June 1973

PLANNING IN A HIERARCHY OF ABSTRACTION SPACES

by

Earl D. Sacerdoti

Proceedings International Joint Conference on Artificial Intelligence (to appear August 1973).

Artificial Intelligence Center

Technical Note 78

SRI Project 1530

The work reported herein was sponsored by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-72-C-0008 with the U.S. Army Research Office.

PLANNING IN A HIERARCHY OF ABSTRACTION SPACES*

by
Earl D. Sacerdoti
Stanford Research Institute
Artificial Intelligence Center
Menlo Park, California 94025

Abstract

A problem domain can be represented as a hierarchy of abstraction spaces in which successively finer levels of detail are introduced. The problem solver ABSTRIPS, a modification of STRIPS, can define an abstraction space hierarchy from the STRIPS representation of a problem domain, and it can utilize the hierarchy in solving problems. Examples of the system's performance are presented that demonstrate the significant increases in problem-solving power that this approach provides. Then some further implications of the hierarchical planning approach are explored.

Key Words: Problem solving, heuristic search, representation, abstraction space, robot planning, hierarchical planning.

I Introduction

General purpose problem solvers, such as STRIPS[1,2†] or GPS,[3] must do their work using general purpose search heuristics. Unfortunately, by using such heuristics, it is not possible to solve any reasonably complex set of problems in a reasonably complex domain. Regardless of how good such heuristics are at directing search, attempts to traverse a complex problem space can be caught in a combinatorial quagmire.

This paper presents an approach to augmenting the power of the heuristic search process. The essence of this approach is to utilize a means for discriminating between important information and details in the problem space. By planning in a hierarchy of abstraction spaces in which successive levels of detail are introduced, significant increases in problem-solving power have been achieved.

Section II sketches the hierarchical planning approach and gives motivation for its use. Sections III and IV describe the definition and use of abstraction spaces by ABSTRIPS (Abstraction-Based STRIPS), a modification of the STRIPS problem-solving system that incorporates this approach. Section V describes the performance of the system. Section VI discusses the implications of this approach for problem solving and for robotics.

II The Motivation for Using Abstraction Spaces in Problem Solving

It was not quite fair to assert in the previous section that a complex problem domain is beyond the combinatorial capability of general purpose problem solvers. A problem solver deals not with the problem domain itself, but with some representation of that domain. So it would be more correct to state that a complex representation exceeds the scope of general purpose problem solvers.

Unfortunately, a straightforward transcription of a complex problem domain will yield a complex representation. However, a well-chosen transcription can lead to a simpler representation. By choosing such a simplifying representation, one can have the problem solver do its work in a context that is simple enough for some useful problem solving to take place.

In other words, the heuristic search through the simplifying representation will be of sufficiently short duration that a goal state in the problem space can be reached. Such a representation displays what McCarthy and Hayes[4] term "heuristic adequacy."

Attempts to achieve simplifying representations, such as the "macro operator," or MACROP, of the STRIPS problem solver,[2] have heretofore tried to preserve, in McCarthy and Hayes' terminology, "epistemological adequacy"; that is, the simplifying representations had to preserve all the detail that was needed to solve the problem at hand. A MACROP simplifies the representation of a problem domain by providing a means of selecting at one time an entire sequence of primitive operators, linked in a semantically sensible manner. But it preserves every detail of the preconditions and effects of its constituent operators.

Such simplifying representations can provide only limited enhancement to the power of a problem-solving system because of a somewhat dismaying fact: For a sufficiently complex problem domain, no epistemologically adequate representation can be heuristically adequate.

Epistemological adequacy implies that every relevant detail is properly dealt with. But attention to detail is precisely what defeats heuristic adequacy. A good heuristic evaluation function will enable a problem solver to reject most of the possible paths in a situation space. But if all the details are attended to, the evaluation function must be applied at all the nodes at which the details are affected. The combinatorics of the expanding search space will enable the problem solver to solve only rather simple problems.

A superior approach to problem solving would be to search first through an abstraction space, a simplifying

* The work reported herein was sponsored by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-72-C-0008 with the U.S. Army Research Office.

† References are listed at the end of this paper.

representation of the problem space in which unimportant details are ignored. When a solution to the problem in the abstraction space is discovered, all that remains is to account for the details of the linkup between the steps of the solution. This can be regarded as a sequence of subproblems in the original problem space. If they can be solved, a solution to the overall problem will have been achieved. If they cannot be solved, more planning in the abstraction space is required to discover an alternative solution.

Polya[5] cites the importance of this approach for human problem solving. It has been used by computer programs to find proofs in symbolic logic[6] (ignoring the nature of the connectives and the ordering of symbols as details) and to detect edges in scenes[7] (using a shrunken picture with less detail).

The concept can readily be extended to a hierarchy of spaces, each dealing with fewer details than the ground space below it and with more details than the abstraction space above it. By considering details only when a successful plan in a higher level space gives strong evidence of their importance, a heuristic search process will investigate a greatly reduced portion of the search space.

The process of abstraction defined in Section III is general in that it is not domain-dependent. But it is highly structured and very dependent on the syntax of the problem domain. It is a first step, providing no capability for a "representational shift" that would restate a difficult problem in terms that render its solution markedly easier. Rather, it employs a series of representational nudges that increase the power of the heuristic search process over a problem space.

III Automated Definition of Abstraction Spaces

The following sections describe the ABSTRIPS system, a modification of the STRIPS problem solver.[1,2] A brief description of the aspects of STRIPS that are relevant to the discussion to follow is presented below.* The reader is encouraged to see Section II of Ref. 2 for a brief but thorough summary of the operation of STRIPS or Ref. 1 for a full description.

Briefly, the representation of a problem domain with which STRIPS deals consists of:

(1) A World Model--The world model is a set of wffs in the predicate calculus, describing facts (e.g., CONNECTS(DOOR1,ROOM1,ROOM2)) or laws (e.g., ∀Rx,Ry,Dx CONNECTS(Dx,Rx,Ry) ⇔ CONNECTS(Dx,Ry,Rx)) of the problem domain.

(2) A Set of Operator Descriptions--Each action in the problem domain is represented by an "operator" for changing one model into another. An operator is defined by a precondition wff, an add list, and a delete list. For an operator to be applicable in a given model, its precondition wff must be satisfied. The add and delete lists describe which wffs are changed when an application of the operator transforms the world model.

