Problem Representation for Refinement

H. ALTAY GUVENIR and VAROL AKMAN
Dept. of Computer Engineering and Information Science, Bilkent University, 06533 Ankara, Turkey
(guvenir@trbilun.BITNET, akman@trbilun.BITNET)

Abstract. In this paper we attempt to develop a problem representation technique which enables the decomposition of a problem into subproblems such that their solution in sequence constitutes a strategy for solving the problem. An important issue here is that the subproblems generated should be easier than the main problem. We propose to represent a set of problem states by a statement which is true for all the members of the set. A statement itself is just a set of atomic statements which are binary predicates on state variables. Then, the statement representing the set of goal states can be partitioned into its subsets each of which becomes a subgoal of the resulting strategy. The techniques involved in partitioning a goal into its subgoals are presented with examples.

Key words. Problem-solving, strategy, problem representation, refinement, machine learning, mechanical discovery.

Introduction

Problem solving has been one of the laboratories of artificial intelligence (Lauriere, 1990). In very simple terms, problem solving involves finding a path from an initial state to a goal state using some kind of search (Ernst and Newell, 1969; Simon, 1983). To solve the same problem for a different initial state one has to go through the same costly search process again. If the same problem will be solved for many different initial states, then solving the problem for each initial state becomes infeasible. Instead, it would be more beneficial to solve the problem in general (that is, independent of the initial states) and then using this general solution, solve the problem for a particular initial state. We will call such a general solution a *strategy*. This paper proposes a problem representation technique which enables the decomposition of a problem into a strategy.[1]

What is a strategy? A strategy for solving a problem is a general solution for that problem, in other words, a solution for all possible initial states. A strategy can be constructed as a decomposition of the problem into easier problems. A *strategy* to solve a problem P can be defined as a sequence of subproblems $P_1, P_2, \ldots, P_n$ such that solving them in sequence is equivalent to solving the problem P, and each of the subproblems P_i is easier than the problem P. Such a decomposition of a problem involves symbolic processing on the description of the problem. This scheme is based on representing a set of problem states by a suitable for such symbolic processing. In the course of the paper we will develop a problem representation scheme which is suitable for mechanically discovering a strategy for a given problem. This scheme is based on representing a set of problem states by a statement which is true for all the members of the set. Here, a statement itself is just a set of atomic statements which are binary predicates on

Minds and Machines **2**: 267–282, 1992.

state variables. Then, the statement representing the set of goal states can be partitioned into its subsets each of which becomes a subgoal of the resulting strategy. At the end of the paper the techniques involved in partitioning a goal into its subgoals are presented with example strategies that are discovered mechanically.

Problem Representation

In the literature a *problem*, is defined by a 3-tuple $P = \langle S, O, G \rangle$ where

S: the set of states,
O: the finite set of operators, and
$G \subset S$: the set of goal states.

Here, each operator $o_i \in O$ is a function o_i: $S \rightarrow S$.

A *problem instance* p_0 is defined as a problem P with a particular initial state $s_0 \in S$, i.e., $p_0 = \langle P, s_0 \rangle$. Then, a *solution* to the problem instance p_0 is sequence of operators $o_1, o_2, \ldots, o_n$ such that $o_n(o_{n-1} \ldots (o_2(o_1(s_0))) \ldots) \in G$. Here $o_i(s)$ is the state obtained by the application of the operator o_i to the state s. A solution to a problem instance p_0 can also be defined as a sequence of states $s_0, s_1, s_2, \ldots, s_n$ such that for each s_i where $0 < i < n$, $o_i(s_{i-1}) = s_i$ for an $o_i \in O$, and finally $s_n \in G$. Finding a solution to the problem instance p_i requires a search in S for a path from s_0 to a state in G. An example problem and a solution for an instance of it are given in Figure 1.

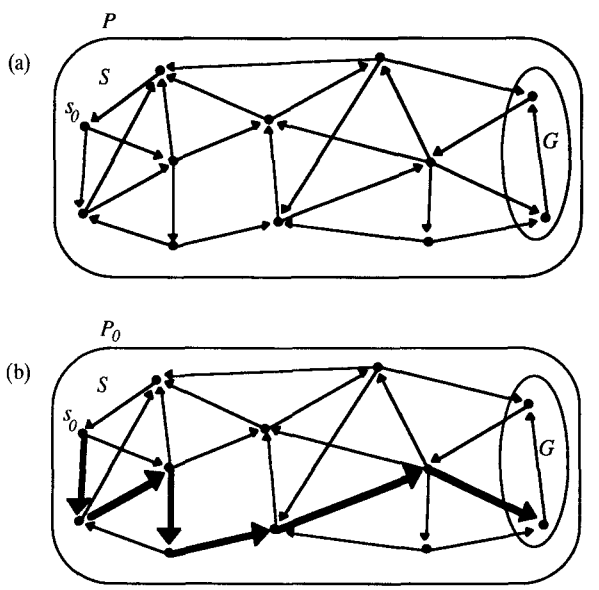

Fig. 1. (a) An example problem; (b) solution to an instance of the problem.

