RSD-TR-14-87

Subgoal Ordering and Goal Augmention for Heuristic Problem Solving

Keki B. Irani
Jie Cheng

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109

August 1987

Center for Research on Integrated Manufacturing

Robot Systems Division
College of Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2110

TABLE OF CONTENTS

1. Introduction 1

2. Previous Subgoal Ordering Strategies 2

3. A Robot Planning Problem 4

4. A Systematic Approach to Subgoal Ordering 7

5. Goal Augmentation 12

6. Integration of Subgoal Ordering & Heuristic Estimation 15

7. Example 18

8. Summary 23

References 23

Subgoal Ordering and Goal Augmentation for Heuristic Problem Solving

Keki B. Irani
Jie Cheng
Division of Computer Science and Engineering
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109, USA
Tel: (313)-764-8517
E-Mail: Keki_B_Irani@um.cc.umich.edu

ABSTRACT:

In order to improve the performance of heuristic search for finding optimal solutions, two high level problem solving strategies, namely, subgoal ordering and goal augmentation, have been developed. The purpose of these two strategies is to make explicit the knowledge embedded in a general problem formulation which can be used to constrain the solution search space. These two strategies have been incorporated into a methodology, which we previously developed for automatically generating admissible search heuristics. The effectiveness of these strategies are demonstrated by the application to robot optimal task planning problems.

paper type: full-paper

track/topic: engineering/reasoning, planning

subtopic: subgoal ordering; goal augmentation; automatic search heuristic generation; robot planning.

1. Introduction

We have now augmented our previously published([IrY85]) general heuristic problem solving methodology by two systematic problem solving strategies, namely, problem subgoal ordering and problem goal augmentation. This paper reports on these two strategies. These strategies, however, are not dependent on our previous work and may be incorporated into any other problem solving methodology. Our goal with these two strategies is to make explicit the knowledge embedded in a problem formulation which can be used to conduct a solution search effectively.

Problem subgoal ordering is an important strategy used to reduce problem space search. Many different subgoal ordering strategies have been reported in the past([Das77], [ErG82], [Sac77], [Sus75], [Tat75], [Wal81], [War74]). However, we present in this paper a novel approach for subgoal ordering. In contrast to the previously proposed approaches, our approach is to preorder the problem subgoals correctly and systematically by efficiently reasoning on the problem formulation. The result of ordering is then imposed on the search control to constrain the search space.

Goal augmentation is an approach to systematically discover the goal information which is not explicitly represented but which can be inferred from the given problem formulation. The augmentation reduces ambiguity and enables more accurate estimation of the search heuristic.

Our research on problem subgoal ordering and goal augmentation are both parts of the effort devoted to developing a general heuristic problem solving methodology. Previously, we achieved a method for systematically modeling problems and automatically generating admissible search heuristics for the A^*-like best-first search(see [IrY85], [IrY86]). Now, the subgoal ordering and the goal augmentation strategies have been

integrated with the search heuristic generation to constrain search by improving the tightness of the heuristic while still preserving its admissibility and monotonicity.

The rest of the paper is organized in the following way. In section 2, the earlier problem subgoal ordering strategies are reviewed. In section 3, a simple robot planning problem is introduced to illustrate the ideas. The subgoal ordering problem is discussed in section 4 while section 5 introduces the principles and procedures for goal augmentation. Section 6 reports the work on the integration of the aforementioned problem solving strategies with the search heuristics automatic generation. An example illustrating the power of the two stratgies is given in section 7. A summary is given in the last section.

2. Previous Subgoal Ordering Strategies

Subgoal ordering has been employed in many early problem solving systems. *GPS* [ErN69] was the earliest to apply the subgoal ordering strategy. Subgoals are arranged as row headings in the "Table of Connection". The system waits to achieve the subgoal heading at a certain row until all the subgoal headings at the higher rows are achieved. A total ordering is thus imposed on the problem subgoals. In ordering problem subgoals this way, one has to be sure that achieving a new subgoal does not violate all the previously achieved subgoals.

Originally, the subgoal headings were determined and ordered by the users. Ernst et al.[ErG82] later developed a procedure DGBS to mechanize this approach. Although DGBS correctly generates subgoal ordering for several problems such as the Fool's disk, Tower of Hanoi, etc., this approach cannot always guarantee correct ordering of the problem subgoals. For example, when the *superfluous operator constraint* is violated, and the table of connection is in a diagonal form, the system is not able to judge the correct

order of subgoals. Another problem with DGBS is that in constructing a table of connection, it must initially consider all the problem operator instances, instead of just the problem operator schemes as usually specified. This may cause DGBS to be computationally intractable when encountering a problem with small number of operator schemes but large number of operator instances.

Some systems, such as *HACKER*[Sus75], *INTERPLAN*[Tat75], *WARPLAN*[War74], order subgoals arbitrarily and then perform destructive reordering in the case that a protection constraint violation is detected. Waldinger[Wal81] suggests a goal regression strategy which amounts to a constructive subgoal reordering. All these strategies make over commitment to subgoal ordering.

In *NOAH*[Sac77], on the other hand, least commitment is made towards subgoal ordering. Subgoals are attempted in parallel unless sequential order constraints are imposed on them by the problem dependent knowledge in the SOUP code. Several critics are then applied to detect and handle interactions among subgoals or to find and eliminate redundancies.

Another approach is the combination of sequential and parallel methods. In this approach, a plan is expanded by choosing a subgoal to be achieved which least interferes with the existing partial solution[Das77].

The subgoal ordering strategies of these systems either make too strong an initial commitment to subgoal ordering without making use of the knowledge of the inter-subgoal constraints at all, or make a commitment to ordering too late. The former results in too much backtracking while the latter results in much redundancy as well as conflicts in partial problem solutions. Consequently, these strategies, do not effectively reduce search effort. Unlike these methods, the method of ordering subgoals that we pro-

pose is based on actively detecting the inherent constraints among problem subgoals.

3. A Robot Planning Problem

In this section, we introduce a robot navigation planning problem. The goal of the problem can be stated as follows: The *Robot* is to move from $room_1$ to $room_4$ while box_1 is to be pushed from $room_1$ into $room_6$. The initial state and the goal state of the problem are depicted in Figure 1.

