# Chapter 4

# Heuristics Viewed as Information Provided by Simplified Models

In Chapter 1 we saw a few examples of heuristic functions that 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.

## 4.1 THE USE OF RELAXED MODELS

### 4.1.1 Where Do These Heuristics Come From?

The word *discovery* carries with it an aura of mystery, because it is normally attached to mental processes that 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 additional 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 quantifier ($\forall n$) that, at least in principle, requires the inequality $h(n) \leqslant h^*(n)$ to be verified for *every* node in the graph. Such exhaustive verification is, of course, not only im-

practical 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 universally quantified statements (say, that there are infinitely many prime numbers) is established by a sequence of inference rules applied to a finite number of axioms without exhaustive enumeration.

An even more surprising aspect of our conviction in the truth of $h_2(n) \leqslant h^*(n)$ 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 of the validity of the assertion $h_2(n) \leqslant h^*(n)$? Clearly, the verification of this assertion is performed in a code where $h^*(n)$ does not possess an explicit representation.

In the case of the Road Map problem our conviction in 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. 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 solution.
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 that 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 a mathematical *function* $h(n)$ to approximate $h^*(n)$, we can actually complete the solution using the relaxed version of the puzzle, *count* the number of steps required, and use this count as an estimate of $h^*(n)$.



This last scheme leads to the general paradigm expounded in this chapter: **Heuristics are discovered by consulting simplified models of the problem domain**. Later we will 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 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, such as that of Figure 1.5. We are required to find an estimate for the cheapest completion path that 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 that we can express as a conjunction of three conditions:

1. Being a graph.
2. Being connected.
3. Being of degree 2.

If we delete the requirement that the completion graph be connected, we get the optimal assignment heuristic of Figure 1.6($a$). Similarly, if we delete the constraint that the graph be of degree 2, we get the minimum-spanning-tree (MST) heuristic depicted in Figure 1.6($b$). An even richer set of heuristics evolves by relaxing the condition that the task be completed by a graph (Pohl, 1973 and 1977).

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 that he visited before. It is not hard to see that under such relaxed 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 that 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 admissible heuristics for the original problem.

### 4.1.2 Consistency of Relaxation-Based Heuristics

It is interesting to note that heuristics generated by optimizations over relaxed models are guaranteed to be *consistent* (or monotone). This is easily verified by

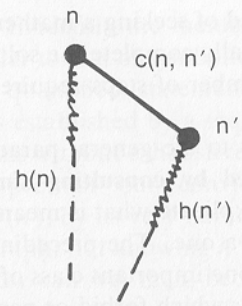


**Figure 4.1**

The consistency of relaxation-based heuristics, that is, $h(n) \leqslant c(n, n') + h(n')$.

inspecting Figure 4.1, 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 $h(n) \leqslant c'(n, n') + h(n')$, where $c'(n, n')$ is the relaxed cost of the edge $(n, n')$. Otherwise $c'(n, n') + h(n')$, instead of $h(n)$, would constitute the optimal cost from $n$.) The relaxed edge-cost $c'(n, n')$ cannot, by definition, exceed the original cost $c(n, n')$. Thus:

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

which, together with the technical condition $h(\text{goal}) = 0$, completes the requirements for consistency (see Section 3.1.5).

This feature has both computational and psychological implications. Computationally it guarantees that $A^*$, guided by any heuristic evolving from a relaxed model, would be spared the effort of reopening CLOSED nodes and redirecting their pointers. 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.

### 4.1.3 Overconstrained, Analogical, and Other Types of Auxiliary Models

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 search strategy must now examine the extra moves available at every decision junction, even if they lead nowhere.

Moreover, relaxation is not the only scheme that may simplify problems. In fact, simplified models can easily be obtained by a process opposite that of relaxation, such as loading the original model with **additional constraints.** Of course, the solutions to overconstrained models would no longer be admissible. However, in satisficing problems where one settles for finding any path to a goal, not necessarily the optimal, simplified overconstrained models may be very helpful. The most straightforward 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, and we may use this cost as a heuristic to guide the search. Alternatively, probability-based models can be used to determine when irrevocable pruning of nodes is "safe." The use of such models is discussed in Section 4.3.

