CMU-CS-78-112

AFOSR-TR- 78-1253

LEVEL II

THE B* TREE SEARCH ALGORITHM: A BEST-FIRST PROOF PROCEDURE

Hans J. Berliner

April, 1978

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR- 78-1253	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): THE B* TREE SEARCH ALGORITHM: A BEST-FIRST PROOF PROCEDURE	5. TYPE OF REPORT & PERIOD COVERED: Interim rept.
6. PERFORMING ORG. REPORT NUMBER: CMU-CS-78-112	
7. AUTHOR(s): Hans J. Berliner	8. CONTRACT OR GRANT NUMBER(s): F44620-73-C-0074, ARPA Order- 2466
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, Department of Computer Science, Pittsburgh, Pennsylvania 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61101E, A02466/7
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, Virginia 22209	12. REPORT DATE: April 1978
	13. NUMBER OF PAGES: 21
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

12 24 p.

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

16 2466 17 84

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper we present a new algorithm for searching trees. The algorithm, which we have named B*, finds a proof that an arc at the root of a search tree is better than any other. It does this by attempting to find both the best arc at the root and the simplest proof, in best-first fashion. This strategy determines the order of node expansion. Any node that is expanded is assigned two values: an upper (or optimistic) bound and a lower (or pessimistic) bound. During the course of a search, these bounds at a node tend to converge, producing natural termination of the search. —> next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data

403084

20. Abstract

As long as all nodal bounds in a sub-tree are valid, B* will select the best arc at the root of that sub-tree. We present experimental and analytic evidence that B* is much more effective than present methods of searching adversary trees.

The B* method assigns a greater responsibility for guiding the search to the evaluation functions that compute the bounds than has been done before. In this way knowledge, rather than a set of arbitrary predefined limits can be used to terminate the search itself. It is interesting to note that the evaluation functions may, measure any properties of the domain, thus resulting in selecting the arc that leads to the greatest quantity of whatever is being measured. We conjecture that this method is that used by chess masters in analyzing chess trees.

CMU-CS-78-112

(2)

THE B* TREE SEARCH ALGORITHM: A BEST-FIRST PROOF PROCEDURE

Hans J. Berliner

April, 1978

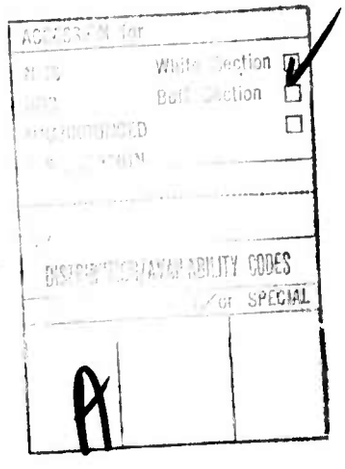
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (Contract F44620-73-c-0074) and is monitored by the Air Force Office of Scientific Research.

ABSTRACT

In this paper we present a new algorithm for searching trees. The algorithm, which we have named B*, finds a proof that an arc at the root of a search tree is better than any other. It does this by attempting to find both the best arc at the root and the simplest proof, in best-first fashion. This strategy determines the order of node expansion. Any node that is expanded is assigned two values: an upper (or optimistic) bound and a lower (or pessimistic) bound. During the course of a search, these bounds at a node tend to converge, producing natural termination of the search. As long as all nodal bounds in a sub-tree are valid, B* will select the best arc at the root of that sub-tree. We present experimental and analytic evidence that B* is much more effective than present methods of searching adversary trees.

The B* method assigns a greater responsibility for guiding the search to the evaluation functions that compute the bounds than has been done before. In this way knowledge, rather than a set of arbitrary predefined limits can be used to terminate the search itself. It is interesting to note that the evaluation functions may measure any properties of the domain, thus resulting in selecting the arc that leads to the greatest quantity of whatever is being measured. We conjecture that this method is that used by chess masters in analyzing chess trees.



I. Introduction

Tree searching permeates all of Artificial Intelligence and much of what is computation. Searches are conducted whenever selection cannot be done effectively by computing a function of some state description of the competing alternatives. The problem with tree searching is that the search space grows as B^D, where B (branching factor) is the average breadth of alternatives and D the depth to which the search must penetrate.

We find it useful to distinguish between searches that continue until they have reached a goal, and those that attempt to solve a problem by iteration; the nth iteration being assumed to take the solving process closer to the solution (which in most cases will never be seen by the search) than the n-1st iteration did. Searches that look for a goal must either succeed or fail. However, searches that work by iteration are expected to produce a meaningful answer at each iteration, for better or for worse.

If a problem has a very large search space and can be solved by iteration (unlike theorem proving which cannot), there is usually no alternative to using the iterative approach. Here, there is a serious problem in bounding the effort so that the search is tractable. For this reason, the search is usually limited in some way (e.g. number of nodes to be expanded, or maximum depth to which it may go). Since it is not expected that a goal node will be encountered, an evaluation function must be invoked to decide the approximate closeness to the goal of a given node at the periphery of the search. This or a similar function can also be used for deciding which tip node to sprout from next in a best-first search. Thus evaluation functions and effort limits appear to be necessary for finding a solution by iteration. However, such conditions on the search appear to cause other problems such as the horizon effect [Berliner, 1973].