A problem is stated to STRIPS as a goal wff. STRIPS must develop a sequence of operator applications that will lead to a world model in which the goal wff is true. A GPS-like means-ends analysis strategy[3] is employed to generate the operator sequence.

A "difference" between the initial model and the goal model is extracted. STRIPS determines which instances of which operators would reduce the difference; the instance that most reduces the difference is selected. If it is applicable in the initial state (i.e., its precondition wff is true in the initial world model), the operator is applied, and a new world model created. If the goal wff is true in the new model, STRIPS is done. If not, the difference between the new state and the goal state is extracted, and the process continues.

If the operator instance that most reduced the difference is not applicable in the initial state (i.e., its precondition wff is not provable in the world model), the precondition is set up as a subgoal wff. STRIPS will then try to develop a sequence of operator applications that will lead to a world model in which the subgoal wff is true. If the subgoal is achieved, the operator instance can be applied as before. If not, another operator instance is selected, and the process continues as before.

Abstraction Spaces in the STRIPS Context

For a practical problem-solving system, one would like to have an abstraction space differ from its ground space enough to achieve a significant improvement in problem-solving efficiency, but yet not so much as to make the mapping from abstraction space to ground space complex and time-consuming.

For the STRIPS system, this criterion is met by having the abstraction spaces differ from their ground spaces only in the level of detail used to specify the preconditions of operators. Although the change in representation provided by this choice may seem intuitively insufficient, it satisfies the criterion well. The world model can remain unchanged; there is no need to delete unimportant details from it because they can simply be ignored. No operators need be deleted in their entirety; if all they do is achieve details, they will never be selected as relevant. Any change to the add or delete lists of the operators would cause the operators' effects to be very different in different spaces. Since the applicability of a particular operator at some intermediate state might depend on any effects of any previously applied operators, the mapping of plans among spaces would be rendered too complex.

---

*In the interests of brevity and clarity, no further mention will be made of the MACROPs in the STRIPS system. A MACROP is the result of generalizing a previously completed plan. Most of its valid subsequences of operators can be extracted for use in further planning. Each such subsequence could be treated by ABSTRIPS like a primitive operator.

Thus, an abstraction space in the STRIPS context differs from its ground space only in the preconditions of its operators. The precondition wffs in an abstraction spaee will have fewer literals than those in its ground space. The literals omitted will be those that are "details" in the sense that a simple plan can be found to achieve them once the more "critical" literals have been achieved. For instance, consider a PUSHTHRUDR operator, whieh describes the effects of a robot pushing a particular object through a doorway into an adjacent room. In a high level abstraction space, the operator would be applicable whenever the object was pushable and a doorway into the desired room existed. In a lower level space, it would also be required that the robot and the object be in the room connected by the doorway with the target room. In a still lower abstraction space, the door would also have to be open. Finally, in the original representation of the problem space, the robot would also have to be next to the box, and the box would have to be next to the door.

For ABSTRIPS to be able to discriminate among various levels of detail, each literal within the preeonditions of each operator in a problem domain is assigned a "criticality" value at the time the domain is first defined. Only the most eritical literals will be in the highest abstraction space, whereas in lower spaces less critical ones will also appear.

Assigning Criticality to the Literals of a Precondition

There are many possible approaches to the assignment of critieality values to the literals of an operator's precondition wff. They span a range from a manual assignment as part of the specification of the problem domain to a completely automatic assignment of criticalities.

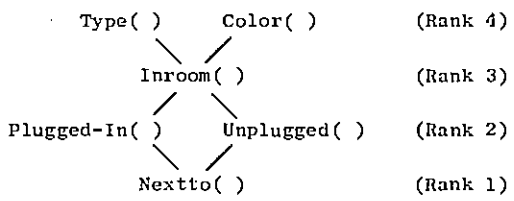
At one extreme, the definition of a problem domain could include an explicit specification of criticalities, reflecting the definer's intuition about the domain. For example, if one were to define a "Turn on tbe lamp(ℓ)" operator, he might say it was essential that ℓ be a lamp. He might say it was very important to be in the room witb the lamp, less important that the lamp's cord be plugged in, and still less important to be next to the lamp. Specifying the criticality value of a literal by a number preceding it in braces, one might define the precondition wff of tbe "Turn on the lamp" operator as

$$\{4\}\text{Type}(\ell,\text{lamp}) \wedge \exists rx(\{3\}\text{Inroom}(\text{Me},rx) \wedge \{3\}\text{Inroom}(\ell,rx)) \wedge \{2\}\text{Plugged-in}(\ell) \wedge \{1\}\text{Nextto}(\text{Me},\ell).$$

At the other extreme, a scheme can be developed to perform an exhaustive analysis of the n! possible orderings of the n literals in a precondition in order to determine which literals can be achieved once other literals are assumed to be true. The results of this analysis can be used to specify the criticality values for literals of the precondition.

For ABSTRIPS, an intermediate approach to criticality assignment was adopted. A predetermined (partial) ordering of all the predicates used in deseribing the problem domain was used to specify an order for examining the literals of the precondition wffs of all the operators in the domain. First, all literals whose truth value could not be changed by any operator in the domain were assigned a maximum criticality value. Then, each remaining literal was examined in an order determined by the partial ordering. If a short plan could be found to achieve a literal from a state in whieh all previously processed literals were assumed to be true, then the literal in question was said to be a detail and was assigned a criticality equal to its rank in the partial ordering. If no such plan could be found, the literal was assigned a criticality greater than the highest rank in the partial order.

For the domain including the "Turn on the lamp(ℓ)" operator, the partial ordering might look like the following:



The Type(ℓ,Lamp) literal could not be changed by any operator in the domain, and so it would be assigned a maximum criticality (6, in this case). The two Inroom literals would be examined next (an arbitrary order can be chosen for literals whose predicates have equal rank in the partial ordering). They cannot be achieved from a state in wbich Type(ℓ,Lamp) is asserted, and so they would be assigned a eritieality greater than the highest rank in the partial order, in this ease 5. Plugged-in (ℓ) can be achieved from a state in which the Inroom literals and the Type literal are true. It can be achieved by a plan to go to the lamp cord, pick it up, bring it to a socket, and plug it in. So it would be assigned a criticality equal to its rank in the partial ordering, namely, 2. Similarly, a plan can be found to achieve Nextto(Me,ℓ) from a state in which the previously processed literals are true, and so it would be assigned a criticality of 1.

