AI Magazine Volume 4 Number 1 (1983) (© AAAI)

On the Discovery and Generation of Certain Heuristics

Judea Pearl

Cognitive Systems Laboratory
School of Engineering and Applied Science
University of California, Los Angeles

Abstract

This paper explores the paradigm that heuristics are discovered by consulting simplified models of the problem domain After describing the features of typical heuristics on some popular problems, we demonstrate that these heuristics can be obtained by the process of deleting constraints from the original problem and solving the relaxed problem which ensues We then outline a scheme for generating such heuristics mechanically, which involves systematic refinement and deletion of constraints from the original problem specification until a semi-decomposable model is identified The solution to the latter consitutes a heuristic for the former

Introduction: Typical Uses of Heuristics

Heuristics are methods and criteria for judging the relative merits of alternative courses of planning or action. There is hardly any intellectual activity which does not rely on heuristics of some kind. The decision to begin reading this paper, for example, reflects a tacit use of heuristics which has lured the reader to invest time and effort in anticipation of certain benefits. Although such anticipations may occasionally be disappointed, on the whole they are essential to planning our everyday activities.

Complex combinatorial problems require the use of heuristics if a reasonably "good" solution is to be produced within practical time constraints. We shall demonstrate this point using three simple problems (readers familiar with the properties of A^* may skip to section Where do these heuristics come from?):

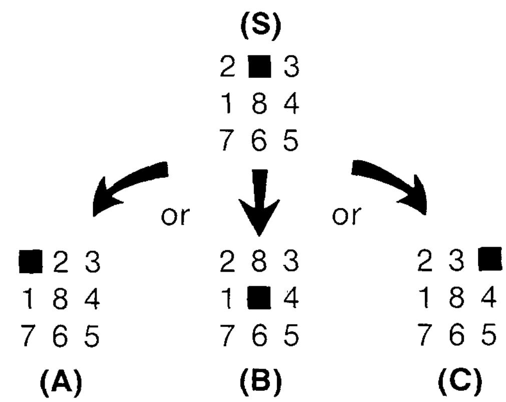


Figure 1 A goal tree in the 8-puzzle

The 8-puzzle. This simple puzzle is a one-person game, the objective of which is to rearrange a given configuration of eight uniquely numbered tiles on a 3×3 board into another given configuration by iteratively sliding one of the tiles into empty location, as in Figure 1 above:

Prepared for *UCLA-Computer Science Department Quarterly,* Spring 1982, this work was supported in part by the National Science Foundation Grant MCS 81 14209

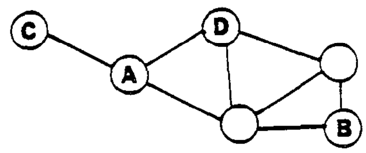


Figure 2

Assume that in the example above the objective is to reach the goal state:

1	2	3
8	■	4
7	6	5

Which of the three alternatives, (A), (B), or (C) appears most promising? The answer can, of course, be obtained by searching the graph associated with the puzzle and finding which of the three states leads to the shortest path to the goal The notorious combinatorial explosion, however, makes this method utterly impractical when the distance to the goal is large and/or when larger boards (e.g., 4×4) are involved. The very search for a solution path requires the use of judgments to decide at any point which search avenue is the most promising

To assist a computer in solving path-finding problems of this type, the programmer is usually required to provide a rule for computing an *estimate* of the proximity between two given configurations. The most popular rules for the 8-puzzle are: $h_1 =$ the number of tiles by which the two configurations differ, and $h_2 =$ the sum of the distances of the mismatched tiles from their proper destinations. (The black position is not counted.) The appropriate distance measure, in this case, is the sum of the coordinate differences, also known as the Manhattan or city-block distance. For instance, in the example above we can compute:

$$h_1(\mathrm{A}) = 2 \quad h_1(\mathrm{B}) = 3 \quad h_1(\mathrm{C}) = 4$$
$$h_2(\mathrm{A}) = 2 \quad h_2(\mathrm{B}) = 4 \quad h_2(\mathrm{C}) = 4$$

These heuristic functions are intuitively appealing and readily computable, and may be used to prune the space of possibilities in such a way that only configurations lying close to the solutin path will actually be explored. An algorithm which exhibits such pruning will be discussed later.

Finding the shortest path in a road map. Given a road map such as the one shown in Figure 2, it is desired to find the shortest path between city A and city B. If the intercity distances are presented in the form of a distance-matrix $d(i,j)$, there is no way for the search program to judge a priori that city C, unlike city D, lies way out of the natural direction from A to B and, consequently, that city D is better candidate from which to pursue the search. At the same time, the preference of D over C is obvious to anyone who glances at the map. What extra information does the map provide which is not made explicit in the distance table?

One possible answer is that the human observer exploits vision machinery to estimate the Euclidean distances in the map and, since the air distance from D to B is shorter than that between C and B, city D appears as a more promising candidate from which to launch the search. That same information can also be used by the machine if each city is assigned a heuristic function $h(*)$ equal to the air distance between that city and the goal B. A tentative choice between pursuing the search from city C or city D should depend, then, on the magnitude of the cost estimate $d(\mathrm{A,C})+h(\mathrm{C})$ relative to the estimate $d(\mathrm{A,D})+h(\mathrm{D})$.

