NILS J. NILSSON Artificial Intelligence Group
Stanford Research Institute, Menlo Park, California

PROBLEM-SOLVING METHODS IN ARTIFICIAL INTELLIGENCE

© 1971

McGRAW-HILL BOOK COMPANY
New York San Francisco St. Louis Düsseldorf
Johannesburg Kuala Lumpur London Mexico Montreal
New Delhi Panama Rio de Janeiro
Singapore Sydney Toronto

A would certainly have missed it and thus would be inadmissible, contrary to our assumption. Thus, we must assume that algorithm A "knew" that there was no less costly path through node n.) The actual cost of an optimal path constrained to go through node n is

$$f(n) = g(n) + h(n)$$

from which we can state that

$$h(n) = f(n) - g(n)$$

Now, as argued above, algorithm A knows that $f(n) \geq f(s)$ and therefore it knows that

$$h(n) \geq f(s) - g(n)$$

Such information available to algorithm A would permit a lower bound estimate of

$$\hat{h}(n) = f(s) - g(n)$$

On the other hand, algorithm A^* used the evaluation function

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$$

We know from Lemma 3-3 that

$$\hat{f}(n) \leq f(s)$$

Thus we know

$$\hat{g}(n) + \hat{h}(n) \leq f(s)$$

Therefore, whatever the $\hat{h}$ function used by A^* it must have satisfied the inequality

$$\hat{h}(n) \leq f(s) - \hat{g}(n)$$

Now, at the time A^* expanded node n, $\hat{g}(n) = g(n)$ by Lemma 3-2 and thus

$$\hat{h}(n) \leq f(s) - g(n)$$

But now we see that, at least for node n, algorithm A used information permitting a lower bound on h at least as large as the lower bound used by algorithm A^*. Thus A^* could not be more informed than A, contradicting our assumption and proving the theorem.

3-10 THE HEURISTIC POWER OF $\hat{h}$

The selection of $\hat{h}$ is crucial in determining the heuristic power of the ordered-search algorithm. Using $\hat{h} \equiv 0$ assures admissibility but results in blind search and is thus usually inefficient. Setting $\hat{h}$ equal to the highest possible lower bound on h expands the fewest nodes consistent with maintaining admissibility.

Often heuristic power can be gained at the expense of giving up admissibility by using for $\hat{h}$ some function that is not a lower bound on h. This added heuristic power then allows us to solve much harder problems. In the 8-puzzle, the function $\hat{h}(n) = W(n)$ (where $W(n)$ is the number of tiles in the wrong place) is a lower bound on $h(n)$, but it does not provide a very good estimate of the difficulty (in terms of number of steps to the goal) of a tile configuration. A better estimate is the function $\hat{h}(n) = P(n)$, where $P(n)$ is the sum of the distances that each tile is from "home" (ignoring intervening pieces). Even this estimate is too gross, however, in that it does not accurately appraise the difficulty of exchanging the positions of two adjacent tiles.

An estimate that works quite well for the 8-puzzle is

$$\hat{h}(n) = P(n) + 3\,S(n)$$

The quantity $S(n)$ is a *sequence score* obtained by checking around the noncentral squares in turn, allotting 2 for every tile not followed by its proper successor and 0 for every other tile, except that a piece in the center scores 1. We note that this $\hat{h}$ function does not provide a lower bound for h.

With this $\hat{h}$ used in the evaluation function $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, we can easily solve much more difficult 8-puzzles than the one we solved earlier. In Fig. 3-8 we show the tree resulting from applying the ordered-search algorithm with this evaluation function to the problem of transforming

2	1	6
4	■	8
7	5	3

into

1	2	3
8	■	4
7	6	5

Again the $\hat{f}$ value of each node is circled, and the uncircled numbers show the order in which nodes are expanded.

The solution path found happens to be of minimal length (18 steps), although since the $\hat{h}$ function is not a lower bound for h, we were not guaranteed finding an optimal path. Note that this $\hat{h}$ function results in a focusing of search rather directly toward the goal; only a very limited amount of spreading occurred, and that was near the start.