Regardless of the method used to determine the criticality values, they define a hierarchy of abstraction spaces. The next section shows how such a hierarchy can be used to aid the planning process.

IV Utilization of Abstraction Spaces in Planning

To take advantage of the hierarchical planning approach offered by the use of abstraction spaces, the ABSTRIPS system--whose flow of control is shown in Figure 1--has a recursive executive program. This program accepts two parameters. The first is a criticality value indicating the abstraction space in which planning is to occur. The second is a list of nodes from the search tree in the higher space, representing a skeleton

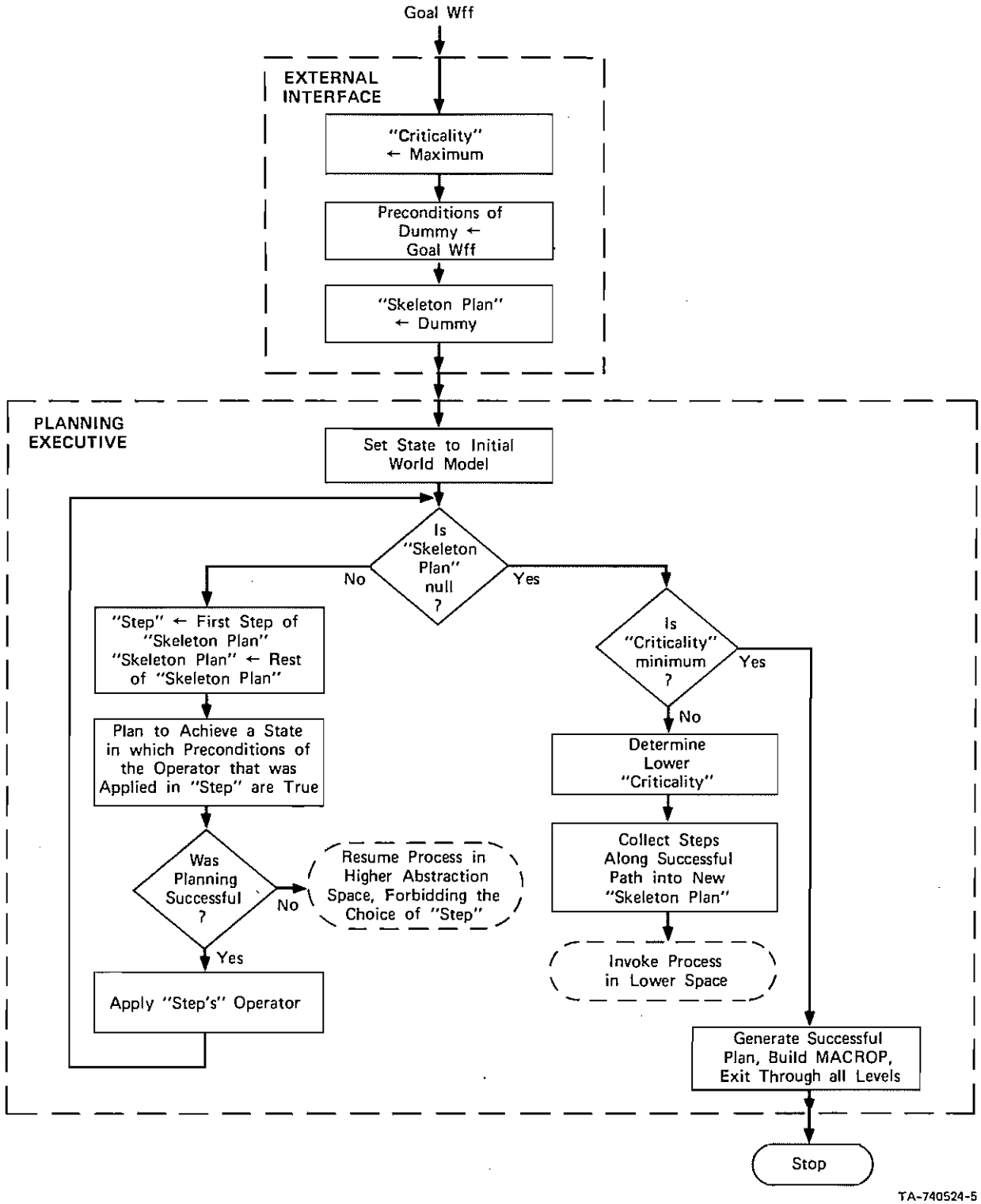


FIGURE 1 FLOW OF CONTROL OF ABSTRIPS

plan. When a new problem is posed to ABSTRIPS, the external interface program sets the preconditions of a dummy operator to the goal wff. The domain's maximum criticality, which was determined when criticalities were assigned, is retrieved. The executive is called with the criticality set to the maximum and the skeleton consisting of the dummy operator.

Within the highest abstraction space, the executive plans to achieve the preconditions of the dummy step in the skeleton plan, i.e., the main goal. When a plan is found, the executive computes the criticality of the next lowest space in which planning is needed, and it builds a skeleton of nodes along the path of the successful plan. The executive then invokes itself recursively. The new invocation solves in turn the subproblems of bridging the gaps between steps in the skeleton plan and of ensuring that the steps in the skeleton plan are still applicable at the appropriate points in the new plan. The final step in the skeleton is always the dummy operator, and so the final applicability check ensures that the original goal has been reached. When all subproblems have been solved, the executive invokes itself for planning in a still lower space. This recursion continues until a complete plan is built up in the problem space itself.

This search strategy might be termed a "length-first" search. It pushes the planning process in each abstraction space all the way to the original goal state before beginning to plan in a lower space. This enables the system to recognize as early as possible the steps that would lead to dead ends or very inefficient plans.

If any subproblem in a particular space cannot be solved, control is returned to the process in its abstraction space. The search tree is restored to its state prior to the selection of the node that led to failure in the ground space. That node is eliminated from consideration, and the search for a successful plan at the higher level continues.

This failure mechanism is analogous to the automatic backtracking feature of the PLANNER language.[5] It has the major defect that when a failure of a lower level process is reported, the process and the context in which the failure occurred are no longer around for analysis. So ABSTRIPS relies heavily on being able to produce good plans at the highest level.

