UCLA-ENG-CSL-8022
April 1980

SCOUT: A SIMPLE GAME-SEARCHING ALGORITHM WITH PROVEN OPTIMAL PROPERTIES

Judea Pearl
Cognitive Systems Laboratory
School of Engineering and Applied Science
University of California, Los Angeles

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

SCOUT: A SIMPLE GAME-SEARCHING ALGORITHM WITH PROVEN OPTIMAL PROPERTIES

Judea Pearl
Cognitive Systems Laboratory
School of Engineering and Applied Science
University of California
Los Angeles, California 90024

ABSTRACT

This paper describes a new algorithm for searching games which is conceptually simple, space efficient, and analytically tractable. It possesses optimal asymptotic properties and may offer practical advantages over α-β for deep searches.

I. INTRODUCTION

We consider a class of two-person perfect information games in which two players, called MAX and MIN, take alternate turns in selecting one out of d legal moves. We assume that the game is searched to a depth h, at which point the terminal positions are assigned a static evaluation function V_0. The task is to evaluate the minimax value, V_h, of the root node by examining, on the average the least number of terminal nodes.

SCOUT, the algorithm described in this paper, has evolved as a purely theoretical tool for analyzing the mean complexity of game-searching tasks where the terminal nodes are assigned random and independent values [1]. With the aid of SCOUT we were able to show that such games can be evaluated with a branching factor of $P^*/(1-P^*)$, where P^* is the root of $x^d+x-1 = 0$, and that no directional algorithm (e.g., ALPHA-BETA) can do better. We have recently tested the performance of SCOUT on a 'real' game (i.e., the game of Kalah) and were somewhat surprised to find that, even for low values of h, the efficiency of SCOUT surpasses that of the α-β procedure [2]. The purpose of this paper is to call attention of game-playing practitioners to the potentials of SCOUT as a practical game-searching tool.

Section II describes the operation of SCOUT in conceptual terms avoiding algorithmic details. Section III presents, without proofs, some of the mathematical properties of SCOUT and compares them to those of the α-β procedure. Finally empirical results are reported comparing the performances of SCOUT and α-β for both random and dynamic orderings.

* Supported in part by NSF Grants MCS 78-07468 and MCS 78-18924.

II. THE SCOUT ALGORITHM

SCOUT invokes two recursive procedures called EVAL and TEST. The main procedure EVAL(S) returns V(S), the minimax value of position S, whereas the function of TEST(S, v, >) is to validate (or refute) the truth of the inequality $V(S) > v$ where v is some given reference value.

Procedure: TEST(S, v, >)

To test whether S satisfies the inequality $V(S) > v$, start applying the same test (calling itself) to its successors from left to right:

If S is MAX, return TRUE as soon as one successor is found to be larger than v; return FALSE if all successors are smaller than or equal to v.

If S is MIN, return FALSE as soon as one successor is found to be smaller than or equal to v; return TRUE if all successors are larger than v.

An identical procedure, called TEST(S, v, $\geq$), can be used to verify the inequality $V(S) \geq v$, with the obvious revisions induced by the equality sign.

Procedure: EVAL(S)

EVAL evaluates a MAX position S by first evaluating (calling itself) its left most successor S_1, then 'scouting' the remaining successors, from left to right, to determine (calling TEST) if 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 (calling EVAL(S_k)) 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. Flow-charts describing both SCOUT and TEST in algorithmic details can be found in [1].

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 general 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 $P^*/(1-P^*)$, as will be demonstrated in Section III.

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. The speed 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 $\overline{V(S_k)} > v$, 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 $V(S_k)$.

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. However, the analysis presented in Section III, as well as the simulation tests, were conducted on the original version described above. These studies show that, even in its unpolished form, SCOUT is asymptotically optimal over all directional algorithms and is somewhat more efficient than the α-β procedure for the game tested (i.e., Kalah).

Recently, Stockman [3] has also introduced an algorithm which examines fewer nodes than α-β. However, Stockman's algorithm requires an enormous storage space for tracing back a large number of potential strategies. SCOUT, by contrast, has storage requirements similar to those of α-β; at any point in time it only maintains pointers along one single path connecting the root to the currently expanded node.

III. ANALYSIS OF SCOUT'S EXPECTED PERFORMANCE

In this section we present, without proofs, some mathematical results related to the expected number of nodes examined by SCOUT and α-β. Additional results, including proofs, can be found in reference [1]. The model used for evaluating these algorithms consists of a uniform tree of height h (h even) and branching factor d, where the terminal positions are assigned random values, independently drawn from a common distribution F. We shall refer to such a tree as a (h, d, F)-tree.