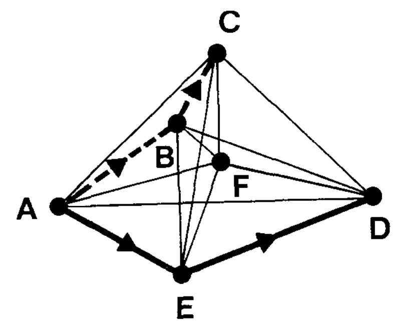


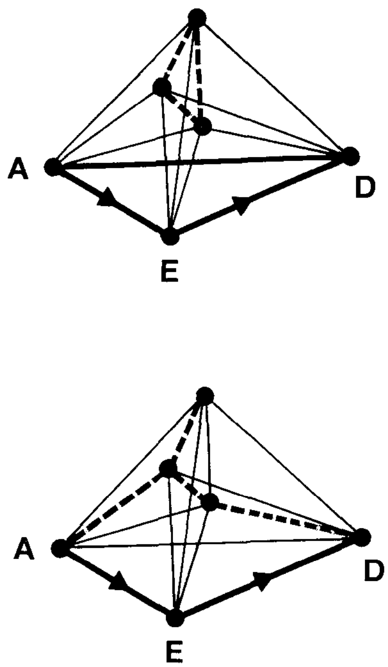
Figure 3

The traveling salesman problem (TSP)

Here we must find the cheapest tour, that is, the cheapest path which visits every node once and only once, and returns to the initial node, in a complete graph of N nodes with each edge assigned a non-negative cost.

It is well known that the TSP is NP-hard and that all known algorithms require an exponential time in the worst case. However, the use of good bounding functions often enables us (using the branch-and-bound algorithm) to find the optimal tour in much less time What is a bounding function? Consider the graph below where the two marked paths ABC and AED represent two partial tours currently being considered by the search procedure Which of the two, if properly completed to form a circuit, is more likely to be part of the optimal solution? Clearly, the overall solution cost is given by the cost of completing the tour added to the cost of the initial subtour, and so the answer lies in how cheaply we can complete the tour through the remaining nodes. However, since the computational effort required to find the optimal completion cost is almost as hard as that of finding the entire optrima tour, we must settle for an *estimate* of the completion cost. Given such estimates, the decision of which subtour to extend first would depend on which one, by combining the cost of the explored part with the estimate of its completion, offers a lower *overall* cost estmate. It can be shown that if at every stage of the search we select for exploration that partial tour with the lowest estimated cost, and if the estimates of the completion costs are consistently optimistic (underestimates), then the first tour to be completed by the search is also the optimal one.

What easily computable function would yield an optimistic, yet not too unrealistic, estimate of the subtour completion cost? People, when first asked to "invent" such a



Figures 4a and 4b

function, usually provide easily computable, but too simplistic answers For example, the cheapest edge or two-edge path connecting the end of the initial subtour, bypassing all unvisited cities or going through one other city, respectively. These functions, while being optimistic, grossly underestimate the completion cost. Upon deeper thought, more realistic estimates are formed, and the two which have received the greatest attention in the literature are: (1) The cheapest 2nd degree graph going through the remaining nodes (Lawler and Wood, 1966), and (2) the cost of the minimum spanning tree (MST) through all remaining nodes (Held and Karp, 1971). The first is obtained by solving the so-called optimal assignment problem using $O(N^3)$ steps, while the second requires $O(N^2)$ steps.

That these two functions provide optimistic estimates of the completion cost is apparent when we conside that completing the tour requires the choice of a path going through all unvisited cities A path is both a special case of a graph with degree 2 and also a special case of a spanning tree. Hence, the set of all completion paths is included in the sets of objects overwhich the optimization takes place, and so, the solution found must have a lower cost than that obtained by optimizing over the set of path only

Figure 4a below shows the shape of a graph that may be found by solving the assignment problem instead of completing subtour AED. The completion part is not a single path but a collection of loops. Figure 4b shows a minimum-spanning-tree completion of subtour AED In this case the object found is a tree containing a node with a degree higher than 2.

Some properties of heuristics

The three examples in the preceding section are typical of a problem-solving method called "state-approach" (Nilsson, 1971), where the search for a solution to a posed problem is formulated as the search for a path in a state-space graph. A path to a given node in such a graph represents a code for a subset of potential solutions: the arcs represent transformations of those codes, which correspond to finer partitions of the parent subsets. In the 8-puzzle the transformations corresponded to the actual legal moves of the game. In the road map and the TSP the transformations consisted of concatenating a partially explored path with one more edge. Tasks such as theorem proving, robot planning, and speech recognition can naturally be represented as path-finding problems using the state-space approach. Even constraint- satisfaction problems, such as the 8-queens problem, which at first glance bear no mention of graphs or paths, can be formulated in state-space if we regard the computations which allow us to scan the space of possible objects systematically as arcs in a graph.

