Subgoal Ordering and Goal Augmentation for Heurila tic Problem Solving

Keki B. Irani and Jie Cheng
Robot Systems Division
Center for Research on Integrated Manufacturing
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122, USA

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 essence 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 is demonstrated by the application to robot optimal task planning problems.

1. Introduction

In previous papers ([IrY85], [IrY87]), we have presented a general heuristic problem solving methodology. Recently, this methodology is augmented by two systematic problem solving strategies, namely, problem subgoal ordering and goal augmentation. This paper reports on these two strategies.

Problem subgoal ordering is an important strategy used to reduce problem space search. Many different subgoal ordering strategies have been reported before (|ErG82|, [Sus75j, [Tat75], fWal8I), [Sac77], [Das77]). However, in this paper we present a novel approach to subgoal ordering. In contrast to the previously proposed approaches, our approach is to preorder the problem subgoals systematically by reasoning on the problem formulation. The result of ordering is then imposed on the search control to constrain the search space.

Goal augmentation is another problem solving strategy which systematically discovers goal information that is not explicitly represented but can be inferred from a given problem formulation. Augmentation can often reduce the ambiguity of the specified problem solving goals which enables more accurate estimation of the search heuristic.

Our research on problem subgoal ordering and goal augmentation are both part of an effort to develop a general heuristic problem solving methodology. Previously, we achieved methods for systematically modeling problems and automatically generating admissible heuristics for A*-like best-first search algorithms (see [IrY85], [IrY87]). Currently, the subgoal ordering and the goal augmentation strategies have been integrated with the heuristic generation to constrain search through improving the tightness of the heuristic estimation.

' This work was supported IN part by AFOSR under contract f J3615-SS-C-5I0S and ID part by SRC under con tract No-07-055

The rest of the paper is organised in the following way. A simple robot planning problem is introduced in section 2. Earlier problem subgoal ordering strategies are briefly reviewed in section 3. Our approaches to subgoal ordering and to goal augmentation are then presented in section 4 and section 5 respectively. The integration of these problem solving approaches with the automatic search-heuristic generation is reported in section 6. A complete example is given in section 7 and the paper is finally summarised in section 8.

2. A Robot Planning Problem

To illustrate our ideas, we introduce a robot navigation planning problem in this section. In the problem, there are three problem objects, one robot and two boxes (see Figure 1). The robot is the only active agent which can change its own position as well as move other objects. The valid actions assumed in this problem setting are (I)the robot going to an object in the same room; (2)the robot going into a connected room; and (3)the robot pushing an adjacent box to a connected room.

The problem's initial state and goal state are described in Figure 1. The solution to be found is an optimal sequence of actions for the robot to perform so that the initial state can be transformed into a goal state with the least possible number of actions. Using our formal problem model, this problem can be specified as follows:

(1) Three problem objects: *robot, boz*$_I$ and box$_3$;

(2) Two problem attributes modeling relevant problem solving aspects, namely, the locations of problem objects in reference to the room configurations and the adjacency relations among problem objects;

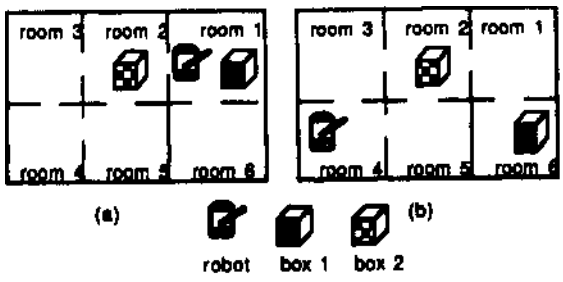

Figure 1. A Robot Planning Problem
(a) Initial state (b) A goal state

(3) Functions which take an object and a state as parameter and return a value representing the status of the object in that state in certain aspects. This example uses two functions, *Inroom* and *Nextto. Inroom{objs)* returns a room identifier indicating in which room the object is located in the state #, while *Neztto(obj,t)* returns a set of objects adjacent to *obj* in the state s;

(4) A goal formula specifying conditions which must be satisfied by any goal state,

$$(Inroom(box_1,s_G)=room_3) \wedge (Inroom(Robot,s_G)=room_4))$$

(5) An initial state, s_{in},

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

(6) A set of problem solving rules modeling valid actions. The rules use the < precondition > < 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. $<_t$ and s_2 are used to represent the states before and after the application of a rule. We assume that if the status of an object with respect to a certain problem aspect is not in the postcondition part of a rule, then it is unaffected by the application of the rule.