Although a problem can be defined formally as above, $P = \langle S, O, G \rangle$, in practice this representation is inadequate if any symbolic processing has to be done on a problem. Also, it is hard to enumerate all the possible problem states and compute the set S. Another difficulty is that the set of goal states G may not be specified explicitly in many problems. For example, the goal states of problems such as the Fool's Disk (Ernst and Goldstein, 1982) and the Rubik's Magic are not given in their definitions. Instead, a statement describing the goal states is given. If the actual goal state is known, the solution is trivial in the Fool's Disk problem. Therefore, in such problems, the difficulty of the problem is to determine the states that satisfy the goal statement.

Solving a Problem in General or Learning a Strategy for a Problem

One general method of attack upon a problem, employed by human problem solvers, is to break down the goal to be attained into a set of subgoals, which together satisfy the conditions of the original problem so that if each subgoal, taken separately, can be attained, the given problem is solved (Newell and Simon, 1972). If each of these subgoals will be attained sequentially (either on a sequential machine or by a human), a solution to a problem in general can be defined in a similar manner to the definition of a solution to a problem instance, given above. A *solution* to a problem $\langle S, O, G \rangle$ in general is a sequence of subgoals $G_0, G_1, \ldots, G_n$ where

G_0 is the set of all possible initial states,
$G_n = G$, and
for each $G_i, 0 \leq i < n$, there is a set of operators $O_i \subset O$ such that for states $s \in G_i$ there is an operator $o \in O_i$ such that $o(s) \in G_{i+1}$.

An example strategy for the problem in Figure 1 and a solution using this strategy are given in Figure 2.

A strategy, then, is a sequence of transitions from one set of states to another using a specified set of operators. Each transition itself is a search in the set of all states S. That is, each transition (or stage) is a subproblem, similar to the main problem. The problem solver using such a strategy, be it a human or a machine, backtracks to the previous stage in case it determines that the current subproblem

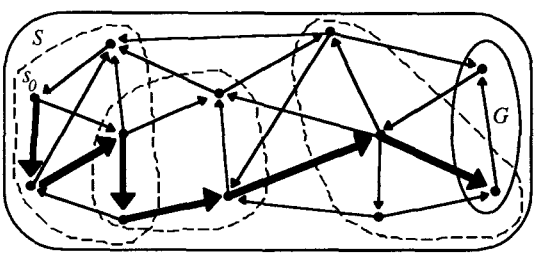

Fig. 2. A strategy for the problem in Figure 1(a).

has no solution; that is, it is impossible to reach a goal state using the given set of operators for that stage. On the other hand, as a special case, if there exists at least one operator given for each state in G_i, then the solution becomes trivial, since no search will be required. The strategies learned for Korf's Macro Problem Solver (MPS) have this property (Korf, 1985). To unify the representation of problems and subproblems, let us represent a problem by a 4-tuple $P = \langle S, I, O, G \rangle$ where I is the set of possible initial states. That is, P is the problem of finding a path in the universe S from any state in I to a state in G, using the operators in O. Therefore, a strategy for solving the problem P can be defined as a sequence of subproblems $P_1, P_2, \ldots, P_n$ such that solving them in sequence is equivalent to solving the problem P, where $P_i = \langle S, I_i, O_i, G_i \rangle$. In order for a strategy to be of any use, its subproblems should be easier than the problem itself. Comparison of problems in terms of their difficulties is usually hard. One way of comparing the difficulties of two problems in the same universe S is to compare their goal states. A problem P_e is *easier* than a problem P_d if the set of goal states of P_d is a subset of that of P_e and every other parameter is the same. That is, the problem $P_e = \langle S, I_e, O, G_e \rangle$ is easier than the problem $P_d = \langle S, I_d, O, G_d \rangle$ if $G_d \subset G_e$ and $I_e \subset I_d$. Notice that this is only a sufficient condition. That is, a problem P_a may be easier than another problem P_b even though the above conditions are not satisfied (e.g., both have the same goal but P_a has more relevant moves). However, this definition will suffice for the purpose of decomposing a problem into a strategy. We can now define a strategy as follows:

DEFINITION. A *strategy* for solving a problem $P = \langle S, I, O, G \rangle$ is a sequence of easier subproblems (or *stages*) $P_1, P_2, \ldots, P_n$ that satisfy the following conditions:

$$\begin{aligned} &\text{i.} \quad I_1 = I \\ &\text{ii.} \quad I_i = G_{i-1}, \quad \text{for } 1 < i \leq n \\ &\text{iii.} \quad G_n = G \\ &\text{iv.} \quad O_i \neq \emptyset, \quad \text{for } 1 \leq i \leq n. \end{aligned} \tag{1}$$

The set of possible initial states I_1 of the first stage is equal to the set of possible initial states of the problem. The set of initial states of any stage is equal to the set of goal states of the previous stage. The set of goal states of the last stage is the set of the goal states of the problem. There must exist some operators that are relevant to solving each of the subproblems. If there is no solution to a subproblem P_i with the given set of operators, then the problem solver backtracks to the previous stage and re-solves P_{i-1} to obtain another state in G_{i-1}. Usually, the set of initial states of a problem is equal to the whole set of problem states, i.e., $I = S$. A strategy for solving such a problem is depicted in Figure 3. An example three-stage strategy for the problem in Figure 1 is shown in Figure 4.