It is desirable to have a search proceed in best-first fashion for several reasons. If we can specify a certain degree of closeness to a goal as a terminating condition, this reduces the degree of arbitrariness in stopping when no goal is encountered. Therefore, Harris [Harris, 1974] advanced the notion of a bandwidth. A reference level together with a bandwidth heuristic would guarantee a solution of value no greater than the bandwidth away from the reference level, providing the search terminated. However, this method results in terminating the search under the artificial condition of posing an *a priori* reference level and bandwidth. For a complex game like chess, an expectation level of (say) maintaining the status quo, is reasonable. If the error bounds are as large as a pawn, the search will continue until it finds that one side must win or lose a pawn. This may be an infinite search, if this condition cannot be met. For smaller bandwidths, spurious fluctuations in the evaluation, which are inevitable as different aspects appear, look promising, and are then ultimately decided upon, can cause the bandwidth condition to be satisfied when it is not at all clear that it should be.

Best-first searches tend to put the searching effort into those sub-trees that seem most promising (i.e. have the most likelihood of containing the solution). However, best-first searches require a great deal of bookkeeping for keeping track

of all competing nodes, contrary to the great efficiencies possible in depth-first searches.

Depth-first searches, on the other hand, tend to be forced to stop at inappropriate moments thus giving rise to the horizon effect. They also tend to investigate huge trees, large parts of which have nothing to do with any solution (since every potential arc of the losing side must be refuted). However, these large trees sometimes turn up something that the evaluation functions would not have found were they guiding the search. This method of discovery has become quite popular of late, since new efficiencies in managing the search have been found [Slate & Atkin, 1977]. At the moment the efficiencies and discovery potential of the depth-first methods appear to outweigh what best-first methods have to offer.

II. The B* Algorithm

In present methods for doing iterative searches, there is no natural way to stop the search. Further, for any given effort limit, the algorithm's idea of what is best at the root, may change so that each new effort increment could produce a radical change in the algorithm's idea of what is correct. To prevent this and to provide for natural termination, the B* search provides that each node has two evaluations: an optimistic one and a pessimistic one. Together, these provide a range on the values that are (likely) to be found in the node's sub-tree. Intuitively, these bounds delimit the area of uncertainty in the evaluation. If the evaluations are valid bounds, they do define a range. If not, some simple corrective processes are possible, and we discuss these later in this paper. In either case, the values in a given sub-tree will tend to be within the range of the root of the sub-tree. As new nodes in the sub-tree are expanded and this information is backed up, this will force a gradual reduction of the range of the root node of any sub-tree until, if necessary, it converges on a single value. This feature of our method augurs well for the tractability of searches. In fact, a simple best-first search in the two valued system would converge; however, as we shall show, a B* search converges more rapidly.

The domain of B* is both 1-person (optimality) searches and 2-person (adversary) searches. We shall explain the B* algorithm using adversary searches, where one player tries to maximize a given function while the other tries to minimize it. In the canonical case where nodes have a single value [Nilsson, 1971], MAX is assumed to be on move at the root, and the arc chosen at the root has a backed-up minimax value that is no worse than that of any other arc at the root. In the two valued system that we introduced above, this condition is slightly relaxed: MAX need only show that the ***pessimistic value of an arc at the root is no worse than the optimistic value of any of the other arcs at the root.*** This is the terminal condition for finding the best arc.

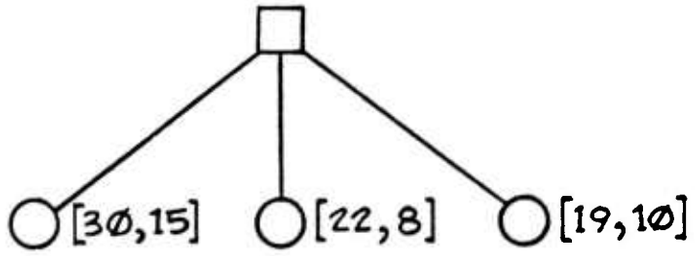


Figure 1 - Start of a B* search

We show the basic situation at the start of a ternary search tree in Figure 1. The optimistic and pessimistic values associated with any node are shown next to it in brackets, the optimistic value being the lefmost of the pair. These values will be updated as the search progresses. In Figure 1, it appears that the leftmost arc has the greatest potential for being the best. It should be noted that if this search were with single (optimistic) valued nodes and this were maximum depth, the search would terminate here without exploring the question of the uncertainty in the evaluation. In the case of B*, there are no terminating conditions other than the one previously enunciated. Thus the search at this point may pursue one of two strategies:

1) It may try to show that the lower bound of the leftmost node is no worse than the upper bound of the other nodes at this depth. We will call this the PROVEBEST strategy.

2) It may try to show that the upper bounds of all the other nodes at depth 1 are no better than the lower bound of the leftmost node. We will call this the DISPROVEBEST strategy.

In either case, the strategy will have to create a proof tree to demonstrate that it has succeeded. We show the simplest cases of the alternate strategies in Figures 2 and 3. In the figures, the numbers inside the node symbols indicate the order of node expansion, and backed up values are shown above the bracketed values they replace. In the case of adversary trees, we insist that one node of every descendant set have a bound equal to that of its parent.