Another factor that determines the heuristic power of the ordered-

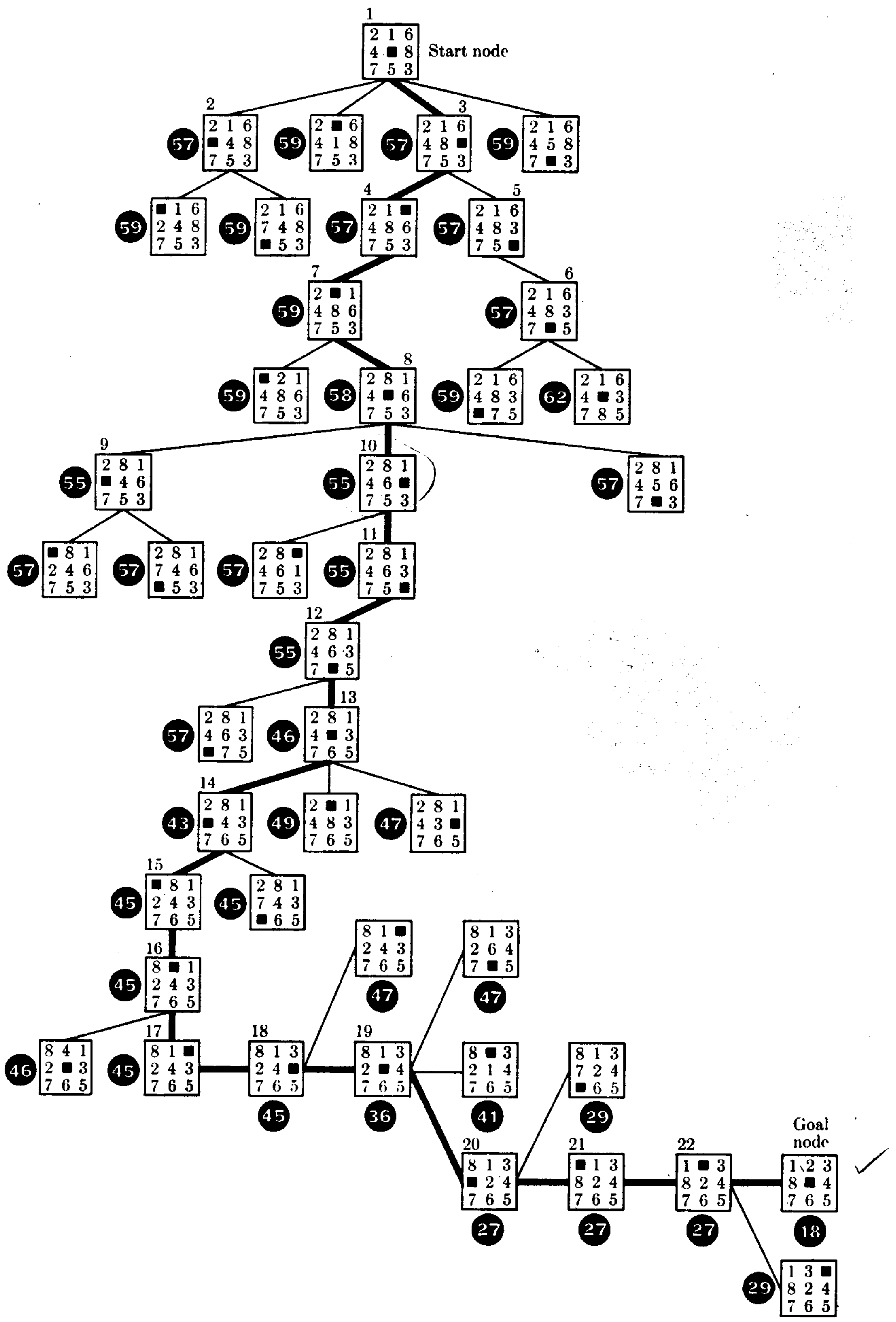

FIG. 3-8 *A tree produced by an ordered search.*

search algorithm is the amount of effort involved in calculating $\hat{h}$. The best $\hat{h}$ would be a function identically equal to h, resulting in an absolute minimum number of node expansions. Such an $\hat{h}$ could, for example, be determined as a result of a separate complete search at every node, but this obviously would not reduce the total computational effort.

Sometimes an $\hat{h}$ function that is not a lower bound on h is easier to compute than one that is. In these cases the heuristic power might be doubly improved—because the total number of nodes expanded may be reduced (at the expense of admissibility) *and* the computational effort is reduced. In certain cases the heuristic power of a given $\hat{h}$ can be increased simply by multiplying it by some positive constant greater than unity. If this factor is *very* large, the situation is as if $\hat{g}(n) \equiv 0$. Such a selection, of course, leads to an inadmissible algorithm but one that might nevertheless perform satisfactorily. In fact, one might intuitively suppose that setting $\hat{g}(n) \equiv 0$ would increase search efficiency in those cases in which *any* path (not necessarily the least costly) to the goal is desired. In the next section we shall present some results counter to such intuition.

To summarize, there are three important factors influencing the heuristic power of an ordered-search algorithm:

1. The cost of the path
2. The number of nodes expanded in finding the path
3. The computational effort required to compute $\hat{h}$

The selection of a suitable $\hat{h}$ then permits one to choose for each problem a desirable compromise among these factors to maximize heuristic power.

3-11 THE IMPORTANCE OF $\hat{g}$

In many problems, we merely desire to find *some* path to a goal node and are unconcerned about the cost of the resulting path. (We are, of course, concerned about the amount of search effort required to find a path.) In such situations, intuitive arguments can be given both for including $\hat{g}$ in the evaluation function and for ignoring it.

Intuitive argument for ignoring $\hat{g}$

When merely any path to a goal is desired, $\hat{g}$ can be ignored since, at any stage during the search, we don't care about the costs of the paths developed thus far. We care only about the remaining search effort required to find a goal node, and this search effort, while possibly dependent on the $\hat{h}$ values of the open nodes, is certainly independent of the $\hat{g}$ values of the open nodes. Therefore, for such problems we should use as the evaluation function $\hat{f} \equiv \hat{h}$.