A HYBRID SSS*/ALPHA-BETA ALGORITHM FOR PARALLEL SEARCH OF GAME TREES

Daniel B. Leifker
Laveen N Kanal

Department of Computer Science
University of Maryland
College Park. Maryland 20742

ABSTRACT

This paper explores the issues that arise when SSS*-like search algorithms are implemented in parallel. There is an important implicit assumption regarding the OPEN list of SSS* (and A*-like algorithms); those states which are guaranteed never to become part of an optimal solution are forced down into the OPEN list and never rise to the top for expansion. However. when multiple processors are introduced in a parallel version of SSS*. these buried states become subject to expansion despite their provable suboptimality. ir such states are not identified and purged, they may exert an enormous drag on the parallel algorithm because considerable processor effort will be wasted, However, the pruning mechanisms of alpha-beta can be adapted by a parallel SSS*; the resulting algorithm HYBRID is suitable for searching game trees and general AND/OR trees in parallel

1 INTRODUCTION

A. The Alpha-Beta and SSS' Algorithms

The alpha-beta game tree search algorithm offers significant potential for search speedup by "pruning", or ignoring game tree branches that cannot affect the final mInimax value of the root. The State Space Search algorithm, or SSS* (Stockman 1977). as originally presented, usually dominated alpha-beta by the "parallel" traversal of subtrees. Whereas alpha-beta was condemned to search strictly in a left-to-right fashion, SSS* sent "probes" simultaneously into the tree, and, in a manner not unlike A* (Nilsson. 1980), maintained an "open" list of partial solutions ordered by descending merits. However, SSS* could also be used to search general problem-reduction representations as well as simple game trees (Stockman and Kanal. 1983). Stockman's original SSS" underwent a few revisions, and the final version (Pearl. 1984) is admissible and always dominates alpha-beta. (We shall henceforth assume that the reader is familiar with the next-state operator G of SSS* as well as list notation for trees.)

B. Essential Components of SSS*

For later comparisons, we give a high-level description of the SSS* algorithm:

ALGORITHM SSS*

(1) Place the start state (i.e., root) on the OPEN list.
(2) If OPEN is empty, exit with failure and halt.
(3) Remove the top state S from OPEN.
(4) If S is final, i.e.. represents a complete solution then return S with success and halt
(5) Apply the next-state operator G to S and add G(S) to the OPEN list (possibly with changed merits).
(6) Go to (2).

Additional detailed discussions of SSS* can be found in (Leifker and Kanal. 1985).

Our goal is to implement a form of SSS* in parallel, using a generalized alpha-beta pruning process to excise suboptimal states from the OPEN list before any processor effort is wasted. This proposed algorithm, which we call HYBRID, is suitable for use in searching general AND/OR trees as well as game trees. Since any game tree can easily be transformed into an AND/OR tree (with strictly alternating levels of AND-nodes and OR-nodes), general AND/OR trees are used in the discussion which follows, although, as in game trees, only top-down expansion of states will be considered Our concepts are unlike previous hybrid algorithms (Campbell and Marsland. 1983), in that there is a true coalescence of alpha-beta and SSS*. not simply a juxtaposition of the two in one program.

II INTRODUCTION OF PARALLELISM

A. Where to Introduce Parallelism

The control part (or driver) of SSS* is relatively straightforward. A state is removed from the top of OPEN and examined. If it is final (i.e.. its root is solved), it is taken as the solution and the algorithm halts, If not. the "next-state" operator G is applied to the state and the result(s) placed back into the OPEN list. If the number of states in OPEN ever drops to zero, the algorithm halts with failure

B. A First Possibility

To implement a form of non-partitioned SSS* in parallel, multiple processors obviously must be introduced somewhere. One of two possible locations is shown in Figure I. where only the top state from OPEN is removed at a time. The processors $V \backslash > P_{2}....PN$ then cooperate to expand this state through the next-state operator G and return the results back to OPEN. If G were a very complex operator having many disjoint, tasks, this arrangement would probably be very attractive. The fundamental algorithm would not be changed, and it would still share in all the formal properties enjoyed by SSS*. The only difference would be that the actions of operator G would be accelerated considerably.

This method of "operator parallelism" must be rejected Tor the following reasons: (I) It forces processors to assume specialized tasks. (2) It can create enormous bottlenecks when work cannot be conveniently partitioned, and (3) it has the potential to cause enormous congestion among the waiting processors because the tasks have been broken down into excessively primitive components.

C The Alternative

The alternative, shown in Figure 2, overcomes many of these pitfalls and is amenable to parallel processing by N processors, even when N is not specified until runtime. The basic approach is to permit each processor to access the OPEN list.

