BRANCH & BOUND FORMULATION FOR SEQUENTIAL AND PARALLEL GAME TREE SEARCHING*: PRELIMINARY RESULTS

Laveen Kanal and Vipin Kumar

The Laboratory for Pattern Analysis
Department of Computer Science
University of Maryland, College Park, MD, 20742

ABSTRACT

In this paper we present a Branch and Bound (B&B) formulation for finding the minlmax value of a game tree and show, that in case a subset with the best bound is chosen for branching, the resulting algorithm is identical to the SSS* algorithm which Stockman [1] has shown to be better than the Alpha-Beta algorithm for game tree searching. We propose parallel Implementations of B&B search for the minimax evaluation of game trees which lead to a parallel implementation of SSS*. The Branch and Bound approach provides a unified way of formulating and analyzing tree searching algorithms such as SSS* and Alpha-Beta. Viewing SSS* this way, its correctness proof Is much easier than given in Stockman, as is a correctness proof for parallel Implementations of SSS*.

Introduction

Viewing game trees as And/Or trees it can be shown [1] that search for the minlmax value of a game tree is equivalent to searching for the maximum valued solution tree, where the value of a solution tree is defined to be the minimum of all its tip values. The algorithm, SSS*, based on State Space Search, was introduced by Stockman for computing the minlmax value of game trees and shown to be better than Alpha-Beta. We formulate a general Branch & Bound procedure to find the optimum solution tree in the space of all possible solution trees represented by a game tree and show that SSS* can be viewed as a particular practical Implementation of the B&B procedure. We also indicate briefly how this formulation applies to the definition and analysis of Alpha-Beta and parallel SSS*.

A B&B procedure can be viewed as a technique for finding the optimum value (maximum or minimum) of a real valued function f, over a possibly infinite domain S, without computing the value of the function for every member of the domain. The original set is repeatedly decomposed (branch operation) into smaller and smaller sets. Lower and upper bounds of the function f for these sets are calculated. Assuming we are searching for the maximum, if the lower bound on f for a set X exceeds the upper bound for any other set Y, set Y can be Ignored (bounding operation) since some member of set X will always have a higher f value compared to all members of set Y. If it can be shown that for any member x of set X there exists y as a member of set Y such that f(y)^f(x), then Y dominates X and set X can be excluded from further consideration without loss of optimality. This dominance relation was used in Kohler & Steiglltz [2] and is further discussed in Ibaraki [3]. Branch and Bound procedures have been widely used for combinatorial optimization problems. Our study of parallel SSS* (and parallel Alpha-Beta) suggests that other B&B procedures for various optimization problems can also be implemented in parallel.

A Branch & Bound Procedure for SSS*

Let S denote the set of all solution trees represented by a game tree, TT denote the collection of sets of solutions trees under consideration for branching and bounding, U be the upper bound defined on a set of solution trees and D be the dominance relation. The selection of the set with maximum upper bound, from IT, is denoted by S(TT). The following B&B algorithm terminates with the optimum solution tree:

1. (Initialize) TT «- S; U(S) «■ + «>.
2. (Select) Choose the set with the highest upper bound: M «- S(TT).
3. (Dominance Test) If the chosen set dominates other sets eliminate them.
4. (Termination Test) If all the solution trees represented by the set M have some unexplored tip nodes in common, go to step 6; else if the set is a singleton, terminate—the optimum decision tree has been found. If the set is not a singleton and does not have common tips go to step 5.
5. (Branch) Divide the set M In several subsets. Associate with each generated subset the upper bound associated with the parent set M. Put all the generated sets into IT.
6. (Evaluate) Explore the tip node ti which is common to all solution trees in the set and update the upper bound: U(M) * Min (U(M), Value(tl)}; put the set back into TT and go to step 2.

Figures 1 & 2 show how the algorithm works.

* This research was supported by NSF Grant ECS-78-22159 to The Laboratory for Pattern Analysis.

The algorithm terminates with the optimum solution tree because ve start with the complete set of solution trees, divide it into smaller and smaller sets, and eliminate a set only if we are certain that the optimum tree does not lie in it. Termination occures when a singleton set having been completely explored (i.e., its upper bound is its true value) is chosen in step 2 as a set with the highest upperbound. At that point one can be certain that no other set has a solution tree better than the one chosen. The algorithm is bound to terminate because we start with a finite set of finite size solution trees. In each iteration of the algorithm we are either eliminating some set of solution trees from further consideration or evaluating some unevaluated tips of some solution trees.