Another class of auxiliary models used in heuristic reasoning is **analogical** or **metaphorical** models. Here the auxiliary model draws its power not from the simplicity of the *problem structure,* but rather from matching the machinery or expertise available to the *problem solver.* For example, the popular game of tic-tac-toe appears simpler to us than its isomorphic number-scrabble game (where the aim is to select three numbers that sum to fifteen, Newell and Simon, 1972), even though the two games are merely different representations of the same underlying graph. The former appears *simpler* because it evokes an expertise available to our visual machinery (e.g., the ability to swiftly detect rows, columns, and diagonals) that has not yet been acquired by our arithmetic reasoner (e.g., to quickly identify sets of number-triplets which sum to fifteen).

Human expertise, because it consumes a fair amount of computational resources, remains narrowly bound to one notational system, usually that in which the expertise was initially acquired. Tic-tac-toe experts were found to

play a rather mediocre game of number scrabble (if forbidden the use of pencil and paper) even when shown the equivalence of the two games. The only players who benefited from their tic-tac-toe expertise were those gifted with the power of visual imagery who were able to visualize the array of dots and crosses which corresponded to each numerical configuration encountered in the number-scrabble game. Evidently, the game-playing expertise was not easily transferable from the notation of square arrays to that of numerical symbols, causing the players to miss threats and opportunites that would otherwise be evoked by imagining the equivalent $3 \times 3$ array configurations.

This example demonstrates the essence of the computational benefits drawn from analogical reasoning, namely, that it is often easier to transform the notational system to fit the evoking mechanism of a preexisting expertise than to organize a duplicate body of expertise around a new notational system. The prevailing use of visual imagery in solving complex problems in mathematics, physics, economics, and programming further attests to the power and generality of this computational principle, even in cases where the analogical transformations are only partially valid. Imagery capitalizes both on the vast amount of common-sense knowledge we have acquired regarding the physical world surrounding us and on the special-purpose machinery evolution has bestowed upon us for processing motor-visual information. Clearly, the use of analogical models by computers would likewise be beneficial only when we learn to build efficient data-driven expert systems for at least one problem domain of sufficient richness, e.g., the manipulation of physical objects. (See Kling, 1971; Funt, 1977; Pearl, 1977b; Lenat, 1977; and Carbonell, 1983.)

## 4.2 MECHANICAL GENERATION OF ADMISSIBLE HEURISTICS

### 4.2.1 Systematic Relaxation

We return now to the relaxation scheme and to show how the deletion of constraints can be systematized to the point that natural heuristics, such as those demonstrated in Chapter 1, can be generated by mechanical means. The constraint-relaxation scheme is particularly suitable for this purpose because it is often convenient to represent problem domains by specifying the constraints that 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 more natural representation of the puzzle would specify the available moves by two sets of conditions, one that must hold true before a given move is applicable and one that 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 that must hold true before the action can be applied
2. An *add list*—a list of predicates that are to be added to the description of the world-state as a result of applying the action
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 will use this representation in formalizing the relaxation scheme for the 8-Puzzle.

We start with a set of three primitive predicates:

$$\begin{array}{lcl} \text{ON}(x, y) & : & \text{tile } x \text{ is on cell } y \\ \text{CLEAR}(y) & : & \text{cell } y \text{ is clear of tiles} \\ \text{ADJ}(y, z) & : & \text{cell } y \text{ is adjacent to cell } z, \end{array}$$

where the variable $x$ is understood to stand for the tiles $X_1, X_2, \ldots, X_8$ and where the variables $y$ and $z$ range over the set of cells $C_1, C_2, \ldots, C_9$. Although the predicate CLEAR can be defined in terms of ON:

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

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

$$\text{ON}(X_1, C_1), \text{ON}(X_2, C_2), \ldots, \text{ON}(X_8, C_8), \text{CLEAR}(C_9)$$

together with the board description:

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

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

MOVE$(x, y, z)$ :

$$\begin{array}{lcl} \text{precondition list} & : & \text{ON}(x, y), \text{CLEAR}(z), \text{ADJ}(y, z) \\ \text{add list} & : & \text{ON}(x, z), \text{CLEAR}(y) \\ \text{delete list} & : & \text{ON}(x, y), \text{CLEAR}(z) \end{array}$$

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 that 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 that 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 $h_1(\cdot)$ ensues.