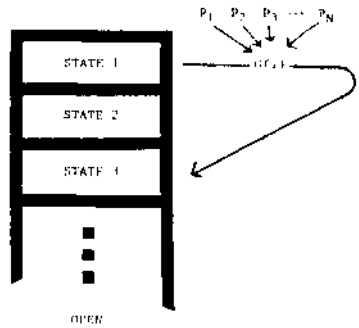

FIGURE 1

Each processor removes the next state from the top of the OPEN list, expands the state by itself, and returns the resulting state(s) back to OPEN This is the method adopted by HYBRID. As one might expect. HYBRID is not simply a variation of SSS* - it represents a new way of heuristic control using the operator G. Consequently, the formal properties of SSS* do not necessarily hold for HYBRID. The remainder of this section defines HYBRID formally and discusses the ramifications of allowing pro cessors to manipulate stales from the interior- of the OPEN list

A special notation must be introduced to facilitate discussion of parallel algorithms. The construction

(OBEGIN - statement-list COEND

indicates concurrent execution (Dijkstra, 1965) in that all statements of - statement-list are executed simultaneously. The notation is augmented here such that processors may be specilied explicitly by attaching their names as labels to statements.

The HYBRID algorithm simply assigns each of the given N processors a copy of the sequential SSS* program. Pormalty, the preliminary version can be defined as:

```
                Algorithm HYBRID
              /* Preliminary version *
(1) Place the start state (i.e., root) on OPEN.

(2) halt := FALSE;  /* initialize */

(3) COBEGIN  /* Each processor 1 through N begins
                identical and concurrent execution.
                Only the code for processor p_k
                (1 <= k <= N) is shown. */

   p_k: BEGIN
        WHILE NOT halt DO
          IF the OPEN list is not empty
            THEN BEGIN
                 Remove a state S from OPEN;
                 If S is final
                   THEN BEGIN
                        halt := TRUE;
                        return(S);
                        END
                   ELSE BEGIN
                        Expand S by operator G;
                        Place results back on
                          the OPEN list by
                          descending merit;
                        END;
                 END;
        END;

COEND
```

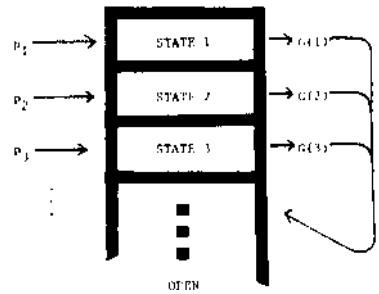

FIGURE 2

The Boolean variable. HALT serves as a global flag to initiate parallel processing and then to terminate it when a solution is found. Presumably the OPEN list and the HALT variable are contained in shared memory and guarded by semaphores. More elaborate synchronization constructs could also be used (Hoare, 1072). but these concerns will not be addressed here. The point to remember is that processors must be in critical sections to change any data in shared memory, and that at most one processor can be in a critical section at a time.

III ANOMALIES OP THE HYBRID ALGORITHM

A Two Problems with the Proposed HYBRID

The proposed version of HYBRID given in Section II is only preliminary because two interesting anomalies occur which must be investigated further: (1) There is no clearly defined time to halt should HYBRID find no solution; and (J) The preliminary version of HYBRID is not admissible (i.e.. if HYBRID halts with a solution, that solution cannot, be guaranteed to be optimal)

B. The First Anomaly: When to Halt?

As defined above, the algorithm is simply allowed to enter an infinite loop if no solution exists for the given tree. Although this is hardly desirable in a practical application. It does illustrate a problem that occurs whenever more than one processor is granted access to an OPEN list. Suppose that HYBRID is executed on a given tree with two processors, p_i and p_j , and that the OPEN list contains one start state when execution begins. Processor P_i, will enter a critical section, remove the state from OPEN, exit the critical section, and begin generating its successors. Processor p_j will then pursue a similar action, but it will find the OPEN list, empty because ;p_i, has not finished expanding the state. If processor p_j is sufficiently naive, it may conclude from the empty list that the given PRR has no solution and then may even attempt to force the entire algorithm to terminate with failure. Since this is clearly not the true state of the computation, HYBRID (as given above) merely instructs all processors to wait idly until (1) a "next" state arrives at the OPEN list for expansion, or (2) the global HALT condition is raised by the discovery of a true solution.

C. The Second Anomaly: Is HYBRID Admissible?

As N (the number of processors) increases, there is also in increase in the likelihood that the algorithm will halt with a non-optimal solution. To see this, recall that the check for termination is made when the state is removed from OPEN, not after it has been generated and about to be placed back onto OPEN. The inadmissibility of HYYBRID, then, although unexpected. Is easily demonstrated: as the number of processors increases, the total number of examined states increases, thus raising the probability that a non-optimal solved state hovering near the top of the OPEN list will be selected for expansion and identified as a final solution. This does not occur in sequential

SSS*. for there the OPEN list is kept strictly sorted by merit, and only the top state is examined at a time.