In this problem, there are three pertinent problem objects, namely, *Robot*, box_1 and box_2. A state of this problem has two aspects. One is that of an object being in a certain room and the other is that of an object being next to other objects. In order to represent the status of the problem objects in a state, we define the functions *INROOM* and *NEXTTO*. Each function takes a problem object and a state as two arguments and returns a value to indicate the corresponding status of the problem object in that state. Assume *obj* is an object and s is a state. Then $INROOM(obj,s)$ gives a room identifier indicating in which room the object is located in the state s. $NEXTTO(obj,s)$ returns a set of objects whose positions are next to *obj* in the state s.

The legal actions in this planning problem are modeled as rules. The rules use the IF <precondition> THEN <postcondition> format. The precondition specifies what must be true for the rule to be applicable. The postcondition specifies the effects of the application of a rule. s_1 and s_2 are used to represent the states before and after the application of a rule. We assume that if the postcondition part of a rule does not contain the status of an object with respect to a certain problem aspect, then that status of the object is unaffected by the rule. The rules for our problem are given below:

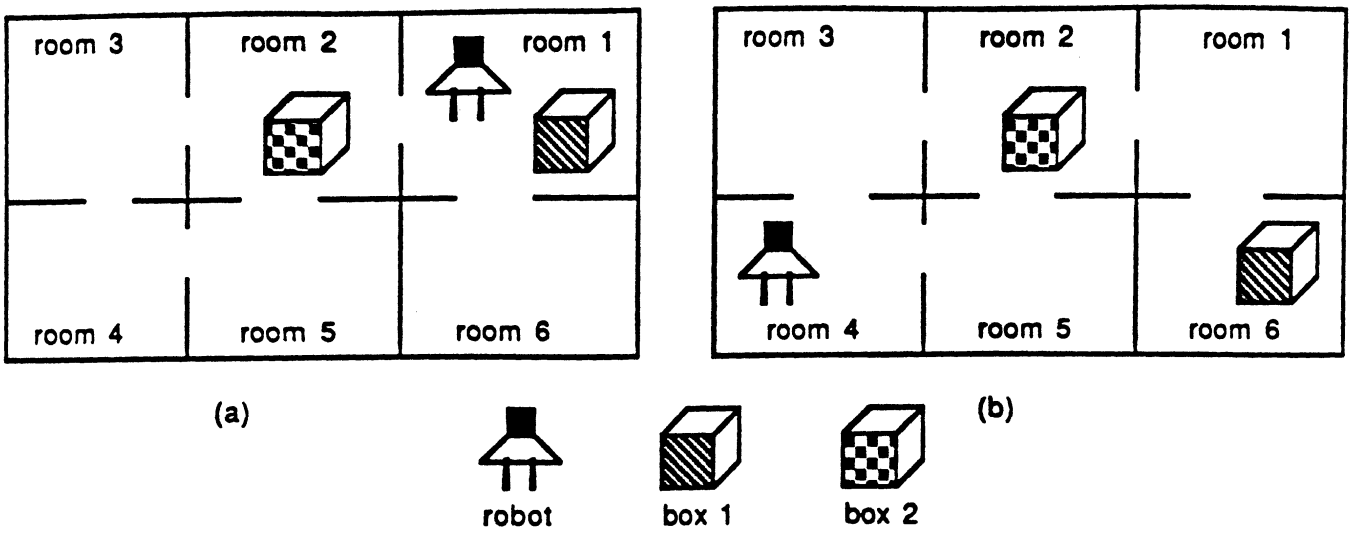


Figure 1. A Robot Planning Problem
(a). Initial state (b). A goal state

R_1 $GOTO(bx^*)$:

IF: $INROOM(Robot,s_1)=INROOM(bx,s_1)$

THEN: $[\ \forall\ by\in(\{box_1,box_2\}-\{bx\})(NEXTTO(by,s_2)=NEXTTO(by,s_1)-\{Robot\})]^{\perp}\ \wedge$

$(NEXTTO(Robot,s_2)=\{bx\})\ \wedge(NEXTTO(bx,s_2)=(\{robot\}\bigcup NEXTTO(bx,s_1)))$

R_2 $GOTHROUGH(rx,ry)$:

IF: $(INROOM(Robot,s_1)=rx)\ \wedge(CONNECTED(rx,ry))$

THEN: $(INROOM(Robot,s_2)=ry)\ \wedge(NEXTTO(Robot,s_2)=\{\})\ \wedge$

$(\ \forall\ by\in\{box_1,box_2\}(NEXTTO(by,s_2)=NEXTTO(by,s_1)-\{Robot\}))$

R_3 $PUSHTHROUGH(bx,rx,ry)$:

IF: $(NEXTTO(Robot,s_1)=\{bx\})\ \wedge(INROOM(Robot,s_1)=rx)\ \wedge$

$(INROOM(bx,s_1)=rx)\ \wedge CONNECTED(rx,ry)$

THEN: $(INROOM(Robot,s_2)=ry)\ \wedge(INROOM(bx,s_2)=ry)\ \wedge$

$(\ \forall\ by\in(\{box_1,box_2\}-\{bx\})(NEXTTO(by,s_2)=NEXTTO(by,s_1)-\{Robot,bx\}))\ \wedge$

$(NEXTTO(Robot,s_2)=\{bx\})\ \wedge(NEXTTO(bx,s_2)=\{robot\})$

---

* bx (or by) and rx (or ry) are the variables which represent a *box* and a *room* respectively.

$^{\perp}$ We are assuming that in a state the robot can be near one box only, while a box can be near both the robot and the other box.

