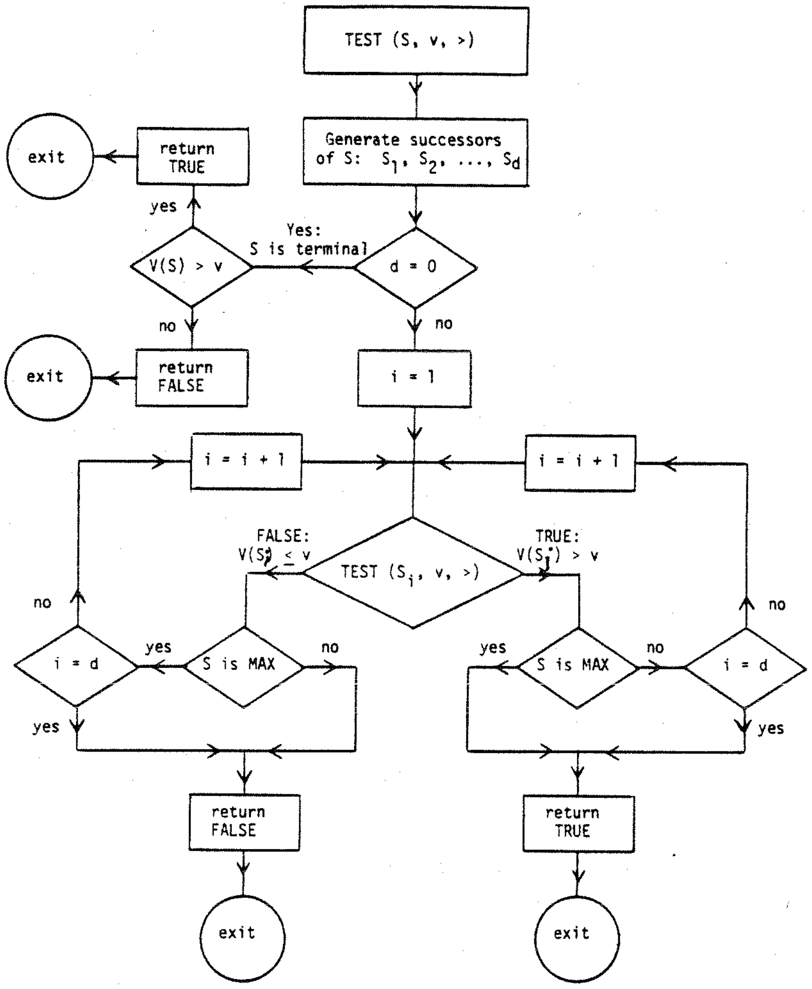


FIG. 5. A flow-chart of the TEST(S, v, $>$) procedure which tests whether the minimax value of position S exceeds a reference v.

When the terminal positions are assigned discrete values then unless $1-P^*$ coincides with one of the plateaus of F_{V_0}, the equation $F_{V_0}(v^*)=1-P^*$ would not have a solution, and the limiting root value would converge to the smallest v' satisfying $F_{V_0}(v')>1-P^*$. Thus all inequality propositions could be tested with a branching factor $d^{1/2}$.

Consider now the minimum number of terminal node examinations required to *evaluate* a game tree. At the best possible case, even if someone hands us for free the true value of S, any evaluation algorithm should be able to defend the proposition '$V(S)=v$' i.e., to defend the pair of propositions '$V(S)\geq v$'

and '$V(S) \leqslant v$'. Since the solution tree required for the verification of an inequality proposition contains $d^{h/2}$ terminal positions and since the sets of terminal positions participating in the defense of each of these inequalities are mutually exclusive, save for the one position satisfying $V_0(t) = V(S)$, we have:

COROLLARY 5. *Any procedure which evaluates a (h, d, F_{V_0})-tree must examine at least $2d^{h/2} - 1$* terminal nodes.

We assumed, of course, that the probability of two or more terminal nodes satisfying $V_0(t) = V(S)$ is zero, and that h is even. This result (in a slightly different form) was also proven by Knuth and Moore [3]. Earlier, Slagel and Dixon [6] proved that the α–β procedure achieves this optimistic bound if the successor positions are perfectly ordered.

