The Heuristic Search under Conditions of Error[1]

Larry R. Harris

Department of Mathematics, Dartmouth College, Hanover, N.H. 03755, U.S.A.

Recommended by U. Montanari

ABSTRACT

By placing various restrictions on the heuristic estimator it is possible to constrain the heuristic search process to fit specific needs. This paper introduces a new restriction upon the heuristic, called the "bandwidth" condition, that enables the ordered search to better cope with time and space difficulties. In particular, the effect of error within the heuristic is considered in detail.

Beyond this, the bandwidth condition quite naturally allows for the extension of the heuristic search to MIN/MAX trees. The resulting game playing algorithm affords many desirable practical features not found in minimax based techniques, as well as maintaining the theoretical framework of ordered searches. The development of this algorithm provides some additional insight to the general problem of searching game trees by showing that certain, somewhat surprising changes in the cost estimates are required to properly search the tree. Furthermore, the use of an ordered search of MIN/MAX trees brings about a rather provocative departure from the conventional approach to computer game playing.

1. Introduction

The heuristic search (Hart et al. [5], Nilsson [9]) has become an important aspect of artificial intelligence because of its wide area of application (Nilsson [9], Pohl [11]) and because of the theoretical framework on which it is based. This framework is in large part due to various restrictions imposed upon the heuristic that guides the search and the resulting effect on the search algorithm itself.

In order to discuss some of these restrictions it is necessary to introduce the following notation. For a node *n* of a tree or graph, the following functions are defined as part of the problem.

[1] Research partially sponsored by Office of Naval Research Contract ONR N00014-73-A-0260.

$k(m, n)$ = the arc cost from node m to node n.
$g(n)$ = the minimum arc cost from the root to node n.
$h(n)$ = the minimum arc cost from node n to a goal.
$f(n) = g(n)+h(n)$ the minimum arc cost from the root to a goal via node n.

Since some of these functions are not known during the actual search, we must be guided by estimates of these functions.

$\hat{g}(n)$ = an estimate of $g(n)$. Note: $\hat{g}(n) = g(n)$ for a tree.
$\hat{h}(n)$ = an estimate of $h(n)$. This is the "heuristic".
$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$ an estimate of $f(n)$.

At each step of the heuristic search the most promising node, the one with minimum $\hat{f}(\ )$ value, is expanded. The algorithm halts when attempting to expand a goal node. In this way the heuristic $\hat{h}(\ )$ orders the search. Using Nilsson's terminology, any node which has been expanded is called CLOSED, and any node not yet expanded is labeled OPEN. The open nodes are maintained on a list called the OPEN list. With this notion of the heuristic search, we can see the effect of certain restrictions placed upon the heuristic.

The most important restriction on the heuristic $\hat{h}(\ )$ is the "admissibility" condition (Hart et al. [5], Nilsson [9]): If $\hat{h}(n) \leqslant h(n)$ for all nodes n, then the heuristic search will always find a minimal cost (optimal) goal.

Another important restriction on the heuristic is the "consistency" condition (Nilsson [9]): If $\hat{h}(m) - \hat{h}(n) \leqslant k(m, n)$, then all nodes expanded by the heuristic search have $\hat{g}(n) = g(n)$ even if the graph is not a tree. This allows for considerable simplification of the algorithm.

This paper introduces another restriction on the heuristic, which allows the heuristic search to better cope with the practical problems of time and space.

2. The "Bandwidth" Condition

When using the heuristic search to solve complex problems time and space difficulties can arise as the number of nodes in the search tree increases. The time required to search the tree or graph can be reduced only by using a heuristic that more closely estimates $h(\ )$. It may be impractical to find such a heuristic that never violates the "admissibility" condition. For many problems, an example of which will be given shortly, it is often easier to find a heuristic that estimates the future cost well, but occasionally overestimates it. In these cases it is reasonable to make use of this inadmissible heuristic to order the search and then to choose from one of two alternatives.

(1) Bound the extra cost of the resulting goal by the extra cost in overestimating $h(\ )$. We may consider accepting a non-optimal solution, if it is much easier to find than an optimal one.

(2) Continue the search beyond the expansion of a goal, reducing the added cost bound until an optimal goal is found.

In either case the search process will run faster than an admissible search since the new heuristic is a more accurate estimator of $h(\ )$.

The "bandwidth" condition requires that

$$h(n) - d \leqslant \hat{h}(n) \leqslant h(n) + e.$$

The above constraints, one of which is a loosening of the admissibility condition, can be used to provide a theoretically sound method of coping with both the space and time difficulties encountered in solving large problems. We also assume that the bandwidth heuristic is exact for any goal. It is clear that, when $e = 0$ and $d > h(n)$ for all nodes, the bandwidth condition reduces to the admissibility condition. The bandwidth condition is similar to Pohl's "bounded error" [10]. However, our approach will be to study the relation between this error bound as applied to each node of the search and how it effects the cost of the goal found by the search. We will also demonstrate that bandwidth heuristics can be used effectively to find optimal goals, in spite of the possibility of over-evaluation.

The upper bound allows us to compute the extra cost of a non-optimal goal found by using $\hat{h}(\ )$ to guide the search. We can bound this added cost (Harris [4]) by noting that some node n^* on the path to the optimal goal is OPEN when the non-optimal goal p was selected for expansion. Thus,

$\hat{f}(p) \leqslant \hat{f}(n^*)$,	p expanded before n^*,
$g(p) + \hat{h}(p) \leqslant g(n^*) + \hat{h}(n^*)$,	definition of $\hat{f}$,
$g(p) + h(p) \leqslant g(n^*) + \hat{h}(n^*)$,	p is a goal so $\hat{h}(p) = h(p) = 0$,
$f(p) \leqslant g(n^*) + h(n^*) + e$,	bandwidth bound,
$f(p) \leqslant f(n^*) + e$,	definition of f,
$f(p) \leqslant f(p^*) + e$,	n^* on the path to p^*.