An algorithm known as A^* (Hart et al., 1968) has become popular in the Artificial Intelligence literature as an efficient way of using heuristics to solve path-finding problems. Unlike conventional shortest-path algorithms, the state-space graph is not available explicitly but rather is generated incrementally during the search itself using the transforamtion rules. A^* uses heuristic information to search the state-space graph in a directed fashion, making explicit only that portion of the graph which is absolutely necessary for finding an optimal solution

We shall say that a node n is *expanded* when all possible transformation rules are applied to it and the resultant nodes, called successors of n, are generated. Any node which is expanded is called CLOSED, and any node generated but not yet expanded is called OPEN. At each step of the search, A^* selects for expansion that OPEN node which has the lowest cost estimate $f(n)$. The estimate $f(n)$ consists of two components:

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

where $g(n)$ is the minimal cost so far encountered from the root to n, and $h(n)$ is an estimate of the cost required to complete the path from n to a goal state. A^* halts when it attempts to expand a node which satisfies the goal condition.

The most significant theoretical result regarding the behavior of A^* is its *admissibiltiy* property: if for every node n, $h(n)$ does not exceed the actual optimal completion cost $h^*(n)$, then A^* terminates with the minimal cost path to a goal. An estimate $h(n)$ satisfying the inequality $h(n) \leq h^*(n)$ for every node in the graph is called an *admissibility heuristic*.

The second important property of A^* is called *consistency*. A heuristic function $h(\bullet)$ is said to be consistent if it satisfies the triangle inequality:

$$h(n) \geq c(n, n') + h(n') \leq (n) \leq n'$$

and:

$$h(n_g) = 0$$

where n' is any successor of n, $c(n, n')$ is the cost of the edge from n to n' and n_g is any node satisfying the goal conditions. The importance of consistency lies in guaranteeing that A^* will never reopen a CLOSED node. The reason is that A^*, when it selects a node for expansion, has already traced the optimal path to that node and so it need not test whether shorter paths may be found in the future. It can be shown that consistency implies admissibility but not vice versa.

The *power* of the heuristic estimate h is measured by the amount of pruning induced by h and depends, of course, on the accuracy of the estimate. If $h(\bullet)$ estimates the completion cost precisely, then A^* will only explore nodes lying along an optimal path. Otherwise A^* will expand any open node satisfying the inequality:

$$g(n) + h(n) < h^*(s)$$

where h^* is the cost of the optimal path from the intial node. Clearly, the higher the value of h the fewer nodes will be expanded by A^*, as long as h remains admissible. In the 8-puzzle example, for instance, since h_2 is generally larger and never lower than h_1, it is a more powerful heuristic and will give rise to a more efficient search.

Where do these heuristics come from?

We have seen a few examples of heuristic functions which were devised by clever individuals to assist in the solution of combinatorial problems. We now focus our attention on the mental process by which these heuristics are "discovered," with a view toward emulating the process mechanically.

The word "discovery" carries with it an aura of mystery, since it is normally attached to mental processes which leave no memory trace of their intermediate steps. It is an appropriate term for the process of generating heuristics, since tracing back the intermediate steps evoked in this process is usually a difficult task. For example, although we can argue convincingly that h_2, the sum of the distances in the 8-puzzle, is an optimistic estimate of the number of moves required to achieve the goal, it is hard to articulate the mechanism by which this function was discovered or to invent additonal heuristics of similar merit.

Articulating the rationale for one's conviction in certain properties of human-devised heuristics may, however, provide clues as to the nature of the discovery process itself. Examining again the h_2 heuristic for the 8-puzzle, note how surprisingly easy it is to convince people that h_2 is admissible. After all, the formal definition of admissibility contains a universal qualifier $(\leq n)$ which, at least in principle, requires that the inequality $h(n) \neq h^*(n)$ be verified for every node in the graph. Such exhaustive verification is, of course, not only impractical but also inconceivable. Evidently the mental process we employ in verifying such propositions is similar to that of symbolic proofs in mathematics, where the truth of unversally quantified statements (say, that there are infinitely many prime numbers) is established by a sequence of inference rules applied to other statements (axioms) without exhaustive enumaration.

An even more suprising aspect of our conviction in the truth of $h_2(n) \leq h^*(n)$ is the fact that $h^*(n)$, by its very nature, is an unknown quantity for almost every node in the graph; not knowing $h^*(n)$ was the very reason for seeking its estimate $h(n)$. How can we, then, become so absolutely convinced in the validity of the assertion $h_2(n) \leq h^*(n)$? Clearly, the verification of this assertion performed in a code where $h^*(n)$ does not possess an explicit representation.

In the case of the road map problem our conviction the admissibility of the air distance heuristic is explainable Here, based on our deeply entrenched knowledge of the properties of Euclidean spaces, we may argue that a straight line between any two points is shorter than any alternative connection between these points and, hence, that the air distance to the goal constitutes an admissible heuristic for the problem. In the 8-puzzle and the Traveling Salesman Problem, however, such a universal assertion cannot be drawn directly from our culture or experience, and must be defended, therefore, by more elaborate arguments based on more fundamental principles.