Let us consider now the more interesting question of estimating $I(h, d, F_{V_0})$, the *expected* number of terminal examinations required for evaluating (h, d, F_{V_0})-game trees. Let $I_D(h, d)$ be the minimal value of $I(h, d, F_{V_0})$ achieved by any directional algorithm under the worst-case F_{V_0}.

$$I_D(h, d) \overset{\Delta}{=} \min_{\substack{A \\ \text{directional}}} \max_{F} I_A(h, d, F). \tag{30}$$

Every algorithm which evaluates a game tree must examine at least as many nodes as that required for testing whether the root value is greater than some reference v. This is so because an evaluation procedure produces a more informative outcome than any inequality test, and moreover, one can always use the value $V(S)$ to deduce all inequality propositions regarding $V(S)$. This fact combined with the optimality of TEST over the class of directional algorithms (see Corollary 4) leads to:

$$I_D(h, d) \geqslant \left(\frac{P^*}{1 - P^*}\right)^h. \tag{31}$$

The right-hand side of (31) is obtained when the terminal positions are assigned continuous values and TEST is given the task of verifying '$V(S) > v^*$'. This leads directly to:

THEOREM 4. *The expected number of terminal positions examined by any directional algorithm which evaluates a (h, d)-game tree with continuous terminal values must have a branching factor greater or equal to $P^*/(1 - P^*)$.*

The quantity $P^*/(1 - P^*)$ was shown by Baudet [1] to be a lower bound for the branching factor of the α–β procedure. Theorem 4 extends the bound to all directional game-evaluating algorithms.

In the next section we will present a straightforward evaluation algorithm called SCOUT which actually achieves the branching factor $P^*/(1 - P^*)$, thus establishing the asymptotic optimality of SCOUT over the class of directional algorithms. including the α-β procedure.

5. Test and, if necessary, evaluate—the SCOUT algorithm

SCOUT evaluates a MAX position S by first evaluating its left most successor S_1, then 'scouting' the remaining successors, from left to right, to determine if