Imagine now that instead of deleting the two conditions, only CLEAR $(z)$ is deleted, whereas 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 $h_2(\cdot)$ 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 solving problems in this model, $h_3$, is mentioned neither in Chapter 1 nor in text books on heuristic search (e.g., Nilsson, 1971). It is not the kind of heuristic that is likely to be discovered by the novice, and it was first introduced by Gaschnig (1979b) 13 years after $A^*$ was tested using $h_1$ and $h_2$ (Doran and Michie, 1966). 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 nontrivial heuristics.

The reader may presume that the space of deletions is now exhausted; deleting ON $(x, y)$ leads again to $h_1$, whereas 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—we may wish to express this relation as a conjunction of two other relations:

$$\text{ADJ}(y, z) \Longleftrightarrow \text{NEIGHBOR}(y, z) \wedge \text{SAME-LINE}(y, z)$$



Deleting any one of these new relations results 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 that is both easy to solve and not too far from the original.

This begs the following question: Can a program tell an easy problem from a hard one without actually trying to solve them? We discuss this question in the next section.

### 4.2.2 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, 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 that 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 features advertising their simplicity.

Most of the preceding examples 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, because 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 $h_2$, with the exception that a *sequence* of operators is now required to satisfy each of the goal conditions. The sequences, however, are again independent in both applicability and effects, a fact that 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 noninterfering with any operator containing the predicate ON($X_j$, $y$) in its add list.

Most automatic problem-solvers are driven by mechanisms that 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 that make the goal different from the current state. The programmer has to specify, though, along what dimensions these differences are measured, which differences 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 three-list description of operators. Actions are brought up for consideration because their add list contains predicates that 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 is 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 $N \times N$ 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, breadth-first, algorithm. Thus the optimal solution for the overall relaxed $N \times N$ Puzzle can be obtained in $O(N^4)$ steps, which is substantially better than the exponential complexity normally encountered in the nonrelaxed version of the $N \times N$ Puzzle.

The relaxed $N \times N$ 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 call **semi-decomposable** models.

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

$$\text{CONNECTED (city } i\text{, city 1)} \qquad i = 2, 3, \ldots, N$$



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 (Nilsson, 1980). 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 that 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 that 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 higher 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. A similar hierarchy of subgoals was also encountered in the Tower of Hanoi problem (Section 1.2.5). 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).

### 4.2.3 Summary

This section outlined a natural scheme for devising heuristics for combinatorial problems. First, the problem domain is formulated in terms of the three-list operators that transform the states in the domain and the conditions that define the goal states. Next, the preconditions that limit the applicability of the operators are refined and partially deleted, and the resulting model submitted to an evaluator that 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 that 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 heuristic. Such a heuristic should be applicable to every instance of the TSP problem and, when incorporated into $A^*$, would yield optimal TSP solutions in shorter times than the MST. 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.

## 4.3 PROBABILITY-BASED HEURISTICS

Probability is a language for **quantifying the unpredictability** of our environment. Probabilistic models specify a set of possible instances of the environment together with their associated degree of likelihood. In the Traveling Salesman problem, for example, the intercity distance table specifies one instance of the environment that the salesman may actually encounter. If, instead of specific distances, the table gives lower and upper bounds between which the actual distances may fall, that table delineates a *set* of possible problem instances. Finally, by specifying the *distribution* of the intercity distances, we are also able to determine whether one instance is more credible than another and by how much. In other words, the distribution quantifies the relative likelihood at which problem instances are expected to be encountered.

Although probability and statistics have developed in the past century into intricate mathematical specialties, probabilistic considerations are not less pervasive in our common, daily activities than other simplification heuristics. We choose one route over another because the former is *more likely* to be shorter, even though it has occasionally turned out longer. We say that a certain situation is *more desirable* than others mostly because similar situations in the past turned out disappointing *less often* (Pearl, 1977a).

Clearly people do tend to retain and invoke likelihood considerations as part of their mental encoding of past experiences. The acquisition and use of these likelihood relationships are governed by heuristic strategies of their own, such as those based on **similarity** and **ease of recall** (Tversky and Kahaneman, 1973, 1974). Normally an event is judged to be likely or probable if it possesses features considered *prototypical* of a large population. Thus we regard a white-skinned African to be a rare event by virtue of the disparity between the skin color observed and that expected from a normally protrayed African. In certain cases, the tendency to attribute to a single instance properties characterizing the population as a whole may even reach paradoxical proportions. For example, uniform sequences of coin-tossing outcomes are normally judged to be "less likely" than sequences with no apparent pattern (Tversky and Kahneman, 1974) simply because a "typical" sequence is expected to appear unorderly. Nevertheless, these mental heuristics perform relatively well in most cases and only highlight the importance of probabilistic estimates in everyday problem-solving activity.