R_1 *GOTO(bz*):*
IF: $Inroom(Robot,s_1)=Inroom(bz,s_1)$
THEN: $\{\forall\ (by \neq bz)(Nextto(by,s_2)=Nextto(by,s_1)-\{Robot\})\}$
$\wedge\ (Nextto(Robot,s_2)=\{bz\})$

$\wedge\ (Nextto(bz,s_2)=(\{robot\} \cup Nextto(bz,s_1)))$

R_2 *GOTHROUGH(rz,ry):*
IF: $(Inroom(Robot,s_1)=rz) \wedge (Connected(rz,ry))$
THEN: $(Inroom(Robot,s_2)=ry) \wedge (Nextto(Robot,s_2)=\{\})$
$\wedge (\forall\ by(Nextto(by,s_2)=Nextto(by,s_1)-\{Robot\}))$

R_3 *PUSHTHROUGH(bz,rz,ry):*
IF: $(Nextto(Robot,s_1)=\{bz\}) \wedge (Inroom(Robot,s_1)=rz)$
$\wedge (Inroom(bz,s_1)=rz) \wedge Connected(rz,ry)$
THEN: $(Inroom(Robot,s_2)=ry) \wedge (Inroom(bz,s_2)=ry) \wedge$
$(\forall\ (by \neq bz)(Nextto(by,s_2)=Nextto(by,s_1)-\{Robot,bz\}))$
$\wedge (Nextto(Robot,s_2)=\{bz\}) \wedge (Nextto(bz,s_2)=\{robot\})$

Although this problem appears to be simple, a heuristic state space search without subgoal ordering is very inefficient. To illustrate this, we use the automatically generated admissible heuristic function([IrY85|) to control the search in solving the problem. For this problem, 78 nodes are produced and 35 nodes are expanded for deriving an optimal solution with a length of 6 rules.

On tracing the search tree generated in solving this problem, we find that the generated problem heuristic prefers to move the robot directly towards its own goal rather than moving the robot to first push the box, into *room** This is due to the fact that the difference in the locations of the robot in any non-goal state and the goal state always dominates the similar difference for the box_x. The inherent ordering constraints between the two problem subgoals are not recognized and used in the heuristic estimation. Consequently, the robot is misled into going into the $room_1$ 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 subgoal ordering constraints and the incompleteness of the goal specification. This motivates us to develop a systematic subgoal ordering approach and a goal augmentation approach. These two approaches reveal implicit knowledge about a problem from its specification, and transform it into explicit problem control constraints to guide heuristic search.

* if (or *by)* and rz(or *ry)* are variables which range over ors and room* respectively

3, Previous Subgoal Ordering Strategies

Subgoal ordering has been employed in many previous problem solving systems as a strategy for reducing search. These systems can be classified by the degree of commitment they make towards subgoal ordering, which reflects how early a decision is made on ordering problem subgoals, and how bold a hypothesis is made on the subgoal ordering constraints with a given amount of information.

Among the early systems, *GPS* [Ern69] is the first to apply subgoal ordering to problem solving. In *GPS,* subgoals are arranged as the row headings in the *Tabic of connection.* The system does not start to achieve a subgoal heading for a certain row until all subgoals heading higher rows are achieved. A total ordering is thus imposed on all problem subgoals, which represents the highest possible commitment towards subgoal ordering. This ordering task needs to be carried out by a user.

Many other systems are also over-committed to subgoal ordering, although they operate in quite a different way from that of GPS. Systems like *HACKER* [Sus75|, *INTERPLAN* [Tat75|, *WARPLAN* [War74], first order subgoals arbitrarily and perform a destructive re-ordering in case a protection constraint violation is detected. Systems like *STRIPS* |FiN71j and *ABSTRIPS* [Sac74] achieve subgoals in the order they are given and backtrack if the given order fails. Waldinger's system [Wal8I] proposes a goal regression strategy which amounts to "constructive subgoal re-ordering".

NOAH [Sac77] is a system which makes "least commitment" towards subgoal ordering. It first attempts problem subgoals in parallel (unless a sequential order constraint is imposed beforehand in the "SOUP" code). Several "critics" are then applied to detect and handle interactions among subgoals or to find and eliminate redundancies.

Another distinctive approach is presented in [Das77], in which a subgoal is chosen to be achieved next if it least interferes with the subgoals already achieved.

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 the 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 reduce search as effectively as expected.