The initial state s_{in} is given by

$$(INROOM(Robot, s_{in})=room_1) \wedge (INROOM(box_1, s_{in})=room_1) \wedge (INROOM(box_2, s_{in})=room_2)$$
$$\wedge (NEXTTO(Robot, s_{in})=\{\}) \wedge (NEXTTO(box_1, s_{in})=\{\}) \wedge (NEXTTO(box_2, s_{in})=\{\})$$

The goal state s_G of the problem is specified by the following formula, called the goal condition formula:

$$(INROOM(box_1, s_G)=room_6) \wedge (INROOM(Robot, s_G)=room_4)$$

Although this problem appears to be very simple, a heuristic state space search can be very inefficient. We tried to use the automatically generated admissible heuristic function([IrY85]) to control the search in solving the problem. The search tree generated is depicted in Figure 2. For this problem, 78 nodes are produced and 35 nodes are expanded for deriving an optimal solution with length of 6 rules.

On tracing the search tree generated in the problem solving, we find that the heuristic prefers to move the robot directly towards its own goal rather than moving the robot to box_1 and then pushing box_1 into $room_6$. This is caused by the fact that the difference in the locations of the robot in any non-goal state and the goal state always dominates that for the box_1. The inherent ordering constraints between the two problem subgoals are not recognized and used. Consequently, the robot is misled into going into $room_4$ directly. It is not until the robot arrives at $room_4$ that the task of moving the box_1 is noticed.

The ineffectiveness of the heuristic in this problem is due to the lack of knowledge about the ordering constraints among the subgoals. This motivates us to develop a systematic subgoal ordering strategy and to incorporate it into the automatic heuristic generation.

4. A Systematic Approach to Subgoal Ordering

If a problem goal can be represented by a predicate formula in a conjunctive normal form, then we can conceive every conjunct as a problem subgoal. There are usually interactions between subgoals which determine the natural order of achieving them in solving a problem. Many types of constraints may exist among problem subgoals. We use only one type of constraint to order subgoals. However, we mention two more types of constraints below to intuitively motivate the third type of constraint which we use for ordering subgoals. Actually, the third type of constraint is a conjunction of the first two.

(1). *A subgoal g_2 cannot be achieved before a subgoal g_1 in any problem solution.* There could be two possible situations. One is when g_2 is satisfied first and the precondition of achieving g_1 cannot be established without violating g_2. The other is if g_2 is satisfied first then any rule which achieves g_1 will force the violation of g_2.

The subgoal ordering constraints in our robot planning problem is an instance of the first situation. If we achieve the subgoal for the robot first, then when we turn to achieve the subgoal for box_1, we will have to retract the established subgoal for the robot.

An example for the second situation is the following. Suppose our goal is to have clothes washed and dried. The actions we can use are 'wash clothes in a washer' and 'dry clothes in a dryer'. If we achieve the 'dry' subgoal first, then although we can still 'wash clothes in a washer' to satisfy the other subgoal, the 'dry' subgoal would be wiped out because the clothes become wet after washing.

(2). *Subgoals g_1 and g_2 cannot be achieved simultaneously by the application of a single rule.* For example, in our robot planning problem, the robot cannot get into $room_4$

while the box_1 is pushed into $room_6$.

(3). *A subgoal g_1 must be achieved before a subgoal g_2 in any problem solution in which both are satisfied.* As mentioned before, this constraint is actually a conjunction of the constraints (1) and (2) and it is this constraint which is used in ordering problem subgoals.

These three subgoal ordering constraints are intuitively clear and practically useful. However, it is not intuitively clear how we can systematically detect these constraints from a problem formulation. In order to automate the process of detecting these kinds of constraints from the basic problem formulation and to order problem subgoals systematically, we have developed relations and procedures. In the following, the proposed approach and the results are presented. First we introduce some notations:

- G represents the goal condition formula which is a ground predicate formula specifying the desired state of affair for a problem. g_i represents a subgoal condition formula which is a conjunct in the goal condition formula. SG is the set of all subgoal condition formulas of G.
- S_{g_i} is a subset the problem states in which the condition g_i holds.
- $R_k(s_1)$ is used to denote the resulting state of the application of the rule R_k to the state s_1 in which R_k is applicable.
- $prec_k$ and $post_k$ are the precondition formulas and postcondition formulas for the rule R_k respectively.
- *problem solution path*: a sequence of states $(s_1,s_2,...,s_n)$ such that s_1 is an initial state, s_n is a goal state, and for every state $s_i (1 \leq i < n)$, there is a rule which can transform s_i into s_{i+1}. *A partial problem solution path in a path* $P=(s_1,s_2,...,s_n)$ is a subsequence $(s_1,s_2,...,s_p)$ such that $p \leq n$.

We now define a binary relation over SG. A theorem is then provided which relates this relation to the type-3 constraint.

Definition: "$\overset{*}{<}$" † is a binary relation over SG. $g_i \overset{*}{<} g_j$ iff

$$\forall\ k\ \forall\ s[((R_k(s) \in S_{g_i} \not\wedge g_j) \wedge (s \in S_{prec_k})) \longrightarrow s \in S_{g_i}]$$

$g_i \overset{*}{<} g_j$ means that for any rule in the problem, if the rule can transform a state, say s, to a new state in which both g_i and g_j are true, then s must satisfy g_i.

The relation $R_{\overset{*}{<}}$ appears to be complicated because quantifiers are used over the rules and the states. However, in constructing this relation, only pattern matching, variable binding and easy evaluations are needed. A procedure called SOC(Subgoal Ordering Constraints) has been developed for constructing $R_{\overset{*}{<}}$. A loose upper bound for the complexity of this procedure is $(m \times n)^2 \times k$, where m, n, and k are the cardinalities of the set of problem objects, set of problem aspects and set of problem rule schemes, respectively.

In the following, we give two definitions and then give a theorem which relates the relation $R_{\overset{*}{<}}$ to the type-3 ordering constraint among problem subgoals.