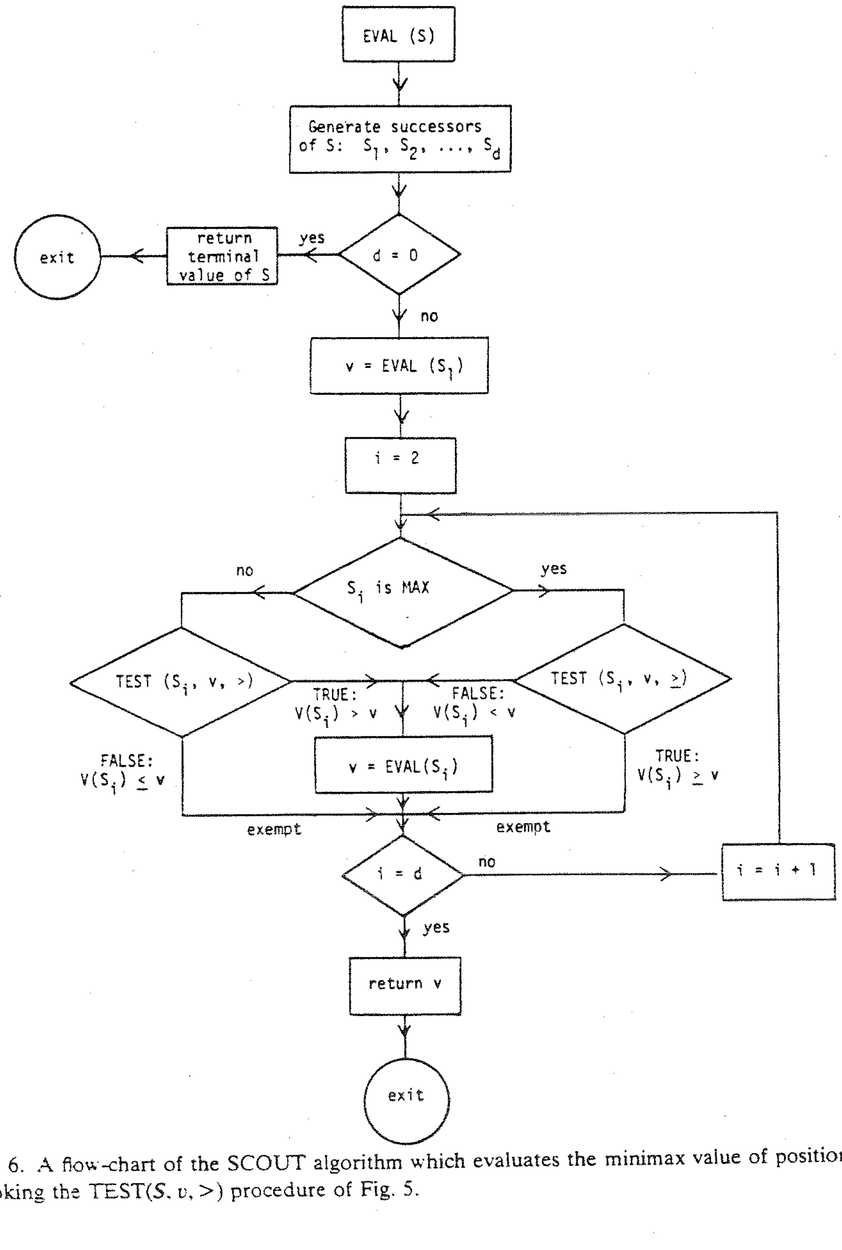


FIG. 6. A flow-chart of the SCOUT algorithm which evaluates the minimax value of position S by invoking the TEST(S, v, $>$) procedure of Fig. 5.

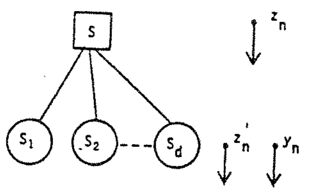
any meets the condition $V(S_k) > V(S_1)$. If the inequality is found to hold for S_k, this node is then evaluated exactly and its value $V(S_k)$ is used for subsequent 'scoutings' tests. Otherwise S_k is exempted from evaluation and S_{k+1} selected for a test. When all successors have been either evaluated or tested and found unworthy of evaluation, the last value obtained is issued as $V(S)$. An identical procedure is used for evaluating a MIN position S, save for the fact that the event $V(S_k) \geq V(S_1)$ now constitutes grounds for exempting S_k from evaluation. The flow-chart of Fig. 6 describes SCOUT in algorithmic details, calling on the TEST algorithm of Fig. 5 to perform the inequality checks.

At first glance it appears that SCOUT is very wasteful; any node S_k which is found to fail a test criterion is submitted back for evaluation. The terminal nodes inspected during such a test may (and in fact will) be revisited during the evaluation phase. An exact mathematical analysis, however, reveals that the amount of waste is not substantial and that SCOUT, in spite of some duplicated effort, still achieves the optimal branching factor $P^*/(1-P^*)$.

Two factors work in favor of SCOUT: (1) Most tests would result in exempting the tested node (and all its descendents) from any further evaluation, and (2) testing for inequality using the TEST(S, v) procedure is relatively speedy. In the worst possible case TEST only consumes an average of $(P^*/(1-P^*))^h$ inspections which according to (31) is below the average consumption of the best directional evaluation procedure. The superiority of TEST stems from the fact that it induces many cutoffs not necessarily permitted by EVAL or any other evaluation scheme. As soon as *one* successor of a MAX node meets the criterion $V(S_k) > v$, all other successors can be ignored. EVAL, by contrast, would necessitate a further examination of the remaining successors to determine if any would possess a value higher than $V(S_k)$.