The detailed presentation of SSS* in Stockman [1] is not as transparent as the above B&B formulation but it can be easily seen that SSS* is equivalent to the B&B procedure presented above. SSS* maintains a list of states of traversals (called OPEN), each representing a set of solution trees and the current upperbound associated with it. State expansion directly corresponds to branching, and purging of states from OPEN corresponds to eliminating the dominated sets of solution trees. Thus SSS* can be considered a practical Implementation of the above B&B procedure. SSS* is of interest because it was proved by Stockman that it never explores a node that Alpha-Beta can Ignore. For practical distributions of tip value assignments Stockman experimentally showed that SSS* explores strictly fewer game tree nodes than Alpha-Beta. We have shown that by defining a particular depth first strategy for selection in the above procedure, one gets the well known Alpha-Beta algorithm. With a different choice of select, branch and evaluate functions we can come up with the "SCOUT" algorithm proposed by J. Pearl [4]. Several other competing algorithms also can be derived in this manner. The above Branch & Bound method provides a unified approach to formulating and analyzing various search algorithms for AND/OR trees.

An Asynchronous Parallel Procedure; Preliminaries

While performing set divisions (node expansions in SSS*) if the upperbounds of all sets (of solutions trees) under consideration are reduced to some value U', then the B&B procedure will still find the true optimum solution tree if its value happens to be less than U^v. However if the value of the optimum tree is greater than or equal to U' then the algorithm will terminate with the value U\ Let BB(S,U') be the value returned by B&B if at some time before it terminates upperbounds of all sets are lowered to U'- and let U* be the value of the optimum tree. Then

$$U^* < U' \Longrightarrow BB(S,U^f) - U^*;$$
$$U^* >- U' \Longrightarrow BB(S,U') - U\backslash$$

The validity of this statement can be easily proven by thinking of the act of lowering the upperbounds of the sets as adding an extra leaf node with value U' to all the solution trees under consideration. Since the optimum value of a solution tree is the minimum of all the values at the tips, this modification will keep the optimum value intact if it was already below U'.

It is easy to show that the number of tip nodes evaluated by B&B (or SSS*) Is a monotonically increasing function of the initial bound associated with the starting set of solution trees. The initial bound represents the guessed value of the optimal solution tree. The more accurate this guess, the more effective the dominance relation becomes. In Alpha-Beta, this corresponds to making more cutoffs when better alpha and beta bounds are known. Experiments have shown that if game tree node values are chosen Independently from a uniform distribution between 0 and m, then the dependence of the number of nodes expanded on the initial bound is as shown in Figure 3. As long as the chosen bound is above the true minimax value, the smaller bound provides the correct minimax value at a significantly lower cost. This fact can be used in a seemingly very efficient asynchronous parallel procedure for searching game trees, which compares favorably with Baudet's parallel implementation of Alpha-Beta [5].

Parallel Implementation of SSS*

Let us assume that the minimax value of a game tree lies between -m and +Hn with uniform probability. If we have just one processor we would start B&B with +Hn as the initial upper bound for the complete set of solution trees. As branching and node expansion proceed, upper bounds for various generated subsets would become lower and lower. At termination we would find that the upper bound of a set (which has just one completely explored solution tree) becomes equal to its own optimum value and Is higher than the upper bounds of all the other sets.

With multiple processors we could start several B&B procedures, one on each available processor, each with a different initial upper bound as shown in Figure 4. Any time a process lowers its current global upperbound it can be guaranteed that the minimax value is less than or equal to this lowered global upper bound. Thus in Fig. 4, if process *n' manages to lower Its upper bound we can guarantee that the minimax value lies between -m and Bn where Bn is the current global upper bound of process n. If process n is allowed to proceed it will terminate with the correct minimax value. (Note that In the beginning we could only guarantee that process 1, with the initial global upper bound of +m, would terminate with the correct optimum value). It is clear that processes 1 through n-1 can safely reduce their global upper bounds to some values below Bn and thus perform the look ahead function which was previously performed by processes 2 to n. Thus among the n processes at least one is guaranteed to find the optimum value and the others perform a look ahead function. At any time a process searching in a lower range finds that its global upper bound is correct, i.e., it can be lowered, it can take over as 'master' and the other processes can perform the look ahead. These parallel processes work independently and communicate only if the bounds are lowered. Therefore this parallel algorithm is well suited for implementation on asynchronous multiprocessors.

REFERENCES

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

[2] Kohler, W.H. and Steiglitz, K. "Characterization and theoretical comparison of branch and bound algorithms for permutation problems" J. ACM 21 (1974) 140-156.

[3] Ibaraki, T. "The Power of Dominance Relations in Branch-and-Bound Algorithm" J. ACM 24 (1977) 264-279.