D. Correcting the Anomalies

Both the anomalies may be easily eliminated. The first anomaly is corrected by halting only when the OPEN list is empty and all processors are idle; the second by maintaining a "tentative optimal merit" (i.e.. a running best solution found so far), and halting only when all remaining states on OPEN are below this threshold. Unfortunately, even with these modifications, HYBRID as it stands is grossly inefficient because of the presence of provably suboptimal states on the OPEN list. This issue is discussed in Section IV.

IV PROVABLY SUBOPTIMAL STATES

A. Efficient Use of Processor Effort

Very often in the course of execution of SSS* the interior of the OPEN list may contain states which are guaranteed not. to be subsumed by any optimal solution. This never causes a problem in sequential SSS*, for there only the top state is examined. However, considerable processor effort will be wasted if any processor is permitted to fetch and expand any of these provably suboptlmal states. It is therefore clear that any efficient version of HYBRID requires a decision procedure to "test" states as they are taken off the OPEN list. If the state is provably suboptlmal. it is discarded. It is expanded only if it has any potential of becoming an optimal solution.

B. Another Look at Alpha-Beta

The alpha-beta algorithm, suitably generalized, can be adapted for use in an efficient version of HYBRID. Although alpha-beta evaluates nodes in a strict left-to-right fashion, the use of alpha values and beta values captures very neatly the concept of suboptimality. If these attributes can be managed by HYBRID, they will provide a quick decision procedure for identifying provably suboptimal branches of the search tree. However, for mnemonic purposes, we shall use the names "floor" and "ceiling" in place of "alpha value" and "beta value"; when discussing the attributes of problem #k. It is convenient to write "floor(k)" and "ceiling(k)". It should be emphasized that this notation is not a function In the mathematical sense, but is rather an attribute which can be accessed and changed.

C. Definitions

Formally, if node #n is terminal and solved with merit m, then floor(n) = celling(n) = m. ir terminal node #n has not yet been expanded, then floor(n) = 0.0 and ceillng(n) = 1.0. If non-terminal node #n has AND successors, then floor(n) is defined as the minimum of its successors' floors, and ccillng(n) is defined as the minimum of its successors' ceilings. If non-terminal node #n has OR successors, then floor(n) is defined as the maximum of Its successors' floors, and ceillng(n) is defined as the maximum of its successors' ceilings. For any given PRR having nodes $p_1,\ldots,p_n$, we define the set Z of provably suboptimal nodes as follows:

(1) If any node p_k has an ancestor p_j- such that p_j is In Z, then p_k is In Z.
(2) If any node p_k has an ancestor p_i such that floor(p_i) Is greater than ceiling(p_k), then p_k is In Z.
(3) If any OR node p_k has an ancestor py and sibling node *Pi* such that floor(p,) Is greater than floor(p_k), and floor(p,) is greater than ceillng(p_i). then pk Is in Z (a "don't-care" cutoff).

There is an important implicit assumption regarding the definition of Z. Initially, Z is empty and grows as the algorithm evaluates more and more terminal nodes. However, once a node becomes provably suboptlmal. it remains provably suboptimal and is removed from all additional consideration. If this were not the case, rule (3) above would give rise to nodes with only transient suboptimality.

D. The Final Version of *HYBRID*

The efficient final version of HYBRID is essentially the same as corrected HYBRID at the end of Section III, with the exception that as each processor in HYBRID removes a state from OPEN, it verifies that no node in the state is a member of Z. If there is such a node, the entire state is discarded and the processor returns to OPEN for another state.

V REMARKS

This is only a high-level description of the HYBRID concept. The detailed mechanisms for practical implementations and the design of appropriate data structures and algorithms to detect provably suboptimal states are current topics of our research.

REFERENCES

[I] Campbell, M. and Marsland, T. A. "A Comparison of Mlnimax Game Tree Search Algorithms", Artificial Intelligence, Volume 20, No. 4, July 1983. p. 347.

[2] Dijkstra, E. "Cooperating Sequential Processes". Technical Report EWP-123. Technological University, Eindhoven. The Netherlands, 1065.

[3] Hoare, C. A. R.. and Perrot, R. "Toward a Theory of Parallel Programming", Operating Systems Techniques, Academic Press, London, 1972.

[4] Leifker. D, B., and Kanal, L. N. "Design and Analysis of Parallel SSS* Algorithms". Technical Report (in preparation), Department of Computer Science, University of Maryland.

[51 Nilsson, N. "Principles of Artificial Intelligence", Tioga Publishing Co.. Palo Alto, California. 1980.

[6] Pearl, J. "Heuristics: Intelligent Search Strategies for Computer Problem Solving", Addlson-Wesley Publishing Co., Reading, Massachusetts, 1984.

[71 Stockman, G. "A Problem-Reduction Approach to the Linguistic Analysis of Waveforms", Ph.D. dissertation, TR-538, Department of Computer Science, University of Maryland, 1977.

[8] Stockman, G.. and Kanal, L. N. "Problem-Reduction Representation for the Linguistic Analysis of Waveforms", IEEE Transactions on Pattern Analysis Machine Intelligence, May 1983.