This requirement has led to two modifications to the search algorithm originally employed by STRIPS. The first is an alteration of the evaluation function used to select which node in the search tree to expand next. STRIPS emphasizes the estimated cost of achieving the goal from the given node and deemphasizes the cost of arriving at the node from the initial state. Thus, it has a tendency to find a slightly longer plan quickly, rather than the cheapest plan more slowly. But each extra step in a high abstraction space is likely to lead to many extra steps in the corresponding plan in the problem space. Thus, for ABSTRIPS, the evaluation function has itself been made a function of the level of abstraction. At the highest level, ABSTRIPS gives equal weight to the cost of reaching a given node and to the estimated cost of reaching the goal from that node. This evaluation function changes incrementally as the level of abstraction decreases, until it reaches the old STRIPS function at the level of the problem space.

The second modification involves postponing the selection of one among several equivalent instances of a relevant operator. During the process of selecting relevant operators to reduce a particular difference, a partial instantiation of the operators' parameters may occur. For example, if the difference were that the robot was not in Room 3, then the operator "Go through a door into a room" might be selected and instantiated to "Go through a door into Room 3." The preconditions of this operator would then be analyzed by the theorem prover to determine which door to choose. If several choices seem equally good to STRIPS (i.e., the states in which the various choices can be applied are equally difficult to reach), then it would arbitrarily pick a door.

For ABSTRIPS, alternative instantiations in an abstraction space might appear equivalent, and yet one choice might be substantially superior when further details are considered. So ABSTRIPS defers its decision when more than one equivalent "best choice" of a relevant operator is found. The partially instantiated relevant operator (e.g., "Go through a door into Room 3") is used in planning. When subsequent analysis in a lower abstraction space reveals a preferred instantiation, that instantiation is then chosen. If this selection should eventually lead to failure, the other instantiations can still be chosen through the backtracking mechanism.

In summary, hierarchical planning using abstraction spaces in a "length-first" search technique postpones extending the search tree through the levels concerned with the detailed preconditions of an operator until it knows that doing so will be highly effectual in reaching the goal (because the operator lies along an almost certainly successful path). By avoiding work on fruitless branches of the search tree, the technique achieves significant efficiencies in the formulation of complex plans.

V Examples of ABSTRIPS' Performance

To clarify the issues raised and the way in which the ABSTRIPS system works, the system's performance is traced through some examples below. The ABSTRIPS system consists of some 370 BBN-LISP functions, which run as compiled code on a PDP-10 computer. All the examples presented were drawn from the environment of the Stanford Research Institute mobile robot. The domain consists of seven rooms interconnected by doorways. Operators have been defined that model the robot's ability to navigate to any object or location within a room, to push boxes within a room or through a doorway, to navigate through a doorway, to block a doorway using a box, and to unblock a doorway. In addition, fictitious operators have been defined to model the opening and closing of doors; these actions are beyond the robot's capabilities. In all, 167 predicate calculus wffs have been defined as axioms to model the robot domain.

The definition of the domain is essentially identical to the one used for the examples in the latest report on the STRIPS system.[2]

Definition of Abstraction Spaces

To enable the system to assign criticality values properly to the literals of the preconditions of the operators, two additional axioms, representing laws about the world, were included in the world model:

$$\forall x.\ \text{PUSHABLE}(x) \supset \text{TYPE}(x,\text{OBJECT})$$

and

$$\forall x.\ \text{STATUS}(x,\text{CLOSED}) \equiv \neg\text{STATUS}(x,\text{OPEN})$$

The criticality determination algorithm required approximately five minutes of running time. The resulting operator descriptions are listed below. The number in braces preceding each literal in the precondition wffs represents the criticality of the literal. The literal will appear in the precondition in abstraction spaces of criticality less than or equal to the number in braces.

GOTOB(bx) Go to object bx.
Preconditions: {6}TYPE(bx,OBJECT),
(Erx)[{5}INROOM(bx,rx) ∧
{5}INROOM(ROBOT,rx)]
Deletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1)
Additions: *NEXTTO(ROBOT,bx)

GOTOD(dx) Go to door dx.
Preconditions: {6}TYPE(dx,DOOR),(Erx)(Ery)
[{5}INROOM(ROBOT,rx) ∧
{6}CONNECTS(dx,rx,ry)]
Deletions: At(HOBOT,S1,S2),NEXTTO(ROBOT,S1)
Additions: *NEXTTO(ROBOT,dx)

GOTOL(x,y) Go to coordinate location (x,y)
Preconditions: (Erx)[{5}INROOM(ROBOT,rx) ∧
{6}LOCINROOM(x,y,rx)]
Beletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1)
Additions: *At(ROBOT,x,y)

PUSHB(bx,by) Push bx to object by
Preconditions: {6}TYPE(by,OBJECT),{6}PUSHABLE(bx),
{1}NEXTTO(ROBOT,bx),
(Erx)[{5}INROOM(bx,rx) ∧
{5}INROOM(by,rx) ∧ {5}INROOM(ROBOT,rx)]
Deletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1),
At(bx,S1,S2),NEXTTO(bx,S1),NEXTTO(S1,bx)
Additions: *NEXTTO(by,bx),*NEXTTO(bx,by),
NEXTTO(ROBOT,bx)

PUSHD(bx,dx) Push bx to door dx
Preconditions: {6}PUSHABLE(bx),{6}TYPE(dx,Door),
{1}NEXTTO(ROBOT,bx),
(Erx)(Ery)[{5}INROOM(ROBOT,rx) ∧
{5}INROOM(bx,rx) ∧ {6}CONNECTS(dx,rx,ry)]
Deletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1),
At(bx,S1,S2),NEXTTO(bx,S1),NEXTTO(S1,bx)
Additions: *NEXTTO(bx,dx),NEXTTO(ROBOT,bx)

PUSHL(bx,x,y): Push bx to coordinate location (x,y)
Preconditions: {6}PUSHABLE(bx),{1}NEXTTO(ROBOT,bx),
(Erx)[{5}INROOM(ROBOT,rx) ∧
{5}INROOM(bx,rx) ∧ {6}LOCINROOM(x,y,rx)]
Deletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1)
At(bx,S1,S2),NEXTTO(bx,S1),NEXTTO(S1,bx)
Additions: *At(bx,x,y),NEXTTO(ROBOT,bx)

GOTHRUDR(dx,rx): Go through door dx into room rx
Preconditions: {6}TYPE(dx,DOOR),{6}TYPE(rx,ROOM),
{2}STATUS(dx,OPEN),
(Ery)[{5}INROOM(ROBOT,ry) ∧
{6}CONNECTS(dx,ry,rx)]
Deletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1),
INROOM(ROBOT,S1)
Additions: *INROOM(ROBOT,rx)