Definition: g_i precedes g_j in a problem solution path P *iff* there exists a partial solution path $(s_1, s_2, \ldots, s_p)$ in P which satisfies:

$$(s_p \in S_{g_i} \not\wedge g_j) \wedge \exists\ k((1 \leq k < p) \wedge (s_k \in S_{g_i}) \wedge \forall\ h((k \leq h < p) \rightarrow (s_h \in S_{g_i} \not\vdash g_j))).$$

Definition: g_i and g_j are said both achieved in a problem solution path P *iff* there is a partial solution path $(s_1, s_2, \ldots, s_p)$ in P such that

---

† In later discussions, the notations $\overset{*}{<}$ and $R_{\overset{*}{<}}$ will be used interchangably.

$$(s_p \in S_{g_i} \wedge g_j) \wedge \forall\, k((1 \leq k < p) \rightarrow s_k \in S_{\neg g_i} \vee \neg g_j)$$

Now, we present the theorem which links the relation $R_{\overset{*}{<}}$ with the type-3 subgoal constraint. The proof is omitted. In the theorem, we assume that the two subgoals g_i and g_j are not both satisfied in the initial state.

Theorem: Let g_i and g_j be two subgoal condition formulas. $g_i \overset{*}{<} g_j$ *iff* g_i precedes g_j in any problem solution in which both subgoals are achieved.

By this theorem, it is clear that if $g_i \overset{*}{<} g_j$, we have to achieve g_i before achieving g_j in any problem solution. Although the relation $R_{\overset{*}{<}}$ is itself not transitive, for a given problem, a partial ordering of subgoals can be created using this relation. A procedure called *GOAL_ORDER* has been developed for that purpose. Every subgoal is assigned a rank by the procedure. All subgoals of a certain rank must be achieved before a subgoal of any higher rank can be achieved. As the example below will show, it is not necessary that if $g_i \overset{*}{<} g_j$, then g_i has a lower rank than that of g_j. However, in that case, the rank of g_j is at least as high as that of g_i. A loose upper bound for the complexity of the procedure *GOAL_ORDER* is $(m \times n)^3 \log(m \times n)$, where m and n are the cardinalities of the set of problem objects and the set of problem aspects, respectively.

With the ranked subgoals, we can construct a goal sequence $G' = G_1, G_2, \ldots, G_n$, where G_i is called the i-th component goal of the goal sequence and is a conjunction of all those subgoal condition formulas with rank i. This sequence imposes constraints on the search for the solution path. It requires that G_i always be achieved before G_j if $i < j$. It also ensures that in achieving G_j, all the component goals G_i's$(i < j)$ which have been achieved will be protected.

We illustrate the result of the procedure *GOAL_ORDER* by giving the following example. Suppose we have a set of subgoals $SG=\{g_1,...,g_6\}$ and the relation $R_{\overset{*}{<}}=\{<g_5,g_6>, <g_2,g_6>, <g_2,g_4>, <g_6,g_1>, <g_6,g_4>, <g_1,g_3>\}$. The procedure *GOAL_ORDER* assigns ranks to the subgoals as follows:

$Rank_1$: g_5, g_2

$Rank_2$: g_6

$Rank_3$: g_1, g_3, g_4

g_5 and g_2 are ranked at the first level because no other subgoal can precede them and there is no ordering constraints among them. g_6 alone is ranked at the second level because it has to be preceded by g_5 and g_2 and it must precede the rest of the subgoals. Finally, g_1, g_3 and g_4 are ranked at the third level because they have to be preceded by all the other subgoals and they themselves can not be ordered further. Notice that since $g_1 \overset{*}{<} g_3$, g_3 has to be preceded by all the subgoals preceding g_1 although g_3 may not relate with any one of them through $R_{\overset{*}{<}}$. However, we cannot rank g_3 at a different level than g_1, because there exist no definite ordering constraints between g_1 and g_4, and g_3 and g_4.

With this ordering result, we can get a goal sequence $G' = G_1, G_2, G_3$, where

$G_1 = g_2 \wedge g_5$

$G_2 = g_6$

$G_3 = g_1 \wedge g_3 \wedge g_4$

In our robot planning example, there are two subgoals as follows:

(1) $INROOM(robot, s_G) = room_4$,

(2) $INROOM(box_1, s_G) = room_6$.

The relation R_{*} is R_{*} $=\{<(2),(1)>\}$. After applying the subgoal ordering algorithm, the subgoals are ranked into two levels and the original problem goal is transformed into the goal sequence $G'=G_1,G_2$, where G_1 is the same as the subgoal (2) and G_2 is the same as the subgoal (1). This ordering complies with the constraints inherent in the problem specifications.

Unlike NOAH which makes least commitment, the approach described in this section represents an undercommitment to the ordering of problem subgoals, which means that subgoals are always arranged in a correct order, although the ordering may not be complete. Therefore, the ordering results can be used to guide the search and never needs to be retracted.

5. Goal Augmentation

In a problem formulation, goals are often not specified completely in the sense that many things are left as "don't care". Therefore, more than one state can be a candidate goal state. However, as far as the optimality is concerned, only some of them can actually qualify to be the goal states.

For the robot planning configuration in our example, for instance, assume the goal is simply that $INROOM(box_1)=room_6$. It seems that the *Robot* can be anywhere in the goal state. However, if an optimal path is to be achieved, three other conditions need to be also satisfied. These three conditions are (1) the *Robot* is in $room_6$, (2) the *Robot* is next to only the box_1, and (3) no object is next to the box_2. The first condition and the third condition are the immediate consequences of achieving the given goal, namely, $INROOM(box_1)=room_6$, while the second condition is a necessary precondition for

achieving the given goal and which is invariant over the rule that achieves the goal.

This reasoning can be used to augment every component goal derived by the subgoal ordering. We have proved that for the goal with one component goal(and the proof can be extended to multiple component goals) that the goal augmentation has two properties. These properties are: (1) every state which satisfies the original component goal condition formula and which is on an optimal solution path, also satisfies the augmented goal condition formula, (2) the number of nodes expanded during the search for the optimal solution path with the the augmented component goal is no more than that with the original component goal. We do not present the theorems and proofs in this paper for space consideration. In the following, we give the algorithm for the augmentation of all component goals of a goal sequence.

$AUGMENT(G', G^*, R)$:

$G' = G_1, G_2, \ldots, G_n$

R is the set of problem rules.

Define $G_l = g_1^l \wedge g_2^l \cdots \wedge g_{n_l}^l . (1 \leq l \leq n)$

Define $G^l = G_1 \wedge G_2 \wedge \cdots \wedge G_l$.