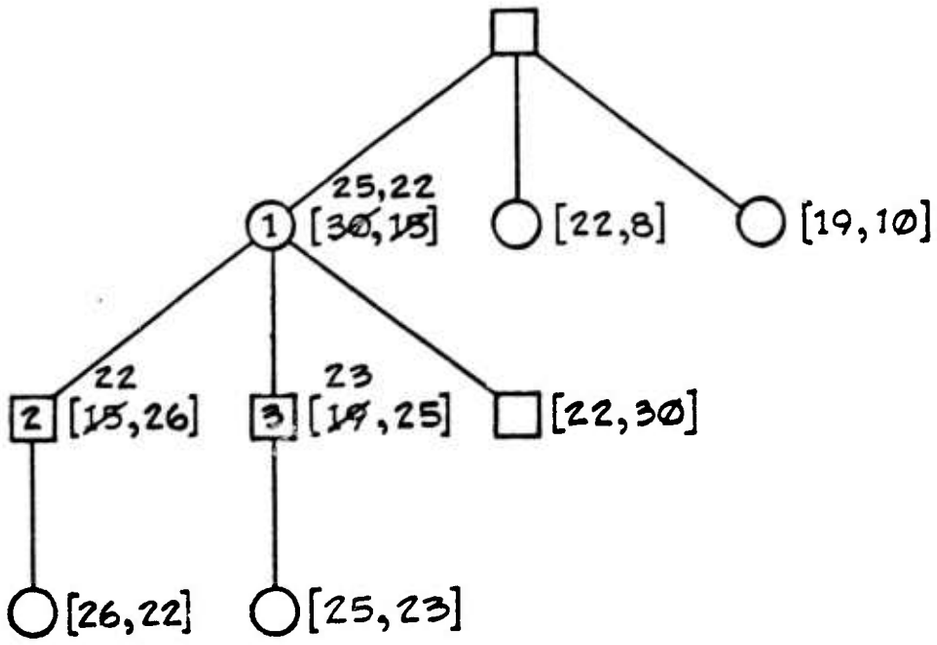


Figure 2 - The PROVEBEST Strategy

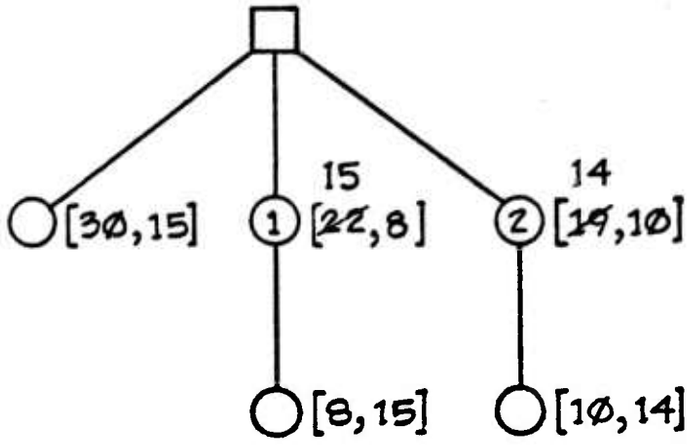


Figure 3 - The DISPROVEREST Strategy

From Figures 2 and 3 it can be seen that, if conditions are right, the seemingly more cumbersome DISPROVEBEST strategy can involve less effort than the PROVEBEST strategy. Further, there is no guarantee that the node with the original best optimistic value will be the ultimate best node. Thus it can be seen that the selection of a method to establish which arc is best at the root will not be a trivial problem.

The B* algorithm addresses itself to this task by doing a best-first proving search. In this search, backing up will occur whenever one of the following conditions is true at a node:

1) The Optimistic and Pessimistic values converge to be equal thus defining the value of that node.

2) There is a more optimistic branch to pursue for either side.

3) The optimistic or pessimistic value achieved is sufficient to be a proof about the sub-tree that it is in.

The combination of these rules assures that the best branch for both sides is always pursued in the search, but only *until it has reached a value sufficient to prove the stated aim of the search.* A small economy is also possible in the generation of descendants. Since any descendant may provide a sufficient condition for causing backup, they may be generated and tested, one at a time, thus saving the cost of doing a complete successor generation at nodes where backup is possible. It should be noted that in cases 2 and 3 above, search may be terminated at a node only temporarily.

In backing up, the best optimistic value of the set of descendants of a node becomes its pessimistic value, and the best pessimistic value of the set of descendants becomes its optimistic value. For MAX, optimistic values are larger than pessimistc, while for MIN optimistic values are smaller than pessimistic. Backing up is applied iteratively as long as there are new values to back up. As backed up values become available, it may be that certain nodes will become logically eliminated from the search. These may be deleted or ignored; it is only a matter of convenience in bookkeeping, as they can not influence the result.

Two features distinguish a B* search from a simple best-first search:

1) While a best-first search only backs up to always sprout from the best minimaxed node, the B* search also backs up whenever one of the bounds of the current branch is sufficient for a proof that the arc at the root which gave rise to this sub-tree is better (worse) than a given reference value. There is a subtle point involved here. It is senseless to extend a branch, the value of which is good enough to be part of a proof; *improving its value will not change the status of the proof.* However, a pure best-first search would not understand this.

2) The B* search can choose a strategy whenever it is at the root of the tree. This allows directing the search effort in such a way that the most meaningful contribution to the proof of which arc is best can be made in the most inexpensive way.

The algorithm requires several reference values for its operation:

1) The *optimistic* and *pessimistic* values at each node delimit the range of

values that can still be achieved by optimal selection of arcs in its sub-tree. These values are updated as the search progresses.

2) At each depth there is kept the value (which we call BESTALT) of the best alternative in the search to this point for the side on move at that depth. These values are updated as the search progresses from the root, by bringing down the value from 2-ply earlier in the tree and updating it if the value of the best alternative at this depth is better. This provides the necessary information for the search to back up when a better alternative is available somewhere.

3) The search may be pursuing the PROVEBEST strategy or the DISPROVEREST strategy, and this must be known throughout the tree since certain decisions depend upon it.