PUSHTHRUDR(bx,dx,rx): Push bx through door dx into room rx
Preconditions: {6}PUSHABLE(bx),{6}TYPE(dx,DOOR),
{6}TYPE(rx,ROOM),{2}STATUS(dx,OPEN),
{1}NEXTTO(bx,dx),{1}NEXTTO(ROBOT,bx),
(Ery)[{5}INROOM(bx,ry) ∧
{5}INROOM(ROBOT,ry) ∧
{6}CONNECTS(dx,ry,rx)]
Deletions: At(ROBOT,S1,S2),NEXTTO(ROBOT,S1),
At(bx,S1,S2),NEXTTO(bx,S1),NEXTTO(S1,bx),
INROOM(ROBOT,S1),INROOM(bx,S1)
Additions: *INROOM(bx,rx),INROOM(ROBOT,rx),
NEXTTO(ROBOT,bx)

OPEN(dx): Open door dx
Preconditions: {6}TYPE(dx,DOOR),{5}STATUS(dx,CLOSED),
{5}NEXTTO(ROBOT,dx)
Deletions: STATUS(dx,CLOSED)
Additions: *STATUS(dx,OPEN)

CLOSE(dx): Close door dx
Preconditions: {6}TYPE(dx,DOOR),{5}STATUS(dx,OPEN),
{5}NEXTTO(ROBOT,dx)
Deletions: STATUS(dx,OPEN)
Additions: *STATUS(dx,CLOSED)

A Detailed Sample Problem

Figure 2 depicts the initial model that was defined for this problem. The robot is in Room RRIL. The door between RRIL and RCLK is closed. BOX1 and BOX2 are both in RPDP. The problem is for the system to plan to achieve a state in which the two boxes are next to one another and the robot is in Room RUNI, as in Figure 3. The goal wff for this problem is: NEXTTO(BOX1,BOX2) ∧ INROOM(ROBOT,RUNI).

STRIPS was able to solve this problem without using abstraction spaces. However, its solution required the exploration of 119 nodes in the search tree, only 23 of which were on the successful path. This exploration took over 30 minutes of computer time. Figure 4(a) depicts the search tree.

ABSTRIPS first examined the problem in an abstraction space in which the only precondition clauses considered were those whose truth value could never be altered by the robot. The difference between the initial state and the goal state was computed. The difference was the goal wff itself. Five relevant operator

---

* The addition clauses preceded by an asterisk are the primary additions of the operator. When STRIPS or ABSTRIPS searches for a relevant operator, it considers only primary addition clauses.

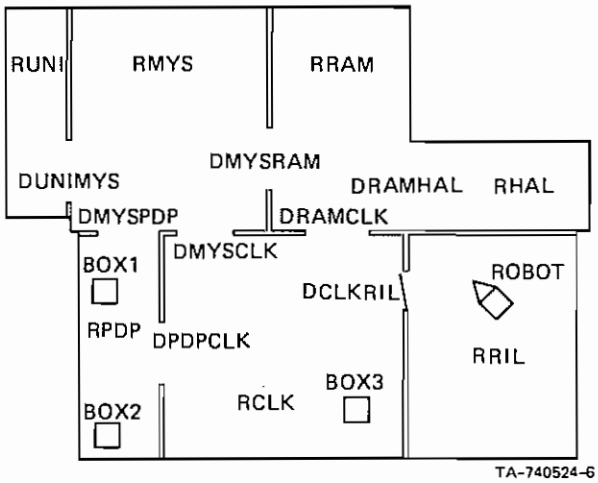


FIGURE 2 INITIAL STATE FOR THE SAMPLE PROBLEM

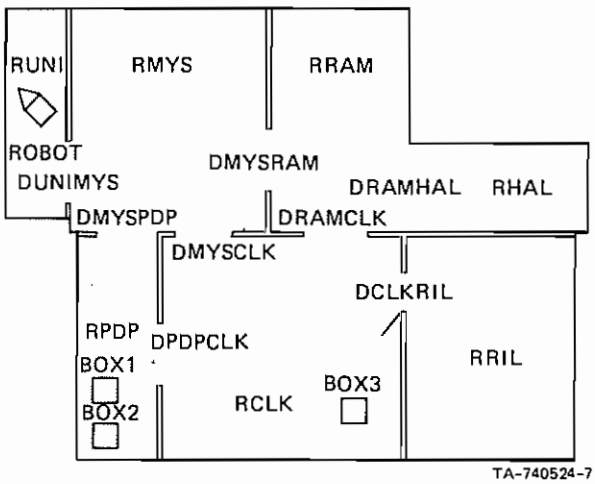


FIGURE 3 A STATE IN WHICH THE GOAL OF THE SAMPLE PROBLEM IS SATISFIED

instances were computed. The first of these, PUSHB (BOX2,BOX1), was examined. Its precondition wff in this abstraction space was true in the initial state; so the operator was applied. This resulted in a new state in which the robot, BOX1, and BOX2 were next to each other. The difference between this state and the goal state was computed and found to be INROOM(ROBOT, RUNI). Two relevant operator instances were found, and the first, GOTHRUDR(Parl2,RUNI), was examined. (Parl2 is an uninstantiated parameter.) Its precondition wff in this abstraction space, TYPE(RUNI,ROOM) ∧ TYPE (Parl2,DOOR) ∧ (Ery)CONNECTS(Parl2,ry,RUNI), was satisfied when Parl2 was instantiated to DUNIMYS. So GUTHRUDR(DUNIMYS,RUNI) was applied, and this generated a state in which the goal wff was true. Figure 4(b) depicts the search tree in the highest abstraction space. The positioning of the nodes suggests the correspondence to the nodes in the STRIPS search tree.

A skeleton plan was built consisting of the nodes at which the two operators were applied. The plan was:

PUSHB(BOX2,BOX1); GOTHRUDR(DUNIMYS,RUNI) .

Planning then began in the space of criticality 5.

The first subgoal was the precondition wff in this abstraction space of the first operator, PUSHB(BOX1, BOX2). The difference between the initial state and the one in which the wff was true was INROOM(ROBOT, RPDP). Operator instances relevant to reducing this difference were GOTHRUDR(Par17,RPDP) and PUSHTHRUDR (ROBOT,Par20,RPDP). The precondition wff of the first was tested, but it was not completely satisfied. There were still differences INROOM(ROBOT,RMYS) or INROOM (ROBOT,RCLK) before GOTHRUDR(Parl7,RPDP) could be applied (i.e., the robot was not yet in a room adjoining RPDP). The PUSHTHRUDR operator was completely inapplicable because the robot is not a pushable object.

