Exploiting Limited Interactions in Plan Optimization

by

Qiang Yang, Dana S. Nau
and James Hendler

Research Report CS-89-33
August, 1989

Exploiting Limited Interactions in Plan Optimization[*]

Qiang Yang[†] Dana S. Nau[‡ § ¶] James Hendler[‡¶§]

Abstract

Past planning systems have generally focused on control structures capable of working in all domains (domain-independent planning) or on specific heuristics for a particular applied domain (domain-dependent planning). An alternate approach is to abstract the kinds of goal and subgoal interactions that occur in some set of related problem domains, and develop planning techniques capable of performing relatively efficiently in all domains in which no other kinds of interactions occur. In this paper we will demonstrate this approach on a particular formulation of multiple-goal planning problems.

In particular, we demonstrate that for cases where multiple-goal planning can be performed by generating individual separate plans for each goal independently and then optimizing the conjunction, we can define a set of limitations on the allowable interactions between goals that allow efficient planning to occur where the restrictions hold. We demonstrate algorithms which are efficient for certain cases of this multiple goal planning approach, and propose a heuristic search algorithm that performs well. We further argue that these restrictions are satisfied across a significant class of planning domains.

The authors' email addresses are qyang@dragon.waterloo.edu, nau@mimsy.umd.edu, and hendler@mimsy.umd.edu.

---

[*]This work was supported in part by an NSF Presidential Young Investigator award to Dr. Nau with matching funds from Texas Instruments and General Motors Research Laboratories, in part by NSF Grant NSFD CDR-88003012 to the University of Maryland Systems Research Center, and in part by ONR grant N00014-88-K-0560 to Dr. Hendler.

[†]Computer Science Department, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

[‡]Computer Science Department, University of Maryland, College Park, Md. 20742, USA

[§]Systems Research Center, University of Maryland.

[¶]Institute for Advanced Computer Studies, University of Maryland.

1. Introduction

One of the most widely used strategies in problem-solving is to decompose a complex problem into several simpler parts. This is particularly true in planning, where a complicated goal is usually decomposed into two or more subgoals to solve. The reason for this is that decomposition tends to divide the exponent of an exponential problem, thus drastically reducing the total problem-solving effort. Korf [14], for example, has demonstrated that if the subgoals are independent, then solving each one in turn will divide both the base and the exponent of the complexity function by the number of subgoals.

The major limitation of the above approach is that although it treats the goals as independent, this condition does not really hold for most AI problems. Instead, the goals or subgoals may interact or conflict with each other.[1] Unfortunately, it appears impossible to achieve both efficiency and generality in handling goal/subgoal interactions. Domain-independent planners attempt to handle interactions which can occur in many possible forms, and thus they sacrifice the gains in efficiency which might possibly be achieved in domains where these interactions are limited. Domain-dependent planners can often do better at dealing with goal/subgoal interactions in their particular domains by imposing domain-dependent restrictions on the kinds of interactions that are allowed—but the restrictions they use are often too restrictive for the planners to be applicable to other domains.

In this paper, we propose an approach which falls in between domain-dependent and domain-independent planning: to abstract the kinds of goal and subgoal interactions that occur in some set of related problem domains, and develop planning techniques capable of performing relatively efficiently in all domains in which no other kinds of interactions occur. We will refer to this approach as *limited-interaction planning.*

The restrictions which we impose on the goal interactions allow us to develop relatively efficient techniques for solving multiple-goal planning problems by developing separate plans for the individual goals, combining these plans to produce a naive plan for the conjoined goal, and performing optimizations to perform to yield a better combined plan. For example, consider the following situation (based on [30]):

John lives one mile from a bakery and one mile from a dairy. The two

[1]The most famous example of this is the "Sussman anomaly," in which solving one goal undoes the independently derived solution to the other.

stores are 1.5 miles apart. John has two goals: to buy bread and to buy milk.

The approach usually taken is to conjoin this into the single goal

```
(GOAL JOHN (AND (HAVE BREAD) (HAVE MILK)))
```

Suppose that we have developed separate plans for the two individual goals (drive to the dairy, buy milk, and come home; and drive to the bakery, buy bread, and come home). Taken together, these two plans will solve the conjoined goal; and the next step is to recognize that the "come home" step of the first plan can be merged with the "get there" step of the second, to produce a better plan.