4) Whenever the search departs from the root, there is defined a reference value called ASPIR. This value is what the proof is about; PROVEBEST trying to prove this sub-tree to be better than the value, while DISPROVEREST tries to prove its sub-tree worse than the value.

The following five decision rules define when the search backs up:

Rule 1 - If the optimistic and pessimistic value at a node are identical then the value of this node is known and the search backs up permanently from this node.

Rules 2 & 4 - When the optimistic value of the node being searched becomes worse than the BESTALT value at that depth, the search backs up to search the alternative. This assures that the search backtracks whenever there is a more optimistic alternative for either side at some earlier node (as in a best-first search).

Rules 3 & 5 - When doing a PROVEBEST search, if the pessimistic value of a MAX node is no worse than ASPIR, or when the optimistic value of a MIN node is no better than ASPIR, this demonstrates that the value of this branch is sufficient to prove that the sub-tree rooted at the root is no worse than ASPIR. A complementary set of tests exists for the DISPROVEREST strategy.

The combination of these rules assures that only nodes where no *more optimistic alternative exists for either side*, nor where *this branch is not clearly already sufficient for a proof* are expanded. The first action is something that depth-first searches cannot do, while the second is something that best-first searches are not aware of.

We now present the B* algorithm. It utilizes the variable CURNODE to keep track of the current node, DEPTH to remember the distance of CURNODE from the root, MAXOPTIM to keep track of the most optimistic value of all successors to CURNODE, MAXPESS to keep track of the best pessimistic value of all successors to CURNODE, and the vector BESTALT to keep track of the value of the side-on-move's best

alternative up to that depth. There are several tests in the algorithm, and all are presented from MAX's point of view. We introduce the operator "'" to indicate that a quantity should be complemented to get MIN's point of view; i.e. >' becomes <, MAXPESS' becomes MAXOPTIM, etc.

1) DEPTH ← 0; CURNODE ← 0; BESTALT[-2] ← 0;

2) if DEPTH < 0 then EXIT.

3) if CURNODE has not been expanded yet then generate, name, and evaluate successors, give each pointer to parent.

4) BESTNODE ← name of successor with best OPTIM value;
ALTERN ← name of successor with second best OPTIM value;
MAXOPTIM ← OPTIM[BESTNODE];
MAXPESS ← Value of the best PESSIM value of all successors;

5) Back up MAXOPTIM and MAXPESS and far as necessary. If a change is made to descendants of root, then (DEPTH ← 0; CURNODE ← 0; go to 4);

6) if MAXOPTIM = MAXPESS then go to 16; ! Rule 1

7) BESTALT[DEPTH] ← BESTALT[DEPTH-2]; ! Bring down BESTALT

8) if DEPTH = 0 then decide STRATEGY.
if STRATEGY = DISPROVEREST then
(ASPIR ← MAXPESS; BESTALT[0] ← MAXPESS; BESTALT[-1] ← OPTIM[ALTERN]);
if STRATEGY = PROVEBEST then
(ASPIR ← OPTIM[ALTERN]; BESTALT[-1] ← MAXPESS; BESTALT[0] ← OPTIM[ALTERN]);

9) if MAXPESS >' BESTALT[DEPTH-1] then goto 16; ! Rule 2: MIN can do better

10) if STRATEGY = PROVEBEST then
if MAXPESS' ≥' ASPIR then goto 16; ! Rule 3: PROVEBEST proof achieved

11) if MAXOPTIM <' BESTALT[DEPTH] then goto 16; ! Rule 4: MAX can do better

12) if STRATEGY = DISPROVEREST then
if MAXOPTIM' ≤' ASPIR then goto 16; ! Rule 5: Leg of DISPROVEREST proof

13) if DEPTH ≠ 0 then
if OPTIM[ALTERN] >' BESTALT[DEPTH] then BESTALT[DEPTH] ← OPTIM[ALTERN];

14) if (DEPTH = 0) and (STRATEGY = DISPROVEREST) then CURNODE ← ALTERN
else CURNODE ← BESTNODE;

15) DEPTH ← DEPTH+1; go to 3.

16) DEPTH ← DEPTH-1; if DEPTH < 0 then ANSWER ← BESTNODE
else (CURNODE ← PARENT[CURNODE]; go to 2);

It should be noted that there is never any point to invoking the DISPROVEREST strategy unless PESSIM[BESTNODE] = MAXPESS. This is because there will be at least one node in the alternative set, the value of which cannot be lowered below

MAXPESS. As long as this value is greater than PESSIM[BESTNODE], then this proof cannot succeed. Also, if two or more successors are tied for the best OPTIM value, BESTNODE is the one with the smallest range.

III. Tests of the B* Algorithm

We have simulated the conduct of searches with several versions of the B* algorithm. In the simulation, adversary trees of constant width were generated, with the range of admissible values at the root and the width of the tree varying over sets of runs. As explained in Appendix A, it is possible to generate such trees so that any node in the tree will have its initial bounds determined as a function of its position in the tree and the run number, regardless of when the node is searched. This guarantees that each algorithm searches the same trees.

In assigning each descendant its bounds, we invoked the proviso that at least one descendant must have an optimistic value equal to that of the parent, and one must have a pessimistic value equal to that of the parent. It was possible in this process, for a descendant to have the same values as its parent.

Searches were performed according to the following scheme. A run consisted of 1600 tree searches. In these there were two principal variables, the range and the width of the tree.

1) The range (the number of discrete values) of the evaluation function was varied from 100 to 6400 by factors of four.

2) The width of branching was varied from 3 to 10 in increments of 1.