(0). If m is the largest index such that $G_1, \ldots, G_m$ are all already satisfied by s_{in}, then $G_l^* = G_l$ for $l = 1, \ldots, m$.

For each $G_l (m+1 \leq l \leq n)$, do (1) to (6):

(1). For each conjunct of G_l, namely, $g_i^l (1 \leq i \leq n_l)$, do (2) to (5):

(2). $AUG_i \leftarrow FALSE$; $R^* = R$.

(3). For each $R_j \in R^*$, if $\exists\, s (s \in S_{prec_j} \wedge_{\neg g_i^l} \wedge R_j(s) \in S_{G^l})$, then $AUG_{ij} \leftarrow post_j'$ ($post_j'$ is $post_j$ with substitution for variables), $R^* = R^* - \{R_j\}$ and do (4)-(5).

(4). for each $prec_{jk}'$(the k-th conjunct of $prec_j$ with substitution), do the following: If $prec_{jk}'$

is not affected by the operator j, then $AUG_{ij} \leftarrow AUG_{ij} \wedge prec^{'}_{jk}$.

(5). $AUG_i \leftarrow AUG_i \vee AUG_{ij}$

(6). $G_l^{'} \leftarrow G_l \wedge (AUG_1 \vee AUG_2 \vee \cdots \vee AUG_{n_l})$. Return.

In the algorithm *AUGMENT*, step (0) finds the first component goal which is not already satisfied in the initial state. The algorithm achieves the augmentation of each component goal G_k mainly in steps (3)-(6). For every subgoal g_i in the component goal G_k, every rule R_j is checked to see whether it is applicable in a state in which the subgoal g_i is not satisfied, and whether its application to such a state can satisfy G_k and all those component goals which precede G_k in the goal sequence. If the rule R_j passes the test, then besides the component goal condition formula G_k, all its preconditions which are unaffected by the application of the rule and all its postconditions are true in the state resulting from the application of R_j. These preconditions and postconditions are conjuncted into AUG_{ij}. If more than one rule passes the test, then the conditions derived from different rules are disjuncted and stored in AUG_i. The disjunctions of all the AUG_i's is the total augmentation which is finally conjuncted with the corresponding component goal in the step(6). A loose upper bound of the complexity of the procedure *AUGMENT* is $(m \times n)^2 \times k$, where m, n, and k are the cardinalities of the set of problem objects, set of problem aspects and set of rules. Since usually only a few rules are related with each possible subgoal in a problem, the computation of the goal augmentation is often very efficient.

We again use our robot planning problem to explain the algorithm. From the subgoal ordering, we derived a goal sequence $G^{'} = G_1, G_2$ with two component goals, namely, $G_1 = INROOM(box_1, s_{G_1}) = room_6$, and $G_2 = INROOM(Robot, s_{G_2}) = room_4$. Since G_1 has only one subgoal and that subgoal can only be achieved by R_3, the

augmentation result is

$$G_1^* = (INROOM(box_1, s_{G_1}) = room_6) \wedge (INROOM(Robot, s_{G_1}) = room_6)$$
$$\wedge (NEXTTO(Robot, s_{G_1}) = \{box_1\}) \wedge (NEXTTO(box_1, s_{G_1}) = \{Robot\})$$
$$\wedge (NEXTTO(box_2, s_{G_1}) = \{\ \})$$

Although the component goal G_2 also has only one subgoal, that subgoal can be achieved by applying either R_2 or R_3. Therefore, the augmentation result is

$$G_2^* = (INROOM(Robot, s_{G_2}) = room_4) \wedge \{[(NEXTTO(Robot, s_{G_2}) = \{\ \})$$
$$\wedge (Robot \notin NEXTTO(box_1, s_{G_2})) \wedge (Robot \notin NEXTTO(box_2, s_{G_2}))]$$
$$\vee [(INROOM(box_2, s_{G_2}) = room_4) \wedge (NEXTTO(box_2, s_{G_2}) = \{Robot\}) \wedge$$
$$(NEXTTO(Robot, s_{G_2}) = \{box_2\}) \wedge (NEXTTO(box_1, s_{G_2}) = \{\ \})]\}$$

6. Integration of Subgoal Ordering and Heuristic Estimation

In our previous research, we proposed a methodology for determining a general and admissible heuristic function $h(e_x)$ for the best-first search for a problem solution(see [IrY85], [IrY86]). According to this methodology, for each state e_x, $h(e_x)$ returns an under-estimated minimum cost for the path from e_x to the goal set. We can now show that the subgoal ordering constraints can be naturally incorporated into the heuristic estimation. The new heuristic estimation is tighter than the original one while the admissibility and monotonicity is still preserved. In this section, we first briefly explain the original heuristic function and then introduce the integration of subgoal ordering with the heuristic estimation. Properties of the new heuristic estimation are then presented.

The original heuristic function $h(e_x)$ is derived as follows: The problem is first decomposed object-wise and parallelly transformed into k simplified problems, where k is the number of problem objects whose status in the state e_x is different from that in the goal state. Each simplified problem contains only one object in its problem space, in goals and in operators. The specifications concerning other objects are simply suppressed. The minimum cost for the optimal path in each simplified problem is then easily derived by conducting an exhaustive search.

Although the derived cost in any simplified problem can be taken as the heuristic for the original problem and this heuristic is both admissible and monotonic, further derivations are made to get a tighter heuristic estimation. Three functions are evaluated. The first gives the maximum value of all the minimum costs for solving the simplified problems. The second is the sum of all the minimum costs divided by the maximum number of objects affected by one operator in the original problem model. The third is the same as the second except that the objects which are affected by all operators are excluded from consideration. The maximum of the three computation results is taken to be the final value of $h(e_x)$ for the state e_x.

The search heuristic described above assures the admissibility and monotonicity. However, as has been demonstrated, it may not be effective in guiding search for some of the problems. The heuristic does not incorporate the knowledge of interactions among problem subgoals. Therefore it sometimes leads search to achieve subgoals in a roundabout order and consequently the search becomes very inefficient. We now introduce a new heuristic function which incorporates the subgoal ordering knowledge.