This provides the desired bound on the added cost of goal p. Such a goal is called *e-optimal* and the algorithm *e-admissible*.

Knowing this bound allows us to use heuristics that may not satisfy the admissibility condition without fear of encountering some degenerate case that could add an arbitrary additional cost to the resulting goal. In this sense bandwidth heuristics are easier to find than admissible heuristics since they are allowed to overestimate the true future cost.

The lower bound of the bandwidth condition allows us to save space by dropping nodes from the open list without surrendering the admissibility of the algorithm. We can eliminate any nodes m from the search if there exists a node q that satisfies the cutoff condition

$$\hat{f}(q) < \hat{f}(m) - (e + d).$$

It is clear from the bounding argument above that we must insure that the path to the optimal goal will not be dropped from the search tree. Assume that it is; then for some node q we have the following:

$$\begin{aligned} \hat{f}(q) &< \hat{f}(n^*) - (e + d), && \text{the drop criterion,} \\ f(q) - d &< f(n^*) + e - (e + d), && \text{bandwidth bounds,} \\ f(q) &< f(p^*), && n^* \text{ on path to } p^*. \end{aligned}$$

This is a contradiction since p^* is an optimal goal.

We can determine a bound on the extra cost for a goal p by comparing $f(p)$ to $\hat{f}(s) - e$ for all open nodes s, since one of these nodes is the n^* used above, in showing that p was e-optimal. If we are not satisfied with this result, we can continue until the bound is lowered to a desirable limit.

Knowing this bound, we may be satisfied with the e-optimal goal, or we may wish to continue the search for the optimal goal. This can be done since all open nodes will eventually have estimated cost greater than $f(p) + e$, where p is the minimum cost goal expanded by the search. In either case the use of a bandwidth heuristic speeds the search by better estimating future cost, and allows the dropping of obviously bad open nodes to conserve storage.

One may wonder whether the use of an admissible heuristic $\hat{h}' = \hat{h} - e$ would find the optimal goal faster than the bandwidth heuristic $\hat{h}$. Unfortunately, the "down weighting" of nodes will add a breadth-first component to the search because nodes deep in the tree cannot be "down weighted" as much as nodes high in the tree, since the $\hat{g}$ values must be accurate. The extreme case of this would be a goal node, whose $\hat{f}$ value cannot be biased at all. This weighting would tend to postpone the expansion of these nodes. It is better to estimate $h(\ )$ as accurately as possible, even if this means occasionally overestimating it.

For may problems it may be impractical to find a single heuristic that can adequately fulfill both of the bandwidth criteria at one time. In these cases we can define two heuristics, one to satisfy each bound.

$$h(n) - d \leqslant \hat{h}2(n) \quad \text{and} \quad \hat{h}1(n) \leqslant h(n) + e.$$

For other cases we may wish e and d to themselves be functions of the node n. The extension to these other cases is straightforward.

3. Example: The Traveling Salesman Problem

As an example of the practical value of the bandwidth heuristics consider the Traveling Salesman Problem of finding the minimum mileage tour of a set of cities. Since the search tree for an N-city problem has $(N - 1)!$ nodes it is impractical to find the optimal tour for large N. Recent ad hoc techniques (Lin and Kerningham [7]) find "good" solutions but can estimate the error only empirically.

The following heuristics can be used to find "good" (e-optimal) solutions. All of the heuristics make use of the reduced mileage matrix defined by the partial tour representing a node. For this problem it is convenient to use two heuristics $\hat{h}1$ and $\hat{h}2$ to fulfill the bandwidth condition. An admissible heuristic $\hat{h}0$ will be used to motivate the definitions of $\hat{h}1$ and $\hat{h}2$.

For a 6-city problem, the partial tour is 1 3,
cities to leave: 3 2 4 5 6, cities to enter 2 4 5 6 1

Reduced Mileage Matrix

$$\begin{array}{cc|ccccc} & & \multicolumn{5}{c}{\text{enter}} \\ & & 2 & 4 & 5 & 6 & 1 \\ \hline 1 & 3 & 3 & 4 & 5 & 2 & X \\ e & 2 & X & 4 & 1 & 2 & 2 \\ a & 4 & 4 & X & 3 & 2 & 1 \\ v & 5 & 1 & 3 & X & 4 & 2 \\ e & 6 & 2 & 2 & 4 & X & 3 \end{array}$$

$$\hat{h}0 = \max(2 + 1 + 1 + 1 + 2, 1 + 2 + 1 + 2 + 1) = 7$$

An admissible heuristic $\hat{h}0$ would be the maximum of the sums of the row and column minima. The heuristic $\hat{h}0$ is admissible since any tour must use one element from each row and column and can do no better than the minimum element. Thus $\hat{h}0(n) \leqslant h(n)$ for all nodes n. A more accurate heuristic would be one that accounts for the likelihood that the minimum elements of all the rows cannot all be used to form a tour. This can be done by taking the weighted average of the two smallest elements in each row.

$$\hat{h}1 = w(2 + 1 + 1 + 1 + 2) + (1 - w)(3 + 2 + 2 + 2 + 2).$$

This heuristic gives some weight to the alternative choices. If it is indeed possible to use only the minimum elements, then this heuristic will overestimate the true future cost. However for the majority of nodes it will provide a much more accurate estimate of the future cost than $\hat{h}0$ and thus it provides a better guide for the search. It is because this heuristic provides a more realistic estimate of $h(\ )$ that the search algorithm runs faster than the admissible search. Of course the resulting solution may be more costly, but we can bound this added cost by knowing how far we can overestimate a particular node.