Thus there were 50 tree searches for each variable pair. For each such run a different search algorithm was tested. Any search that penetrated beyond depth 100, or which put more than 30,000 nodes into its nodes dictionary was declared intractable and aborted.

Several observations could be made from the data. In general, tree size grew with width. Range, on the other hand, turned out to be a non-monotonic function. Searches with the smallest and largest ranges required the least effort in general. Searches of range 400 were hardly ever the largest for any given width and algorithm, while searches of width 1600 were hardly ever the smallest for any given width and algorithm. We cannot interpret this result beyond it indicating that there seems to be a range value for which searches will require most effort and that ranges above and below this will require less.

As we tested the basic B* algorithm presented earlier, potentially useful variations suggested themselves. Earlier, we indicated that the search would prove a given arc best using PROVEBEST, or an arc worse than the best arc using DISPROVEREST. This involves setting ASPIR, the value that must be achieved in the proof, at the optimistic value of the best alternative at the root for PROVEBEST, and at the pessimistic value of the best arc at the root for DISPROVEREST (see step 8 of algorithm). However, this can create wasted effort if, for instance, the range of

the sub-tree being worked on is considerably narrower than the sub-tree that would be searched under the other strategy. By setting ASPIR somewhere between the above values, some proving of each type could occur, creating an overall saving of effort. This did, in fact, prove to be the case.

We tested several distinct variations of the B* algorithm. These related to where ASPIR was set and the crileria for selecting the strategy at the root. The variations were:

1) Number of alternatives considered when making strategy decision at root.

   2 - Best plus one alternative.

   3 - Best plus two alternatives.

   A - All alternatives.

   T - Only when alternative(s) were tied with Best.

2) Criterion applied to decide strategy.

   D - If the sum of the squares of the depths from which came the knowledge of the optimistic bounds of the alternatives was less than the square of the depth from which knowledge of the best arc came, then the best alternative was chosen, else the best arc. This favors exploring sub-trees which have not yet been explored deeply.

   R - Criterion information ("D" above or unity) was divided by the range of the node (thus favoring the searching of nodes with larger ranges).

3) Value ASPIR was set to:

   L - At the limit for each strategy.

   M - In the middle between the limits for both strategies.

These alpha-numeric keys are used to label column headings in Table I to show which algorithm is being tested. BF indicates the results of running a best-first search on the same data, and these are used as a base for comparison.

The categories on the left are based upon how the best-first search did on a given tree. A given tree is in the intractable category, if any of the algorithms tested found it intractable. The entries in the table indicate the ratio of effort, in terms of nodes visited, compared to how the best-first search did on the set; e.g. .50 means that half the effort was required. The last row indicates the number of intractable searches for each version.

TABLE I
Effort Compared to Best-First for Various Implementations of B*

SIZE\ ALG	BF	TR	2DL	2DRL	2DM	2DRM	3DL	3DM	ADM	2DRMX
≤50	1.00	.95	.82	.83	.82	.81	.83	.83	.84	.84
≤200	1.00	.89	.65	.65	.65	.62	.64	.62	.64	.71
≤1000	1.00	.88	.60	.55	.61	.56	.51	.48	.47	.64
>1000	1.00	.72	.50	.48	.51	.48	.36	.35	.32	.56
Intractable	1.00	.77	.71	.67	.70	.64	.62	.59	.52	.69
No. Intract.	226	112	106	85	100	76	96	83	71	81

The data support several conclusions:

1) There is a slight but definite advantage to having ASPIR in the center of the range. The columns with "M" generally outdo those with "L" in the same position. We should note here that we did try some methods of varying the exact position of ASPIR between the limits, but found the mean to be as good as any.

2) The greater the number of parameters considered when making the strategy decision between PROVEBEST and DISPROVEREST, the better the result.

3) In general, the larger the tree, the more pronounced the effect of a good algorithm.

4) The right-most column headed "2DRMX" is a test of what would happen if the nodal bounds were not valid. Here we used algorithm "2DRM" but allowed 5% of all nodes to have their successors have a range which was 50% larger than the parent; i.e. 25% on either side. The net effect of this appears to be a 5 - 10% increase in the amount of effort. Clearly, it is possible to have more frequent aberations, but the effect here does not seem to be serious.

It can be seen that the more flexible the algorithm is in being able to assign methods for solution, the better the results, especially for large trees. Further, it seems to us that with additional effort on improving the strategy selection criterion, the best algorithm could become twice as good as the "ADM" algorithm. Since the method of selecting strategies and assigning limits in this experiment is essentially syntactic (there is no use made of the semantics of the domain being searched), it seems reasonable to suppose that the availability of semantic information would allow even better decisions with consequent improvement in the search effort required.

We examined many of the cases where intractable searches occured. These are due to the stringent way that values are assigned to descendants. When the range of a node gets rather small, and there are a relatively large number of descendants, the probability that at least one will have the same limits as its parent is extremely high. This prevents any progress toward a solution at such a node, and if the probability of this occuring is high enough, the probability of a string of such occurences can be quite high too. This was borne out when we did a run of the best algorithm with the additional proviso that any node for which the range was

reduced to 2 or less arbitrarily received a value equal to the mean of its optimistic and pessimistic value. For this change, the number of intractable searches went from 71 to 4, and each of these was due to overflow of the nodes dictionary rather than exceeding the maximum depth. This method is somewhat reminiscent of Samuel's idea [Samuel, 1959] of terminating search at a node when Alpha and Beta are very close together.