We denote the component goal sequence generated by subgoal ordering to be $G_1, G_2, \ldots, G_m$, the sequence after the augmentation to be $G_1^*, G_2^*, \ldots, G_m^*$, and finally

we denote the goal sequence for heuristic estimation to be

$$G_1^s, G_2^s, \ldots, G_m^s = G_1^*, G_2^* \wedge G_1, \cdots, G_m^* \wedge (\bigwedge_{i=1}^{m-1} G_i)$$

In the search process for the problem solution, for any state e_x being evaluated, we define an *effective goal subsequence* $G_{e_x}^s$ which is the remaining sequence of component goals to be fulfilled. Formally,

$$G_{e_x}^s = G_{i_x}^s, G_{i_x+1}^s, \ldots, G_m^s \quad iff \quad \forall\, i((1 \leq i < i_x) \rightarrow e_x \in S_{G_i}) \wedge e_x \notin S_{G_{i_x}}$$

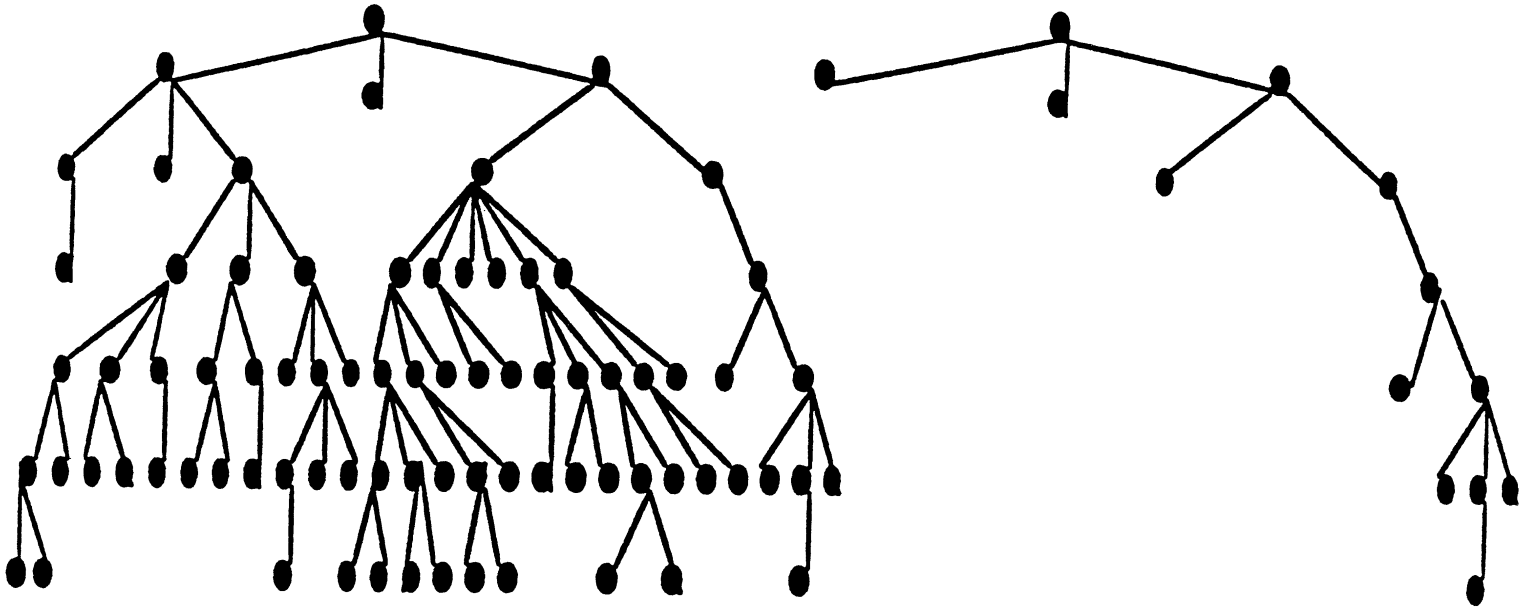
In the following, we informally introduce the new heuristic function. When a state, say e_x, is to be evaluated, the problem is decomposed into k simplified problems as before where k is the number of problem objects which do not have the same status in e_x and in all the component goals of the effective goal subsequence $G_{e_x}^s$. In the simplified problem for an object a, the effective goal subsequence is relaxed such that only the specification about the status of a is retained in each of the elements of the sequence. The cost of the optimal solution path passing through each of the elements of this simplified sequence is then determined. The final value of the new heuristic $h^+(e_x)$ is the maximum of the three values computed in the same way as before.

The properties of the heuristic function $h^+(e_x)$ are presented in the following two theorems.

Theorem: The heuristic function $h^+(e_x)$ as given above is admissible and monotonic.

Theorem: The heuristic function $h^+(e_x)$ as given above is tighter than the original heuristic estimation $h(e_x)$.

With the new heuristic function, the search becomes very efficient. As can be seen from Figure 2(a), without the subgoal ordering and goal augmentation, the search goes



78 nodes generated
35 nodes expanded
(a)

13 nodes generated
6 nodes expanded
(b)

Figure 2. Comparison of Search Tree Diagrams
(a). without subgoal ordering and goal augmention
(b). with subgoal ordering and goal augmentation

astray for a long time before it touches the right path. For this problem, 78 nodes are produced and 35 nodes are expanded for deriving an optimal solution with length of 6 rules. However, as shown by Figure 2(b), with the subgoal ordering incorporated, the heuristic value discriminates against the misleading path at the very outset. The search tree generated for this problem contains only 13 nodes of which, only 6 nodes are expanded. This demonstrates the advantage of the subgoal ordering when it is incorporated into heuristic estimation.

7. Example

In this section, we give an example to illustrate the application of the subgoal ordering, goal augmentation and the integration of these two strategies with the admissible

heuristic generation. The example problem is again a robot navigation planning problem. The problem is both described in Figure 3 and specified by the problem model formalism in the following. The results of applying each strategy are shown and finally, the performance of the search with using and without using the subgoal ordering and the goal augmentation are compared.