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: $v_1 < v_2 < \ldots < v_M$, then the root value converges to a definite limit iff $1-P^* \neq F(v_i)$ for all i, in which case the limit is the smallest v_i satisfying $1-P^* < F(V_i)$.

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

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

is called the branching factor corresponding to the algorithm A.

Definition: Let C be a class of algorithms capable of searching a general (h, d, F)-tree. An algorithm A is said to be asymptotically optimal over C if for all d, F, and $B \in C$, $r_A(d, F) \leq r_B(d, F)$.

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.

Theorem 2: 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)-tree, has a branching factor $d^{1/2}$ if $v \neq v^*$ and $P^*/(1-P^*)$ if $v = v^*$, where v^* satisfies $F(v^*) = 1-P^*$ and P^* is the root of $x^d+x-1 = 0$.

Theorem 3: TEST is asymptotically optimal over all directional algorithms which test whether the root node of a (h, d, F)-tree exceeds a specified reference v.

Corollary 1: Any procedure which evaluates a (h, d, F)-tree must examine at least $2d^{h/2}-1$ nodes.

Corollary 2: 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 [3] to be a lower bound for the branching factor of the α-β procedure. Corollary 2 extends the bound to all directional game-evaluating algorithms.

Theorem 4: 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 of $P^*/(1-P^*)$.

Theorem 5: The expected number of terminal examinations performed by SCOUT in evaluating a (h, d, F)-game with discrete terminal values has a branching factor $d^{1/2}$, with exceptions only when one of the discrete values, v^*, satisfies $F(v^*) = 1-P^*$.

Corollary 3: For games with discrete terminal values satisfying the conditions of Theorem 5, the SCOUT procedure is asymptotically optimal over all evaluation algorithms.

The improvement in efficiency due to the discrete nature of the terminal value manifests itself only when the search depth h is larger than log M/log [d(1-P*)/P*], where M is the quantization density in the neighborhood of $V_0 = v^*$.

The branching factor of α-β is less tractable than that of SCOUT. At the time this paper was first written the tightest bounds on $r_{\alpha-\beta}$ were those delineated by Baudet [3] giving the lower bound $r_{\alpha-\beta} \geq P^*/(1-P^*)$ (a special case of Corollary 2) and an upper bound which is about 20 percent higher over the range $2 \leq d \leq 32$. Thus, SCOUT was the first algorithm known to achieve the bound P*/(1-P*) and we were questioning whether α-β would enjoy a comparable asymptotic performance. Moreover, it can be shown that neither SCOUT nor α-β dominate one another on a node-by-node basis; i.e., nodes examined by SCOUT may be skipped by α-β and vice versa [1].

The uncertainty regarding the branching factor of the α-β procedure has recently been resolved [5]. Evidently, α-β and SCOUT are asymptotically equivalent; $r_{\alpha-\beta}$ equals P*/(1-P*) for continuous valued trees and $d^{1/2}$ for games with discrete values.

For low values of h the branching factor is no longer an adquate criterion of efficiency and the comparison between SCOUT and α-β must be done empirically. The following table represents the number of node inspections spent by both algorithms on the game of Kalah (1-in-a-hole version) [2]:

It appears that as the search depth increases SCOUT offers some advantage over α-β. Experiments with higher numbers of stones in each hole indicate that this advantage may deteriorate for large d. We suppose, therefore, that SCOUT may be found useful in searching games with high h/d ratios.

REFERENCES

[1] Pearl, J. "Asymptotic Properties of Minimax Trees and Game-Searching Procedures." UCLA-ENG-CSL-7981, University of California, Los Angeles, March 1980, to be published in Artificial Intelligence.

[2] Noe, T. "A Comparison of the Alpha-Beta and SCOUT Algorithms Using the Game of Kalah." UCLA-ENG-CSL-8017, University of California, Los Angeles, April 1980.

[3] Stockman, G. "A Minimax Algorithm Better Than Alpha-Beta?" Artificial Intelligence 12, 1979, 179-196.

[4] Baudet, G. M. "On the Branching Factor of the Alpha-Beta Pruning Algorithm." Artificial Intelligence 10, 1978, 173-199.

[5] Pearl, J. "The Solution for the Branching Factor of the Alpha-Beta Pruning Algorithm." UCLA-ENG-CSL-8019, University of California, Los Angeles, May 1980.

Search Depth	Random Ordering			Dynamic Ordering		
	SCOUT	α-β	% Improvement	SCOUT	α-β	% Improvement
2	82	70	-17.0	39	37	-5.4
3	394	380	-3.7	62	61	-1.6
4	1173	1322	+11.3	91	96	-1.1
5	2514	4198	+40.1	279	336	+17.0
6	5111	6944	+26.4	371	440	+15.7