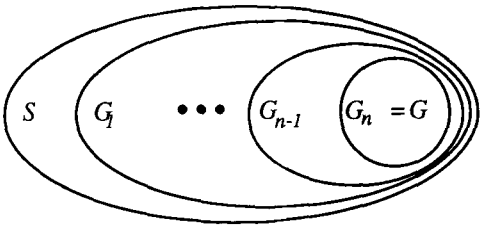

Fig. 3. Goals of subproblems forming a strategy.

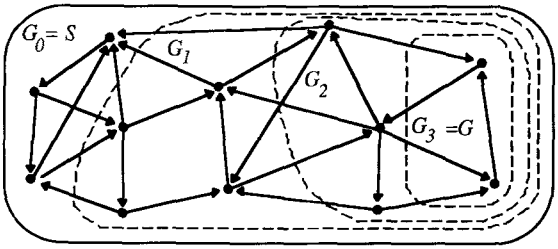

Fig. 4. Subgoals of a strategy for the problem in Figure 1(a).

Representation of the Set of States

As mentioned earlier, neither the set of states S nor the set of goal states G are given in the problem description explicitly. Problem states are described in terms of some lower level problem variables. Here, problem variables are the parameters of the problem that can be changed by the operators. For example, in the well-known Towers of Hanoi Problem (THP) (Banerji, 1980), a state is described by the values of some lower level components, that is, disks and pegs. One possible way to represent a state in THP is to give the positions of all the disks. The set of goal states is given by a statement which is true only for the goal states. The goal states of the THP are described by the statement: "*all the disks are on peg C.*" Similarly, the goal statement of the Mod-3 puzzle[2] in Guvenier and Ernst (1990) is "*all the cells have the same value.*" The goal of the Rubik's Cube puzzle,[3] on the other hand, is "*all the faces of the cube have a solid color.*" In general, a statement describes a set of states, those states which *satisfy* the statement. A statement specifies some relations among the state variables. Therefore, it is natural to represent a state by a vector of state variables.

DEFINITION. A *state* is a vector of state variables $(s_1, s_2, \ldots, s_m)$ where each s_i is chosen from a set of values V_i. Note that $S \subset V_1 \times V_2 \times \cdots \times V_m$.

This is similar to Korf's definition (in Korf, 1985). For example, if the positions of the disks are the state variables in the THP, the vector $\langle A, C, B, A \rangle$ represents the state in which the first and the fourth disks are on peg A, the second disk is on

peg C and the third disk is on peg B. However, in Korf's representation the state vector for the Rubik's cube is composed of the position and the orientation values of 26 cubicles, whereas here a state vector contains the color values of each of the 54 facets.

To represent a statement about states we propose the following definitions:

DEFINITION. An *atomic statement* is a predicate with two arguments. The arguments can be constants or state variables.

For example, disk1 = C is an atomic statement in the THP. Similarly, in the Mod-3 puzzle, s11 = s12 is an atomic statement which indicates that the upper left and upper middle cells have the same values. In the Rubik's cube puzzle the atomic statement F2 = F9 indicates that the center facet (F9) in the front face has the same color as the upper middle facet (F2) of the front face; see Figure 9 for naming of the facets.

DEFINITION. A *statement* is a set of atomic statements. A statement is interpreted as the conjunction of its elements.

A statement $Q(s)$ represents a set of problem states $S_q = \{s \mid Q(s)\}$. Therefore, $Q(s) \Rightarrow s \in S_q$. For instance, the statement $Q(s) = \{\text{disk2} = \text{C}, \text{disk2} = \text{disk3}, \text{disk3} = \text{C}\}$ represents the set of states in which both disk2 and disk3 are on the peg C in the THP. The statement

$$\{\text{F1} = \text{F2}, \ldots, \text{F1} = \text{F9},\ \text{F2} = \text{F3}, \ldots, \text{F2} = \text{F9},\ \ldots,\ \text{F8} = \text{F9}\}$$

in Rubik's Cube represents the set of states in which all the facets in the front face have the same color. Similarly, the set of goal states of the Mod-3 puzzle can be represented by the statement $G(s) = \{s11 = s12,\ s11 = s13, \ldots, s32 = s33\}$.

An empty statement is **true** for all states and therefore represents the set of all problem states, S. The set union of two statements is equivalent to their logical conjunction. That is, if $Q(s)$ represents the set of states S_q and $R(s)$ the set S_r, then the statement $Q(s) \cup R(s)$ represent the set of states $S_q \cap S_r$. If $Q(s)$ is a subset of $R(s)$, then every problem state s that satisfies $R(s)$ also satisfies $Q(s)$;

Table I. Statements and the sets they represent

Statements	Problem states	Logical meaning
$Q(s) = \emptyset$	$\{s \mid Q(s)\} = S$	$Q(s) =$ **true**
$Q(s) = R(s) \cup T(s)$	$\{s \mid Q(s)\} = \{s \mid R(s)$ and $T(s)\}$	$Q(s) = R(s)$ & $T(s)$
$Q(s) \subset R(s)$	$\{s \mid R(s)\} \subset \{s \mid Q(s)\}$	$R(s) \Rightarrow Q(s)$

that is, $R(s)$ logically implies $Q(s)$. The relations between the statements and the sets of states are shown in Table I.