If we try to articulate the rationale for our confidence in the admissibility of h_2 for the 8-puzzle, we may encounter arguments such as the following:

1. Consider any solution to the goal, not necessarily an optimal one. In order to satisfy the goal conditions, each tile must trace some trajectory from its original location to its destination, and the overall cost (number of steps) of the solution is the sum of the costs of the individual trajectories. Every trajectory must consist of at least as many steps as that given by the Manhattan-distance between the tile's origin and its destination, and, hence, the sum of the distances cannot exceed the overall cost of the solutions.
2. If I were able to move each tile independently of the others, I would pick up tile #1 and move it, in steps, along the shortest path to its destination, do the same with tile #2, and so on until all tiles reach their goal locations. On the whole, I will have to spend at least as many steps as that given by the sum of the distances. The fact that in the actual game tiles tend to interfere with each other can only make things worse. Hence, . .

The first argument is analytical It selects one property which must be satisfied by every solution, such as in providing a homeward-trajectory for every tile, and asks for the minimum cost required for maintaining just that property The second argument is operational. It describes a procedure for solving a similar, auxiliary problem where the rules of the game have been relaxed . Instead of the conventional 8-puzzle whose tiles are kept confined in a 2-dimensional plane, we now imagine a relaxed puzzle whose tiles are permitted to climb on top of each other. Instead of seeking heuristic *function* $h(n)$ to approximate $h^*(n)$, we can actually compute the solution using the relaxed version of the puzzle, count the number of steps required, and use this count as an estimate of $h^*(n)$.

The last scheme leads to the general paradigm expounded in this paper: *Heuristics are discovered by consulting simplified models of the problem domain* We shall later explicate what is meant by a *simplified* model and how to go about finding such a one. The preceding example, however, specifically demonstrates the use of one important class of simplified models: that generated by *removing constraints* which forbid or penalize certain moves in the original problem We shall call models obtained by such constraint-deletion processes *relaxed models*

Let us examine first how the constraint-deletion scheme may work in the Traveling Salesman Problem. We are required to find as estimate for the cheapest completion path which starts at city D, ends at City A, and goes through every city in the set S of the unvisited cities. A path is a connected graph of degree 2, except for the end-points, which are of degree 1, a definition which we can express as a conjunction of three conditions: (1) being a graph, (2) being connected, (3) being a of degree 2. If we delete the requirement that the completion graph be connected, we get the assignment heuristic of Figure 4a Similarly, if we delete the constraint that the graph be of degree 2, we get the minimum-spanning-tree (MST heuristic depiced in Figure 4b.) An even richer set of heuristics evolves by relaxing the condition that the task completed by a graph.

An alternative way of leading toward the MST heuristic is to imagine a salesman employed under the following cost arrangement: he has to pay from his own pocket for any trip in which he visits a city for the first time, but can get a free ride back to any city which he visited before. It is not hard to see that under such relxed cost conditions the salesman would benefit from visiting some cities more than once and that the optimal tour strategy is to pay only for those trips which are part of the minimum-spanning-tree. If instead of this cost arrangement the salesman gets one free ride *from* any city visited twice, the optimal tour may be made up of smaller loops, and the assignment problem ensues. Thus, by adding free rides to the original cost structure we create relaxed problems whose solutions can be taken as heuristics for the original problem.

It is interesting to note that heuristics generated by optimizations over relaxed models are guaranteed to be *consistent* This is easily verified by inspecting Figure 5 below, where $h(n)$ and $h(n')$ are the heuristics assigned to nodes n and n', respectively. These heuristics stand for the minimum cost of completing the solution from the corresponding nodes in some relaxed model common to both nodes. $h(n)$, representing an optimal solution (geodesic), must satisfy $h(n) \leq c'(n,n') + h(n')$, where $c'(n,n')$ is the relaxed cost of the edge (n,n'), or else $c'(n,n') + h(n')$, instead of $h(n)$, would constitute the optimal cost from n. The relaxed edge-

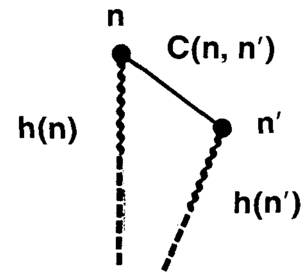


Figure 5

cost $c'(n,n')$ cannot, by definition, exceed the original cost $c(n,n')$, thus:

$$h(n) \leq c(n,n') + h(n')$$

which, together with the technical condition $h(n_g) = 0$, completes the requirements for consistency.

This feature has both computational and psychological implications. Computationally, it guarantees that a search algorithm, A^*, guided by any heuristic evolving from a relaxed model, would be spared the effort of reopening CLOSED nodes or even testing whether a newly generated node has been expanded before. The pointers assigned to any node expanded by such algorithms are already directed along the optimal path to that node.

Psychologically, if we assume that people discover heuristics by consulting relaxed models, we can explain now why most man-made heuristics are both admissible and consistent. The reader may wish to test this point by attempting to generate heuristics for the 8-puzzle or the TSP that are admissible but not consistent. The difficulties encountered in such attempts should strengthen the reader's conviction that the most natural process for generating heuristics is by relaxation, where consistency surfaces as an automatic bonus.