Because the ultimate test for the success of heuristic methods is that they work well "most of the time," and because probability theory is our principal formalism for quantifying concepts such as "most of the time," it is only natural that probabilistic models should provide a formal ground for quantitatively evaluating the performance of heuristic methods. Moreover, it is equally natural that probabilistic models, whenever available, be consulted in the process of devising heuristic methods, or in selecting the parameters that govern these methods, so as to guarantee that only a small fraction of problem instances will escape an adequate treatment.

The following sections represent a brief overview of some probability-based heuristics. Some examples of these heuristics were already encountered in Section 3.2 where both the rating of nodes for expansion and the termination conditions were determined by probabilistic considerations based on the distribution of the random variable $h^*$. Additional examples are presented in subsequent chapters.

### 4.3.1 Heuristics Based on the Most Likely Outcome

Heuristic methods are often based on estimates of some unknown variables of the problem instance. For example, $A^*$ is driven by the function $h$ which estimates the unknown value of $h^*$ (the minimal cost required for completing the solution path). If the problem domain is characterized by a probabilistic model, it is sometimes possible to *calculate* the most-likely value of the unknown variable and use this value in constructing the heuristic function.

Probability calculus provides a machinery for performing such calculations. Consider, for example, the problem of finding the cheapest path in a graph where all arc costs are known to be drawn independently from a common distribution function with mean $\mu$. If $N$ stands for the number of arcs remaining between a node $n$ and the goal, then for large $N$ the law of large numbers (Feller, 1968) asserts that the cost of any path to the goal is likely to fall in the neighborhood of $\mu N$. Therefore, if we are sure that only one path leads from $n$ to the goal, we can take the value $\mu N$ 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, $\mu N$ may no longer be the best cost estimate for the cheapest path. However, if the structure of these solution paths is fairly regular, probability calculus can still be invoked to obtain that best cost estimate. Section 5.4 contains an example of such a computation. It will be shown there that in a tall uniform binary tree where every branch assumes the cost 1 or 0 with probabilities $p$ and $1-p$, respectively, the cost of the cheapest path converges to $\alpha^*(p)N$, if $p > \frac{1}{2}$, where $\alpha^*$ is a predetermined function of $p$ (see Figure 5.8). Hence, if the value of $p$ summarizes all the information available regarding costs in the unexplored portion of the tree, then $f = g + \alpha^* N$ would be the natural criterion for rating nodes in $A^*$.

By their very nature, such probability-based estimates are not guaranteed to be admissible. Occasionally these estimates may grossly overestimate the completion cost $h^*$ and, in turn, 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 for finding a "good" solution *most* of the time, probability-based heuristics may be perfectly acceptable (see exercise 5.4). They can also be incorporated into various hybrid strategies to determine the appropriate pruning criteria (see Section 5.4) and lead to dramatic reductions in the expected search time.

### 4.3.2 Heuristics Based on Sampling

Sampling is perhaps the oldest and the most widely practiced of all heuristic methods. Generally speaking, it involves inferring some property of a large set of elements from the properties of a small, randomly selected, subset of elements.

The economical benefits of sampling are widely studied in text books on statistics (e.g., Winkler and Hays, 1975). For example, if a manufacturer wishes to test whether a batch of 1 million nails contains no more than 1,000 defective units, he need not test the entire batch; a random sample of a few hundred nails may suffice to reflect the quality of the entire batch with an acceptable level of confidence. Significantly, the number of samples required for reaching a given level of confidence is fairly independent of the size of the batch and so, the larger the batch, the greater the economical savings achieved by sampling over that of exhaustive testing.