Operators and Their Properties

An operator has two important parts: its precondition, and its effect on the state it is applied. Therefore, we will represent an operator by a pair $o = \langle PC(s), A \rangle$ where $PC(s)$ is the *precondition statement*, possibly empty, and A is the set of assignments which are made to the state variables by the application of the operator. Formally, an operator o is a function $o: \{s \mid PC(s)\} \rightarrow S$.

DEFINITION. An operator o is *safe* over a statement $Q(s)$ if when o is applied to a state s satisfying $Q(s)$, the resulting state $o(s)$ also satisfies the statement $Q(s)$. Formally, o is safe over $Q(s)$ if

$$\forall s[Q(s) \Rightarrow Q(o(s))] .$$

For example, any operator that moves disk1 is safe over the statement {disk3 = C} in the THP. Similarly, operator o13, which increments the values of cells in first row and third column by one modulo 3, is safe over the statement {s11 = s12} in the Mod-3 puzzle. The operators F+ and F−, which rotate the front face 90° in the positive direction (counterclockwise) and in the negative direction, respectively, are safe over the statement {F1 = F2, F1 = F3, . . . , F8 = F9} in the Rubik's cube puzzle.

DEFINITION. An operator o is *irrelevant* to going from $Q(s)$ to $R(s)$ if o is safe over $Q(s)$, and when applied to a state that satisfies $Q(s)$ but not $R(s)$, then the resulting state $o(s)$ will never satisfy $R(s)$. That is, the application of o to a state that does not satisfy $R(s)$, will not generate a state satisfying $R(s)$. Formally, o is irrelevant to going from $Q(s)$ to $R(s)$ if o is safe over $Q(s)$ & $\sim R(s)$.

For example, any $disk_1$ move in the THP is irrelevant to going from {disk3 = C} to {disk2 = C, disk2 = disk3, disk3 = C}. Operator o13 in the Mod-3 puzzle is relevant to going from {s11 = s12} to {s11 = s12, s23 = s33}. Similarly the operators B+ and B−, that rotate the back face, are irrelevant to going from ∅ to {F2 = F9} in the Rubik's Cube puzzle.

DEFINITION. An operator o is *relevant* to going from $Q(s)$ to $R(s)$ if o is safe over $Q(s)$ and not irrelevant to going from $Q(s)$ to $R(s)$. That is, if o is relevant to going from $Q(s)$ to $R(s)$, then there is a chance that $R(o(s))$ will be true if s satisfies $Q(s)$ but not $R(s)$.

For example, any $disk_2$ move in the THP is relevant to going from {disk3 = C} to {disk2 = C, disk2 = disk3, disk3 = C}. Operator o13 in the Mod-3 puzzle is

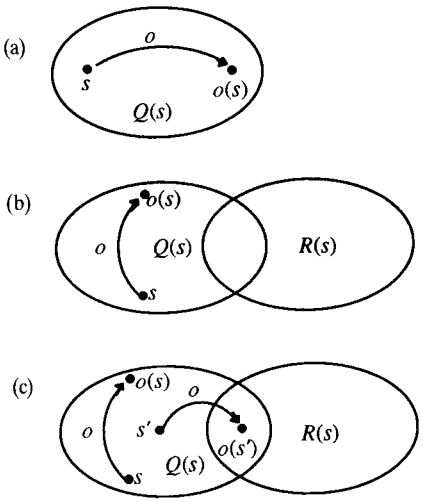

Fig. 5. Properties of operators. (a) o is safe over $Q(s)$; (b) o is irrelevant to going from $Q(s)$ to $R(s)$; (c) o is relevant to going from $Q(s)$ to $R(s)$.

relevant to going from {s11 = s12} to {s11 = s12, s11 = s13, s12 = s13}. The operators U+ and U−, that rotate the upper face, are relevant to going from ∅ to {F2 = F9} in the Rubik's Cube puzzle.

Safety, relevancy and irrelevancy of operators are illustrated in Figure 5.

DEFINITION. An operator o is *potentially applicable* to a set of states represented by $Q(s)$ if the precondition statement $PC(s)$ of o does not conflict with $Q(s)$. That is, there are some states that satisfy both $Q(s)$ and $PC(s)$. Formally,

$$\exists s[Q(s) \& PC(s)] .$$

For example, in the THP the operator o2AB (move disk2 from peg A to peg B), whose precondition statement is {disk1 = C, disk2 = A}, is potentially applicable to all states in {disk1 = C} or {disk3 = C}, but is not potentially applicable to any state in {disk2 = C}.[4]

Problem Representation by Statements

The sets of states of a problem can be represented by statements as defined above. Therefore, we can represent a problem as a 4-tuple $P = \langle S, I(s), O, G(s) \rangle$ where $I(s)$ and $G(s)$ are statements representing the set of initial states and the set of goal states, respectively.