Ernst et al. [ErG82] developed a procedure, *DGBS*, to

mechanize the *GPS* approach. *DGBS* represents a reasonable commitment towards problem subgoal ordering. It tries to detect inherent problem subgoal ordering constraints so that they can be put in order properly before the problem solving actually starts. However, *DGBS* is very inefficient in its construction of the Table of connection. The procedure has to process all the problem operator *instances* initially, instead of just considering problem operator *schemes* as usually given in a problem specification. This may cause *DGBS* to be computationally intractable when tackling a problem with a few operator *schemes* but a large number of operator *instances*. Another problem with *DGBS* is that it cannot guarantee a correct ordering. When the 'superfluous operator constraint" is violated, and the table of connection is in a diagonal form, the system is unable to order subgoals properly.

The key weakness of all these previous subgoal ordering approaches is their inability to properly reveal and make use of the relationship between problem operator structures and problem subgoal orderings. As will be shown later, our research result presents an improvement on this issue over these approaches.

4. A Systematic Approach to Subgoal Ordering

If a problem goal is 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 such subgoals which determine the natural order of achieving them in solving a problem and these interactions are called *subgoal ordering constraints.*

Many types of ordering constraints may exist among problem subgoals. We are mainly interested in one such type. However, we discuss two more types of constraints below to intuitively motivate the third type of constraint, which we use for ordering subgoals.

(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, if g_2 is satisfied first, then all those rules which can achieve g_1 will not be applicable unless g_2 is destroyed. The other is when, if g_2 is satisfied first, then any rule which can achieve g_1 will force the violation of g_2 as a side effect of that rule.

The subgoal ordering constraint in the robot planning example 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_u we will have to retract the established subgoal for the robot.

An example of 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 instance, in our robot planning example, the robot cannot get into $room_A$ while the boz) is pushed into $room_6$.

(3). *A subgoal g1 must be achieved before a subgoal g_2 in any problem solution in which both are satisfied.* This constraint is actually a conjunction of the constraints (1) and (2).

These three subgoal ordering constraints are intuitively clear. However, it is not intuitively clear how one 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 affairs for a problem. g_t 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 of problem states satisfying g_1.
- $R_k(s)$ is used to denote the resulting state of the application of the rule R_k to state *s*.
- $prec_k$ and $post_k$ are the precondition formulas and postcondition formulas for the rule R_k respectively.
- *problem solution path:* a sequence of states $(s_1, s_2, \ldots, s_n)$ such that 8j is an initial state, s_n is a goal state, and for every state $s_i (1 \le i < n)$, there is a rule which can transform s_i into s_{i-1}.
- o *partial solution path in a problem solutton path* $P = (s_1, s_2, \ldots, s_n)$ **is a subsequence** $(s_1, s_2, \ldots, s_p)$ **where** $p \le n$.

In order to explicitly represent the relation between the problem rule specifications and subgoal ordering constraints, we give the following definitions.

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

$$(s_p \in S_{g_i} \wedge_{g_j}) \wedge \exists k\{(1 \le k < p) / \forall h((1 \le h < p) \rightarrow (s_h \in S_{\neg g_i \wedge \neg g_j}))\}$$

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

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

Definition: " < " is a binary relation over *SG. g,* $<g_j$ iff

$$\forall k \forall s\{((R_k(s) \in S_{g_i \wedge g_j}) \wedge (s \in S_{prec_k})) \rightarrow s \in S_{g_i}\}$$

$g_i \overset{\cdot}{<} g_j$ 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,* and *g,* are true, then *s* must satisfy g_i.

The relation $\overset{\cdot}{<}$ appears to be complicated because of the quantifiers used. However, to construct *-", only pattern matching and variable binding are needed. A procedure called SOC (Subgoal Ordering Constraints) has been developed to construct <. A loose upper bound for the complexity of this procedure is $(m \times n)^3 \times k$, where *m, n,* and *k* are the cardinalities of the set of problem objects, the set of problem aspects and the set of rules, respectively.

Now, we present a theorem which links the relation $\overset{*}{<}$ with the type-3 subgoal constraint. The proof is omitted. In the theorem, we assume that the 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, one can see that if $g_i \overset{*}{<} g_j$, then g_i has to be achieved before g_j in any problem solution. Although the relation $\overset{*}{<}$ 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 this purpose. Every subgoal is assigned a rank by the procedure, and all subgoals of a certain rank must be achieved before a subgoal of any higher rank can be achieved. 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 be always achieved before G_j if $i < j$.