6. Analysis of SCOUT's expected performance

Let S be a MAX node rooting an n-cycle tree $(h = 2n)$ with a uniform branching degree d. Let z_n denote the expected number of terminal examinations undertaken by SCOUT. These examinations consist of those performed during the EVAL(S_k) phases $(k = 1, \dots, d)$ plus those performed during the TEST$(S_k, v, >)$ phases $(k = 2, \dots, d)$. Since the subtrees emanating from the successors of S all have identically distributed terminal values, the number of



positions examined in each EVAL(S_k) phase have identical expectations, called z'_n. Let v_k be the test criterion during the TEST$(S_k, v, >)$ phase, and let $y^+_n(k)$

and $y_n^-(k)$ have the same interpretations as in Section 3. The event that S_k is found to satisfy the criterion $V(S_k) > v_k$ would consume a mean expenditure of $y_n^+(k) + z_n'$ inspections while a successor found to refute this test would consume, on the average, only $y_n^-(k)$ inspections. Thus, if q_k stands for the probability that successor S_k would require an evaluation, we have:

$$\begin{aligned} z_n &= z_n' + \sum_{k=2}^{d} q_k(z_n' + y_n^+(k)) + \sum_{k=2}^{d} (1-q_k)y_n^-(k) \\ &= z_n'[1 + \sum_{k=2}^{d} q_k] + \sum_{k=2}^{d} y_n(k). \end{aligned} \tag{32}$$

Since S_k would require an evaluation iff $V(S_k) > \max[V(S_1), V(S_2), \ldots, V(S_{k-1})]$ and all the node-values at any given level are independent, identically distributed and continuous random variables, we have:

$$q_k = \frac{1}{k}, \qquad k = 2, \ldots, d. \tag{33}$$

Moreover, since we are interested in a worst case analysis, each $y_n(k)$ can be replaced by its highest possible value. This occurs when the probability that any given terminal position t satisfies $V(t) > v_k$ is equal to the fixed point probability P^*. From (13) and (14) $y_n(k)$, in such a case, would be given by $(P^*/(1-P^*))^{2n-1}$ and we can write:

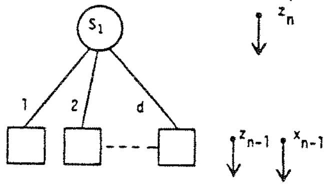
$$z_n = z_n'\zeta(d) + \left(\frac{P^*}{1-P^*}\right)^{2n-1}(d-1) \tag{34}$$

where

$$\zeta(d) = \sum_{k=1}^{d} \frac{1}{k}. \tag{35}$$

Note that this approximation is not too pessimistic in light of the fact that for large n the values of all nodes converge rapidly toward the limiting value v^* and, therefore, most tests would employ a threshold level v_k from the neighborhood of v^*.

To compute the solution of (34) we now examine the expected number of inspections z_n' employed while evaluating any of the successors of S, say S_1. Since S_1 is a MIN position a successor would be submitted for evaluation iff its



value is proven to be *below* the threshold level propagating from the left. Each evaluation would require an average of z_{n-1} inspections and each test would consume at most an average of $[x_{n-1} \approx (P^*/(1-P^*))^{2n-2}]$ terminal inspections. Consequently, using an argument similar to the one above, we obtain:

$$z'_n = z_{n-1}\zeta(d) + \left(\frac{P^*}{1-P^*}\right)^{2n-2}(d-1) \tag{36}$$

which, combining (34) and (36), yields:

$$z_n = z_{n-1}\zeta^2(d) + (d-1)\left(\frac{P^*}{1-P^*}\right)^{2n-2}\left[\zeta(d) + \left(\frac{P^*}{1-P^*}\right)\right]. \tag{37}$$

(37) is a linear difference equation of the form:

$$z_n = \alpha z_{n-1} + K\beta^n \tag{38}$$

with

$$\begin{aligned} z_0 &= 1, \\ K &= (d-1)\left(\frac{1-P^*}{P^*}\right)^2\left[\zeta(d) + \left(\frac{P^*}{1-P^*}\right)\right], \\ \beta &= \left(\frac{P^*}{1-P^*}\right)^2, \\ \alpha &= \zeta^2(d). \end{aligned} \tag{39}$$