Difficulties of problems can be compared by checking their initial and goal statements as well. A problem P_e is *easier* than a problem P_d if

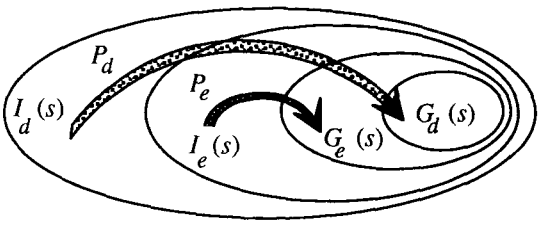

Fig. 6. Comparison of difficult and easy problems.

$$I_e(s) \Rightarrow I_d(s) \text{ and} \qquad (2)$$
$$G_d(s) \Rightarrow G_e(s) .$$

In other words, if the possible initial states of P_e are also initial states of P_d and the goal states of P_d are also goal states of P_e, then P_e is easier than P_d. Again, this is only a sufficient condition. Comparison of difficulties of two problems is depicted in Figure 6. In this figure, the problem P_d of obtaining a state that satisfies $G_d(s)$ from a state that satisfies $I_d(s)$ (the bigger arrow) is more difficult than the problem P_e of obtaining a state that satisfies $G_e(s)$ from a state that satisfies $I_e(s)$ (the smaller arrow).

Refining a Problem into a Strategy

Refinement is based on a decomposition of the goal statement of a problem into subgoals by partitioning the goal statement $G(s)$ into subgoals $G_1(s)$, $G_2(s), \ldots, G_n(s)$, and finding the relevant operators $O_1, O_2, \ldots, O_n$ for each subgoal such that $P_1, P_2, \ldots, P_n$ where $P_i = \langle S, S_{i-1}(s), O, G_i(s) \rangle$, satisfy the following conditions which are derived from (1):

$$\begin{array}{ll} \text{i.} & G_0(s) = I(s) \\ \text{ii.} & G_n(s) = G(s) \\ \text{iii.} & O_i \neq \emptyset , \quad \text{for } 1 \leqslant i \leqslant n \end{array} \qquad (3)$$

The formal description of the refinement algorithm is given in Figure 7. The refinement algorithm is based on grouping those statements that have exactly the same set of relevant operators into subgoals. The heuristic used here is that if a set of atomic statements have exactly the same set of relevant operators, then there is a high amount of interaction between them and, therefore, they should be satisfied at the same time. For example, the atomic statements F2 = F9 and U6 = U9 in the Rubik's Cube problem have exactly the same set of relevant moves, namely, {F+, F−, U+, U−}. Therefore, both of these atomic statements must be satisfied at the same time. If there is an atomic statement for which no relevant operators are found, the problem is considered to be 'unsolvable'. If all the operators in O are relevant to every atomic statement in $G(s)$, then no refinement is possible, and the refinement algorithm terminates by returning the problem unchanged.

refine(<S, $I(s)$, O, $G(s)$>):

1. For each atomic statement $g_i(s)$ in $G(s)$, find the set of operators O_i that are relevant to going from $I(s)$ to $\{s \mid g_i(s)\}$.

   If there is any atomic statement $g_i(s)$ with no relevant operators, then return "problem is unsolvable."

2. Form statements $G_i(s)$ by grouping the atomic statements with the same relevant operators into one statement.

   If all atomic statements are grouped into a single statement, then the problem cannot be refined; hence return <S, $I(s)$, O, $G(s)$>.

3. For each statement $G_i(s)$, determine the set of operators OS_i that are safe over and potentially applicable to $I(s) \cup G_i(s)$, and form a list of candidates <$G_i(s)$, O_i, OS_i>.

4. While the list of candidates is not empty do:
   Choose the candidate <$G_i(s)$, O_i, OS_i> such that $|OS_i|$ is the largest.
   Let *rest* be **refine**(<S, $I(s) \cup G_i(s)$, $G(s) - G_i(s)$, OS_i>).
   If *rest* is not unsolvable, then return <S, $I(s)$, $G_i(s)$, O_i> followed by *rest*,
   else remove the first candidate from the list of candidates.
   end of while.
   Problem cannot be refined, return <S, $I(s)$, O, $G(s)$>.

Fig. 7. The refinement algorithm in pseudo-code.

In the third step of the refinement algorithm, the set of operators, OS_i, that are safe over and potentially applicable to both $I(s)$ and $G_i(s)$ (i.e., $I(s) \cup G_i(s)$ as sets of atomic statements) are determined for each $G_i(s)$. This is to find the operators that can be used to solve the rest of the problem, if the statement $G_i(s)$ is selected to be the goal of the first stage. Each statement $G_i(s)$ is a candidate to be the first subgoal. A list of candidates with their relevant operators and safe operators is formed.