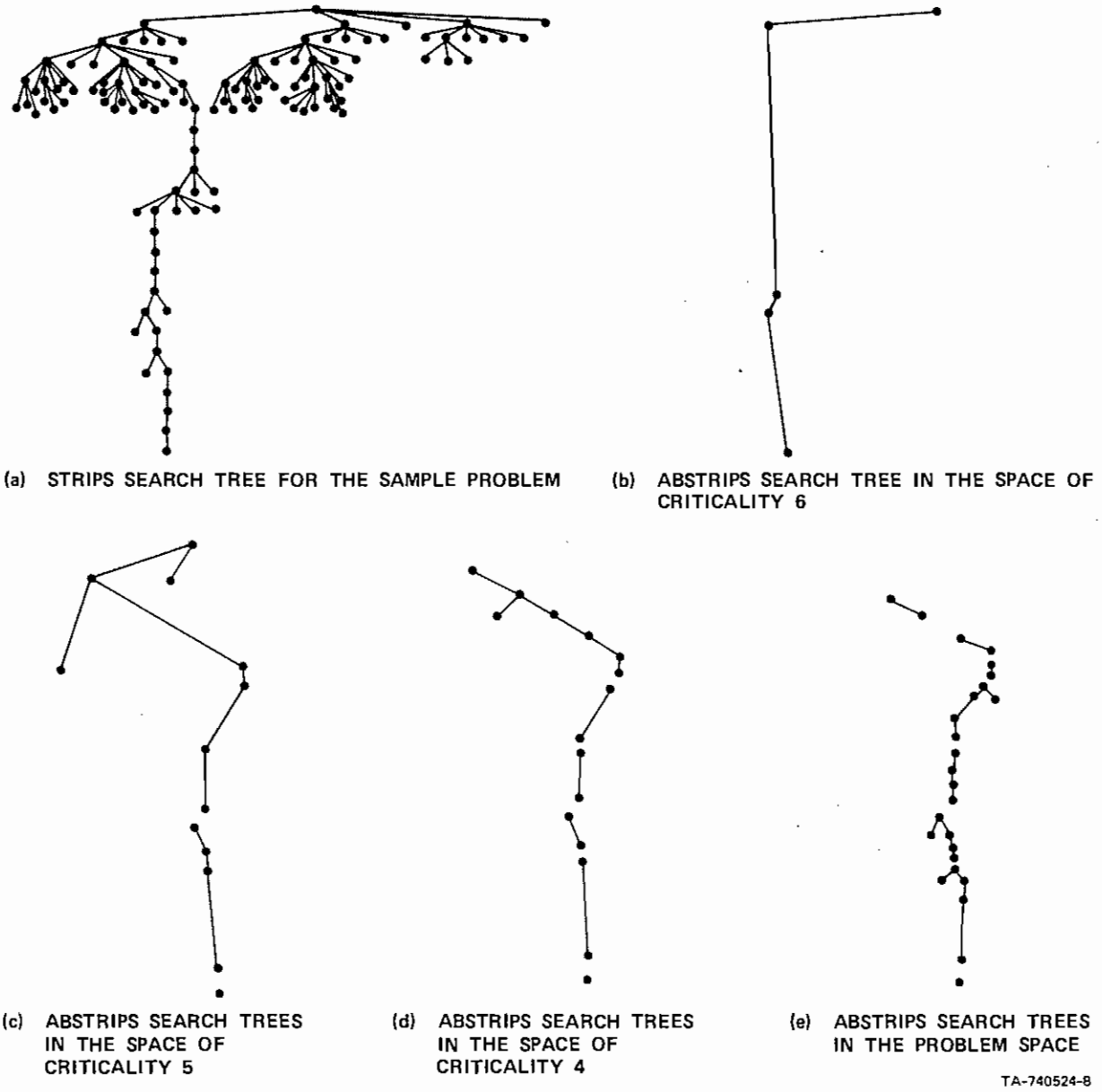
Then ABSTRIPS tried to reduce the differences that would render GOTHRUDR(Parl7,RPDP) applicable. Four relevant operators were found. The first was GOTHRUDR (Par22,RMYS), and its precondition wff was not satisfied either (the robot was not in a room adjoining RMYS). The second relevant operator was GOTHRUDR(Par22,RCLK), and its precondition wff was satisfied when Par22 was instantiated to DCLKRIL. So GOTHRUDR(DCLKRIL,RCLK) was applied, producing a state in which GOTHRUDR(DPDPCLK, RPDP) was applicable. That operator was applied, producing a state in which the initial subgoal, the precondition wff of PUSHB(BOX2,BOX1), was true. The PUSHB operator was then applied.

Then a new subgoal was set up, in which the preconditions of GOTHRUDR(DUNIMYS,RUNI) in this space were true. The difference between the current state and the subgoal state was INROOM(ROBOT,RMYS). GOTHRUDR(Par27, RMYS) was selected as a relevant operator, and its preconditions were satisfied when Par27 was bound to DMYSPDP. So GOTHRUDR(DMYSPDP,RMYS) was applied, producing a state in which the subgoal was satisfied. The operator associated with this subgoal, GOTHRUDR(DUNIMYS, RUNI), was applied, and the goal state was again reached. Figure 4(c) shows the search trees in this space.

The following new skeleton plan was built up: GOTHRUDR(DCLKRIL,RCLK); GOTHRUDR(DPDPCLK,RPDP); PUSHB (BOX2,BOX1); GOTHRUDR(DMYSPDP,RMYS);GOTHRUDR(DUNIMYS, RUNI). The planning process was then reinvoked in an abstraction space of criticality 2.

The first subgoal, the precondition wff of the first step in the skeleton plan, GOTHRUDR(DCLKRIL,RCLK), was not satisfied in the initial model. The difference was STATUS(DCLKRIL,OPEN). An analysis showed that it could be eliminated by applying GOTOD(DCLKRIL) and then OPEN (DCLKRIL). This resulted in a state that satisfied the first subgoal. So GOTHRUDR(DCLKRIL,RCLK) was applied.