[4] Pearl, J. "Asymptotic Properties of Minimax Trees and Game-Tree Searching Procedures" Artificial Intelligence 14 (1980) 113-138.

[5] Baudet, G. "The design and analysis of algorithms for asynchronous multiprocessors", Ph.D. Dissertation, Carnegie-Mellon Univ. Pittsburgh, PA, Nov. 1976.

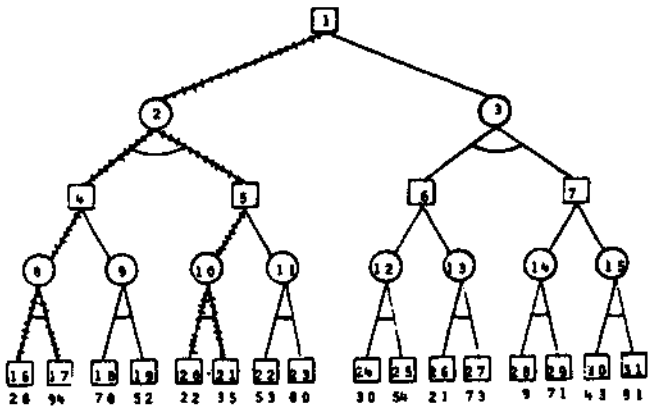

Fig 1 - An AND/OR tree with hatch marks showing a solution tree.

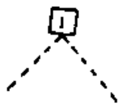

Fig 2.a - Root node represents total set of solution trees.

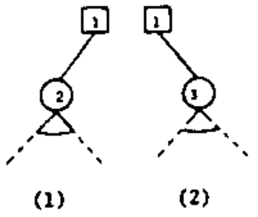

Fig 2.b - Branching divides initial total set of solution trees into two disjoint subsets.

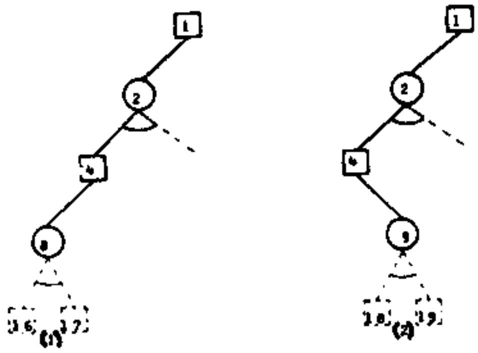

Fig 2.c - Further division is performed on sets of Fig 2.b.1. All the solution trees represented by 2.c.1 have nodes 16 and 17 in common. At the next step, we do not divide this set but evaluate one of its tip nodes (say node 16) to get a better upper bound on its value.

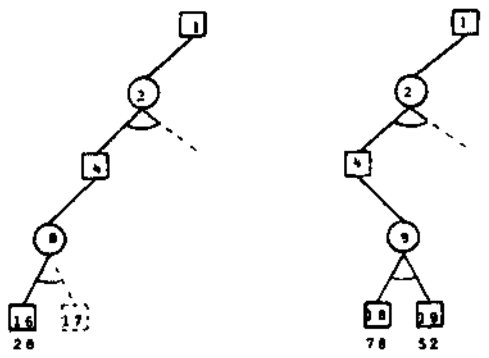

Fig 2.d - After exploring node 16 of the set of Fig 2.c.1 and nodes 18 and 19 of the set 2.c.2 we note that solution tree set of Figure 2.c.2 dominates the set of Figure 2.c.1. This happens because min {value(16),value(17)} ≤ min {value (18),value(19)}.

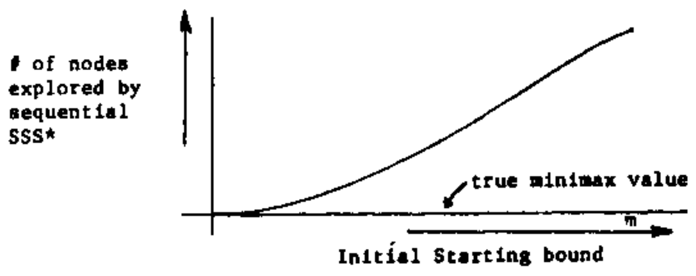

Fig 3. # of nodes explored is monotonic function of initial starting bound.

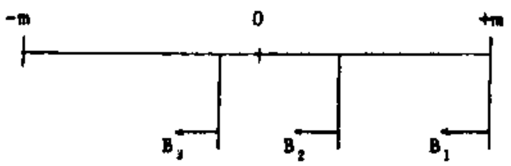

Fig 4. ith process is started with B_i initial bound.