The weight w can also be made proportional to the depth of the node to indicate the greater accuracy of the heuristic when the reduced matrix becomes smaller. This gives a depth first bias to the search that is critically dependent on the node configuration. This selective depth bias is extremely helpful in solving the large problems shown at the end of this section.

The heuristic $\hat{h}2$ is an estimate on how big $h(n)$ can get for a particular node n. For the Traveling Salesman Problem, given any partial tour, we need only estimate the cost of the rest of the tour from this point to calculate this bound. Using the reduced cost matrix we could simply sum the major diagonal, or for a lower and therefore better estimate we could take the minimum choice from each city until a tour is complete. Each of these will have $d = 0$ since we know the minimum future cost must be lower than or equal to the cost of a particular tour. Typically $\hat{h}2$ is only calculated when there is a need to drop nodes.

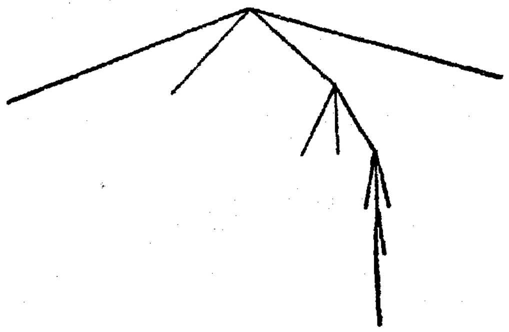


FIG. 1. 6-city bandwidth; 5-admissible, 6 expansions.

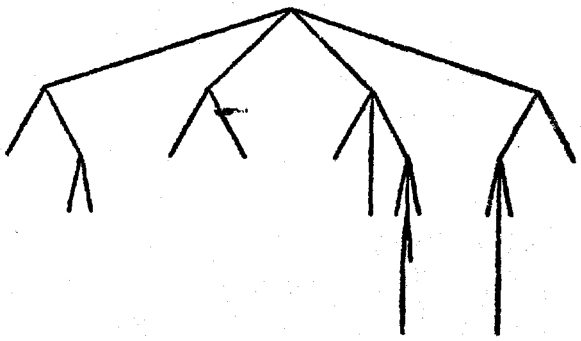


FIG. 2. 6-city bandwidth; optimal goal, 14 expansions.

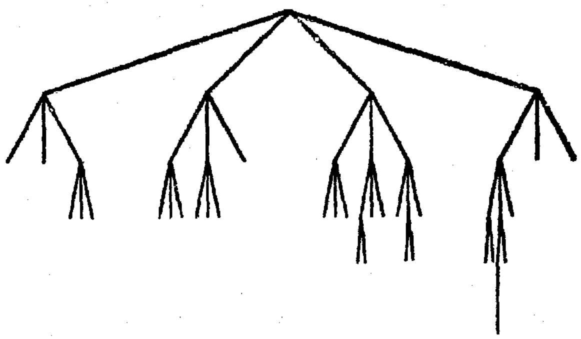


FIG. 3. 6-city admissible; optimal goal, 18 expansions.

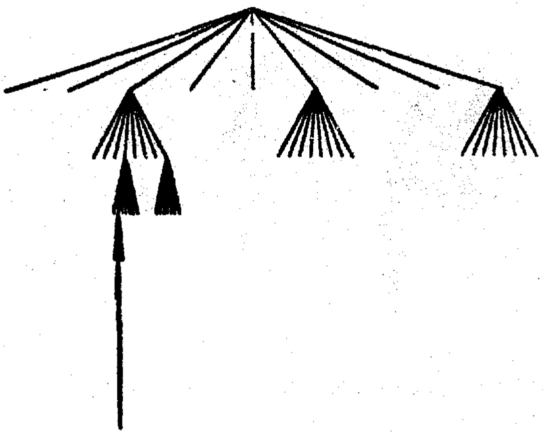


FIG. 4. 11-city bandwidth; optimal goal, 14 expansions.

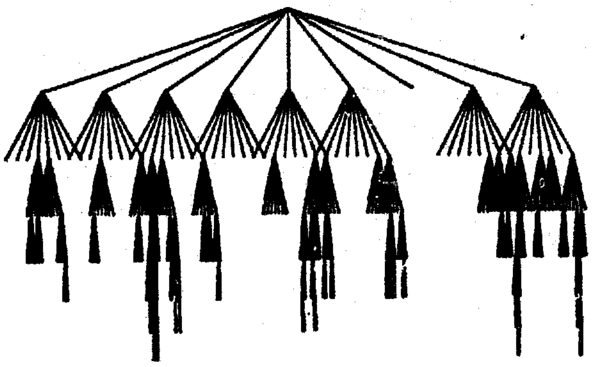


FIG. 5. 11-city; admissible; no goal found: 500 open nodes.

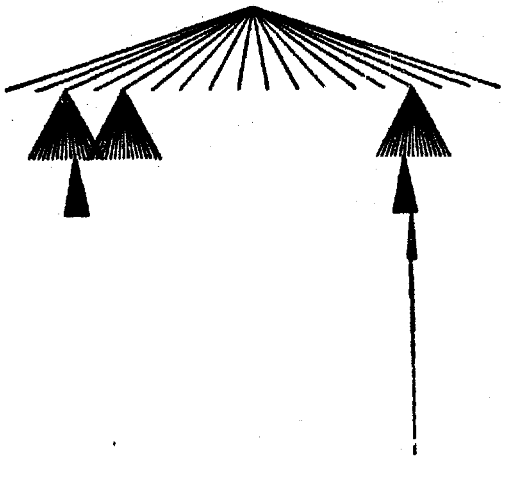


FIG. 6. 20-city bandwidth; 4-admissible, 42 expansions.