The first subgoal is determined in the fourth step. In order for a candidate to be selected as the first subgoal the remaining part of the problem, $G(s) - G_i(s)$, must be solvable. The candidate with the maximum number of safe operators (the one which leaves the largest number of operators to solve the rest of the problem) is tried first. The heuristic used here is that *the larger the number of available moves in a problem, the more likely that it will be solvable*. The test of solvability is done by trying to refine the rest of the problem recursively. If the result indicates that the rest of the problem is unsolvable, then the next candidate is tried. Otherwise, the result of the refinement is a list whose first element is the subproblem

representing the selected candidate and the rest of the list is the refinement found for the rest of the problem. If all the candidates are exhausted, the refinement algorithm terminates unsuccessfully, returning the problem unchanged.

The initial statement of the first subproblem is the same as the initial statement of the main problem, that is, $I_1(s) = I(s)$. The initial statement of each remaining subproblem is the goal statement of the preceding subproblem; that is, $I_i(s) = G_{i-1}(s)$, cf. (1). Also, each subgoal statement generated by the refinement algorithm is a subset of the goal statement of the main problem. That is, for each subproblem $I_i(s) \Rightarrow I(s)$ and $G(s) \Rightarrow G_i(s)$. Therefore, each subproblem generated by the refinement algorithm is easier than the main problem, cf. (2).

Determination of Properties of an Operator over Statements

The refinement algorithm makes use of properties such as safety and relevancy of an operator over a given statement. These properties depend on the effects of the assignments of the operator on the relations representing the atomic statements. This kind of knowledge is problem- or, in general, domain-dependent, and should therefore be separated from the strategy learning mechanism and put into a *domain-dependent knowledge base* (DDKB). For example, the DDKB for Mod-3 puzzle should include facts such as incrementing a value modulo 3 three times will not change its value, or facts such as if $x = y$, then $f(x) = f(y)$ for any function f.

A DDKB is designed to answer a question in the form

"Does $Q(s)$ imply $r(o(s))$?"

The input to DDKB is a statement $Q(s)$, an atomic statement $r(s)$ and an operator o; the output is "yes" if $r(o(s))$ can be inferred from $Q(s)$, "don't know" otherwise.

Given such a DDKB, the safety of an operator o over a statement $Q(s)$ can be determined by asking the question "Does $Q(s)$ imply $q_i(o(s))$?" for each atomic statement $q_i(s) \in Q(s)$. If the answer is "yes" for all atomic statements, then o is safe over $Q(s)$.

In the first step of the refinement algorithm, operators that are relevant to going from an initial statement $I(s)$ to an atomic statement $g_i(s)$ of the goal are sought. In order to determine the relevancy of an operator o, the question "Does $I(s)$ & $\sim g_i(s)$ imply $g_i(o(s))$?" is asked. If the answer is "yes", then the operator o is irrelevant, otherwise it is considered to be relevant to going from $I(s)$ to $\{g_i(s)\}$.

Some Example Strategies

The algorithm given above has been implemented in Allegro Common Lisp on a

P_1: $I_1(s) = \emptyset$
O_1 = {o21, o22, o31, o32};
$G_1(s)$ = {s11 = s12, s23 = s33};

P_2: $I_2(s)$ = {s11 = s12, s23 = s33};
O_2 = {o23, o33};
$G_2(s)$ = {s11 = s13, s21 = s31, s22 = s32};

P_3: $I_3(s)$ = {s11 = s12, s11 = s13, s21 = s31, s22 = s32, s23=s33};
O_3 = {o12, o13};
$G_3(s)$ = {s11 = s21, s22 = s23};

P_4: $I_4(s)$ = {s11 = s12, s11 = s13, s11 = s21, s21 = s31, s22 = s32,
s22 = s23, s23 = s33};
O_4 = {o11};
$G_4(s)$ = {s11 = s22};

Fig. 8. Refinement of Mod-3 puzzle into a four-stage strategy.

NeXT system and tested for several problems. The Mod-3 puzzle is decomposed into the four step strategy shown in Figure 8.[5] The initial statement of the first step is empty, which represents the logical value **true**. Therefore, $I_1(s)$ is true for any state; i.e., the initial state can be any state. The goal of the first stage is to get the first two cells in the first row (s11 and s12) equal and also the last cells in the second and third rows (s23 and s33) equal. For example,

1 1 0
2 0 2
0 1 2

is such a state. Note that the goal of the first stage is easier to satisfy than the goal of the main problem. Also, only the operators {o21, o22, o31, o32} will be used in the search for the first step; this reduces the branching factor from 9 to 4. Therefore stage 1 is easier than the whole problem; the same is true for the other three stages as well.

When solving stage 2 it is known that all the initial states satisfy $G_1(s)$ and that statement should not be violated at this step; therefore $G_1(s)$ is the statement about the initial states of stage 2. Similar conditions hold for the remaining stages as well. If, in the second stage, no state satisfying $G_2(s)$ can be found with the operators o23 and o33, then the problem solver backtracks to the first stage and finds another state satisfying $G_1(s)$.

The decomposition of the Rubik's Cube puzzle is given in Figure 9. The resulting strategy has 5 stages. Although all the stages are easier than the whole problem, their difficulties are not uniform. The first stage is the easiest and the level of difficulty increases towards the later stages. Such a decomposition resembles the strategy that would be learned by a person who is just introduced to the puzzle.