Each of the remaining subgoals of the process in this abstraction space were immediately satisfiable, and so each step of the skeleton plan was applied in turn, resulting in a state in which the original goal was satisfied. The skeleton plan produced was GOTOD (DCLKRIL); OPEN(DCLKRIL), followed by all the steps of the previous skeleton plan. Figure 4(d) shows the search trees in this space.



(a) STRIPS SEARCH TREE FOR THE SAMPLE PROBLEM

(b) ABSTRIPS SEARCH TREE IN THE SPACE OF CRITICALITY 6

(c) ABSTRIPS SEARCH TREES IN THE SPACE OF CRITICALITY 5

(d) ABSTRIPS SEARCH TREES IN THE SPACE OF CRITICALITY 4

(e) ABSTRIPS SEARCH TREES IN THE PROBLEM SPACE

TA-740524-8

FIGURE 4 SEARCH TREES FOR THE SAMPLE PROBLEM

Finally, planning took place in the ground space, the space including literals of criticality 1. The first three steps of the skeleton plan were applied in turn. But the preconditions of GOTHRUDR(DPDPCLK,RPDP) were not satisfiable in a state in which the robot had just come through DCLKRIL. The difference was NEXTTO (ROBOT,DPDPCLK), and analysis indicated that it could be eliminated by applying GOTOD(DPDPCLK), enabling GOTHRUDR(DPDPCLK,RPDP) to be applied.

The next subgoal, the preconditions of PUSHB(BOX2, BOX1), was not satisfied at this point. The difference was NEXTTO(ROBOT,BOX2), which could be eliminated by an application of the first relevant operator selected, GOTOB(BOX2). After PUSHB(BOX2,BOX1) was applied, the next two subgoals failed because the robot was not next to the appropriate door. An analysis similar to the one that occurred with DPDPCLK was performed, enabling ABSTRIPS to finish the plan with an operator to go to and an operator to go through DMYSPDP and DUNIMYS.

Note that the planning in this space is just as if STRIPS were given seven small problems to solve consecutively, without the benefit of MACROPS. The search trees for the ground space are shown in Figure 4(e). The entire planning process for ABSTRIPS produced 60 nodes, 54 of which were on the successful path in one space or another. This process required 5:28 of computer time. This is less than one-fifth of the time required by the nonhierarchical STRIPS.

Other Examples

The set of tasks from the most recent report on STRIPS[2] was run on ABSTRIPS. The running times and the search trees are compared with those from the STRIPS system in Table 1. Figure 5 plots the planning time as a function of plan length for STRIPS and ABSTRIPS on an extended set of problems from the robot domain.

Table 1

COMPARISON OF PLANNING TIMES AND SEARCH TREES

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
ABSTRIPS					
Time to find plan (minutes)	1:54	2:55	2:24	2:30	6:41
Total nodes in search trees	25	34	30	33	63
--by spaces*	5,5,5,10	5,7,7,15	3,4,11,12	5,7,7,14	5,17,16,25
Nodes on solution path	24	32	28	32	54
--by spaces*	5,5,5,9	5,7,7,13	3,4,10,11	5,7,7,13	5,11,15,23
Operators in plan	4	6	5	6	11
STRIPS					
Time to find plan (minutes)	1:40	5:44	4:34	9:47	>20:00†
Total nodes in search tree	10	33	22	51	--
Nodes on solution path	9	13	11	15	--
Operators in plan	4	6	5	7	--
STRIPS with MACROPs					
Time to find plan (minutes)	1:40	2:06	5:18	3:00	5:49
Total nodes in search tree	10	9	14	9	14
Nodes on solution path	9	9	9	9	14
Operators in plan	4	6	5	6	11

*The number of nodes from the search tree in each space, from the one of highest criticality to the problem space itself.

†STRIPS had not solved Problem 5 after 20 minutes.

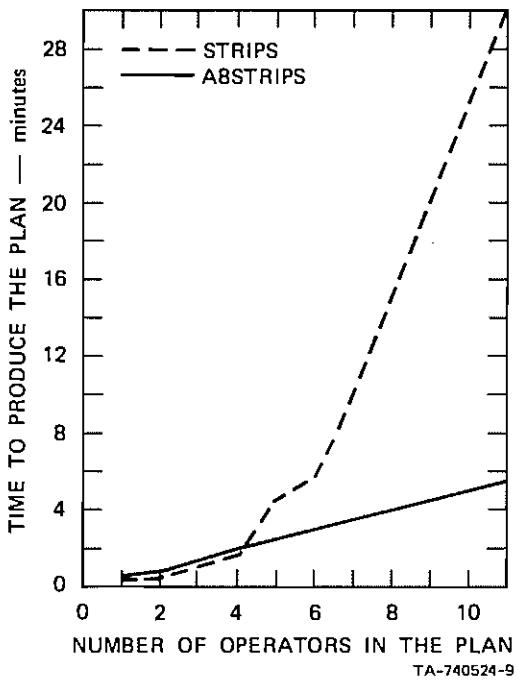


FIGURE 5 PLANNING TIME AS A FUNCTION OF PLAN LENGTH

VI Further Implications of the Use of Abstraction Spaces in Planning

This paper has shown how the representation of a problem domain as a hierarchy of abstraction spaces dramatically improved the performance of a problem solver. This section briefly considers the implications of such a hierarchical representation for some other problem areas in robotics and problem solving.

Learning Task-Specific Knowledge

General-purpose problem solvers have tended to be weak problem solvers. Because the heuristics they use to guide the search through the problem space must be generally applicable, they are not especially powerful in any particular task domain. On the other hand, special purpose programs to solve problems in a particular domain have been notably successful. The HEURISTIC DENDRAL program[9] and the game playing programs display far more problem-solving power in their particular domains of competence than a general purpose problem solver could muster. This competence is derived to a large degree from the great amount of task-specific knowledge that has been incorporated into their search heuristics.

Unfortunately, while these special purpose programs display intelligent behavior within their limited domain, they are worth little in any other domain. Can a more generally intelligent system be constructed that, when presented with task-specific knowledge (basic to which

is the description of the problem domain), can incorporate that knowledge into its search heuristics?

The process of automated definition of abstraction space offers a possible approach. By applying a general purpose problem solver to a particular domain in the most general manner described in Section III, a task-specific detail hierarchy can be built up. The ability of a system to discriminate important considerations from mere details is an important aspect of task-specific knowledge.

