The B* Tree Search Algorithm: A Best-First Proof Procedure†

Hans Berliner

Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA 15213, U.S.A.

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.*

1. 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.

† This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

We find it useful to distinguish between searches that continue until they have reached a goal, and those that can stop short of a goal. When a goal reaching search has been completed, there should be no further secrets in the problem presented. The path to the goal (and presumably alternate paths in the case where an opponent exists and has choices) are kno... Thus, once a step down the solution path is taken, all future steps are also known.

The case is quite different when no goal is reached. Here, an evaluation function is required to produce some measure of the nearness of the goal. Since a given search usually does not encounter a goal, it will have to be repeated after the action associated with the "solution" has been taken. Thus, the ultimate solution to the problem proceeds in steps, where new problems are along what is believed to be the path to the goal. We will call this type of search *iterative*.[1] 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 problems such as theorem proving where the search must continue until a proof is found), 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 [2].

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, and achieve this condition, then this reduces the degree of arbitrariness in stopping when no goal is encountered. Therefore, Harris [7] advanced the notion of a bandwidth. A goal together with a bandwidth condition around the value of the goal would guarantee a solution of value no worse than the bandwidth away from the goal, providing the search terminated. However, selecting goals (or sub-goals in case the goal is deemed too remote) and bandwidths that give the search a chance to terminate in a reasonable fashion is extremely difficult. Further, while the bandwidth condition does produce a certain degree of discipline in stopping, it adds further arbitrariness to the search process.

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,

[1] We want to assure that this definition of iteration is not confused with *iterative deepening* [15], a method now in popular use for controlling the depth of a depth-first search.

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 [15]. At the moment the efficiencies and discovery potential of the depth-first methods appear to outweight what best-first methods have to offer. However, both methods have some glaring deficiencies.

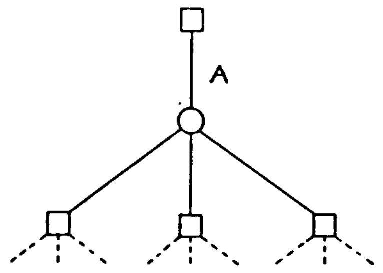

FIG. 1.

In the situation of Fig. 1, both best-first and depth-first searches will expend their allotted effort and come to the conclusion that the arc labelled A is best. Clearly, no search is required for this conclusion, since it is the only arc from the root. To overcome this waste of effort, the developers of performance programs (notably chess programs) test for this condition before embarking on any search.

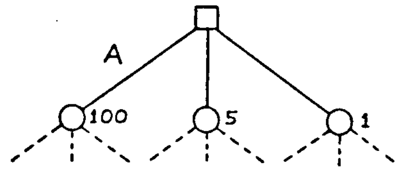

FIG. 2.

In Fig. 2 the situation is different as there are three legal successors to the root node. Numbers at the descendant nodes are intended to show how an evaluation function would appraise the relative merits of the three nodes. Here the depth-first search has no recourse but to do its search, and (if the initial evaluation is reasonably correct) eventually report that arc A should be chosen. A best-first search on the other hand, would probably expend its entire effort in the sub-tree of arc A, and eventually come to the conclusion that it was best. These incongruities require no further explanation.

2. 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 the best arc 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, the values in a given sub-tree will be within the range specified at the root of the sub-tree. As new nodes in a given sub-tree are expanded and this information is backed up, the range of the root node of that sub-tree will be gradually reduced 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 if all bounds are valid. However, as we shall show, a B* search converges more rapidly.

The domain of B* is both 1-person 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 [11, pp. 137–140], 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.

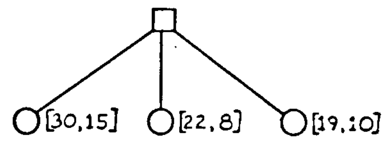

FIG. 3. Start of a B* search.

We show the basic situation at the start of a 2-person ternary search tree in Fig. 3. The optimistic and pessimistic values associated with any node are shown next to it in brackets, the optimistic value being the leftmost of the pair. These values will be updated as the search progresses. In Fig. 3, 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 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. When at the root, the B* search may pursue one of two strategies:

(1) It may try to raise the lower bound of the leftmost (most optimistic) node so that it is not worse than the upper bound of any of its sibling nodes. We will call this the PROVEBEST strategy.