Before proceeding to outline how systematic relaxations can be used to generate heuristics mechanically, it is important to note that not every relaxed model is automatically simpler than the original. Assume, for example, that in the 8-puzzle, in addition to the conventional moves, we also allow a checker-like jumping of tiles across the main diagonals. The puzzle thus created is obviously more relaxed than the original, but it is not at all clear that the complexity of searching for an optimal solution in the relaxed puzzle is lower than that associated with the original problem. True, adding shortcuts makes the search graph of the relaxed model somewhat shallower, yet it also becomes bushier: A_* must now examine the extra moves available at every decision junction, even if they lead nowhere.

Moreover, relaxation is not the only scheme which may simplify problems. In fact, simplified models can easily be obtained by a process opposite to that of relaxation, such as loading the original model with additional constraints Of course, the solutions to over-constrained models would no longer be admissible. However, in problems where one settles for finding any path to the goal, not necessarily the optimal, simplified over-constrained models may be very helpful The most popular way of constraining models is to assume that a certain portion of the solution is given a-priori This assumption cuts down on the number of remaining variables which of course, results in a speedy search for the completion portion For example, if one arbitrarily selects an initial subtour through M cities in the TSP, an overconstrained problem ensues which is simpler than the original because it involves only $N - M$ cities The cost associated with the solution to such a problem constitutes an upper bound to h^* and can be used to cut down the storage requirement of A^*. Every node in OPEN whose admissible evaluation $f(n) = g(n) + h(n)$ exceeds that upper bound can be permanently removed from memory without endangering the optimality of the resulting solution.

A third important class of simplified models can be obtained by probabilistic considerations In certain cases we may possess sufficient knowledge about the problem domain to permit an estimate of the *most probable* cost of the completion path Consider, for example, the problem of finding the cheapest cost path in a tree where all arc costs are known to be drawn independently from a common distribution function, with mean μ. If N stands for the number of arcs remaining between a node n and the goal, then for large N the cost of any path to the goal is knownref be highly peaked about μN Therefore, if we are sure that only one path leads from n to the goal, we can take the value μN as an estimate of h^* and use $f = g + \mu N$ as a node-rating function in A^* If several paths lead from n to the goal set and if the structure of these paths is fairly regular, probability calculus can be invoked to estimate the most likely cost of the cheapest one among them, and that estimate can be used as h in A^*. Alternatively, probabilistic models often predict that reasonable solution paths are likely to exhibit certain distinctive properties in their behavior, e.g , that the cost along the path will increase gradually at a predetermined rate Hence, an irrevocable search strategy can be employed which prunes away any path found to behave at variance with such expectations (Karp and Pearl, 1982).

By their very nature, probability-based models cannot guarantee the optimality of the solution. Although, in most cases, they produce accurate estimates of the completion cost, occasionally these turn out to be grossly overestimated, which may cause A^* to terminate prematurely with a suboptimal solution On the other hand, if one does not insist on finding an exact optimal solution all the time but settles instead for finding a "good" solution most of the time, probability-based heuristics can, in many cases, reduce the complexity of combinatorial problems from exponential to polynomial.

Another class of simplified models used in heuristic reasoning is analogical or metaphorical models. Here the auxilary model draws its power not from a structural simplicity inherent in the *problem,* but rather from matching the machinery and expertise accumulated by the *problem solver* For example, the game of tic-tac-toe appears simpler to us than its isomorphic number-scrabble game (Newell and Simon, 1972) because the former evokes an expertise gathered by our visual machinery which has not yet been acquired by our arithmetic reasoner. Likewise, the use of visual imagery in solving complex mathematical or programming problems takes advantage of the special purpose machinery which evolution has bestowed upon us for processing visual information and manipulating physical objects The use of analogical models by computers would only be beneficial when we learn how to build an efficient data-driven expert system for at least one problem domain of sufficient richness, e g , physical objects.

Mechanical Generation of Admissible Heuristics

We return now to the relaxation scheme and to show how the deletion of constraints can be systematic to the point that natural heuristics, such as those demonstrated in the Introduction Section, can be generated by mechanical means. The constraint-relaxation scheme is particularly suitable for this purpose because many problem domains are conveniently formalizable by the explicit representation of the constraints which govern the applicability and impact of the various transformations in that domain. Take, for instance, the 8-puzzle problem. It is utterly impractical to specify the set of legal moves by an exhaustive list of pairs, describing the states before and after the application of each move A much more natural representation of the puzzle would specify the available moves by two sets of conditions, one which must hold true before a given move is applicable and one which must prevail after the move is applied. In the robot-planning program STRIPS (Fikes and Nilsson, 1971), for example, actions are represented by three lists: (1) a *precondition-list,* a conjunction of predicates which must hold true before the action can be applied; (2) an *add-list,* a list of predicates which are to be added to the description of the world-state as a result of applying the action; and (3) a *delete-list,* a list of predicates that are no longer true once the action is applied and should, therefore, be deleted from the state description. We shall use this representation in formalizing the relaxation scheme for the 8-puzzle.