To get another benchmark for comparing B*, we ran a depth-first alpha-beta search on the same data. Here, we allowed the forward prune pardigm, since the bounds on any node were assumed valid. In a search without the two-value system, each node expansion could bring a value any distance from the value of its parent. Since this cannot happen under the two-value scheme it is logical to not search any node the range of which indicates it cannot influence the solution. In order to prevent the search from running away in depth, we used the iterative deepening approach [Slate & Atkin, 1977] which goes to depth N, then to depth N+1, etc., until it finds a solution or becomes intractable. Searches were started with N=1. The results showed that depth-first typically expands three to seven times as many nodes as the best-first algorithm. Although it did manage to do a few problems in fewer nodes than the best B* algorithm, it was unable to solve any problem of depth greater than 19, and became intractable on almost twice as many searches as the best-first algorithm. In contrast, the best algorithm solved some problems as deep as 94 ply, though no doubt shallower solutions existed.

IV. Considerations that Led to the Discovery of the Algorithm

In the course of working on computer chess, we have had occasion to examine the standard methods for searching adversary trees. The behavior of these algorithms appeared more cumbersome than the searches which I, as a chess master, believed myself capable of performing. The real issue was whether a well defined algorithm existed for doing such searches.

1) Our initial motivation came from the fact that all searches that were not expected to reach a goal required effort limits. Such effort limits, in turn, appeared to bring on undesirable consequences such as the horizon effect. While there are patches to ameliorate such ideosyncracies of the search, the feeling that these were not "natural" algorithms persisted.

2) There are two meaningful proposals to overcome the effort limit problem. Harris [Harris, 1973] proposed a bandwidth condition for terminating the search. However, this shifts the limiting quanlity from a physical search effort limit, to a error in measurement limit which, as indicated earlier, has other problems. Another attempt to avoid these problems was to use a set of maximum depths in a depth-first search for terminating searches which qualified moves for other searches [Adelson-Velskij, et. al., 1975]. This is, in effect, a fail-soft approach to effort limits. When there are a number of effort limits, the hope is that everything of importance will somehow be covered. There are no reports of how this approach worked out, but it would appear to have the same essential limitations as all the other effort limited searches. This is borne out by the fact that the authors have now implemented another method of searching for their chess

program KAISSA. In none of the existing tractable search procedures is there a natural terminating condition without any parameters which specify under what conditions to halt.

3) We have noted that standard searches may at times investigate a very large number of nodes that have no apparent relevance to a solution. Consider the following situation: If there is only one legal successor to the root node, any iterative solution technique can easily check for this condition and indicate this is the best successor without further analysis. However, if there is only one *sensible* arc, a depth-first program will still insist on refuting all other arcs at the root to the prescribed depth, while a best-first program may investigate the one good arc *ad infinitum*. Usually, it is possible to determine that the one sensible arc is best without going at all deep in the search. It appears that some essential ingredient is missing. We have felt for some time that the notion of level of aspiration (as first put forward in [Newell, 1955]) was the key to the proper construction. The Alpha-beta search procedure appears to have such a level of aspiration scheme. However, this scheme has an aspiration level for each side, and that only serves to bound those branches that can be a part of the solution. To us, a level of aspiration is a focal point that each side tries to push a little in the favorable direction. We attempted this construction in the search scheme of CAPS-II [Berliner, 1974], which relied heavily on notions of optimism, pessimism and aspiration. These are the type of *semantic* or *domain-dependent* notions that should control a search. However, we performed depth limited depth-first searches in CAPS. Without the best-first requirement there was no need to keep track of best alternatives, nor to maintain the optimistic and pessimistic values at each node.

4) We have always liked the way the search could be terminated at the root node, when the backed up (sure) value of one alternative is better than the optimistic values of all the alternatives. This is the forward prune paradigm, and while it can be used to keep the search from investigating branches that appear useless at any depth, it only terminates the search if applicable at the root. However, when a global ASPIR and local optimistic and pessimistic values exist, it is possible to decide that a particular sub-tree at any depth cannot affect a given proof attempt. This is like a forward prune, only the search may return to this node at a later stage for another proof attempt.

5) Protocols of chess masters analyzing chess positions [De Groot, 1965] show a phenomenon known as progressive deepening. Roughly, this appears to be the investigating of a line of play, abandonment of the investigation of this line, and the subsequent return to the investigation of the line, but with the analysis being followed to a greater depth in the tree. The deepening process may occur several times during the analysis. Since humans investigate very sparse trees and chess masters play chess very well, it was thought that this procedure (whatever it consisted of) should be an effective way of managing the search. The real question was whether there was an actual search algorithm, or whether the deepening was the result of ad hoc procedures. I have held to the former view.

In fact, De Groot came very close to discovering our algorithm. In "Thought and Choice in Chess" [De Groot, 1965], (pp. 28-32), he outlines a proof procedure involving the basic strategies for demonstrating that a move is better than its nearest competitor, and shows that this seems to be at the core of many of the protocols he collected. However, he fails to relate it to a tree searching procedure, and in fact speculates that the subjects are only using this scheme as a basis for their investigations (which may be correct).

V. Evaluation Functions and Meaningful Bounds

During the course of our investigations, we have attempted to apply the B* algorithm to some optimization problems, notably the 8-puzzle [Nilsson, 1971]. During this effort, we succeeded in creating lower bounding functions which were monotonic and several times more sensitive than any previously published for this particular problem. However, we could not devise a really useful upper bounding function. Such a function should form a reasonable range together with the lower bounding function and should be monotonic. The most difficult 8-puzzle configurations can be solved in 30 steps [Schofield, 1967]. Our best upper bounding function "grabbed" at about 8 ply from a solution. Thus problems of depth 12 or so could be solved easily by B*, but for deeper problems the upper bounding function was not able to contribute to the solution.