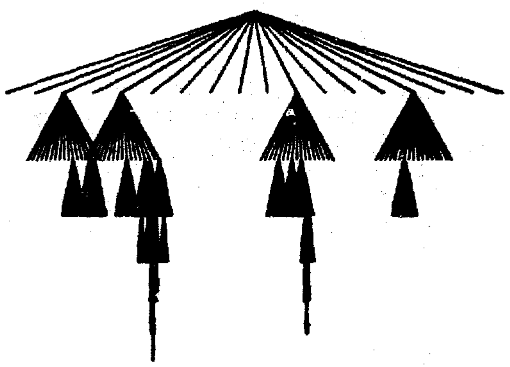


FIG. 7. 20-city admissible; no goal found: 500 open nodes.

Thus for each node n, we know that we can do no worse than $\hat{h}2(n)$. Knowing this we can drop any node m if there exists a node q such that $\hat{f}2(q) < \hat{f}1(m) - (e + d)$ since any path through m must cost more than a path through q.

Several search trees for solutions to the Traveling Salesman Problem appear in the preceding figures. They visually demonstrate the search efficiencies of the admissible heuristic $\hat{h}0$ and the bandwidth heuristic $\hat{h}1$.

Since the data for these problems is symmetric, the search space can be halved by forcing the last city to appear in the last half of the tour. This factor was used by both heuristics, and is responsible for the reduced branching in the upper half of the trees. A simple 6-city problem from Little et al. [8] is shown first, to demonstrate how the bandwidth heuristic can be used to find the optimal tour (Figs. 1–3). It is followed by a more difficult 11-city problem (Figs. 4–5) and the 20-city problem of Croes [3] (Figs. 6–7) to indicate the hopelessness of this admissible heuristic for large problems, as well as the accuracy of the bandwidth heuristic. The weight w in the definition of $\hat{h}1$ was made a function of depth as discussed above.

4. Searching Game Trees

Given the success of ordered searches in so many problem domains it seems reasonable to extend the heuristic search to game playing. The advantages of ordering the search should be as apparent in game playing as in the Traveling Salesman Problem. Unfortunately special problems occur which require significant changes in the algorithms in order to maintain certain desirable properties.

We would like to be able to make the optimal move from a given board configuration. This would require being able to prove that we can win or tie from an arbitrary board configuration, since someone must be able to force a win or tie in a 2-person 0-sum game. It is not that we cannot devise algorithms to perform this proof (Nilsson [9], Chang and Slagle [2]), but that for any interesting game the algorithms are impractical.

If we develop admissible algorithms for game trees, such as Nilsson's ordered search procedure for AND/OR trees, critics respond that such an algorithm is impractical for any interesting game such as checkers or chess. On the other hand, algorithms that run in a reasonable amount of time are criticized because they don't always make the "best" move, and that they lack the theoretical framework of the admissible algorithms.

The use of a bandwidth heuristic allows us to work somewhere between these two extremes. We begin by reducing our aim in two ways; first, by looking for an $(e + d)$-optimal goal instead of an optimal one and second, by looking only for the first move towards such a goal. The bandwidth heuristic search can achieve these more modest goals.

The standard technique for finding a good move from a given board position in 2-person games is the alpha–beta minimax (Samuel [12]) and its variations (Slagle [13], Nilsson [9]). It is helpful to note some of the possible shortcomings of minimaxing to motivate an improved technique. The following detrimental factors arise when minimaxing.

(1) The minimax technique implicitly assumes that the opponent is using the same evaluation of board positions as the program when it minimizes on alternate ply levels. This assumption must be made in order to determine what move the opponent will make at each of these levels. It is clear that this is an unwarranted assumption that can lead to serious errors in play.

(2) Related to this problem is the fact that minimaxing makes permanent decisions on the basis of comparing board evaluations. If the evaluator misorders two boards, the minimax will never recover. We are forced to admit that the evaluator *must* misorder some nodes or else we could play optimally with no tree search at all. In light of the inaccuracy of the board evaluator, it seems unwarranted to make irrevocable decisions based upon it.

(3) Minimaxing indicates what line of play looks most promising in a particular subtree, but does not provide an error estimate of how good the move is with respect to the entire move tree. It is desirable to provide such a definition of a "good" move.

(4) Since minimaxing regenerates a portion of the move tree each time the program is to move, there is little consistency in play from move to move. While substantial savings in computation can be obtained by ordering the generation of the tree (alpha–beta), there is little tendency to continue a particular line of attack on the next move because the new search is based solely upon values generated from 2 levels deeper in the tree. Since these values tend to vary somewhat from their ancestors the line of play suggested earlier may not be followed.

The bandwidth heuristic search provides a means of coping with each of these problems inherent to minimaxing. Before presenting the algorithm in detail, some definitions are required. First we define the cost functions based upon one cost unit per move. For OPEN nodes of the search tree:

$\hat{g}(n) = g(n)$,	cost (number of moves) from the root to node n,
$h(n) - d \leqslant \hat{h}(n) \leqslant h(n) + e$,	estimate of future cost,
$\hat{f}(n) = \hat{h}(n) + \hat{g}(n)$,	total estimated cost via node n.

For terminal nodes of the search tree:

$\hat{h}(\text{WIN}) = 0$ and $\hat{h}(\text{LOSS}) = N$ where N is a large number.