We start with a set of three primitive predicates:

ON(x, y)	:	tile x is on cell y
CLEAR(y)	:	cell y is clear of tiles
ADJ(y, z)	:	cell y is adjacent to cell z,

where the variable x is understood to stand for the tiles $x_l, X_2, \quad, X_8$ and where the variables y and z range over the set of cells $C_1, C_2, \quad ., C_9$. Although the predicate CLEAR can be defined in terms of ON:

$$\text{CLEAR}(y) \Leftrightarrow \forall(x) \sim \text{ON}(x, y),$$

it is convenient to carry CLEAR explicitly as if it were an independent predicate Using these primitives, each state will be described by a list of 9 predicates, such as:

ON(X_1, C_1), ON(X_2, C_2), ..., ON$(X_8 C_8)$, CLEAR (C_9), together with the board configuration:

$$\text{ADJ}(C_1, C_2),\ \text{ADJ}(C_1, C_4),\ \ldots,$$

The move corresponding to transferring tile x from location y to location z will be described by the three lists:

MOVE (x, y, z) :

precondition list	:	ON(x, y), CLEAR(z), ADJ(y, z)
add list	:	ON(x, y), CLEAR(y)
delete list	:	ON(x, y), CLEAR(z)

The problem is defined as finding a sequence of applicable instantiations for the basic operator MOVE(x, y, z) that will transform the initial state into a state satisfying the goal criteria.

Let us now examine the effect of relaxing the problem by deleting the two conditions, CLEAR(z) and ADJ(y, z), from the precondition list. The resultant puzzle permits each tile to be taken from its current position and be placed on any desired cell with one move. The problem can be readily solved using a straightforward control scheme: at any state find any tile which is not located on the required cell. Let this tile be X_1, its current location Y_1, and its required location Z_1 Apply the operator MOVE(X_1, Y_1, Z_1), and repeat the procedure on the prevailing state until all tiles are properly located.

Clearly, the number of moves required to solve any such problem is exactly the number of tiles which are misplaced in the initial state. If one submits this relaxed problem to a mechanical problem-solver and counts the number of moves required, the heuristic $h_1(\bullet)$ ensues.

Imagine now that instead of deleting the two conditions, only CLEAR(z) is deleted while ADJ(y, z) remains. The resultant model permits each tile to be moved into an adjacent location regardless of its being occupied by another tile This obviously leads to the $h_2(\bullet)$ heuristic: the sum of the Manhattan-distances.

The next deletion follows naturally; let us retain the condition CLEAR(z) and delete ADJ(y, z). The resultant model permits transferring any tile to the empty spot, even when the two cells are not adjacent. The problem of reconfiguring the initial state with such operators is equivalent to that of sorting a list of elements by swapping the locations of two elements at a time, where every swap must exchange one marked element (the blank) with some other element. The optimal solution to this swap-sort problem can be obtained using the following "greedy" algorithm:

> If the current empty cell y is to be covered by tile x, move x into y Otherwise (if y is to remain empty in the goal state), move into y any arbitrary misplaced tile Repeat.

The resulting cost of this model, h_3, is mentioned neither in the Introductory Section 1 nor in textbooks on heuristic search. It is not the kind of heuristic that is likely to be discovered by the novice, and it was first introduced by Gaschnig (1979) eleven years after A^* was exemplified using h_1 and h_2. Although h_3 turns out to be only slightly better than h_1, its late discovery, coupled with the fact that it evolves so naturally from the constraint-deletion scheme, illustrates that the method of systematic deletions is capable of generating non-trivial heuristics.

The reader may presume that the space of deletions is now exhausted; deleting ON(x, y) leads again to h_1, while retaining all three conditions brings us back to the original problem. Fortunately, the space of deletions can be further refined by enriching the set of elementary predicates. There is no reason, for instance, why the relation ADJ(y, z) need be taken as elementary—one may wish to express this relation as a conjunction of two other relations:

$$\text{ADJ}(y, z) \Leftrightarrow \text{NEIGHBOR}(y, z) \bigwedge \text{SAME-LINE}(y, z)$$

Deleting any one of these new relations will result in a new model, closer to the original than that created by deleting the entire ADJ predicate.

Thus, if we equip our program with a large set of predicates or with facilities to generate additional predicates, the space of deletions can be refined progressively, each refinement creating new problems closer and closer to the original. The resulting problems, however, may not lend themselves to easy solutions and may turn out to be even harder than the original problem. Therefore, the search for a model in the space of deletions cannot proceed blindly but must be directed toward finding a model which is both easy to solve and not too far from the original.

This begs the following question: Can the program tell an easy problem from a hard one without actually trying to solve them? This will be discussed the the next section.

Can a Program Tell an Easy Problem When It Sees One?

We now come to the key issue in our heuristic-generation scheme. We have seen how problem models can be relaxed with various degrees of refinements. We have also seen that relaxation without simplification is a futile excursion. Ideally, then, we would like to have a program that evaluates the degree of simplification provided by any candidate relaxation and uses this evaluation to direct the search in model-space toward a model which is both simple and close to the original. This, however, may be asking for too much. The most we may be able to obtain is a program that recognizes a simple model when such a one happens to be generated by some relaxation. Of course, we do not expect to be able to prove mechanically propositions such as "This class of problems cannot be solved in polynomial-time " Instead, we should be able to recognize a subclass of easy problems, those possessing salient feature advertising their simplicity.