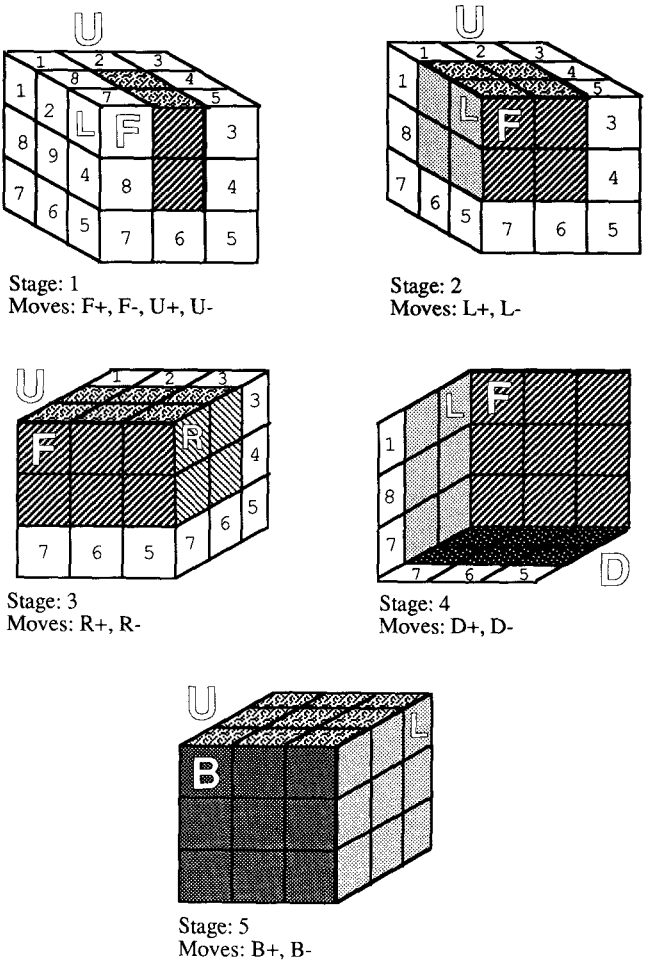

Fig. 9. Decomposition of the Rubik's Cube puzzle into 5 subproblems.

However, after fooling around with the cube and acquiring a few useful combinations of operators, the same person would come up with a different and a finer grain decomposition. Such a useful combination of operators is called a macro.[6] For example, such a person would learn that the macro U − L + U + D − F + F + D + U − L + U + is a useful one since it rotates only three side cubies and does not modify the other cubies. The person would generalize this macro and use all its symmetric versions for other sides. Similarly, the macro U + R + U − L − U + R − U − L + rotates only three corner cubies in the upper face and does not modify the others; and its other symmetric forms can be used for other faces. With these new macros at his disposal the person should be able to decompose the puzzle into a finer grain strategy. We can expect a similar behavior from our refinement algorithm. The decomposition of the Rubik's Cube puzzle with the enhanced set of moves (12 operators and 48 macros) is a 15-stage strategy as