The restrictions required for our approach to be applicable are limiting, but not as severely limiting as the domain-dependent heuristics used by many application-specific planners. Our goal has been to develop restrictions with the following properties:

1. the restrictions are stateable in a clear and precise way (rather than simply referring to general knowledge about the characteristics of a particular domain of application);
2. the resulting classes of planning problems are large enough to be useful and interesting;
3. the classes of problems allowed are "well-behaved" enough that planning may be done with a reasonable degree of efficiency.

This paper provides background for the limited-interaction approach to planning, presents one set of restrictions satisfying the above criteria, and argues that these restrictions are satisfied across a significant class of planning domains. It also discusses the complexity of the resultant planning problems, and demonstrates that limited-interaction multiple-goal planning can be performed efficiently under these restrictions.

2. Background

As pointed out in the introduction, one of the major problems with planning is how to handle interactions among goals or subgoals. One approach which has been used to circumvent this problem is the condition of *linearity*. This condition is satisfied in a planning problem if the subgoals are all independent and can be achieved sequentially in any arbitrary order. Linear planners start by generating plans for the subgoals as if the planning problem were linear. Thus, to use a linear planner in domains where subgoals or goals interact strongly, it is necessary to add ways to detect and resolve the conflicts. As an example, STRIPS [7] did linear planning for compound goals which are conjuncts of component goals.

An alternative assumption is that it is better *not* to order operators than to order them arbitrarily. This results in the *least-commitment strategy*, in which an order between two operators is not assigned unless absolutely necessary (for example, this could occur if an action for one goal deletes an expression that was a precondition of another goal or subgoal). The plan thus developed is a partially ordered set of actions. Most of the well-known planning systems (for example, [24, 27, 19, 29, 4], fall into this catagory.

Although the domain-independent nonlinear planners are more efficient in handling conflicts than their linear counterparts, there is still usually too much computation involved: the problem requires exponential time in most interesting cases [4]. Such extensive computation is usually not feasible for planning in real-world domains. This exponentiality is of particular difficulty to systems dealing with multiple goals: as more subgoals are added to a single conjoined goal, the solution time is drastically increased.

One way to tackle this problem is to use explicit domain knowledge to lessen the computational burden of detecting and resolving the goal interactions in planning. Such domain-dependent planning systems have been built for many practical problems. Some recent examples include

1. Military command and control planning applications [1, 11, 3];

2. Route planning [15, 10];

3. Autonomous vehicle navigation [2, 16];

4. Automated manufacturing [5, 6, 9, 20].

Another approach is to restrict the number of goals among which interactions can occur. Vere's DEVISER [29] approached this problem by using temporal scopings associated with goals and actions. Much of the planning behavior in the DEVISER system involved setting up temporal constraints and comparing them to the durations.

Vere's approach can be thought of as a special case of a more general idea: that by placing appropriate kinds of restrictions on goal and subgoal interactions, it may be possible to achieve efficient planning across a range of applications. This idea, which we call "limited-interaction planning," is the primary motivation behind the current paper.

One example of this limited-interaction approach can be found in multiple goal planning problems. We consider the situation where, instead of generating a single conjoined goal, plans are generated for each goal concurrently, and then an "optimization" step is performed to merge the resultant plans into a single global plan. The problem of finding the best plan is still NP-hard, but provided that the interactions in the plans satisfy our restrictions, we have heuristic methods which appear to perform quite efficiently in the average case. For this approach to be useful, the set of planning problems satisfying our allowable set of interactions must be broad enough to be useful and interesting—and we argue that this is the case.

3. Problem Statement

In this paper, a plan is defined to be a partially ordered set of actions. Each action has a set of preconditions and postconditions[2]. Actions can have costs, and the cost of a plan is the sum of the costs of the actions.

Let G be a goal which is the conjunction of a number of other goals $G_1, G_2, \ldots, G_g$. For the example given in Section 1, `(HAVE BREAD)` and `(HAVE MILK)` are both goals for the conjunctive goal `(AND (HAVE BREAD) (HAVE MILK))`. One way to try to achieve G would be to achieve each of the individual goals independently, and try to combine the plans for the subgoals into a "global plan" for G. In this paper, we assume that the plans for the individual goals have already been found, and we look at how to combine them into a global plan.

Depending on what kinds of interactions occur among the actions in the plans, it might or might not be possible for the plans to be combined. In this paper, we consider only the following kinds of interactions.