We illustrate the result of the procedure *GOAL_ORDER* by giving the following example. Suppose we have a set of subgoals $SG = \{g_1, \ldots, g_8\}$ and the relation $\overset{*}{<} = \{<g_5,g_8>, <g_2,g_8>, <g_2,g_4>, <g_6,g_1>, <g_8,g_4>, <g_1,g_3>\}$. The procedure *GOAL_ORDER* will then assigns $Rank_1$ to g_2 and g_5, $Rank_2$ to g_6, and $Rank_3$ to g_1, g_3 and g_4.

g_2 and g_5 are ranked at the first level since no other subgoal can precede them and there are no ordering constraints among them. g_6 alone is ranked at the second level because it is preceded by g_5 and g_2 and it precedes the rest of the subgoals. Finally, g_1, g_3 and g_4 are ranked at the third level because they are preceded by all the other subgoals and they themselves cannot be ordered further. Notice that since $g_1 \overset{*}{<} g_3$, g_3 has to be preceded by all the subgoals preceding g_1 even though g_3 may not relate with any one of them through $\overset{*}{<}$. 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 the relation $\overset{*}{<}$, we get a goal sequence $G' = G_1, G_2, G_3$, where $G_1 = g_2 \wedge g_5$, $G_2 = g_6$, and $G_3 = g_1 \wedge g_3 \wedge g_4$.

In our robot planning example, the subgoals are:

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

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

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

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 optimality is concerned, only some of them can actually qualify to be goal states.

For the robot planning configuration in our example, for instance, assume the goal is simply that *Inroom(box)=roomfi.* It appears that the *Robot* can be anywhere in the goal state. However, if an optimal path is pursued, three other conditions also need to be satisfied. These conditions are (1) the *Robot* is in *rooms* (2) the *Robot* is adjacent to only the box_u 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(boxi)=room^,* while the second condition is a necessary precondition for achieving the given goal and 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 proven two properties of goal augmentation. The properties are: (1) every state which satisfies the original component goal condition and which is on an optimal solution path, also satisfies the augmented goal condition, (2) the number of nodes expanded during the search for the optimal solution path with the augmented component goal is no more than that with the original component goal. Proofs are omitted for consideration of space. 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.
$G_l = g_1^l \wedge g_2^l \cdots \wedge g_{n_l}^l . (1 \le l \le n)$
$G^l = G_1 \wedge G_2 \wedge \cdots \wedge G_l$.

(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 $G_l (m+1 \le l \le n)$, do (1) to (6):

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

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

(3). For each $R_j \in R^*$, if $\exists\, s (s \in S_{prec_j} \wedge \neg g_i^l \wedge R_j(s) \in S_{G^{l-1}})$, 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). $G_l' \leftarrow G_l \wedge (AUG_1 \vee AUG_2 \vee \cdots \vee AUG_{n_l})$.

(7). Return.

In this algorithm, 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,* 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,* is not satisfied, and whether its application to such a state can satisfy G_k and all those component goals preceding G_k in the goal sequence. If 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 conjoined into AUG_j. If more than one rule passes the test, then the conditions derived from different rules are disjoined and stored in AUG_t. The disjunction of all the AUG_j's is the total augmentation which is finally conjoined with the component goal G_k in 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 rule schemes 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 illustrate the algorithm. From the subgoal ordering, we derived a goal sequence $G = G_1, G_2$ with two component goals, namely,

$$G_1 = Inroom(box_1, s_{G_1}) = room_6,$$
$$G_2 = Inroom(Robot, s_{G_2}) = room_4.$$

Since G_1 has only one subgoal and that subgoal can only be achieved by R_3, the augmentation result is

$$G_1^* : (Inroom(box_1, s_{G_1}) = Inroom(Robot, s_{G_1}) = room_6) \wedge (Nextto(Robot, s_{G_1}) = \{box_1\}) \wedge (Nextto(box_2, s_{G_1}) = \{\}) \wedge (Nextto(box_1, s_{G_1}) = \{Robot\})$$

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

$$G_2^* : (Inroom(Robot, s_{G_2}) = room_4) \wedge (((Nextto(Robot, s_{G_2}) = \{\}) \wedge (Robot \notin Nextto(box_1, s_{G_2})) \wedge (Robot \notin Nextto(box_2, s_{G_2}))) \vee ((Inroom(box_2, s_{G_2}) = room_4) \wedge (Nextto(box_2, s_{G_2}) = \{Robot\}) \wedge (Nextto(Robot, s_{G_2}) = \{box_2\}) \wedge (Nextto(box_1, s_{G_2}) = \{\})))$$

•• Integration of Subgoal Ordering, Goal Augmentation and Heuristic Estimation

In our previous research, we proposed a methodology for determining a general and admissible heuristic function *h($)* for best-first search (see [IrY85], [IrY87]). According to this methodology, for each state s, *h(s)* returns an underestimated minimum cost for the path from * to the goal set. In this section, we show that subgoal ordering and goal augmentation can be naturally incorporated into heuristic estimation. The new heuristic estimation is tighter than the original one while the admissibility and monotonicity is still preserved. We first explain the original heuristic function and then describe the integration of subgoal ordering, goal augmentation and heuristic estimation.