The implications to problems in **computational complexity** and algorithm design are straightforward. Assume, for instance, that we are given a huge database containing $N$ binary digits and we are asked to compute the *proportion* of digits that are 1. Clearly, there is no way of computing the exact value of the proportion without inspecting all $N$ digits, thus spending $N$ computational steps. However, if we are willing to tolerate a small error, a constant number of computations may suffice. We need only choose a random sample of $n$ digits and compute the proportion of 1's among the $n$ samples. For large values of $N$, the celebrated Bernoulli Theorem (Uspensky, 1937) guarantees that if the



underlying proportion is $\pi$ and the sample proportion is $\nu$, then the deviation between the two is bounded by the following relation

$$P(|\pi - \nu| \geqslant \epsilon) \leqslant 2e^{-n\epsilon^2/2}$$

Hence, the number of observations necessary to guarantee with sufficiently high probability ($\eta$) that the error $|\pi - \nu|$ will remain below $\epsilon$ is given by

$$n = \frac{2}{\epsilon^2} \log \frac{2}{1-\eta}$$

This number depends only on the precision and confidence parameters, $\epsilon$ and $\eta$, but remains constant with increasing $N$. Thus the computational savings produced by our willingness to tolerate some degree of imprecision can be enormous; a reduction of complexity from linear time to constant time.

Moreover, if $N$ increases indefinitely, we can choose the sample size to be any divergent function of $N$, say $n = g(N)$, and guarantee that as $N \to \infty$ the deviation between $\nu$ and $\pi$ will vanish with probability approaching 1; that is,

$$P(|\pi - \nu| < \epsilon) \to 1 \qquad \forall \epsilon > 0$$

Colloquially, we can say that our willingness to tolerate some small errors, guaranteed to "vanish almost surely," causes a complexity reduction from $N$ to $g(N)$. The function $g(N)$ can, of course, be made to diverge very slowly, e.g., like $\log N$, $\log \log N$, or even slower.

This technique is not limited to the task of computing proportions but is applicable to the computation of every statistical property of the database population: the mean, the variance, the median, the mode, etc., Weide (1978) has extended the applications of sampling methods to problems of sorting, searching, and selection.

In Section 4.3.3 we describe heuristic methods that are based on the assumption that problem instances are generated by known distribution functions; the search parameters are tailored to fit the distribution functions so as to solve *almost* all problem instances and, simultaneously, exhibit limited *average* search times as computed by the assumed distributions. In the event that these distributions are not known, sampling techniques can again be resorted to for **estimating the shape** of these distributions or their main parameters. In some cases, however, sampling-based estimates of certain input properties can be used to guide **probabilistic search algorithms** that achieve much stronger guarantees:

1. Every problem instance will be solved *exactly*.
2. The average search time will be limited for *every problem instance*, irrespective of any assumed distribution.

Using such estimates, Rabin (1976) has shown that the closest-pair among any set of $N$ points in a Euclidean space can be found in linear expected time; a

substantial improvement over the quadratic time required by an exhaustive examination of all pairs.

### 4.3.3 Probability-Based Heuristics in the Service of Semi-Optimization Problems

Assume that we are given an unsorted list of $N$ distinct real numbers and we are asked to find the largest element in the list. There is no way that we can perform this task without first inspecting all $N$ elements and, if each inspection counts as one computational step, it is clear that finding the maximal element requires $N$ steps.

Let us now assume that we are also permitted to consult an **oracle** who, for reasons of his own, reveals to us the true value of the maximal element in the list but not its position. With this added information we can inspect the elements sequentially and stop as soon as we find an element that matches the value given by the oracle. In the worst case, we will still need to inspect all $N$ elements but, if we are lucky, the search may terminate after the first inspection. On the average, assuming that the elements are arranged at random (or that our inspection proceeds at random), the time required for identifying the maximal element is $N/2$. A speedup factor of 2, then, is the utility of consulting an oracle, which turns the optimization problem into a satisficing one.

Let us further assume that, instead of finding the exact maximal element, we are willing to settle for any element in the **$\epsilon$-neighborhood** of the maximal, that is, if $X_m$ is the actual value of the maximal element, then any number in the set $\{X : X_m - X \leqslant \epsilon\}$ would qualify for a solution. What is now the benefit associated with knowing the value of $X_m$?

