A SEARCH PROCEDURE FOR PERFECT INFORMATION GAMES OF CHANCE: ITS FORMULATION AND ANALYSIS

Bruce W. Ballard
Dept. of Computer Science
Duke University
Durham, N.C. 27706

ABSTRACT

An algorithm is developed for searching the trees of "perfect information" games involving chance events. Many dice games (e.g. backgammon, craps, and monopoly and similar board games), and some card games (e.g. casino blackjack), have this property. For depth 3 trees, empirical observation reveals a search reduction of more than 50 percent, while closed-form analysis reveals a best-case complexity of O(N**2). This represents a substantial savings over the O(N**3) behavior of the "obvious" search strategy.

I INTRODUCTION

Many games involving chance events, such as the roll of dice or the drawing of playing cards, can be modeled by introducing "probability" nodes into standard minimax trees. We use the symbols + and - to denote maximizing and minimizing nodes, respectively, and * (pronounced "star") to denote a probability node. We define the value of a * node as the weighted average of the values of its successors, which may occur with differing probabilities. We shall develop and evaluate the performance of an algorithm to search *-minimax trees efficiently. In this paper, we assume that all descendents of a * node are equally likely. The algorithm we present can be extended, in a direct way, to the more general case.

For the most part, *-minimax trees, as we shall call them, retain the properties of ordinary minimax trees. In particular, they pertain to 2-person, 0-sum, perfect information games. By "perfect information" we mean that neither player conceals information about the current state of the game, or possible future states, that could be useful to the other player. Many dice games (e.g. craps, backgammon, and monopoly and similar board games) satisfy these criteria, as do some card games (e.g. casino blackjack).

Figure 1 gives an example *-minimax tree. Backed-up values for non-terminal nodes are shown in parentheses. The value of the * node has been computed as (2 - 4) / 2 = -1.

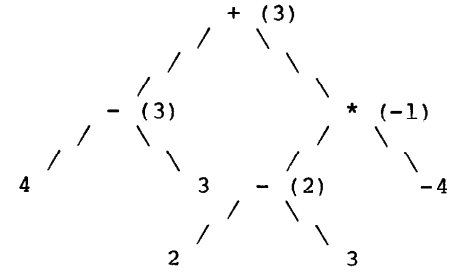

Figure 1 - A Sample *-Minimax Tree

II THE *-MINIMAX SEARCH PROBLEM

In searching *-minimax trees, we want to retain the alpha-beta "cutoff" power of ordinary minimax trees. However, the presence of * nodes provides opportunities for additional forms of cutoffs. Recognizing that lower and upper bounds on the value of a * node can be derived by exploring one or more of its children, we have devised an algorithm which can reduce search complexity by more than 50 percent with random ordering of successor nodes, and by an order of magnitude with optimal ordering.

As an example of a possible "* cutoff", suppose the (leaf) values of a particular tree are integers between 0 and 10, inclusive, and that a * node with 4 equally likely successors has had 2 of its successors searched. This situation is shown in Figure 2.

This research has been partially supported by AFOSR, Air Force Command, AFOSR 81-0221. The author wishes to express appreciation to Dr. Donald Loveland and Tom Truscott for discussing portions of an earlier draft of this paper.

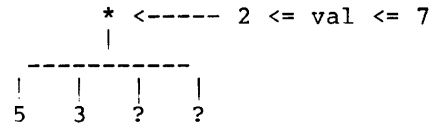

Figure 2 - Interim Bounds on a * Node

Knowing the values of these 2 children, we can say that the smallest possible value for the * node is (5 + 3 + 0 + 0) / 4, or 2. Similarly, the greatest possible value of the * node is (5 + 3 + 10 + 10) / 4, or 7. Thus, a cutoff can occur if the alpha value passed to * is >= 7, or if the beta value is <= 2. We shall formulate a search strategy to take advantage of this form of potential cutoff. In addition, our strategy will compute new alpha and beta values for use below * nodes.

III AN ALGORITHM FOR *-MINIMAX TREES

Let L and U denote lower and upper bounds on all possible game (leaf) values. Let V1, V2, ..., VN be the values of the N successors of a * node, whose i-th successor is about to be searched. After returning from the i-th node, a cutoff will occur if

$$\frac{(V1+...+Vi-1) + Vi + U*(N - i)}{N} <= \text{alpha}$$

or if

$$\frac{(V1+...+Vi-1) + Vi + L*(N - i)}{N} >= \text{beta}$$

Alpha and beta values for the i-th successor are given by

```
Ai = N*alpha - (V1+...+Vi-1) - U*(N - i)
Bi = N*beta  - (V1+...+Vi-1) - L*(N - i)
```