Its solution is given by:

$$z_n = \alpha^n + K\beta\,\frac{\beta^n - \alpha^n}{\beta - \alpha}. \tag{40}$$

Clearly, it is the relative size of α and β which governs the asymptotic behavior of z_n for large values of n. However, since for all d we have:

$$\frac{P^*(d)}{1-P^*(d)} > \zeta(d)$$

(e.g., for $d \to \infty$, $P^*/(1-P^*) = O(d/\log d)$ while $\zeta(d) = O(\log d)$) β would become the dominant factor, and we can write:

$$z_n \sim \frac{K}{\beta - \alpha}\beta^{n+1} \tag{41}$$

or equivalently (with $h = 2n$):

$$I_{\text{SCOUT}}(h, d, F) \sim \frac{(d-1)}{\dfrac{P^*}{1-P^*} - \zeta(d)}\left(\frac{P^*}{1-P^*}\right)^h. \tag{42}$$

THEOREM 5. *The expected number of terminal examinations performed by* SCOUT *in the evaluation of (h, d)-game trees with continuous terminal values has a branching factor*:

$$r_{\text{SCOUT}} = \frac{P^*}{1-P^*}. \tag{43}$$

So far, our analysis was based on the assumption that the terminal nodes may be assigned continuous values. We will now demonstrate that I_{SCOUT} is substantially reduced if the terminal nodes are assigned only discrete values.

Let's ignore the rare case where $1-P^*$ coincides exactly with one of the plateaus of F. When coincidence does not occur, we showed in Section 2 that the values of all nodes at sufficiently high levels converge to the same limit, given by the lowest terminal value v' satisfying $F_{V_0}(v') > 1-P^*$. This convergence has two effects on the complexity of SCOUT as analyzed in (32): first, q_k is no longer equal to $1/k$ but rather, converges to zero at high n for all $k > 1$. The reason for this is that in order for $V(S_k)$ to be greater than $V(S_1)$ (which is most probably equal to v') it must exceed $V(S_1)$ by a finite positive quantity and, at a very high h, finite differences between any two nodes are extremely rare. Second, the threshold levels v_k against which the TEST$(S_k, v, >)$ procedures are performed are no longer close to v^* but differ from it by finite amounts. Under such conditions the proposition '$V(S_k) > v_k$' can be tested more efficiently since $r_{\text{TEST}} = d^{1/2}$ (see (29)).

Applying these considerations to the analysis of z_n in (32) gives:

$$r_{\text{SCOUT}} \sim d^{1/2} \tag{44}$$

and we obtain:

THEOREM 6. *The expected number of terminal positions examined by the* SCOUT *procedures in evaluating a (h, d, F_{V_0})-game with discrete terminal values has a branching factor*:

$$r_{\text{SCOUT}} = d^{1/2} \tag{45}$$

with exceptions only when one of the discrete values, v^*, satisfies $F_{V_0}(v^*) = 1-P^*$.

COROLLARY 6. *For games with discrete terminal values satisfying the conditions of Theorem 6, the* SCOUT *procedure is asymptotically optimal over all evaluation algorithms.*

Of course, the transition from $r_{\text{SOLVE}} = P^*/(1-P^*)$ in the continuous case to $r_{\text{SOLVE}} = d^{1/2}$ in the discrete case does not occur abruptly. When the quantization levels are very close to each other it takes many more levels before SCOUT begins to acquire the lower branching factor of $d^{1/2}$. In fact, using the discussion of Section 1, it is possible to compute at what height SCOUT begins to act more efficiently. For example, if the terminal values are integers, uniformly distributed from 1 to M, we know that at very high levels of the tree the values of all nodes will converge to I^*, where I^* is the lowest integer satisfying $F_{V_0}(I^*) > 1 - P^*$. The probability that a node n cycles away from the bottom would acquire this value is:

$$P[V_n(S) = I^*] = F_{V_n}(I^*) - F_{V_n}(I^* - 1). \tag{46}$$

If $F_{V_n}(I^*)$ and $F_{V_n}(I^* - 1)$ are very close to each other, $P[V_n(S) = I^*]$ will be governed by the linear regions of the curves in Fig. 2. Therefore, we can write:

$$P[V_n(S) = I^*] \approx \frac{\mathrm{d}F_{V_n}(v)}{\mathrm{d}F_{V_0}(v)}\bigg|_{F_{V_0}=1-P^*} [F_{V_0}(I^*) - F_{V_0}(I^* - 1)]$$

and, using (5) and (10):

$$P[V_n(S) = I^*] \cong \left[\frac{d(1-P^*)}{P^*}\right]^{2n} 1/M. \tag{47}$$

In order for the TEST procedures nested in SCOUT to achieve a branching factor of $d^{1/2}$ the parameter P_{n-1} appearing in (15) must be sufficiently close to zero. But this is achieved when $P[V_n(S) = I^*]$ approaches unity, i.e., when h satisfies:

$$h \geqslant \frac{\log M}{\log \dfrac{d(1-P^*)}{P^*}} \overset{\Delta}{=} h_0(M, d). \tag{48}$$

Thus, above the critical level $h_0(M, d)$ it becomes fairly sure that every node has a minimax value I^* and, consequently, the SCOUT procedure would have to expand only $d^{1/2}$ nodes per level. Note that the critical height increases logarithmically with the number of quantization levels M.

Several improvements could be applied to the SCOUT algorithm to render it more efficient. For example, when a TEST procedure issues a non-exempt verdict, it could also return a new reference value and some information regarding how the decision was obtained in order to minimize the number of nodes to be inspected by EVAL. The main reasons for introducing SCOUT have been its conceptual and analytic simplicity and the fact that it possesses the lowest branching factor of any algorithm known to date. However, the potential of SCOUT as a practical game-searching procedure should not be dismissed altogether. Recent simulation studies using he game of Kalah show[2]

[2]Peter Homeier, personal communication.

that the efficiency of SCOUT, even in its unpolished version, compares favorably with that of the α–β procedure.

7. On the branching factor of the ALPHA–BETA (α–β) procedure

The reader is assumed to be familiar with the basic features of the ALPHA–BETA (α–β) pruning method. Descriptions of the method can be found in the text-books by Nilsson [4, Section 4] and Slagel [5, pp. 16–24]. A historical survey of the development of the concept is given by Knuth and Moore [3, Section 5].

The fact that the number of terminal nodes examined by α–β may vary from $(d^{\lfloor h/2 \rfloor} + d^{\lceil h/2 \rceil} - 1)$ to d^h was shown by Slagle and Dixon [6] and elaborated by Knuth and Moore [3].

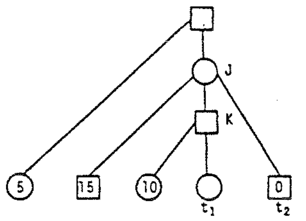
The analysis of expected performance using uniform trees with random terminal values has begun with Fuller *et al.* [2] who obtained formulas by which the average number of terminal examinations can be computed. Unfortunately, the formulas are very complicated and would not facilitate an asymptotic analysis. Simulation studies conducted by Fuller *et al.* led to the estimate:

$$r_{\alpha\text{-}\beta} \approx d^{0.72}.$$

Knuth and Moore [3] have analyzed a less powerful but simpler version of the α–β procedure by ignoring deep cutoffs. They have shown that the branching factor of this simplified model is $O(d/(\log d))$ and speculated that the inclusion of deep cutoffs would not alter this behavior substantially. However, the gap between the upper and lower bounds for the branching factor remained appreciable, even for the simplified model.

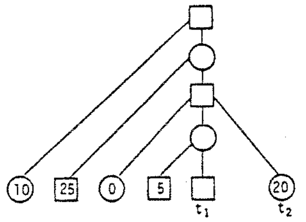
A more recent paper by Baudet [1] contains several improvements. Starting by considering possible equalities between terminal values, Baudet derived a general formula for $I_{\alpha\text{-}\beta}$ (deep cutoffs included) from which the branching factor can be estimated. In particular, Baudet shows that for bivalued terminal positions $r_{\alpha\text{-}\beta}$ could be as high as $P^*/(1-P^*)$ (a special case of (19)); and for the continuous case that $r_{\alpha\text{-}\beta}$ is lower bounded by $r_{\alpha\text{-}\beta} \geqslant P^*/(1-P^*)$ (a special case of (31)). A tighter upper bound for $r_{\alpha\text{-}\beta}$ was then computed which significantly narrowed the gap left by Knuth and Moore to less than 20% in the range $2 \leqslant d \leqslant 32$.