Clearly the answer depends on how early we can find an element falling above $X_m - \epsilon$. A well-known result in probability theory states that if we are conducting a series of independent trials to be terminated upon the first success and if the probability of success in each trial is $p$, then the expected number of trials performed is $1/p$. Consequently, if $p(X_m, \epsilon)$ stands for the probability that a typical element in the list falls in the range $X_m - \epsilon \leqslant X \leqslant X_m$ given that $X \leqslant X_m$, then the expected number of tests conducted is at most $[p(X_m, \epsilon)]^{-1}$. If the $X$'s are continuous bounded random variables independently drawn from a common distribution $F_X(x)$ and if $\epsilon$ is very small, then $p(X_m, \epsilon)$ is easily calculated to give

$$p(X_m, \epsilon) = P[X_m - \epsilon \leqslant X \leqslant X_m \mid X \leqslant X_m]$$

$$= \frac{F_X(X_m) - F_X(X_m - \epsilon)}{F_X(X_m)}$$

$$\approx \frac{F'_X(X_m)}{F_X(X_m)}\,\epsilon$$



and the expected number of tests $E(Z_N)$ is bounded by

$$E(Z_N \mid X_m) \leqslant [P(X_m, \epsilon)]^{-1} \approx \frac{F_X(X_m)}{F'_X(X_m)}\,1/\epsilon$$

which is a constant value, inversely proportional to $\epsilon$, but independent of $N$. Thus the benefits of knowing $X_m$ have been amplified significantly by tolerating an $\epsilon$ worth of suboptimality; instead of $N/2$, we can now find a solution in constant expected time.

Assume now that an oracle is not available, but instead we have a **probabilistic model** from which we may infer an estimate $K$ for $X_m$. We can still conduct a sequential inspection of the items on the list but with a modified stopping criterion. Instead of stopping at the interval $X_m - \epsilon \leqslant X \leqslant X_m$ which now is unknown, we can stop as soon as we get a number $X$ exceeding $K - \epsilon$, where $K$ is our guess for the value of $X_m$. A simple analysis, similar to the preceding one, shows that the expected number of inspections under this new stopping criterion is bounded by

$$E(Z_N) \leqslant \frac{F_X(X_m)}{F'_X(K)}\,1/\epsilon$$

As in the case of the oracle, this bound is also a constant. However, the number finally found is no longer guaranteed to lie in the $\epsilon$-neighborhood of $X_m$ unless we make sure that $X_m$ does not exceed the guess $K$. In case $X$ is a bounded random variable with a known distribution, we can easily meet this guarantee by choosing $K = x_o$, where $x_o$ is the highest possible value that $X$ may attain; that is,

$$x_o = \sup\{x : F_X(x) < 1\}$$

Moreover, if $F'(x_o)$ is too small, we still have the option of choosing $K = x_o - \epsilon'$ $(\epsilon' < \epsilon)$ and modifying the stopping rule to the interval $X \geqslant K - \epsilon + \epsilon'$, again guaranteeing the $\epsilon$-admissibility of the solution found together with the boundness of $E(Z_N)$.

However, if $X$ is an unbounded random variable, there is no way to guarantee that the actual maximal value $X_m$ will not exceed any fixed stopping limit $K$ and, therefore, we must abandon absolute guarantees of near optimality in every scheme short of exhaustive search. One may be tempted, at this point, to replace the near-optimality requirement with the requirement of *approximate-optimality;* accepting occasional solutions outside the $\epsilon$-neighborhood of $X_m$ but insisting that these occasions not occur too often. This can be accomplished by choosing a stopping bound $K(N)$ sufficiently large so that $P[X_m - K(N) < \epsilon]$ be made smaller than some $\delta > 0$. However, this will usually work against our second objective of keeping $E(Z_N)$ bounded or, at least, growing at a rate slower than $N$. The value of $K(N)$ will often be so high that the expected number of inspections needed until finding a sample exceeding $K(N)$ would be proportional to $N$ (see exercises 4.4a and 4.5a). Permitting

the error $\epsilon$ to grow proportionally to $X_m$ may sometimes alleviate this problem (see exercises 4.4b and 4.5b).

The purpose of the preceding discussion was to demonstrate how the willingness to tolerate some degree of suboptimality coupled with knowledge regarding the distribution of the input data can sometimes result in dramatic reduction of search complexity. The demonstration was made simple and feasible by three major assumptions:

1. That the items on the list are independent and identically distributed (i.i.d.) random variables.
2. That testing each candidate requires the same amount of computation.
3. That these random variables are bounded.