The original heuristic function *h(s)* is derived as follows: The problem is first transformed into *k* simplified problems, where *k* is the number of problem objects whose status in the state *a* is different from that in the goal state. Each simplified problem contains only one object in its problem space, with all specifications concerning other objects suppressed. The minimum cost for the optimal path in each simplified problem is then either derived by conducting an exhaustive search in this small space, or simply retrieved from the store of previous derivation results.

Although the derived cost in any simplified problem can be taken as the heuristic for the original problem, 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 any 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*(s)* for the state s.

The search heuristic described above assures admissibility and monotonicity. However, the heuristic does not incorporate the knowledge of interactions among problem subgoals and is very sensitive to the completeness of the goal specification. In the following, we describe a a new heuristic function which incorporates subgoal ordering knowledge and goal augmentation.

We first denote the component goal sequence generated by subgoal ordering to be $G_1, G_2, ..., G_m$, the sequence after the augmentation t o $G_1', G_2', \ldots, G_m'$, d the goal sequence for heuristic estimation to be

$$G_1^A, G_2^A, ..., G_m^A = G_1', G_2' \wedge G_1, \cdots, G_m' \wedge (\bigwedge_{i=1}^{m-1} G_i)$$

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

$$G_s^A = G_{i_s}^A, G_{i_s+1}^A, \ldots, G_m^A \text{ iff}$$
$$\forall\ i((1 \le i < i_s) \rightarrow s \in S_{G_i}) \wedge s \notin S_{G_{i_s}}$$

The new heuristic function can now be informally defined. When a state, say s, 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 s and in all the component goals of the effective goal subsequence G. In the simplified problem for an object o, 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 $A^+(s)$ is the maximum of the three values computed in the same way as before.

The heuristic function h+*($)* is also admissible and monotonic. Furthermore, it provides consistently tighter

heuristic estimation than the original function *h(s)*. The proof is omitted. With the new heuristic function, the search becomes very efficient. As can be seen from Figure 2(a), for the problem given in Figure 1, without subgoal ordering and goal augmentation the search can go 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 subgoal ordering and goal augmentaion, 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 6 modes are expanded.

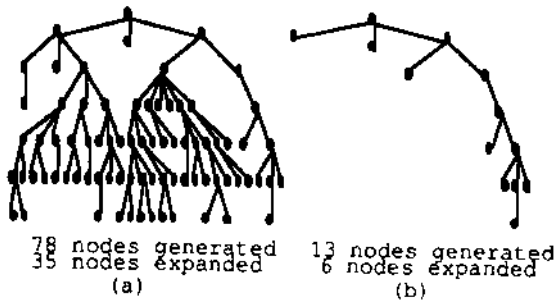

Figure 2. Comparison of Search Tree Diagrams
(a) without subgoal ordering and goal augmentation
(b) with subgoal ordering and goal augmentation

7. Example

In this section, we give an example to illustrate the application of our proposed integrated problem solving approaches. This example is also a robot navigation planning problem. The problem is described in Figure 3 and specified by the problem model formalism. The results are shown for each stage and finally, the performance of the search, with and without subgoal ordering and goal augmentation, are compared.

There are seven problem objects in this problem, one *Robot*, one *box* and five *doors*. The problem has three aspects. Two of them are the same as in the previous example and we still use functions *Inroom* and *Nextto* to represent them. A new problem aspect is the status of a door, for which we represent by the function *D_state*. *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'.