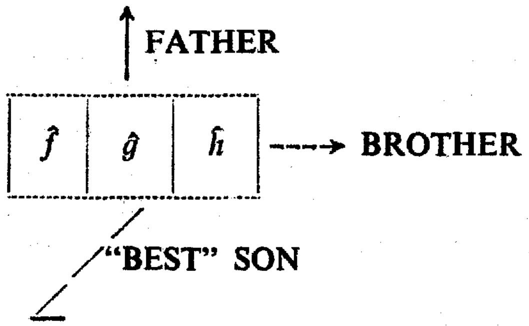
For closed nodes of the search tree these functions are defined recursively by backing the values of the "best" son up the tree. From this it is clear that $\hat{h}(\ )$ estimates the number of moves to a win. Note that this means a low value is good for the program, the opposite of the minimax evaluator. Special care is taken when estimating losing nodes since strictly speaking, $h(\text{LOSS})$ is infinite. For the purposes of the algorithm we merely assume that $h(\text{LOSS}) = N$.

A terminal node p is a *goal* if it is a win in fewer than $N - e$ moves, i.e. $h(p) = 0$ and $f(p) < N - e$.

A node n is said to be *accessible* if all the opponent's moves on the path to n represent the best move for him in terms of *true* cost. That is, for each MIN node m on the path to $n, f(m) \geqslant f(b)$ for all brothers b of m.

4.1. The bandwidth heuristic search algorithm

A linked search tree is maintained using the following node specification.



Closed nodes of the search tree typically contain the 3 pointers plus values for $\hat{f}(n)$, $\hat{g}(n)$ and possibly $\hat{h}(n)$. Typically the linked list of brothers is sorted in increasing or decreasing order for MAX or MIN levels respectively. Tip nodes of the search tree must also contain the board configuration or a list of moves from which the board can be constructed.

The bandwidth heuristic search proceeds as follows.

(1) Select a node p for expansion by following the "best" son chain to a tip node (the node p).

(2) If p is a goal node (a Win), stop and make the 1st move towards p.
(3) If p is a terminal node (a Loss), resign or play towards p expecting to lose.
(4) Otherwise, expand p (on both a MIN and MAX level) using the drop criterion (see (8) below) to avoid the needless generation of the grandsons of p (similar to the alpha beta cutoff). Note: a check must be made that a terminal node is not generated on the first expansion of p.
(5) Evaluate $\hat{h}(\ )$ and calculate $\hat{f}(\ )$ for all grandsons of p. Add the generated nodes to the search tree.
(6) Recalculate $\hat{f}(p)$ using the recursive definition of $\hat{f}(\ )$.
(7) Recalculate $\hat{f}(\ )$ for all ancestors of p and resort the brother lists to maintain the proper order.
(8) Drop any node n (and the subtree below n) that has a brother b such that

$$n \text{ on a MIN level and } \hat{f}(b) > \hat{f}(n) + (e + d),$$
$$n \text{ on a MAX level and } \hat{f}(b) < \hat{f}(n) - (e + d).$$

(9) Go to (1).

The algorithm runs in reasonable time because it is not required to find the optimal accessible goal, but only the first move towards an $(e + d)$-optimal accessible goal of cost less than N.

In order to prove that the Bandwidth Heuristic Search finds the 1st move to an $(e + d)$-optimal, accessible goal, we first prove two important facts concerning the search tree.

LEMMA 1. *Within any subtree of the search tree, a node on the path to an optimal accessible goal, if such a goal exists, cannot be dropped.*

Proof. By induction down the tree. Initially the root of the subtree is on the path to its optimal accessible goal. As the subtree is expanded we show that a node n^* on the path to an optimal accessible goal p^* cannot be dropped on either a MAX or a MIN level.

MAX level: Assume that n^* is dropped by a brother node b.

$$\begin{array}{rl}
\hat{f}(b) < \hat{f}(n^*) - (e + d), & \text{the "drop" criterion,}\\
g(b) + \hat{h}(b) < g(n^*) + \hat{h}(n^*) - (e + d), & \text{definition of } \hat{f},\\
g(b) + h(b) - d < f(n^*) + h(n^*) + e - (e + d), & \text{bounds on } \hat{h},\\
f(b) < f(n^*), & \text{definition of } f,\\
f(b) < f(p^*), & n^* \text{ on path to } p^*.
\end{array}$$

This contradicts the optimality of p^*.

MIN level: Assume that n^* is dropped by node b.

$$\begin{array}{rl}
\hat{f}(b) > \hat{f}(n^*) + (e + d), & \text{the "drop" criterion,}\\
g(b) + \hat{h}(b) > g(n^*) + \hat{h}(n^*) + (e + d), & \text{definition of } \hat{f},\\
g(b) + h(b) + e > g(n^*) + h(n^*) - d + (e + d), & \text{bounds on } \hat{h},\\
f(b) > f(n^*), & \text{definition of } f.
\end{array}$$

This contradicts the accessibility of p^*.

LEMMA 2. *For all nodes n of the search tree the following bound is maintained:*

$$f(n) - d \leqslant \hat{f}(n) \leqslant f(n) + e.$$

Proof. By induction up the search tree. Clearly the above bound is true for all tip nodes of the search tree since we can immediately apply the bandwidth conditions to $\hat{h}(\ )$. The desired bound on $\hat{f}(\ )$ remains true for non-tip nodes as values are backed up the search tree on either a MIN or a MAX level. The son of node n that is backed up (has the "best" $\hat{f}(\ )$ value) is labeled s, and the truly "best" son (determined by $f(\ )$ value) is labeled s^*.