In more elaborate search problems, the values associated with each candidate solution (e.g., the costs of solution paths in a tree) will not be i.i.d.; each candidate may have a different probability of leading to a solution and may require a different amount of exploration effort. The expected search time then depends on the *order* in which tests are applied and optimizing this order may not be simple (see exercise 4.5 for a simple ordering optimization and Simon and Kadane (1975) for more elaborate cases). Search problems are further complicated by the fact that the exploration of any given candidate solution usually sheds light on the qualities of related solutions and may be performed incrementally, that is, we have the option of abandoning a partially explored candidate to attend another. Finally, the distribution of the solution weights very often varies with the problem size $N$ and so, one may not be able to find a stopping bound $K(N)$ that falls in the $\epsilon$-neighborhood of the optimal solution while, at the same time, guaranteeing that a solution meeting this bound will be found in a reasonable time.

Despite these complications, however, the essential steps of our exercise point to a general method of devising heuristics, applicable to a large class of optimization problems. We first use probabilistic considerations to estimate in what neighborhood the optimal solution is most likely to fall, and then devise an efficient search for identifying a solution within that neighborhood. This method succeeds if the following two conditions prevail:

1. The density of $X_m$ becomes highly concentrated about some predetermined quantity $K(N)$. In other words,
$$P[(1-\epsilon)K(N) \leqslant X_m \leqslant (1+\epsilon)K(N)] \to 1$$
2. In almost every problem instance, the target $\epsilon$-neighborhood around $K(N)$ should contain at least one solution that is discoverable by an easy search algorithm.

In Section 5.4 we analyze in detail a problem environment where these two conditions hold. More specifically, we show there that the costs along many solution paths in the neighborhood of $K(N)$ are likely to increase gradually at a predetermining rate $\alpha^*$ and hence, an irrevocable search strategy can be employed which prunes away any path found to be at variance with this expectation. This probability-based pruning strategy reduces the average complexity from exponential to linear.



## 4.4 BIBLIOGRAPHICAL AND HISTORICAL REMARKS

The idea of connecting heuristics with the use of simplified models was first communicated to me by Stan Rosenschein. Relaxation methods of obtaining lower bounds to optimization problems have been practiced extensively in operations research in conjunction with the branch-and-bound techniques (Lawler and Wood (1966), and Held and Karp (1970)). A simple way of obtaining relaxed problems is to enlarge the region over which a function is minimized until the boundaries become mathematically more tractable, e.g., piece-wise linear or totally unbounded. However, the most popular use of relaxation lies in **integer programming** problems (Shapiro, 1979) where we exploit the fact that it is often easier to minimize a function over a continuous domain (using the tools of calculus or linear programming) than the domain of integers. The solution to the relaxed problem can then be used as a lower bound in a branch and bound algorithm for the original problem (see exercise 4.1).

Rather than ignoring constraints altogether, an alternative way of relaxing problems by mathematical manipulations is to penalize the cost function for some constraint violations and reward it when some constraints are satisfied. This can be done most conveniently whenever the constraints are represented in the form of numerical inequalities (or equalities) where one may add to the cost function a penalty term proportional to the amounts by which the inequalities are violated. This technique was coined **"Lagrange relaxation"** by Geoffrion (1974) and is surveyed by Fisher (1981).

The auxiliary-problem approach was formally introduced to nonnumeric, AI-type problems by Gaschnig (1979b), Guida and Somalvico (1979), and Banerji (1980). Gaschnig described the space of auxiliary problems as "subgraphs" and "supergraphs," obtained by deleting or adding edges to the original problem graph. The swap-sorting problem was contrived by Gaschnig to exemplify supergraphs of the 8-Puzzle. Guida and Somalvico use propositional representation of constraints similar to that of Section 5.1.2 and propose the use of relaxed models for generating admissible heuristics. A slightly different formulation is also given in Kibler (1982).

Valtorta's thesis (1981) contains a proof of the consistency of relaxation-based heuristics, and an analysis of the overall complexity of searching both the original and the auxiliary problems. It shows that if the auxiliary problem is searched by a breadth-first strategy, then the overall complexity will be worse than simply executing a breadth-first search on the original problem. The use of systematic deletions in search of decomposable problems was proposed by Pearl (1982a). Other approaches to automatic generation of heuristics are reported by Ernst and Goldstein (1982); Korf (1982); and Lenat (1983).

An effective planning system (named ABSTRIPS) using constraints relaxation was implemented by Sacerdoti (1974). ABSTRIPS first synthesizes a global abstract plan