We have speculated about why the construction of the upper bounding function was so difficult. It appears that, since the function to be optimized is the cost of the solution path, it is always possible to get good estimates of the lower bound since it involves estimating the elements required for a hypothetical, but frequently unattainable path. No similar notion pertains for upper bounds, since longest paths, while forming an upper bound and being monotonic, are too far removed from the value of an ultimate solution to be a useful bound. However, for relatively short paths (or nearby sub-goals) it is possible that useful upper bounding functions can be constructed. The guiding principle for those that we were able to construct is to use a pattern-based approach; i.e. a certain pattern was recognized as being embedded at a node and requiring at most N steps for a solution. We feel that this distinction in the way effective bounding functions can be constructed is extremely important, and could very well account for why humans do such a good job at sub-optimizing tasks.

Actually, the notion of an optimal path to a goal implies that the search procedure traverse such a path. Such a procedure could not be iterative, else it could stop short of a goal. Thus it seems that optimality tasks are just not well suited to B*'s capabilities. Finding an optimal path is approximately equivalent to finding the shortest mate in a game of chess, and this is seldom relevant to making the best move. An iterative algorithm prefers to find a good start on a path, which may be optimal, but in any case meets a satisficing criterion, and can be found with a reasonable or mimimal effort (few nodes). Optimization problems just do not fit well into such a mold. On the other hand, adversary situations are apparently much easier to handle, since one person's optimistic function is the other's pessimistic one.

We consider the basic issue here to be what constitutes a solution. If (as is almost

always the case) 1-person problems deal with optimizing the cost of some path function, then there appears to be little hope for applying B* to such problems unless better upper bounding functions can be found. However, when iterative solutions are desired or are the only ones that are tractable, the B* algorithm can be used to find a series of first steps in the right direction.

The B* search can easily be structured to fit a given task. For instance, in chess an ASPIR and evaluation functions can be chosen to support the search to determine whether a given set of non-terminal goals is achievable. Further, and De Groot presents some evidence to this point, humans probably change the aspiration level (and goals) at times when returning to the root. There is good evidence that the evaluation functions may be changing too [Berliner, 1977a].

Applying these ideas to the solution of 1-person problems leads us to believe that certain problems for which optimizing the cost is not the correct formulation may be solved by B*. Such a problem could exist when, for instance, it is most important to get a solution for minimal computational resources (nodes visited in the search). This would be the way humans would solve many such tasks. In an incomplete information environment, this could be a reasonable enterprise. We propose two examples:

1) Not analyzing which of several plausible replies an opponent would make in a game of chess, when all the moves to that point are clearly best.

2) In a robot navigation environment, not trying to plan a complete path when the whole terrain cannot be viewed at the time a first solution is attempted.

VI. Discussion and Summary

There are two things that distinguish the B* algorithm from other known tree search procedures:

1) The optimistic and pessimistic value system allows for termination of a search without encountering a goal node, and without any effort limit.

2) The option to exercise either of two search strategies allows the search to spread its effort through the shallowest portion of a tree where it is least expensive, instead of being forced to pursue the best alternative to great depths, or pursue all alternatives to the same depth.

In pursuit of the latter, it is best to have the aspiration level somewhere between the best pessimistic value at the root and the optimistic value of the best alternative. This allows both searchs strategies to be employed effectively. Use of the depth from which the current evaluation has come, and the present range of a node also are useful in determining the best strategy, as no doubt, would be the domain-dependent knowledge associated with an evaluation (not merely its magnitude).

It is interesting to compare the basic features of B* with those of well known search

algorithms. Consider the A* search algorithm [Nilsson, 1971]. It could easily operate under the two value system in a mode that is satisfied to find the best arc at the root, and the cost of the path without finding the complete path itself. This algorithm would be equivalent to B* using only the PROVEBEST strategy, and being able to halt search on a branch only when a goal was reached or if the upper and lower bounds on the branch became equal; i.e. the cost of the path is known. Another step in the direction of iteration would be to only use the PROVEBEST strategy and allow the search to halt when a best node at the root had been identified. In this mode the exact cost of the path would not be known. This produces the best-first algorithm used for the column headed BF in Table I. Finally, the full-fledged B* algorithm working with both strategies discovers the best node without the exact cost of the path. However, it does enough shallow searching so that it explores considerably fewer nodes than any of the algorithms described above.

Having the two strategies without the two value system has no meaning at all, since there is no way of pronouncing one node at the root better than any other without having an effort limit. Just using a depth-first iterative deepening procedure, although it spreads the search over the shallower portions of the search tree, investigates too many non-pertinent nodes.

Today's search algorithms rely on assigning a single value to a node, under the assumption that each node expansion will bring in new and useful information that can be backed up and used to produce a more informed opinion about the node's sub-tree. However, this ignores the variability about the estimate that is made by the terminal evaluation function. It is precisely for this reason that chess programs indulge in quiescence searches when the variability at a terminal node is considered too high. Our method can thus be considered to carry a specification of variability of the evaluation for every node in the tree. Thus any posed issue (as represented by its variability) cannot be abandoned until it can be shown to be irrelevant to determining the best solution.