MIN level:

$$\begin{array}{rl}
\hat{f}(s) \leqslant \hat{f}(s^*), & s \text{ backed up in preference to } s^*, \\
f(s) - d \leqslant \hat{f}(s) \leqslant f(s^*) + e, & \text{inductive hypothesis,} \\
f(s) - d \leqslant \hat{f}(n) \leqslant f(s^*) + e, & \text{value of } s \text{ backed up to } n, \\
f(n) - d \leqslant \hat{f}(n) \leqslant f(s^*) + e, & f(n) \leqslant f(s) \text{ on a MIN level,} \\
f(n) - d \leqslant \hat{f}(n) \leqslant f(n) + e, & f(n) = f(s^*) \text{ since } s^* \text{ is "best" son.}
\end{array}$$

MAX level:

$$\begin{array}{rl}
\hat{f}(s^*) \leqslant \hat{f}(s), & s \text{ backed up in preference to } s^*, \\
f(s^*) - d \leqslant \hat{f}(s) \leqslant f(s) + e, & \text{inductive hypothesis,} \\
f(s^*) - d \leqslant \hat{f}(n) \leqslant f(s) + e, & \text{value of } s \text{ backed up to } n, \\
f(s^*) - d \leqslant \hat{f}(n) \leqslant f(n) + e, & f(n) \geqslant f(s) \text{ on a MAX level,} \\
f(n) - d \leqslant \hat{f}(n) \leqslant f(n) + e, & f(n) = f(s^*) \text{ since } s^* \text{ is "best" son.}
\end{array}$$

With these two facts regarding the search tree we can easily prove the admissibility of the algorithm.

THEOREM. *If an accessible goal exists, then the Bandwidth Heuristic Search will find the first move to an* $(e + d)$*-optimal, accessible goal.*

Proof. The algorithm cannot loop indefinitely since the $\hat{h}$ value for a node can never decrease by more than e, and the $\hat{g}$ values increase as expansions take place. Eventually the $\hat{f}$ value of the "best" tip node will reach the value of an accessible goal and halt in step (2), or exceed N and halt in step (3).

Under the conditions of the theorem the algorithm cannot halt in step (3) since Lemma 1 insures that a node on the path to the optimal accessible goal p^* exists in the search tree, and Lemma 2 insures that all nodes on this path will be rated less than N. Therefore a loss, with a known value greater than N cannot be chosen for expansion since the MIN level at the root will always prefer the path to p^*.

Therefore the algorithm halts in step (2) attempting to expand a goal node p. Consider where the path to p and the path to p^* intersect. If they do not intersect at the root, then the first move to p is also towards p^* and the theorem holds.

If the paths intersect at the root then let q be the first node on the path to p, and let n^* be the first node on the path to p^*. Note that the root is a MIN level.

$\hat{f}(q) \leqslant \hat{f}(n^*)$,	p was chosen for expansion,
$f(q) - d \leqslant f(n^*) + e$,	Lemma 2,
$f(q) - d \leqslant f(p^*) + e$,	n^* on the path to p^*,
$f(q) \leqslant f(p^*) + (e + d)$,	q.e.d.

Thus the error introduced by the imprecision of the heuristic is bounded, and the result pertains to the entire move tree, not just to the search tree.

4.2. The alternating definition of $\hat{f}(\ )$

In the standard heuristic search the purpose of the cost function $\hat{g}(\ )$ is to add a breadth-first component to the search. A particular path in the tree will continue to be expanded only if the decrease in $\hat{h}(\ )$ outweighs the increase in $\hat{g}(\ )$. In order to keep this desirable tendency for the bandwidth search it is necessary to alter the common definition of $\hat{f}(\ )$ on MIN levels.

If $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$ for all levels in the tree, then the maximizing player will see a false improvement along a line that is expanded even when the $\hat{h}$ value remains constant. We must force a breadth-first component to the opponent's choices as well as the program's choices. To do this we must alter the fact that on MIN levels going deeper in the tree makes the opponents moves look *better* instead of worse. This is clearly the opposite of the case for the program's choices, and of heuristic searches in general. The following backup technique will provide the necessary breadth-first component.

Node n on a MAX level: Define node m to be that son of n that is the estimated best move for the opponent. That is, $\hat{f}(m) \geqslant \hat{f}(k)$ for all sons k of n.

$$\hat{h}(n) = \hat{h}(m), \quad \hat{g}(n) = \hat{g}(m), \quad \hat{f}(n) = \hat{h}(n) + \hat{g}(n).$$

Node n on a MIN level: Define node m as the estimated best move for the program. That is, $\hat{f}(m) \leqslant \hat{f}(k)$ for all sons k of n.

$$\hat{h}(n) = \hat{h}(m), \quad \hat{g}(n) = \hat{g}(m), \quad \hat{f}(n) = \hat{h}(n) - \hat{g}(n).$$

If we do not use this alternating definition of $\hat{f}(\ )$, the search will tend to investigate only 1 or 2 of the opponents moves at each level. Given the inaccuracies inherent to the heuristic, it is clear that we must provide for a more conservative search.

Using this alternating definition of $\hat{f}(\ )$, Lemma 2 will strictly be satisfied only on MAX levels, but this is all that the proof of admissibility required.

4.3. Finding bandwidth heuristics

Before considering the Bandwidth Heuristic Search any further we must answer two questions which immediately come to mind. First, can we really find a bandwidth heuristic for a game such as chess or checkers? Second, do we really expect the algorithm to halt by expanding a goal each time the computer is to make a move?

The first question should be rephrased to ask: "For what values of e and d can we find a bandwidth heuristic for a game such as chess?" It is clear that for large e and d bandwidth heuristics can easily be found, and in fact the minimax board evaluators satisfy the conditions for some e and d.