(2) It may try to lower the upper bounds of all the other nodes at depth 1, so that none are better than the lower bound of the leftmost node. We will call this the DISPROVEREST 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 Figs. 4 and 5. In the figures, the numbers inside the node symbols indicate the order of node

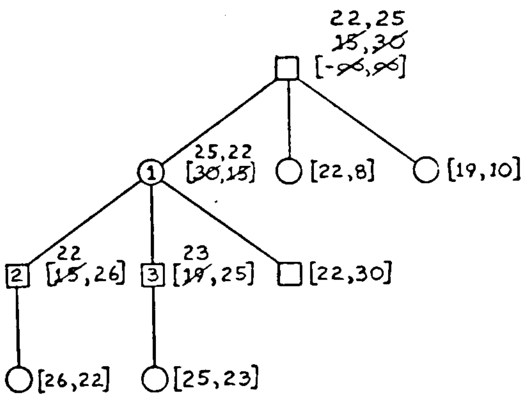

FIG. 4. The PROVEBEST strategy.

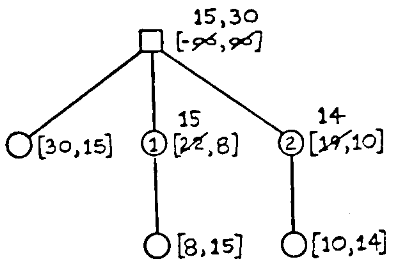

FIG. 5. The DISPROVEREST strategy.

expansion, and backed up values are shown above the bracketed value they replace.

From Figs. 4 and 5 it can be seen that, if conditions are right, the seemingly more cumbersone DISPROVEREST 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 is not a trivial problem.

The B* algorithm addresses itself to this task by doing a best-first proving search. The search decides on a strategy whenever it is at the root. Then it pursues the strategy by always selecting the best branch in the chosen sub-tree until it reaches a leaf node. That node is then expanded. Backing up of the search will occur whenever the expansion of a node produces a value that changes the bounds of its parent node. This change is rippled back until no further changes take place; whereupon the forward search is resumed. The search terminates when the pessimistic value of the best node at the root is no worse than the optimistic value of all its siblings.

Since new values are always backed up, any value which survives back to the root will cause a re-examination of the strategy. This assures that the search will not continue searching a branch the *value of which is already sufficient for the proof that is being attempted.* 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 leaf nodes. It should be noted that backing up from a node in no way implies that the search will not go back to this node later, as is typical in best-first searches.

When the search backs 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 pessimistic, 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. When backing up, the search resumes from the last node the values of which were changed. If this node is an immediate descendant of the root, then the question of what strategy to pursue is re-examined.

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

(1) A best-first search is intent on finding a goal and thus only backs up to always sprout from the best minimaxed node. However, since the B* search is only interested in finding the best first step toward a goal, it will only continue this finding of the best minimaxed node as long as the proof is not complete. There is a

subtle point involved here. It is senseless to extend a branch, the value of which is sufficient for 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. Later in the paper, we discuss criteria that have been applied experimentally for making the strategy selection.

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, OPTIM and PESSIM at each node to remember the bounds, PARENT to point to the name of the node's parent, MAXOPTIM to keep track of the most optimistic value of all successors to CURNODE, and MAXPESS to keep track of the best pessimistic value of all successors to CURNODE. There are two tests in Step 4 of the algorithm which are presented from MAX's point of view. We introduce the operator " ' " to indicate that the sense of a test should be complemented to get MIN's point of view at alternate level of the tree; i.e. >' becomes <, and <' becomes >.

```
(1) DEPTH ← Ø; CURNODE ← Ø; OPTIM[Ø] ← −∞; PESSIM[Ø] ← ∞;
(2) if CURNODE has not been expanded yet then generate and evaluate
    successors, giving each a name and a pointer to CURNODE.
(3) BESTNODE ← name of successor of CURNODE 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;
(4) if (MAXOPTIM <' PESSIM[CURNODE])
      or (MAXPESS >' OPTIM[CURNODE]) then
    begin                                  ! Back-up (values and search)
        PESSIM[CURNODE] ← MAXOPTIM;
        OPTIM[CURNODE] ← MAXPESS;
        if DEPTH > Ø then
        begin
          CURNODE ← PARENT[CURNODE];
          DEPTH ← DEPTH−1;
          go to 3;
        end
        else if DEPTH = Ø then             ! Check for termination
              if PESSIM[BESTNODE] ≥ OPTIM[ALTERN] then
                exit with ANSWER = BESTNODE;
    end;
(5) if DEPTH = Ø then
      begin
        decide STRATEGY;
        if STRATEGY = DISPROVEREST then CURNODE ← ALTERN
        else if STRATEGY = PROVEBEST then CURNODE ← BESTNODE;
      end;
(6) if (DEPTH ≠ Ø) then CURNODE ← BESTNODE;
(7) DEPTH ← DEPTH+1; go to 2.               ! Go forward
```

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. In the presented algorithm, if two or more successors are tied for the best OPTIM value, BESTNODE is the one with the smallest range.