In view of the fact that the SCOUT algorithm was found to achieve the lower bound $P^*/(1-P^*)$, we were first led to believe that α–β, which appears to be much more economical than SCOUT, also achieves this bound and that the uncertainty concerning the actual branching factor of α–β has finally been eliminated. However, after several futile attempts to prove the superiority of α–β over SCOUT, we found counter-examples demonstrating that SCOUT's extra caution in testing prior to evaluation may sometimes pay off, causing it to skip nodes which would be visited by α–β. In the diagram below, the node marked t_1 would be examined by the α–β procedure but ignored by SCOUT.



When J is submitted to the test TEST(J, 5, >), the zero value assigned to node t_2 causes the test to fail, whereas during the TEST(K, 5, >) phase, t_1 is skipped by virtue of its elder sibling having the value 10. α–β, on the other hand, has no way of finding out the low value of t_2 before t_1 is examined.

The converse situation can, of course, also be demonstrated. The diagram below shows how a node (t_1) which is visited by SCOUT is cut off by α–β. However, the asymptotic performance of SCOUT is at least as good as that of



α–β by virtue of Theorem 4 and the fact that α–β is directional.
Our inability to demonstrate the asymptotic equivalence of SCOUT and α–β on a node by node basis leaves the branching factor of the α–β procedure enigmatic and renders its asymptotic optimality unsettled. We wish to conjecture, though, that α–β probably does reach the branching factor $P^*/(1-P^*)$ and that it is, therefore, asymptotically optimal over all directional algorithms. It would simply be too amusing to find a wasteful procedure such as SCOUT outperforming the α–β procedure.[3]

However, the uncertainty regarding the branching factor of the α–β procedure only pertains to continuous valued trees. We shall next demonstrate that when the terminal positions are assigned discrete values, the α–β procedure attains the absolute minimal branching factor of $d^{1/2}$.

The fact that at high levels almost all nodes attain the same minimax value,

[3]This conjecture has recently been confirmed (see Pearl, J., "The Solution for the Branching Factor of the Alpha–Beta Pruning Algorithm," UCLA-ENG-CSL-8019, School of Engineering and Applied Science, University of California, Los Angeles, May 1980).

v^*, makes it increasingly probable that the α–β cutoff conditions[4] are met successfully at all nodes where they are applicable and this, in turn, gives rise to a branching factor of $d^{1/2}$. For, consider the top m cycles of a $(n+m)$-cycle tree, if all cutoff conditions are met at this portion of the tree, only $2d^m - 1$ nth level nodes need be expanded. Therefore, denoting by x_n the expected number of terminal positions examined by α–β in evaluating any nth level node, and by $P_{\alpha\text{-}\beta}(n, m)$ the probability that all cutoff conditions are satisfied whenever applicable, we can write:

$$\frac{x_{n+m}}{x_n} \leq (2d^m - 1)P_{\alpha\text{-}\beta}(n, m) + d^{2m}[1 - P_{\alpha\text{-}\beta}(n, m)]. \tag{49}$$