For complex games such as chess it is unlikely that heuristics that can be proved to satisfy the bandwidth conditions can be found. In this case the e-admissibility of the algorithm becomes academic. However, the practical advantages of the bandwidth search process remain unaffected. The e and d then become parameters of the search to vary the density of the search tree.

The loosening of the admissibility condition to the bandwidth condition was originally meant to simplify the task of finding a suitable heuristic for a specific problem, not make it more difficult. The main distinction made by the bandwidth search is that it readily admits that there are inaccuracies in the heuristic, and that we must cope with these in a more subtle way than a single comparison of heuristic values.

The answer to the second question indicates the change in outlook implied by the bandwidth search. The algorithm is not meant to start and stop each move as is the case when minimaxing. The bandwidth heuristic search expands the move tree continuously, even while it is the opponent's turn to move. Of course as actual moves are made the unused sections of the search tree can be discarded. The algorithm only suggests moves when it is forced to stop because of time or space constraints. Under these conditions programs can play as well as current techniques with a near zero response time, since much of the tree search could take place while the opponent is thinking! The expansion of a goal must take place at some time during the game, but probably not before move decisions have to be made.

We must therefore devise a decision procedure for picking a move when the algorithm has not halted, but time or space constraints force us to make a move. The first guess one might have is to make the move towards the node to be expanded next. That is, the accessible node with the minimum $\hat{f}(\ )$ value. However this is very dangerous since this node could be a misevaluated line of play that would be discarded immediately after its expansion. A safer technique would be to move towards the optimal accessible node as determined by $\hat{h}(\ )$ values. This gives a strong bias toward lines of play that have been looked into deeply, since no cost is assigned to a move. It also allows the program to save more of the move tree for use next move, as well as keeping the program from making serious mistakes.

Let us now view the bandwidth search in light of our discussion of minimaxing. The bandwidth search, as would any ordered search, uses heuristic values as a *means* of guiding the search, not as an *end* in determining the best move. That is, the error due to a misevaluation of two nodes is not critical, as with the minimax based algorithms, since the misevaluation can be corrected after further expansion. The bandwidth search quite naturally follows forces or obvious lines of play to an arbitrary depth in the move tree. It is not restricted to an a priori depth bound.

Since the accessibility of a node is defined in terms of TRUE cost the bandwidth search does not assume the opponent evaluates boards in a manner similar to the program. The bandwidth search does assume that the opponent is trying to win the game in the fewest moves possible. This is a much more plausible assumption than the one mentioned above for minimaxing.

A "good" move is defined as the first move towards an $(e + d)$-optimal accessible goal. If the estimated cost of this first move is less than $N - (e + d)$, then the program must win from this position. If the value is greater than N, then the program can expect to lose. However, if the value is in the "uncertainty zone" $[N - (e + d), N]$, as it will be most of the time, then it is unknown who can win and the game should be continued.

In essence, we are saying that the heuristic should be accurate on a coarse scale, but may not necessarily be accurate on a fine scale. That is, for $\hat{h}(\ )$ values sufficiently far apart the ordering of the heuristic should be the same as the $h(\ )$ ordering. Sufficiently far apart is, of course, defined as one bandwidth $(e + d)$. On a fine scale we admit the inaccuracy of the heuristic and reserve judgement until the nodes have been expanded further. Thus, when $\hat{h}(\ )$ values appear in the "uncertainty zone", all is not lost, we must simply continue to expand the search tree until the heuristic values become more exact.

4.4. Empirical results

In order to better isolate the effect of the bandwidth search, extensive tests were conducted using the games of chess and Four Score, an interesting variant of three dimensional tic-tac-toe. In both cases existing programs that used a fixed ordered alpha–beta minimax were modified to use the bandwidth heuristic search. The new version then competed against the old program as well as human opponents. When two programs completed identical time and space constraints were imposed on each, and both programs used the identical heuristic evaluator. Since all the programs are deterministic it is only possible to play two unbiased games, one with each program going first.

For chess the bandwidth search noticeably improved the play of the program and easily won both games, playing white once and black once. At no time in either game did the minimax program mount a serious offensive attack although its defense was acceptable. In order to more accurately assess the improvement in chess play a local chess master, rated at 2200, was asked to play each program until he had some feel for their ratings. His estimate was 1100–1200 for the minimax program, and 1400–1500 for the bandwidth search program.

The bandwidth search fared well in Four Score competition, winning while moving first and second. When playing first it was able to effect an 8 move combination in the middle game, which was beyond the horizon of the minimax. Four Score has no standardized rating scheme, thus no meaningful numeric comparison is possible.

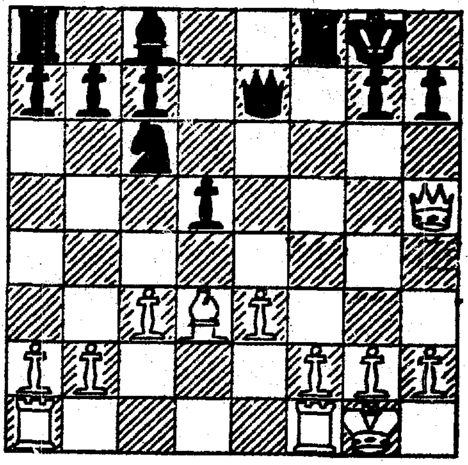


FIG. 8.

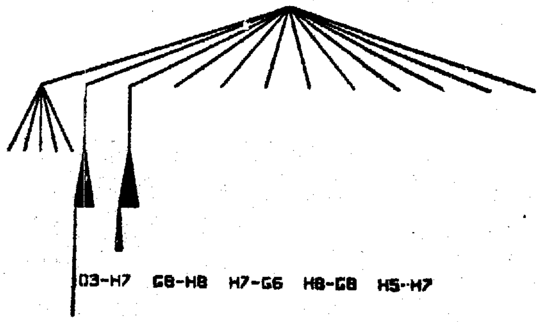


FIG. 9. Bandwidth search; optimal goal, 10 expansions.