There are 7 problem objects involved in this problem, one *Robot*, one *box* and 5 *doors*. The problem has three aspects. Two of them are the same as in the previous example and we still use the functions *INROOM* and *NEXTTO* to represent them. A new problem aspect is the status of a door, for which we choose to use the function *D-STATE* to represent. *D-STATE* takes a door object and a state as its arguments and returns the status of that door in that state. The value of the door status is either 'open' or 'closed'.

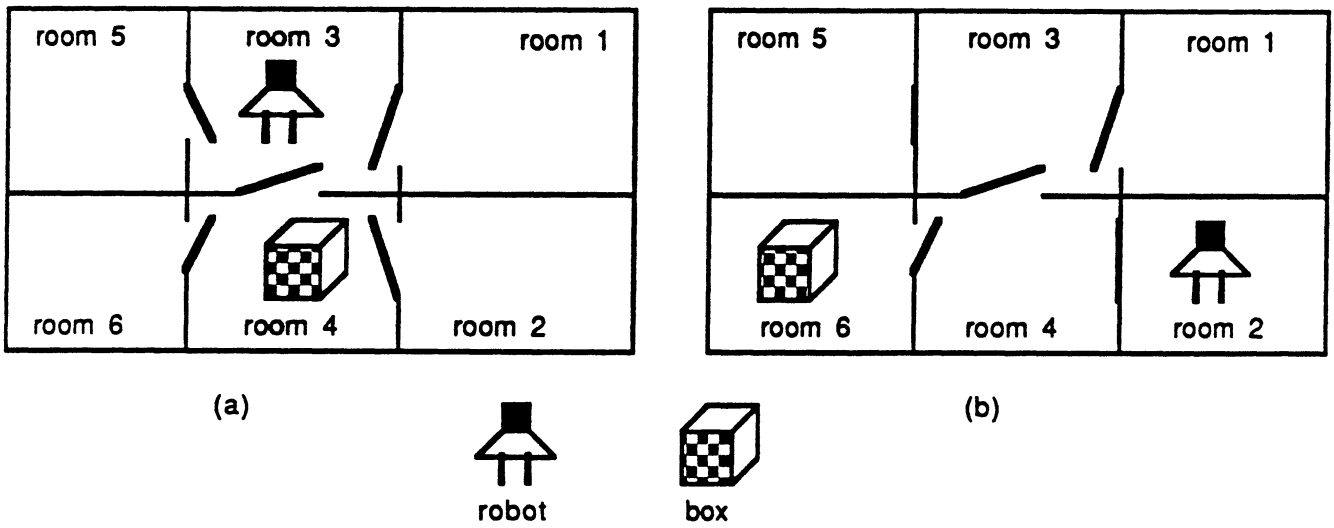


Figure 3. A Robot Planning Example
(a). An initial state (2). A goal state

There are 6 pertinent actions that are modeled as rules. The first rule is for the *Robot* to walk to a *box* if they are in the same room. The second rule is for the *Robot* to walk to a *door* if the *Robot* is in one of the rooms connected by that door. The third rule is for the *Robot* to go from one room to another if these two rooms are connected by a door and the door is open. The fourth rule is for the *Robot* to push a box from one room to the other. For this rule to be applicable, besides satisfying the preconditions for action 3, the *Robot* must also be near the *box* to be pushed. The last two rules are opening and closing a door by the *Robot* which stands next to that door. It is assumed that the cost of applying each rule is a constant. Therefore, a best solution is one composed of least number of rules.

The rules are specified as follows:

R_1 $GOTO(bx^*)$:

IF: $INROOM(Robot,s_1)=INROOM(bx,s_1)$

THEN: $(\forall\ dx\in\{door_{13},...,door_{46}\}(NEXTTO(dx,s_2)=\{\}))^{\perp}\ \wedge$

$(NEXTTO(Robot,s_2)=\{bx\})\ \wedge(NEXTTO(bx,s_2)=\{robot\})$

R_2 $GOTO(dx)$:

IF: $\exists\ rx,ry\,[(INROOM(Robot,s_1)=rx)\ \wedge(CONNECT(dx,rx,ry))]$

THEN: $(\forall\ obj\in(\{box,door_{13},...,door_{46}\}-\{dx\})(NEXTTO(obj,s_2)=\{\})\ \wedge$

$(NEXTTO(Robot,s_2)=\{dx\})\ \wedge(NEXTTO(dx,s_2)=\{Robot\})$

R_3 $GOTHROUGH(dx,ry)$:

IF: $\exists\ rx\,[(INROOM(Robot,s_1)=rx)\ \wedge(CONNECTED(dx,rx,ry))]\ \wedge(D\_STATE(dx,s_1)=open)$

THEN: $(INROOM(Robot,s_2)=ry)\ \wedge(\forall\ obj(NEXTTO(obj,s_2)=\{\}))$

R_4 $PUSHTHROUGH(bx,dx,ry)$:

---

* bx(or by), rx(or ry) and dx(or dy) are the variables which represent a *box*, a *room* and a *door* respectively.

$^{\perp}$ We are assuming that in any state, the robot cannot be near both a door and a box and the box is never near any door.

IF: $(NEXTTO(Robot,s_1)=\{bx\}) \wedge (\exists\ rx[(INROOM(Robot,s_1)=INROOM(bx,s_1)=rx)) \wedge$

$CONNECTED(dx,rx,ry)] \wedge (D\_STATE(dx,s_1)=open)$

THEN: $(INROOM(Robot,s_2)=ry) \wedge (INROOM(bx,s_2)=ry) \wedge$

$(\forall\ dx \in \{door_{13},\ldots,door_{46}\}(NEXTTO(dx,s_2)=\{\}) \wedge$

$(NEXTTO(Robot,s_2)=\{bx\}) \wedge (NEXTTO(bx,s_2)=\{robot\})$

R_5 $OPEN(dx)$:

IF: $(D\_STATE(dx,s_1)=closed) \wedge (NEXTTO(robot,s_1)=\{dx\}) \wedge$

$(\exists\ rx,ry[(INROOM(Robot,s_1)=rx) \wedge CONNECT(dx,rx,ry)])$

THEN : $D\_STATE(dx,s_2)=open$

R_6 $CLOSE(dx)$:

IF: $(D\_STATE(dx,s_1)=open) \wedge (NEXTTO(robot,s_1)=\{dx\})$

$(\exists\ rx,ry[(INROOM(Robot,s_1)=rx) \wedge CONNECT(dx,rx,ry)])$

THEN: $D\_STATE(dx,s_2)=closed$

The initial state s_{in} is specified as follows:

$(INROOM(Robot,s_{in})=room_3) \wedge (INROOM(box,s_{in})=room_4) \wedge$

$(D\_STATE(door_{13},s_{in})=open) \wedge \cdots \wedge (D\_STATE(door_{46},s_{in})=open)$

The problem goal state s_G is specified by the goal condition formula $G=g_1 \wedge g_2 \wedge g_3 \wedge g_4$, where

g_1:$(INROOM(Robot,s_G)=room_2)$,

g_2:$(INROOM(box,s_G)=room_6)$,

g_3:$(D\_STATE(door_{35},s_G)=closed)$,

g_4:$(D\_STATE(door_{24},s_G)=closed)$.

When trying to solve this problem by heuristic search without using the subgoal ordering and goal augmentation, we find that the efficiency is very poor. However, there exist inherent ordering constraints among the problem subgoals and there also exists

implicit goal state information in this problem formulation. By applying the subgoal ordering and the goal augmentation strategies, we can detect and make use of this information to enable a tighter search heuristic and to make the search efficient.

Using the procedure SOC, the relation $R_{*<}$ is constructed as follows:

$$R_{*<}=\{<g_1,g_4>,<g_2,g_1>,<g_3,g_1>\}$$

Applying the procedure *GOAL_ORDER* to the set of problem subgoals with the relation $R_{*<}$, we get a goal sequence $G'=G_1,G_2,G_3$, where $G_1=g_2 \wedge g_3$, $G_2=g_1$ and $G_3=g_4$.

The procedure *AUGMENT* is then applied to that goal sequence and the following results are derived. To make it easy to understand, we have simplified the augmentation results.

$$\begin{aligned}
G_1^* = &(INROOM(box,s_{G_1})=room_6) \wedge (D_STATE(door_{35},s_{G_1})=closed) \wedge \\
&[(D_STATE(door_{46},s_{G_1})=open) \wedge (INROOM(Robot,s_{G_1})=room_6) \wedge \\
&(NEXTTO(Robot,s_{G_1})=\{box\}) \wedge (NEXTTO(box,s_{G_1})=\{Robot\}) \wedge \\
&(\forall\ dx(NEXTTO(dx,s_{G_1})=\{\})] \vee \\
&[(INROOM(Robot,s_{G_1})\in\{room_3,room_5\}) \wedge (NEXTTO(Robot,s_{G_1})=\{d_{35}\} \wedge \\
&NEXTTO(door_{35},s_{G_1})=\{Robot\} \wedge \forall\ (dx\neq door_{35})(NEXTTO(dx,s_{G_1})=\{\})]
\end{aligned}$$

$$\begin{aligned}
G_2^* = &(INROOM(Robot,s_{G_2})=room_2) \wedge (D_STATE(door_{24},s_{G_2})=open) \wedge \\
&(\forall\ obj(NEXTTO(obj,s_{G_2})=\{\})
\end{aligned}$$

$$\begin{aligned}
G_3^* = &(D_STATE(door_{24},s_{G_3})=closed) \wedge (INROOM(Robot,s_{G_3})=room_2) \wedge \\
&(NEXTTO(Robot,s_{G_3})=\{door_{24}\}) \wedge (NEXTTO(door_{24},s_{G_3})=\{Robot\})
\end{aligned}$$

Finally, we incorporate the above results into the heuristic generation and conduct the heuristic search to solve for the optimal solution path. The effectiveness of the search heuristic is greatly improved. Using the original methodology without subgoal ordering and goal augmentation, 321 nodes are generated and 138 nodes expanded. However, with the new methodology, 140 nodes are generated and 60 nodes expanded.

8. Summary

This paper is a contribution to the development of a general methodology for automated heuristic problem solving. We have presented an improved procedure for subgoal ordering and a novel procedure for goal augmentation. We then outline a procedure to integrate these two strategies with our previous methodology of automatic generation of admissible search heuristic. The combined package is a new methodology of general problem solving.

REFERENCES

[Das77] Dawson, C., and Siklossy, L., "The Role of Preprocessing in Problem Solving Systems", *Proc. IJCAI 5,* Cambridge, Mass., August 1977.

[ErG82] Ernst, G. W., and Goldstein, M. M., "Mechanical Discovery of Classes of Problem Solving Strategies", *JACM*, Vol. 29, No.1, January 1982. pp. 1-23.

[ErN69] Ernst, G. W., and Newell, A., *GPS: A Case Study in Generality and Problem Solving.* ACM Monograph Series. Academic Press, New York, 1969.

[IrY85] Irani, K.B and Yoo, Suk I., "Problem Solving: A Methodology for Modeling and Generating Heuristics". *Second International Conference on Artificial Intelligence Applications.*, December 1985.

[IrY86] Irani, K.B. and Yoo, S. I., "A Methodology for Solving Problems: Problem Modeling and Automatic Heuristic Generation". Submitted to *IEEE Transactions on PAMI*, 1986.

[Sac77] Sacerdoti, E. D., "A Structure for Plans and Behavior", Elsevier North-Holland, New York, 1977.

[Sus75] Sussman, G. J., *A Computer Model of Skill Acquisition*, New York: American Elsevier. 1975.

[Tat75] Tate, A., "Interacting Goals and Their Use", *The proceeding of the 4th IJCAI*, 1975, pp. 215-218.

[Wal81] Waldinger, R., "Achieving Several Goals Simultaneously". Readings in AI, pp. 250-271. (1981).

[War74] Warren, David H.D., "WARPLAN: A System for Generating Plans", Department of Computational Logic Memo 76, U. of Edinburgh, July, 1974.