Most of the examples discussed above possess such features. They can be solved by "greedy," hill-climbing methods without backtracking, and the feature that makes them amenable to such methods is their *decomposability* Take, for instance, the most relaxed model for the 8-puzzle, where each tile can be lifted and placed on any cell with no restrictions. We know that this problem is simple, without actually solving it, by virtue of the fact that all the goal conditions, ON(X_1, C_1), ON(X_2, C_2), ..., can be satisfied *independently* of each other. Each element in this list of subgoals can be satisfied by one operator without undoing the effect of previous operators and without affecting the applicability of future operators.

Similar conditions prevail in the model corresponding to h_2, except where a sequence of operators is required to satisfy each of the goal conditions. The sequences, however, are again independent in both applicability and effects, a fact which is discernible mechanically from the formal specification of the model, since the subgoals themselves define the desired partition of the operators. Any operator whose add-list contains the predicate ON(X_i, y) will be directed only toward satisfying the subgoal ON(X_i, C_i) and can be proven to be non-interfering with any operator containing the predicate ON(X_j, y) in its add-list.

Most automatic problem-solvers are driven by mechanisms which attempt to break down a given problem into its constituent subproblems as dictated by the goal description. For example, the General-Problem-Solver (Ernst and Newell, 1969) is controlled by "differences," a set of features which make the goal different from the current state. The programmer has to specify, though, along what dimensions these differences are measured, which difference are easier to remove, what operators have the potential of reducing each of the differences, and under what conditions each reduction operator is applicable. In STRIPS, most of these decisions are made mechanically on the basis of the 3-list description of operators. Actions are brought up for consideration by virtue of their add-list containing predicates which can bridge the gap between the desired goal and the current state. If the current state does not possess the conditions necessary for enacting a useful difference-reducing transformation, a new subgoal must be created to satisfy the missing conditions Thus the complexity of this "end-means" strategy increases sharply when subgoals begin to interact with each other.

The simplicity of decomposable problems, on the other hand, stems from the fact that each of the subgoals can be satisfied independently of each other; thus the overall goal can be achieved in a time equal to the number of conjuncts in the goal description multiplied by the time required for satisfying a single conjunct in isolation It is a version of the celebrated 'divide-and-conquer' principle, where the division is dictated by the primitive conjuncts defining the goal conditions. For example, in an NxN-puzzle we have N^2 conjuncts defining the goal state configuration, and the solution of each subproblem, namely finding the shortest path for one tile using a relaxed model with single-cell moves, can be obtained in $O(N^2)$ steps even by an uninformed, breadthfirst, algorithm. Thus the optimal solution for the overall relaxed NxN-puzzle can be obtained in $O(N^4)$ steps, which is substantially better than the exponential complexity normally encountered in the non-relaxed version of the NxN-puzzle.

The relaxed NxN-puzzle is an example of a complete independence between the subgoals, where an operator leading toward a given subgoal is neither hindered nor assisted by any operator leading toward another subgoal. Such complete independence is a rare case in practice and can only be achieved after deleting a large fraction of the applicability constraints. It turns out, however, that simplicity can also be achieved in much weaker forms of independence which we shall call *semi-decomposable* models.

Take, for example, the minimum-spanning-tree problem. If the goal is defined as a conjunction of N-1 conditions:

CONNECTED (city i, city 1) $i = 2, 3, \ldots, N$

and each elementary operator consists of adding an edge between a connected and an unconnected city, we have a semi-decomposable structure. Even though no operator may undo the labor of previous operators or hinder the applicability of future operators, some degree of coupling remains since each operator *enables* a different set of applicable operators This form of coupling, which was not present in the relaxed 8-puzzle, may make the cost of the solution depend on the order in which the operators are applied. Fortunately, the MST problem possesses another feature, *commutativity,* which renders the greedy algorithm "cheapest-subgoal-first" optimal. *Commutativity* implies that the internal order at which a given set of operators is applied does not alter the set of operators applicable in the future. This property, too, should be discernible from the formal specification of the domain model and, once verified, would identify the "greedy" strategy which yields an optimal solution.

Another type of semi-decomposable problem is exemplified by the swap-sort model of the 8-puzzle where the subgoals interact with respect to both applicability and effects. Moving a given tile into the empty cell clearly disqualifies the applicability of all operators which move other tiles into that particular cell. Additionally, if at a certain stage the predicate specifying the correct position of the empty cell is already satisfied, it would be impossible to satisfy additional subgoals without first falsifying this predicate, hopefully on a temporary basis only

In spite of these couplings, the feature which renders this puzzle simple, admitting a greedy algorithm, is the existence of a partial order on the subgoals and their associated operators such that the operators designated for any subgoal g may influence only subgoals of lower order than g, leaving all other subgoals unaffected. In our simple example, establishing the correct position of the blank is a subgoal of a higher order than all the other subgoals and should, therefore, be attempted last. To find such a partial order of subgoals from the problem specification is similar to finding a triangular connection matrix in GPS; programs for computing this task have been reported in the literature (Ernst and Goldstein, 1982).