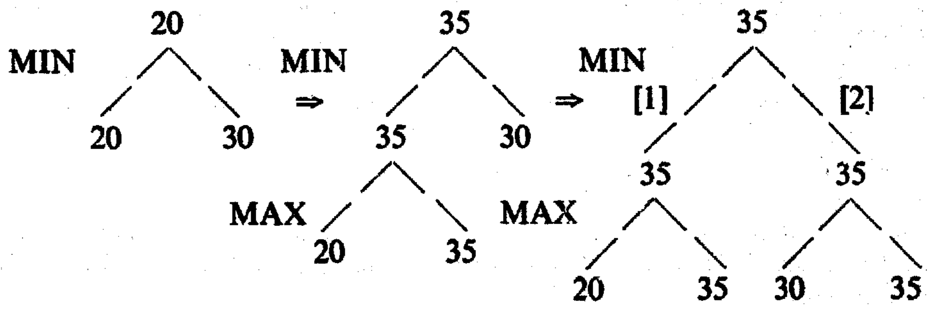
Since it is known that minimax chess programs can exceed the 1200 rating it is clear that the chess heuristic used in this experiment is not the best possible. Since the bandwidth search makes more extensive use of the heuristic than does minimaxing, it seems intuitive, although unsupported, that the bandwidth search should improve at least as fast as the minimax search, for every improvement in the heuristic.

The above chess example indicates the main difference in approach to search game trees made by the heuristic search. Berliner [1] has criticized minimax techniques for requiring too much work to find tactical lines from positions in which chess masters find them instantly. He suggested the board that has a mate in five (Fig. 8). He estimates minimax programs, even with near optimal cutoffs, must evaluate over 100 positions to find the mating line. The point is not that the search will not find the mate, but that if this board appeared itself 5 ply deep in a search tree, that this amount of effort is too much to expend at that depth, and the program would not correctly value this position. Chess masters, on the other hand, would intuitively know the mate threat upon encountering the board, even in a deep search.

The bandwidth search finds the mate almost directly, as indicated by the search tree, and expends little effort (10 expansions) in doing so. If this board appeared deep in a tree search, the line *could* be exploited because expansion of a line this thin presents no problem at any level of the search. This ordered search technique has found mating sequences as deep as 19 ply since these sequences typically are very sparse and mostly forced.

4.5. Double expansion

It is possible to define an ordered search along the same lines as the Bandwidth Search without using double expansion. The following case demonstrates a more subtle effect that would cause a single expansion algorithm to order the search incorrectly.



When the heuristic is overly sensitive to the last move, the single expansion search can actually reverse the initial ordering of the nodes. Move [2] will be considered next since it was expanded last, even though move [1] is the better play as initially indicated by the heuristic. This problem becomes more acute as the number of possible moves increases, since the entire list of moves can be reversed on a MIN level before progressing beyond the next MAX level.

5. Conclusion

The "bandwidth" conditions for the heuristic search provide a convenient means for coping with the practical constraints of time and space, as well as for maintaining the admissibility of the search.

For game trees the bandwidth conditions lead to an effective search procedure that has many desirable features over the conventional minimax techniques. Moreover, the bandwidth search brings about a totally different view of computer game playing. Instead of regenerating sections of the move tree each time the program is to move, the bandwidth search expands the move tree over a continuum of time. This allows for more coherent play from move to move, as well as the capability of "thinking" while the opponent is making his move. The bandwidth search minimizes the critical dependence upon the heuristic and eliminates the need for fixed a priori limitations of the search process.

Another important aspect of this approach to game playing is the capability of the program to interact with a human partner in ordering the search of the move tree. All of these areas remain open for future research.

REFERENCES

1. Berliner, H. J. Some necessary conditions for a master chess program. *Proc. Third International Joint Conf. on Artificial Intelligence* (1973), 77.
2. Chang, C. L., and Slagle, J. R. An admissible and optimal algorithm for searching AND/OR graphs. *Artificial Intelligence* 2 (1971), 117–128.
3. Croes, G. A method for solving traveling salesman problems. *Operations Research* 6 (1958), 791–812.
4. Harris, L. R. A model for adaptive problem solving applied to natural language acquisition. Technical Report TR-133, Computer Science Dept., Cornell University (1972).
5. Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basic for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Sci. and Cybernetics* 4 (1968), 100–107.
6. Lawler, E., and Wood, D. Branch and bound methods: A survey. *Operations Research* 14 (July–August 1966), 699–719.
7. Lin, S., and Kernigham, B. W. A heuristic algorithm for the traveling salesman problem. Bell Telephone Laboratories Computer Science Technical Report No. 1 (April 1972).
8. Little, J. D., Murtz, K. G., Sweeney, D. W., and Karel, C. An algorithm for the traveling salesman problem. *Operations Research* 11 (November–December 1963), 972–989.
9. Nilsson, N. J. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York (1971).
10. Pohl, I. First results on the effect of error in heuristic search. *Machine Intelligence* 5 (1969), 219–236.
11. Pohl, I. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1 (1970), 193–204.
12. Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM J. Res. Develop.* 3 (1959), 210–229.
13. Slagle, J. R. Game trees, M and N minimaxing and the M and N alpha-beta procedure. Lawrence Radiation Laboratory AI Report No. 3, Livermore, Calif. (November 1970).

Received September 1973; revised version received March 1974