where "alpha" and "beta" are the alpha-beta values of the present * node. These equations suggest that A and B be initialized by

```
A1 = N * (alpha - U) + U
B1 = N * (beta  - L) + L
```

and updated by

```
An+1 = An + U - Vn
Bn+1 = Bn + L - Vn
```

The following Star1 procedure implements this strategy, making use of (1) a Term procedure, to evaluate terminal positions; (2) an Eval procedure which, depending on which player is to move next, invokes either Max or Min; and (3) a procedure to generate the successors of a node.

```
Star1(board, alpha, beta)
{
 local A, B, i, v, vsum, AX, BX, s[];

 determine the N successors s1,s2,...,sN
 if (N == 0)
    return(Term(board));

 A = N * (alpha - U) + U;
 B = N * (beta  - L) + L;
 vsum = 0;
 for (i=1; i<=N; i++) {
     AX = max(A, L);
     BX = min(B, U);
     v = Eval(s[i], AX, BX);
     if (v <= A)
        return(alpha);
     if (v >= B)
        return(beta);
     vsum = vsum + v;
     A = A + U - v;
     B = B + L - v;
 }
 return(vsum / N);
}
```

An example of the way in which * nodes use and create new alpha-beta values is suggested by the partially searched *-minimax tree given in Figure 3.

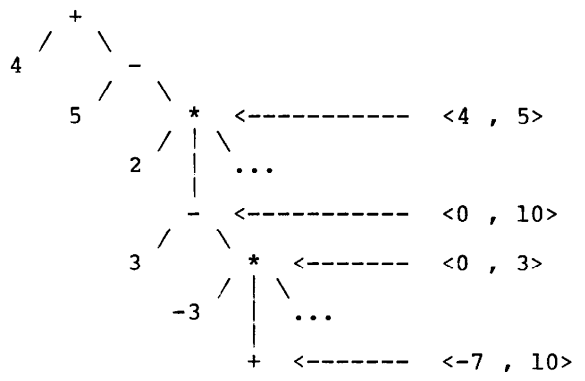

Figure 3 - A Partially Searched *-Minimax Tree
(assuming -10 <= Leaf Values <= +10)

IV AN "IMPROVED" ALGORITHM

In typical *-minimax games, where players take turns in making moves, a given * node has either all + successors or all - successors. Consider a particular * node whose succesors are all - nodes. If this node is worse than a previously searched * node, a preliminary "probing" of just one child of each - node can substantially reduce the number of nodes explored before a cutoff occurs. If Wi denotes the value of some child of the i-th - node, and Vi denotes the (true) value of the i-th - node, we will obtain a cutoff below the * node if

$$\frac{(V1+\ldots+Vi-1)+Vi+(Wi+1+\ldots+WN)}{N} <= alpha$$

which yields an Ai value of

$$Ai = alpha*N-(V1+\ldots+Vi-1)-(Wi+1+\ldots+WN)$$

From this formula we develop a modified Star2Min procedure (given in Ballard[82]) for * nodes with all - children. A similar Star2Max procedure applies to * nodes having all + children.

V ANALYSIS OF *-MINIMAX ALGORITHMS

Most analyses of ordinary alpha-beta have considered so-called "complete" N-ary trees, where all leaves occur at a fixed depth D and all non-terminal nodes have exactly N successors. In most of the *-minimax games we have studied, chance events are used primarily to determine a player's set of legal moves. We therefore define the class of *-complete N-ary trees by inserting a * node above each node of a complete N-ary tree, and giving these * nodes N-1 additional successor nodes of the same type. The leftmost part of a *-complete 2-ply binary tree appears in Figure 4. We have investigated the efficiency of Star1 and Star2 on depth 3 *-complete N-ary trees, since they correspond to trees allowing the simplest cutoffs of standard alpha-beta.

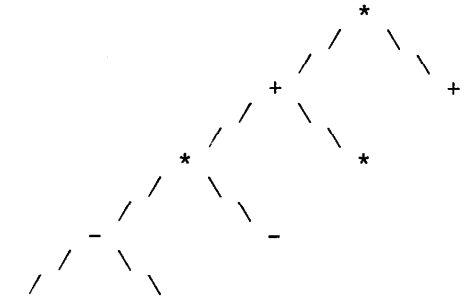

Figure 4 - The Leftmost Portion of a 2-Ply *-Complete Binary Tree

As an attempt to capture the sorts of dependencies that occur in practice, we follow Fuller et al[73] by assigning distinct, uniformly spaced values to the arcs below a node, and defining leaf values as the sum of the arc values on the path to it from the root. In Ballard[82] we derive the following:

Result 1: The asymptotic best-case behavior of algorithm Star1 on the + node of a *-complete 2-ply N-ary tree is to examine approximately 0.7211 N**3 of the N**3 leaves beneath it.