There are six types of actions in this problem that are modeled as rules. The costs of all rule applications are assumed to be the same 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: $(\forall\ dx(Nextto(dx,s_2)=\{\}))\ \wedge$
$(Nextto(Robot,s_2)=\{bx\})\ \wedge\ (Nextto(bx,s_2)=\{robot\})$

R_2 *GOTO*(dx):
IF: $\exists\ rx,ry[(Inroom(Robot,s_1)=rx)\ \wedge(Connect(dx,rx,ry))]$

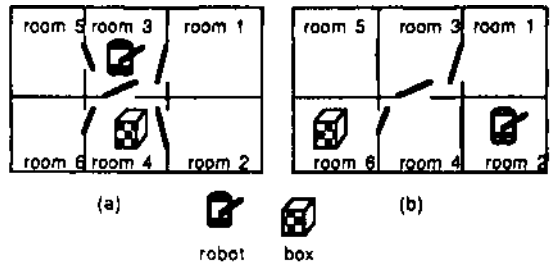

Figure 3. A Robot Planning Example
(a). An initial state (2). A goal state

THEN: $(\forall\ obj \neq dx)(Nextto(obj,s_2)=\{\})\ \wedge$
$(Nextto(Robot,s_2)=\{dx\})\ \wedge(Nextto(dx,s_2)=\{Robot\})$

R_3 *GOTHROUGH*(dx,ry):
IF: $\exists\ rx[(Inroom(Robot,s_1)=rx)\ \wedge(D\_state(dx,s_1)=open)$
$\wedge\ (Connect(dx,rx,ry))]$
THEN: $(Inroom(Robot,s_2)=ry)\ \wedge(\forall\ obj)(Nextto(obj,s_2)=\{\}))$

R_4 *PUSHTHROUGH*(bx,dx,ry):
IF: $(Nextto(Robot,s_1)=\{bx\})\ \wedge(D\_state(dx,s_1)=open)$
$\wedge(\exists\ rx[(Inroom(Robot,s_1)=Inroom(bx,s_1)=rx))\ \wedge$
$Connect(dx,rx,ry)]$
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: $(D\_state(dx,s_1)=open)\ \wedge(Nextto(robot,s_1)=\{dx\})$
$\wedge(\exists\ rx,ry[(Inroom(Robot,s_1)=rx)\ \wedge Connect(dx,rx,ry)])$
THEN: $D\_state(dx,s_2)=closed$

The initial state s_{in} is specified as follows:
$(Inroom(Robot,s_{in})=room_3)\ \wedge(Inroom(box,s_{in})=room_4)$
$\wedge(D\_state(door_{13},s_{in})=\cdots=(D\_state(door_{46},s_{in})=open)$

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

In this problem, there exist inherent ordering constraints among subgoals and implicit goal state information in the problem specification. By applying subgoal ordering and goal augmentation strategies, we can detect and make use of the underlying information to make search efficient.

With the procedure SOC, the relation $\dot{<}$ is derived to be $\{<g_1,g_4>,<g_2,g_1>,<g_3,g_1>\}$. Applying *GOAL_ORDER* to the set of subgoals with the relation $\dot{<}$, we then 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.

$$G_1' = (Inroom(box, s_{G_1}) = room_8) \wedge (D\_state(d_{35}, s_{G_1}) = closed)$$
$$\wedge [(D\_state(d_{46}, s_{G_1}) = open) \wedge (Inroom(Robot, s_{G_1}) = room_8)$$
$$\wedge (Nextto(Robot, s_{G_1}) = \{box\}) \wedge (Nextto(box, s_{G_1}) = \{Robot\})$$
$$\wedge (\forall\, dx\, (Nextto(dx, s_{G_1}) = \{\}) \vee [(Nextto(d_{35}, s_{G_1}) = \{Robot\})$$
$$\wedge (Inroom(Robot, s_{G_1}) \in \{room_3, room_4\})$$
$$\wedge (Nextto(Robot, s_{G_1}) = \{d_{35}\})$$
$$\wedge\ \forall\ (dx \neq d_{35})(Nextto(dx, s_{G_1}) = \{\}]$$

$$G_2' = (Inroom(Robot, s_{G_2}) = room_2) \wedge (D\_state(d_{24}, s_{G_1}) = open)$$
$$\wedge (\forall\, obj\, (Nextto(obj, s_{G_2}) = \{\})$$

$$G_3' = (D\_state(d_{24}, s_{G_1}) = closed) \wedge (Inroom(Robot, s_{G_2}) = room_2)$$
$$\wedge (Nextto(Robot, s_{G_1}) = \{d_{24}\}) \wedge (Nextto(d_{24}, s_{G_1}) = \{Robot\})$$

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.

s. 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 outlined 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.l, January 1982. pp. 1-23.

(ErN69l 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.

[IrY87] Irani, K.B. and Yoo, S. I., "A Methodology for Solving Problems: Problem Modeling and Automatic Heuristic Generation". Submitted to *IEEE Transactions on PAW,* 1987.

[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.