The observant reader may have noted that since each proof tree is an AND/OR tree, *all* arcs must be searched at alternate levels of the tree corresponding to the side against whom the proof is directed. At such a point, it is not always optimal to search the "best" arc first as indicated in Step 6 above. There are cases where it is superior to search the branch involving the most difficult proof first, since if that fails, then the whole proof at this juncture will fail, and the search can be redirected to another sub-tree. Reasoning along similar lines is presented in Harris [8, p. 160]. We are at present working on a general method to define when the best node should be searched next and when the one with the greatest pessimistic value, for all levels of the tree.

3. 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 I, 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 value 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 are 32 basic conditions, and 50 tree searches were performed for each condition. 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 about the test 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 range 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.

We tested several distinct variations of the B* algorithm. These related to the criteria for selecting the strategy at the root. The variations are explained below; although we did not test all combinations of these conditions.

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

2 — Best plus one alternative.
3 — Best plus two alternatives.
A — All alternatives.

(2) Criterion applied to decide strategy.

D — If the sum of the squares of the depths from which the optimistic bounds of the alternatives had been backed up was less than the square of the depth from which the value of the best arc had been backed up, 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).

These alpha-numeric keys are used to label column headings in Table 1 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. 0.50 means that half the effort was required. All but the last row indicate the effort ratio; the last row indicates the number of intractable searches for each version.

TABLE 1. Effort compared to best-first for various implementations of B*

Size/Alg	BF	2D	2DR	3D	AD	2DRX
<50	1.00	0.82	0.81	0.83	0.84	0.84
<200	1.00	0.65	0.62	0.62	0.64	0.71
<1000	1.00	0.61	0.56	0.48	0.47	0.64
>1000	1.00	0.51	0.48	0.35	0.32	0.56
Intractable	1.00	0.70	0.64	0.59	0.52	0.69
No. Intractable	226	100	76	83	71	81

The data support several conclusions:

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

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

(3) The right-most column headed "2DRX" is a test of what would happen if the nodal bounds were not valid. Here we used algorithm "2DR" 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.

Thus it is quite likely that changing Step 4 of the algorithm to allow the backing up of values outside the expected range as well as those within the range is quite acceptable. This will result in the search changing its mind about what is going on more often, but will allow new data to have an impact on the search. The degree to which the search will vacillate would appear to depend upon how often an out-of-bounds datum was discovered; however, with a moderate rate of such occurrences, this effect would appear minimal.

Using the largest amount of data for strategy selection appears to produce the best results. 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 "AD" 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 occurred. 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 occurring is high enough, the probability of a string of such occurrences 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 [12] 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 paradigm, since the bounds on any node were assumed valid. In a search without the two-value system, each node expansion could produce 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; thus the use of the forward prune paradigm. In order to prevent the search from running away in depth, we used the iterative deepening approach [15] 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 BF 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 BF algorithm. In contrast, the best B* algorithm solved some problems as deep as 94 ply, although it is conceivable that shallower solutions existed.

4. 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 idiosyncracies 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 [7] proposed a bandwidth condition for terminating the search. However, this shifts the limiting quantity from a physical search effort limit, to a distance from the goal 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 in the chess program KAISSA (see [1]). For a set of maximum depths, first all moves, then all material threatening moves plus captures, then all captures and checks, and finally only captures were considered. With an effort limit for each category of moves, it was hoped that everything of importance would 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 program. 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*. We have demonstrated these cases in Section 1. 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 by Newell in [10]) 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. The correct level of aspiration specifies the value required for a "solution". We attempted such a construction in the search scheme of CAPS-II (see [3]), which relied heavily on notions of optimism, pessimism and aspiration. However, we performed depth limited depth-first searches in CAPS. Without the best-first requirement there was no need to keep track of 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 (static) values of all the other 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. In B*, the forward prune of all remaining alternatives at the root (when PESSIM[CURNODE] $\geqslant$ OPTIM[ALTERN]) not only terminates the search, but it provides a logic for all actions taken in the search. Thus, when the final test succeeds, a massive forward prune of all of the tree not connected with the solution is effected.

(5) Protocols collected by De Groot [6] of chess masters analyzing chess positions 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" [6, 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, or to any notion of optimistic and pessimistic limits.