Result 2: The asymptotic best-case behavior of algorithm Star2 on the + node of a *-complete 2-ply N-ary tree is to examine approximately 1.7211 N**2 of the N**3 leaves beneath it.

The latter result is encouraging because, like the O(N**2) best case alpha-beta result for depth 3 trees, it shows that a wise algorithm can hope to reduce search complexity by a factor of N. For the most part, we achieved this reduction without increasing the complexity of the algorithm, its overhead, or the additional space needed. Table 1 indicates search savings for various values of N. Note the rapid convergence of Star1 toward the 72.11 percent region predicted by the asymptotic figure given above.

Since average-case analysis is much more difficult than best-case analysis, we decided to investigate expected-case performance by empirical means. Using the UNIX pseudo-random number generator, we generated and gathered statistics on 1000 *-complete trees for each of several branching factors. Table 2 summarizes the results of Star1 performance. It can be seen that the average case savings is about 21 percent, or roughly 2/3 of the best-case savings (given above). Table 3 summarizes the results of Star2 performance. An interesting result was that roughly half the * nodes for which a cutoff occurred were cut off during the probing phase. Also, we see that for a branching factor greater than about 20, Star2 looks at fewer than half the leaves explored by Star1.

VI INCORPORATING *-MINIMAX SEARCH INTO A GAME-PLAYING PROGRAM

In programming actual minimax games, adjustments are often made to a pure alpha-beta search because of the overwhelming size of most search trees. In particular, a static evaluation function is generally used to rank successor nodes in what appears (before searching) to be best-to-worst order, hoping to assure early cutoffs; a depth bound is often maintained in some form to preclude searching prohibitively deep nodes; forward pruning is performed, meaning that some nodes which look unpromising are not searched at all; a transposition table is maintained to avoid searching the same position more than once if it appears in several places ("transpositions") in the search tree; and so forth. In practice, we would expect such modifications to be made to the *-minimax procedures as well, although the underlying algorithms need not be changed.

N	2	4	6	8	10	20	30	40
Procedure Star1								
Number	5	40	138	336	670	5560	18990	45320
Percent	62.5	62.5	63.9	65.6	67.0	69.5	70.3	70.8
Procedure Star2								
Number	5	25	58	105	166	677	1532	2732
Percent	62.5	39.1	26.9	20.5	16.6	8.5	5.7	4.3

Table 1 - Best-Case Leaf Exploration of Star1 and Star2 for Various *-Complete 2-Ply Trees

N	2	4	6	8	10	20	30	40
Number	7.1	53.9	178	418	810	6389	21382	50425
Percent	88.8	84.1	82.5	81.6	81.1	79.9	79.2	78.8

Table 2 - Average-Case Leaf Exploration of Star1 for Various *-Complete 2-Ply Trees

N	4	6	8	10	20	30	40
Cutoffs							
Probing	1.3	2.0	2.8	3.5	8.1	12.6	17.4
Regular	0.7	1.5	2.5	3.5	8.4	13.4	18.3
Leaves Seen							
Number	48	139	293	531	3341	10109	22390
Percent	75.4	64.5	57.3	53.1	41.8	37.4	35.0

Table 3 - Average-Case Leaf Exploration of Star2 for Various *-Complete 2-Ply Trees

VII SUMMARY

We have developed an algorithm for searching trees pertaining to "perfect information" games involving chance events. We analyzed the average-case complexity of the algorithms empirically, and observed a savings of more than 50 percent. Closed-form analysis reveals a best-case complexity of O(N**2), a substantial savings over the O(N**3) behavior of the "obvious" search stragety. Our strategy can be adapted, as ordinary "alpha-beta" searching has been, to take advantage of special features of a particular game.

REFERENCES

Ballard, B. W. A search procedure for *-minimax trees. Technical Report, Dept. of Computer Science, Duke University, Durham, N.C. (1982).

Baudet, G. M. On the branching factor of the alpha-beta pruning algorithm. Artificial Intelligence 10 (1978) 173-199.

Fuller, S. H., Gaschnig, J. G. and Gillogly, J. J. An analysis of the alpha-beta pruning algorithm. Dept. of Computer Science Report, Carnegie-Mellon University (July 1973).

Knuth, D. E. and Moore, R. W. An analysis of alpha-beta pruning. Artifical Intelligence 6 (1975) 293-326.

Newborn, M. M. The efficiency of the alpha-beta search on trees with branch-dependent terminal node scores. Artifical Intelligence 8 (1977) 137-153.