The advantage of the two-value system is that it provides a method for naturally terminating a search. It also allows the critical test which will pronounce one arc at the root better than all the rest. However, it clearly requires good estimating functions for its success. In difficult adversary domains such as chess this appears doable, and we have constructed reasonable functions of this type for chess tactics. The key is that in such situations, one side's optimism is the other's pessimism. For domains in which optimality searches are usually done, it is difficult to find useful upper bounding functions for path costs involving long paths. Therefore, B* can probably only be used for such searches, when some other criterion of success such as a reasonable solution at low computational cost is desired. This is probably close to the criterion humans use in approaching such problems, since they do not have the facilities to deal with the combinatorics of even mildly difficult problems.

Clearly, evaluation functions are very important. The B* search transfers the responsibility for determining how much effort to spend (which has previously been the responsibility of the search parameters, i. .e. depth limit, effort limit,

etc.) to the evaluation functions which now determine the effort limit due to their crispness and ability to narrow the range between optimistic and pessimistic. In the final analysis, the B* search is a conversation between an evaluation function and a control procedure which terminates when enough has been discovered in the search to justify a selection at the root. If the evaluation function estimates do not validly bound the actual value of a node, then errors in arc selection can occur. However, there is no reason why these should be more severe than errors produced by any estimating function which is not applied at domain defined terminal nodes. Unfortunately, very little appears to have been done toward making a science of the construction of sensitive evaluation functions, since the highly significant work of Samuels [Samuels, 1959 and 1969]. We have been investigating how such evaluation functions can be constructed of many layers of increasingly more complex primitives in connection with the 8-puzzle and backgammon [Berliner, 1977b]. In the latter great amounts of knowledge need to be brought to bear, since search is not very practical.

The proof schemas cited in De Groot, some of the protocol analysis (particularly pp. 213-217), and the fact that humans search very small, narrow trees lead us to believe that the B* search is, in fact, what is being called progressive deepening. In performing a search, the B* algorithm may go down a branch several times, each time looking to see whether a value sufficient for a proof can be found. The search will abandon a branch when:

1) The branch is no longer best.

2) The proof is established.

In the first case, the deepening stops only to be resumed at the now best branch, possibly several ply nearer the root. In the second case, the deepening stops and the search reverts to the root to determine whether the proof is complete (it may not be if ASPIR is in the middle, as explained in section III). Such phenomena could easily give rise to the notion of a best-first search with progressive deepening since the jumping around is observed at the level of the protocol, without the underlying logic being apparent. Thus the B* algorithm fulfills all the basic conditions.

BIBLIOGRAPHY

Adelson-Velskiy, G. M., Arlasarov, V. L., and Donskoy, M. V. (1975), "Some Methods of Controlling the Tree Search in Chess Programs", *Artificial Intelligence*, Vol. 6, No. 4, 1975.

Berliner, H.J. (1973), "Some Neccessary Conditions for a Master Chess Program" *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pp. 77-85, August 1973.

Berliner, H. J. (1974), *Chess as Problem Solving: The Development of a Tactics Analyzer*, Ph. D. Dissertation, Computer Science Department, Carnegie-Mellon University, March 1974.

Berliner, H.J. (1977a), "On the Use of Domain-Dependent Descriptions in Tree Searching", in *Perspectives on Computer Science*, A. K. Jones (Ed.), Academic Press, 1977.

Berliner, H. J. (1977b), "BKG -- A Program that Plays Backgammon", *Computer Science Dept., Carnegie-Mellon University*, 1977.

De Groot, A.D. (1965), *Thought and Choice in Chess*, Mouton and Co., 1965.

Harris, L. (1974), "The Heuristic Search Under Conditions of Error", *Artificial Intelligence*, Vol. 5, No. 3, pp. 217-234, 1974.

Newell, A. (1955), "The Chess Machine: An Example of Dealing with a Complex Task by Adaptation", *Proceedings Western Joint Computer Conference*, pp. 101-108, 1955.

Nilsson, N. J. (1971), *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.

Samuel, A. L. (1959), "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, Vol. 3, No. 3, 1959, pp. 210-229.

Samuel, A. L. (1969), "Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress", *IBM Journal of Research and Development*, Nov. 1967, pp. 601-617.

Schofield, P. D. A. (1967), "Complete Solution of the 'Eight-Puzzle'", in *Machine Intelligence 1*, N. L. Collins & D. Michie (Eds.), American Elsevier Publishing Co., 1967.

Slate, D. J. and Atkin, L. R. (1977), "CHESS 4.5 -- The Northwestern University Chess Program", in *Chess Skill in Man and Machine*, P. Frey (Ed.), Springer-Verlag, 1977.

APPENDIX A - How Io Generale Canonical Trees of Uniform Width

We here show how to generate canonical trees which are independent of the order of search. We note that a tree can receive a unique name by specifying the range of values at its root, the width (number of immediale successors at each node), and the iteration number for a tree of this type. To find a unique name for each node in such a tree, we nole that if we assign the name "0" to the root, and have the immediate descendants of any node be named

(parentname*width+1),(parentname*width+2), -- (parentname*width+width)

then this provides a unique naming scheme. Now it is self-evident that the bounds on a node that has not yet been sprouted from must be a function of its position in the tree (name) and the name of the tree. Thus, if we initialize the random number generator that assigns values to the immediate descandants of a node as a function of its original bounds, its name, the width, and the iteration number, then the descendants of node "X" will look the same for all trees with the same initial parameters, regardless of the order of search or whether a node is actually ever expanded. The actual function we use to initialize the random number generator is (Parentname+width)*(iterationnumber+range). This avoids initializing at zero since width and range are never zero. The bounds of the parent node serve as bounds on the range of values that the random number generator is allowed to produce.