A further aspect of task-specific knowledge is the facility for negotiating those areas of the search space that are easily traversible. In the hierarchical representation framework, easily traversihle areas correspond to subproblems of achieving details, once the more critical aspects of a prohlem have been solved.

The ABSTRIPS system determines that a given literal is a detail when it has built a small plan to achieve a state in which it is true. That small plan can be saved as a MACROP, to be used as the first-choice relevent operator whenever the detail needs to be achieved. The relatively small number of MACROPs formed in this way, when added to the set of basic operators, constitute a basic body of knowledge about how to solve prohlems in a particular task domain.

Planning with Multiple Outcome Operators

The use of a hierarchical representation can greatly simplify the process of creating conditional plans, plans with information gathering operators, and plans with loops. This is because the outcomes of these operators are uncertain only to a particular level of detail. Thus, in a higher abstraction space a simple specification can adequately model the preconditions and effects of the operators, although some of the effects may have to be described in terms of uninstantiated parameters. A drawback to this approach is that, as noted in Section III, the mapping of plans among spaces becomes difficult when the effects of operators are abstracted. Nevertheless, the simplicity of representation of these rather complex operators renders this scheme attractive.

As an example, in planning to drive to the airport to catch a plane, one would use a "Park the car" operator. Such an operator might have the effect of "If Lot A is not full, park inside Lot A. Else if Lot B is not full, park inside Lot B. Else drive around, and then park the car." If one plans at a high level of abstraction to drive to the airport, he does not consider the "Park the car" operator in its full complexity. Rather, he considers an image of the operator in an abstraction space in which no uncertainties exist. It might have the simple precondition At(Car,Airport) and might cause the clause Parked-in-lot (Car,Parameter37) to be added to the model. Further planning could continue without considering as separate cases states in which Parked-in-lot(Car,Lot A) or Parked-in-lot(Car,Lot B) were true.

An Integrated Robot System

A primary motivation for building the STRIPS system, and its offspring ABSTRIPS, was to build plans for a mobile robot. In the Stanford Research Institute robot system, the operator descriptions are models for actions that the robot can actually take. The actions modeled are termed "intermediate level actions" (ILAs). When they are executed, they invoke "low level actions" (LLAs), which are concerned with initiating and monitoring motion of the robot. These routines in turn pass commands to, and receive information from, a program in a PDP-15 computer, which communicates with the robot itself via a radio link.

The ground space as viewed by ABSTRIPS is in fact just another abstraction space from the point of view of plans built up from hasic operations at lower levels. The problem solver can be extended to handle successively finer levels of detail until a ground space is reached in which the only remaining details are to roll the robot around. This offers the enticing possibility of a fully integrated planning and exccution system. But the interaction of planning and execution would require that the plans that such a system built be differed from the traditional form of plan built by problcm solvers.

For a system that deals with complex prohlems in a rcal world, as opposed to a simulated one, it is undesirable to solve an entire problcm with an epistemologically adequate plan. There are too many reasonably likely outcomes for each real-world operation. The number of hypothetically possihle states of the world attainable by a particular plan will grow exponentially with the length of the plan. Most of the effort of such a system would be spent reasoning about world states that would never he achieved, and very little of it would be spent moving the robot toward its goals.

It is desired that the system's planning efforts focus on reasoning about states of the world that are likely to be traversed in thc course of robot execution. Thus, the overall planning should be roughed out in an abstraction space that ignores enough levels of detail so that the rough plan is fairly certain to succeed.

A few steps of the plan can be used as a skeleton, to which more detailed steps are added in a manner similar to ABSTRIPS. These new steps are fairly certain to succeed at the level of detail to which they are specified. Even more detailed steps can be fillcd in for the beginning portion of this subplan, and the process can continue until a short subplan of low-level rohot commands is built. These can be executed in sequence. Any deviations between the actual state of the world and the hypothesized results of the subplan will hopefully be mere details to the space that is an abstraction of the robot commands. Thus, the remaining steps of the plan in this spaee, as well as all higher spaces, are still on the solution path.

Further building and extending of the various subplans can then take place, including a new bottom-level subplan to move the robot. This subplan will accurately reflect the precise results of previous execution, and so it will be fully appropriate for achieving the ultimate goal. The process of alternatively adding detailed steps to the plan and then actually executing some steps can continue until the goal is achieved.

If a grievous failure occurs at some point in execution and nondetails in higher models no longer reflect the actual state of the world, subplans at affected levels of detail can propagate the failure up to an abstraction space in which the deviation from the predicted world model was a detail. Replanning can he initiated from this level of abstraction, thus reusing the results of as much as possible of the previous planning.

Therefore, by using a hierarchy of ahstraction spaces to mask uncertainties in the real world effects of planned operations, an effectively integrated robot planning and executing system can be created. By dealing with a hierarchy of short, simple plans, such a system will be able to cope effectively with truly complex problems.

Acknowledgements

The author is indebted to Richard Fikes, Peter Hart, and Nils Nilsson for their enthusiastic encouragement and intellectual support. The research reported in this paper was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-72-C-0008 with the U.S. Army Research Office.

References

1. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, Nos. 3/4, pp. 189-208 (1971).

2. R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol. 3, pp. 251-288 (1972).

3. G. Ernst and A. Newell, GPS: A Case Study in Generality and Problem Solving, ACM Monograph Series (Academic Press, New York, New York, 1969).

4. J. McCarthy and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," iu Machine Intelligence 4, B. Meltzer and D. Michie, eds., pp. 463-502 (American Elsevier Publishing Company, New York, New York, 1969.

5. G. Polya, How to Solve It, p. 8 (Princeton University Press, Princeton, New Jersey, 1945).

6. A. Newell, J. C. Shaw, and H. A. Simon, "Report on a General Problem Solving Program," Proceedings of the International Conference on Information Processing, UNESCO, Paris, pp. 256-264 (1960).

7. M. D. Kelly, "Edge Detection in Pictures by Computer Using Planning," in Machine Intelligence 6, B. Meltzer and D. Michie, eds., pp. 397-409 (American Elsevier Publishing Company, New York, New York, 1971).

8. C. Hewitt, "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," Ph.D. Thesis, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts (1972).

9. B. Buchanan, G. Sutherland, and E. Feigenbaum, "HEURISTIC DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry," in Machine Intelligence 4, B. Meltzer and D. Michie, eds., pp. 209-254 (American Elsevier Publishing Company, New York, New York, 1969.