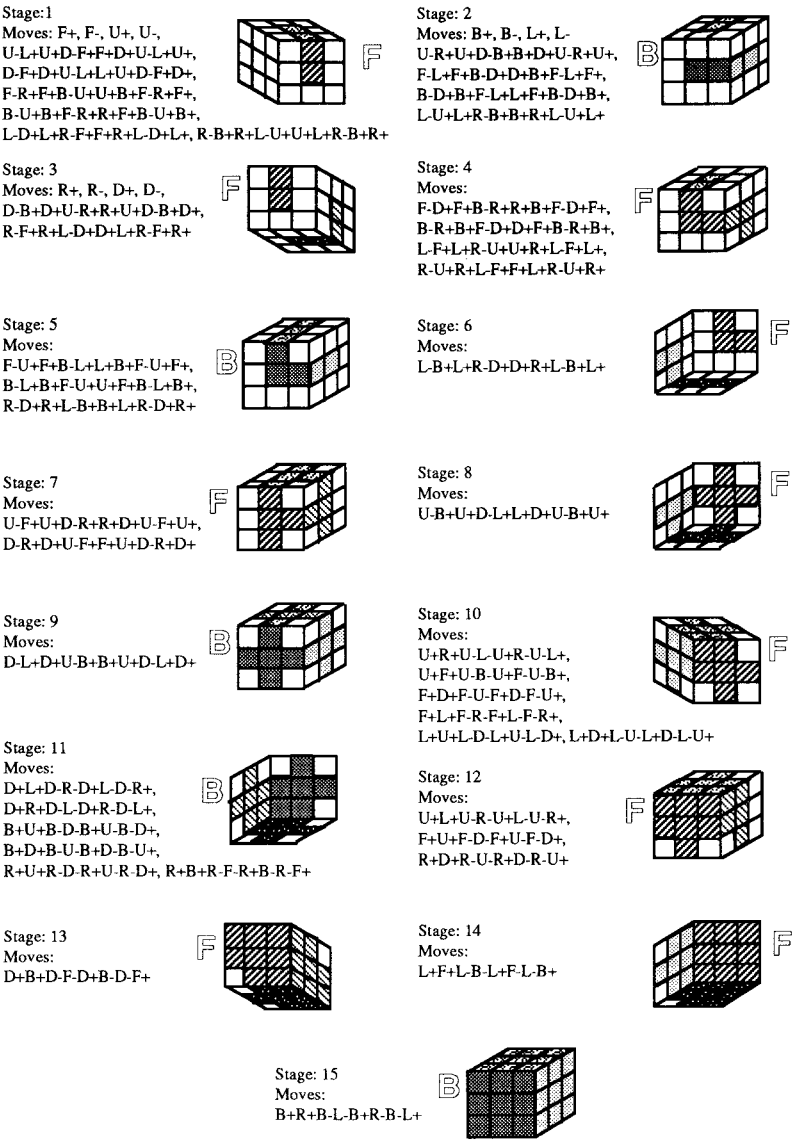

Fig. 10. Refinement of the Rubik's Cube puzzle into a 15-stage strategy.

shown in Figure 10. Note that the goal of the first stage is the same as the goal of the first stage of the strategy in Figure 9. The second stage of first strategy is completed in the 10th stage of the second one. Up to the 10th stage all the side cubies are put into their goal positions. The third stage in the first strategy corresponds to the 12th stage, and fourth stage to the 14th stage in the second strategy. The second strategy is more useful than the first one since its stages are easier than the stages of the first strategy. This improvement in the decomposition

reflects the improvement we would expect to see in human problem solving after acquiring useful macro moves (Nourse, 1981).

Conclusion

Solving a problem independent of its initial states requires developing a strategy for that problem. Although a strategy can be in any form, we defined here a strategy as a sequence of easier subproblems. Statements, which are sets of atomic statements, are proposed to represent sets of problem states. With this representation, a strategy can be learned by decomposing the goal statement into its subsets each of which corresponds to a subgoal.

An algorithm for decomposing a problem into a sequence of subproblems is given. Using this algorithm, along with a goal statement for each stage a set of relevant operators is learned. The algorithm is tested on problems with different characteristics. The strategies learned by this algorithm are similar to the ones developed by humans.

Sometimes the decomposition of a problem does not yield a useful strategy. This stems from the fact that the operators modify a large number of state components in the representation of the problem. As in human problem solving, macro moves are helpful in decomposing such problems. This refinement algorithm has been shown to reflect the same improvement by decomposing the problem into easier subproblems.

Acknowledgements

We are grateful to the editor of *Minds and Machines* and two anonymous referees for their invaluable comments on earlier drafts of this paper. We wish to thank George W. Ernst (Case Western Reserve University) and Ranan B. Banerji (Temple University) for useful conversations.

Notes

[1] A brief precursor to this paper was presented by the first author to the *Working Session on Algebraic Approaches to Problem Solving and Representation*, Philips Laboratories, Briarcliff, NY, 1990.

[2] The Mod-3 puzzle is played on a 3×3 board, where each cell can take any integer value between 0 and 2 (inclusive). An operator consists of playing on a cell, which will increment by modulo 3 the value of each cell that is in the same row or the same column as the cell that is being played on. The goal is to have the same value on every cell.

[3] Rubik's Cube is a trademark of Ideal Toy Corporation, Hollis, NY.

[4] We assume that each variable is single-valued.

[5] Note that in Figure 8 atomic statements of the initial statements are not shown in the goal statements.

[6] The idea of composing a sequence of operators and viewing the sequence as a single operator in machine problem solving goes back as far as Amarel's paper on representations for the Missionaries and Cannibals problem (Amarel, 1968).

References

Amarel, S. (1968), 'On Representation of Problems of Reasoning about Actions', in D. Michie, ed., *Machine Intelligence* **3**, Edinburgh, Scotland: Edinburgh University Press.

Banerji, R. B. (1980), *Artificial Intelligence: a Theoretical Approach*, New York, NY: North-Holland.

Ernst, G. W. and Goldstein, M. M. (1982), 'Mechanical Discovery of Classes of Problem-Solving Strategies', *Journal of ACM* **29**, pp. 1–23.

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

Guvenir, H. A. and Ernst, G. W. (1990), 'Learning Problem Solving Strategies Using Refinement and Macro Generation', *Artificial Intelligence* **44**, pp. 209–243.

Korf, R. E. (1985), *Learning to Solve Problems by Searching for Macro-Operators*, Boston, MA: Pitman Advanced Publishing Program.

Lauriere, J.-L. (1990), *Problem Solving and Artificial Intelligence*, London: Prentice-Hall.

Newell, A. and Simon, H. A. (1972), *Human Problem Solving*, Englewood Cliffs, N.J.: Prentice Hall.

Nourse, J. G. (1981), *The Simple Solution to Rubik's Cube*, New York: Bantam Books.

Simon, H. A. (1983), 'Search and Reasoning in Problem Solving', *Artificial Intelligence* **21**, pp. 7–29.