Some of the protocols (particularly pp. 213–217) appear to us to support the B* paradigm very strongly. What is being called progressive deepening, is really nothing other than the retracing of a branch of the search which may have been abandoned for any of the several reasons that exist for backing up. At the level of the protocol, only the jumping around is noticed. This is then interpreted as a best-first search with progressive deepening since the underlying logic is not readily apparent.

5. Evaluation Functions and Meaningful Bounds

Most existing 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 solution.

Because of this, 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 search parameters such as depth limit, effort limit, bandwidth, 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.

In the course of our investigations, we have attempted to apply the B* algorithm to some optimization problems, notably the 8-puzzle [11]. 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 also be monotonic. The most difficult 8-puzzle configurations can be solved in 30 steps [14]. 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 during the early stages of the search.

We also spent some time on the travelling salesman problem. In both these problems the function to be optimized is the cost of the solution path. In both, we found that the greatest difficulty was in getting good upper bounding functions. This appears to be due to the fact that a maximum path length turns out to not be a good upper bound, while refined estimates of minimum path length are quite reasonable lower bounds. However, for relatively short paths (or nearby sub-goals) it was possible to construct useful upper bounding functions. The guiding principle here was 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.

Optimality searches require that the search determine the minimum cost for reaching a goal. Since an iterative procedure could stop short of a goal, it would need extremely tight bounding functions to be able to "see" the goal from a great enough distance to be useful in most cases. 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, if one considers only tractable searches. 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 effort (few nodes). Optimization problems just do not fit well into such a mold.

On the other hand, satisficing searches appear very well suited to B*'s capabilities. With knowledgable bounding functions to guide the process, we feel convinced that B* would produce satisficing solutions to 8-puzzle problems very much as humans do; with few nodes expanded and solution paths of somewhat greater than optimal length. Further, adversary situations appear still easier to handle than one person situations, since one adversary's optimistic function is the other's pessimistic one. It should be noted that good B* bounding functions will in all likelihood not measure what an optimality function would measure. Thus for chess, a good function might measure attack potential rather than distance to a mate, and for a graph traversal problem, ease of traversing a local sub-graph rather than total path length. This tends to partition the total task into small segments which (hopefully) would be bounded by what is being measured. Thus, in the chess example the search would terminate when the attack issue is resolved (rather than continuing the analysis into the endgame), and in the path example the search would terminate once a local graph had been resolved with a convenient connection to another part of the total graph. In this way the evaluation functions can keep measuring the next thing to be optimized, and thus lead the process through a series of sub-optimal paths on the way to what is hoped to be the best ultimate goal. If new issues arise during a particular search, and De Groot presents some evidence that they do in chess searches, humans change their aspiration level (which could mean changing the evaluation function). We present analytic evidence for such changes in [4].

We have constructed reasonable bounding functions for chess tactics, although we did not then know of the B* algorithm [3]. The key for such constructions, is that one side's optimism is the other's pessimism. For instance, our evaluation function calculates the optimistic value of a tactical move to be the current material balance plus the sum of all our recognized threats against material. The pessimistic value is the material balance minus the sum of all the opponent's recognized threats.

When evaluation function estimates do not validly bound the actual value of a node, errors in arc selection can occur. However, there is no reason why these should be more severe than errors produced by any other search technique using an estimating function which is applied at leaf nodes that are not terminal in the domain *per se*. Thus, if an arc at the root is chosen because an estimate was in error somewhere in the tree, this would be no different than in searches with a single evaluation function. If the error was detected prior to termination, data from our simulations seem to indicate that small and infrequent intrusions of this type (which would happen when relatively good bounding functions do exist) have little effect on the magnitude of the search effort.

The issue of when a B* search fails to terminate is more serious and could still stand further investigation. In cases where the range of the best node at the root remains overlapped with the ranges of at least one of its sibling nodes even after a considerable search, several options are possible. One could select the arc with the greatest average value, or temper this with some function of the depth of investigation of each competing arc. Such contingencies probably arise in human searches ard are resolved by humans in such situations by various means that we have yet to understand. I am particularly struck by a quote from former World Chess Champion Alexander Alekhin in [6, p. 409]: "Well, in case of time pressure I would play 1. B×N/5". A clear case of using an external criterion to resolve some small remaining uncertainty.

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 [12, 13]. 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 [5]. In the latter great amounts of knowledge need to be brought to bear, since search is not practical because of a very large branching factor.

6. Comparison to other Search Algorithms

It is interesting to compare the basic features of B* with those of well-known search algorithms. Consider the A* search algorithm [11]. 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 1. 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.

Finally, it should be noted that the optimistic and pessimistic values at a node correspond exactly to alpha and beta in an alpha-beta minimax search. That is they delimit the range of acceptable values that can be returned from their sub-tree.