On the other hand, the event of meeting all cutoff conditions is subsumed by the event that all the $2d^m - 1$ nodes expanded attain the limit value v^*, and consequently:

$$P_{\alpha\text{-}\beta}(n, m) \geq P[V_{n-1}(S) = v^*]^{(2d^m-1)}.$$

Now, letting $m = n^2$ and recalling (6) that $P[V_{n-1}(S) = v^*]$ approaches unity at a super exponential rate:

$$1 - P[V_{n-1}(s) = v^*] \leq (\delta)^{d^{n-n_0}} \qquad \text{for } n > n_0$$

where δ is a fraction strictly smaller than 1, and n_0 a function of δ, $F_{V_0}(v_i)$ and $F_V(v_{i-1})$ (see Theorem 1), we obtain:

$$\lim_{n\to\infty} d^{2m}[1 - P_{\alpha\text{-}\beta}(n, m)] = 0$$

and from (49):

$$\frac{x_{n+n^2}}{x_n} = O(2d^{n^2}).$$

The effective branching factor for the entire $(n+n^2)$-cycle tree, even assuming that every node expanded at the nth cycle requires the examination of all d^{2n} terminal nodes under it, becomes:

$$r_{\alpha\text{-}\beta} = \lim_{n\to\infty} [2d^{n^2} \cdot d^{2n}]^{1/2(n+n^2)} = d^{1/2}.$$

We summarize this result by stating:

THEOREM 7. *The expected number of terminal positions examined by the* ALPHA–BETA *procedure in evaluating a* (h, d, F)*-game with discrete terminal values has a branching factor* $r_{\alpha-\beta} = d^{1/2}$ *with exceptions only when one of the discrete values,* v^*, *satisfies* $F(v^*) = 1 - P^*$.

[4][2] contains an elaborate description of the α–β cutoff conditions, using a notation similar to ours.

Corollary 7. *For games with discrete terminal values satisfying the conditions of Theorem 7, the α–β procedure is asymptotically optimal over all evaluation algorithms.*

Paralleling our discussion of the SCOUT algorithm, α–β too does not acquire the more efficient branching factor of $d^{1/2}$ by an abrupt transition from the continuous to the discrete case. If the terminal values are drawn from M equally likely integers, (48) provides an estimate for the height $h_0(M, d)$ at which the search would become more efficient. Note, however, that it is not the total number of quantization levels $v_1, v_2, \ldots, v_M$ which affects the search efficiency but rather the distances of $F_{V_0}(v_i)$ and $F_{V_0}(v_{i-1})$ from $1-P^*$. Thus, coarser quantizations in the neighborhood of v^* have a more significant role in speeding up the α–β procedure.

Recently, Stockman [7] has introduced a non-directional algorithm which examines fewer nodes than α–β. The magnitude of this improvement has not been evaluated yet, but the superiority of Stockman's algorithm could be one of the following two types. It may either possess a reduced branching factor. or it may exhibit a marginal improvement at low h's which disappears on taller trees. If the superiority is of the former type, it must be singular to the continuous case because in the discrete case, Corollary 7 states that α–β it asymptotically optimal over all algorithms, directional as well as non-directional.

It would still be interesting, though, to find out if any non-directional algorithm can solve a (h, d, P^*)-game with branching factor lower than $P^*/(1-P^*)$. If such an algorithm exists it could be incorporated into SCOUT (replacing TEST) and thus enabling it to evaluate continuous valued game trees with a branching factor lower than $P^*/(1-P^*)$.

REFERENCES

1. Baudet, G.M., On the branching factor of the alpha–beta Pruning algorithm, *Artificial Intelligence* **10** (1978) 173–199.
2. Fuller, S.H., Gaschnig, J.G. and Gillogly, J.J., An analysis of the alpha–beta Pruning algorithm. Department of Computer Science Report, Carnegie-Mellon University (July 1973).
3. Knuth, D.E. and Moore, R.N., An analysis of alpha–beta Pruning, *Artificial Intelligence* **6** (1975) 293–326.
4. Nilsson, N.J., *Problem-Solving Methods in Artificial Intelligence* (McGraw-Hill, New York. 1971).
5. Slagle, J.R., *Artificial Intelligence: The Heuristic Programming Approach* (McGraw-Hill, New York, 1971).
6. Slagle, J.R. and Dixon, J.K., Experiments with some programs that search game trees, *J. ACM* **2** (1969) 189–207.
7. Stockman, G., A minimax algorithm better than alpha–beta?, *Artificial Intelligence* **12** (1979) 179–196.

Received 2 January 1980; revised version received 12 March 1980