Conclusions

This paper outlined a natural scheme for devising heuristics for combinatorial problems. First, the problem domain is formulated in terms of the 3-list operators which transform the states in the domain and the conditions which define the goal states. Next, the preconditions which limit the applicability of the operators are refined and partially deleted, and a relaxed model submitted to an evaluator which determines whether it is semi-decomposable. The space of all such deletions constitutes a meta-search-space of relaxed models from which the most restrictive semi-decomposable element is to be selected The search starts either at the original model from which precondition conjuncts are deleted one at a time, or at the trivial model containing no preconditions to which restrictions are added sequentially until decomposability is destroyed. The model selected, together with the "greedy" strategy which exploits its decomposability, constitutes the heuristic for the original domain

Although the effort invested in searching for an appropriate relaxed model may seem heavy, the payoffs expected are rather rewarding. Once a simplified model is found, it can be used to generate heuristics for *all* instances of the original problem domain For example, if our scheme is applied successfully to the TSP model, it may discover a $O(N^2)$ heuristic superior to (more constrained than) the MST Such a heuristic will be applicable to every instance of the TSP problem and, when incorporated into A^*, will yield optimal TSP solutions in shorter times than the MST heuristic We currently do not know if such heuristics exist Equally challenging are routing problems encountered in communication networks and VLSI designs which, unlike the TSP, have not been the focus of a long theoretical research, but where the problem of devising effective heuristics remains, nonetheless, a practical necessity.

Future progress in this area hinges on developing techniques for recognizing simple, decomposable problems when such are present, and on manipulating the space of deletions systematically in order that such problem be, in fact, synthesized

Bibliographical and historical remarks

The ideas expressed in the preceding sections have been developed independently by serveral people, including: J Gaschnig, M Somalvico, M. Valtorta, D. Kibler and myself

The notion of viewing heuristics as information provided by simplified models was first communicated to me by Stan Rosenschein in 1979. Aside from its popular use in operations research (Lawler and Wood, 1966), (Held and Karp, 1972), the auxiliary problem approach was formally introduced to AI-type problems by Gaschnig (1979), Guida and Somalvico (1979) and Banerji (1980). Gaschnig described the spaces of auxiliary problems as "subgraphs" and "supergraphs" obtained by deleting or adding edges to the originial problem graph. Guida and Somalvico use propositional representation of constraints similar to that of the "Mechanical Generation of Admissible Heuristics" section and propose the use of relaxed models for generating admissible heuristics. A slightly different formulatio is also given in Kibler (1982).

Sacerdoti's planning system ABSTRIPS (Sacerdoti, 1974) also uses constraints relaxation for creating simplified problem spaces. ABSTRIPS first synthesizes a global abstract plan, then searches for a detailed mode of its implementation. The abstract planning phase differs from the fully detailed one in that the operators invoked bby the former lack some of the preconditions spelled out in the latter Although the program does not make explicit use of a numerical evaluation function, the search scedule is determined by progress achieved in the abstract planning phase and so, in effect, it can be thought of as being guided by the advice of a relaxed model

Valtorta (1981) presents a proof of the conistency of relaxation-based heuristics, and an analysis of the overall complexity of searching both the original and the auxiliary problems He shows that if the auxuliary probelms are solbed by the blind search, then the overall complexity will be worse than simply executing a breadth-first search on the original problem. This result emphasizes the importance of searching for decomposable structures where optimal solutions can be found by "greedy" algorithms without resorting to breadth-first search. The use of systematic deletions in search for decomposable problems was proposed by Pearl (1982) and is currently pursued at UCLA Other approaches to automatic generation of heuristics are reported by Banerji (1980) and Ernst and Goldstein (1982)

References

Banerji, R B 1980. *Artificial Intelligence A Theoretical Approach,* Amsterdam:North Holland

Ernst, G W and Goldstein, M M. 1982 Mechanical Discovery of Classes of Problem-Solving Strategies *JACM,* 29(1):1 23.

Ernst, G. W. and Newell, A. 1969 *GPS: A Case Study in Generality and Problem Solving,* New York: Academic Press

Fikes, R E and Nilsson, N J. 1971. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving." *Artificial Intelligence,* 2:184-208.

Gashnig, J 1979. A problem similarity approach to devising heuristics: First results *Proc IJCAI 6,* 301-307

Guida, G and Somalvico, M 1979. A method fo computing heuristics in problem solving *Information Sciences,* 19:251-259

Hart, P , Nilsson, N , and Raphael, B. 1968 A formal basis for heursitc determination of minimum cost paths *IEEE Trans System Sci Cybernetics,* 4:100–107

Held, M and Karp, R 1971. The traveling salesman problem and minimum spanning trees *Operation Research,* Vol 19

Karp, R M and Pearl, J. 1983 Searching for the cheapest path in a tree with random costs UCLA-ENG-CSL-8275 (To appear in *Artificial Intelligence*

Kibler, D 1982 Natural generation of admissible heuristics University of California at Irvine, Dept of Computer Science

Lawler, E L and Wood, D. E. 1966. Branch-and-bound methods: A survey *Operations Research,* 14:699–719

Newell, A and Simon, H 1972 *Human problem solving* New York:Prentice-Hall

Nilsson, N. J 1971 *Problem solving methods in artificial intelligence* New York:McGraw-Hill

Pearl, J 1982. On the discovery and generation of certain heuristics. *UCLA Computer Science Quarterly,* Vol 2

Sacerdoti, E. D. 1974 Planning in a hierarchy of abstraction spaces *Artificial Intelligence,* 5:115–135