7. 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 selecting a strategy, it is good to consider the present range of a node, and the depth from which its current bounds have come. This allows some gauging of the cost of employing a particular strategy. Further, domain-dependent knowledge associated with an evaluation (not merely its magnitude), would no doubt also aid in strategy selection.

I have had a number of discussion with colleagues about whether B* is a Branch and Bound (BB) algorithm. The view that it is has support in that some BB algorithms do employ best-first methods and some do use upper and lower bounds to produce their effects. The BB technique derives its main effect by eliminating the search of branches that can not contain the solution. B* will never knowingly visit such branches either. In B*, superceded nodes can be (and are in our implementation) permanently eliminated from the search as a matter of course.

However, in our view, B* is definitely not a BB algorithm. The main strength of the B* algorithm is the ability to pursue branches that are known to *not be best*, and no other algorithm that we know of can opt for such a strategy. Therefore, we assert B* is quite different from the class referred to as BB algorithms. The reader may wish to judge for himself by perusing a comprehensive reference such as [9].

There are a number of issues left for investigation: (1) it is important to discover how to construct good bounding functions; (2) more light should be shed on the question of how much of an effect is caused when the value of a new node is not within the bounds of its parent; (3) there seems to be a more optimal strategy than always pursuing the best branch lower in the tree.

Given the fact that all search techniques are limited by the accuracy of the evaluations of leaf nodes, B* appears to expand fewer nodes than any other technique available for doing iterative searches. B* achieves its results because it is continually aware of the status of alternatives and exactly what is required to prove one alternative better than the rest. In this it seems very similar to the underlying method that humans exibit in doing game tree searches, and we conjecture that it is indeed this method that the B* algorithm captures.

Appendix I. How to Generate 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 immediate 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 note 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 descendants 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.

ACKNOWLEDGMENT

The author wishes to acknowledge the help of the following persons who provided assistance in discussing the properties of the algorithm, improvement in its formulation, and improvement in the readability of this paper: Allen Newell, Herbert Simon, Charles Leiserson, Andrew Palay, John Gaschnig, Jon Bentley, and the referees, whose comments were most helpful.

REFERENCES

1. Adelson-Velskiy, G. M., Arlasarov, V. L. and Donskoy, M. V., Some methods of controlling the tree search in chess programs, *Artificial Intelligence* 6(4) (1975).
2. Berliner, H. J., Some necessary conditions for a master chess program, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (August 1973) 77–85.
3. Berliner, H. J., Chess as problem solving: The development of a tactics analyzer, Dissertation, Comput. Sci. Dept., Carnegie-Mellon University (March 1974).
4. Berliner, H. J., On the use of domain-dependent descriptions in tree searching, in: Jones, A. K. (Ed.), *Perspectives on Computer Science* (Academic Press, New York, 1977).
5. Berliner, H. J., BKG — A program that plays backgammon, Comput. Sci. Dept., Carnegie-Mellon University (1977).
6. De Groot, A. D., *Thought and Choice in Chess* (Mouton and Co., Der Haag, 1965).
7. Harris, L., The heuristic search under conditions of error, *Artificial Intelligence*, 5 (1974) 217–234.
8. Harris, L. R., The heuristic search: An alternative to the alpha–beta minimax procedure, in: Frey, P. (Ed.), *Chess Skill in Man and Machine* (Springer-Verlag, Berlin, 1977).
9. Lawler, E. L. and Wood, D. E., Branch-and-bounds methods: A survey, *Operations Res.* 14 (1966) 699–719.
10. Newell, A., The chess machine: An example of dealing with a complex task by adaptation, *Proceedings Western Joint Computer Conference* (1955) 101–108.
11. Nilsson, N. J., *Problem-Solving Methods in Artificial Intelligence* (McGraw-Hill, New York, 1971).
12. Samuel, A. L., Some studies in machine learning using the game of checkers, *IBM J. Res. Develop.* 3 (1959) 210–229.
13. Samuel, A. L., Some studies in machine learning using the game of checkers, II — Recent Progress, *IBM J. Res. Develop.* (1967) 601–617.
14. Schofield, P. D. A., Complete solution of the 'Eight-puzzle', in: Collins, N. L. and Michie, D. (Eds.), *Machine Intelligence* 1 (American Elsevier Publishing Co., New York, 1967).
15. Slate, D. J. and Atkin, L. R., CHESS 4.5 — The Northwestern University Chess Program, in: Frey, P. (Ed.), *Chess Skill in Man and Machine* (Springer-Verlag, Berlin, 1977).