1. Let A be a set of actions $\{a_1, a_2, \ldots, a_n\}$. Then there may be a *merged action* $m(A)$ capable of accomplishing the effects of all actions in A. The cost of $m(A)$ could be either higher or lower than the sum of the costs of the other actions—but it is only useful to consider merging the actions in A if this will result in a lower total cost. Thus, although we allow the case where

[2]As discussed later in the paper, a plan may also be associated with certain constraints, such as one that requires two or more actions to occur at the same time.

$\text{cost}(m(A)) \geq \sum_{a \in A} \text{cost}(a)$, we can ignore it for the purposes of planning. Thus, we only consider A to be mergeable if $\text{cost}(m(A)) < \sum_{a \in A} \text{cost}(a)$; and in this case we say that an *action-merging interaction* occurs.

One way in which an action-merging interaction can occur is if the actions in A contain various sub-actions which cancel each other out, in which case the action $m(A)$ would correspond to the set of actions in A with these sub-actions removed. If the cost of each action is the sum of the costs of its sub-actions, then the cost of $m(A)$ is clearly less than the sum of the costs of the actions in A.

Note that even though a set of actions may be mergeable, it may not always be possible to merge that set of actions in a given plan. For example, suppose a and a' are mergeable, but in the plan P, a must precede b and b must precede a'. Then a and a' cannot be merged in P, because it would require b to precede itself.

2. An *action-precedence* interaction is an interaction which requires that an action a in some plan P_i must occur before an action b in some other plan P_j. This can occur, for example, if b removes one of the preconditions necessary for a, and there is no other action which can be inserted after b to restore this precondition.

   Much previous work in planning has dealt with deleted-condition interactions. Some action-precedence interactions are expressible as deleted-condition interactions, and conversely, some deleted-condition interactions can be resolved by imposing precedence orderings. Deleted-condition interactions can often be resolved in other ways as well—and thus, in general, they are more difficult to deal with than action-precedence interactions. However, there is a significant class of problems, including certain kinds of automated manufacturing problems and certain kinds of scheduling problems, where action-precedence interactions are the only form of deleted-condition interactions. Examples of such problems appear later in this section.

3. Plans for different goals may sometimes contain some of the same actions. The *identical-action* interaction occurs when an action in one plan must be identical to an action in one of the other plans.

4. Sometimes, two different actions must occur at the same time. We call such an interaction a *simultaneous-action interaction.* This is different from

the identical-action interaction, because these simultaneous actions are not identical. An example would be two robotic hands working together in order to pick up an object.

Depending on what interactions appear in a given planning problem, it may or may not be possible to combine the plans into a global plan. Of the kinds of interactions disussed above, three of them (the action-precedence, identical-action, and simultaneous-action interactions) place constraints on how a set of plans might be combined into an overall global plan. The other kind of interaction (the action-merging interaction) places no constraint on how the plans might be combined, but instead allows possible optimizations of the global plan once it has been created. Thus, the only kinds of interactions which might make it impossible to combine a set of plans into a global plan are the action-precedence, identical-action, and simultaneous-action interactions. The problem of finding out whether or not a set of plans can be combined into a global plan we call the *multiple-goal plan existence problem.*

As an added complication, each goal G_i may have several alternate plans capable of achieving it, and thus there may be several different possible identities for the global plan for G. The least costly plan for G_i is not necessarily part of the least costly global plan, because some more costly plan for G_i may be mergeable in a better way with the plans for the other goals. For example, Fig. 1 shows the results of merging a plan P_1 with two different plans P_2 and P_2'.

We define the *multiple-goal plan optimization problem* to be the problem of choosing which plan to use for each goal, and which actions to merge in these plans, so as to produce the least costly global plan for G.

Problems involving optimizing multiple-goal plans occur in a large number of interesting problem domains, such as automated manufacturing and factory scheduling. In these domains multiple goals must be achieved within the context of a set of constraints (deadlines, machining requirements, etc.) The general class of *all* such problems clearly will not fit within the confines of the restrictions specified in this paper (for example, we have not yet extended our approach to deal with scheduling deadlines), but significant and useful classes of problems can be found which satisfy these restrictions. Several specific examples are given below.

Example 1. Consider again the shopping example given in Section 1, in which John has two goals: `(HAVE BREAD)` and `(HAVE MILK)`. To achieve the `(HAVE BREAD)` goal, a plan could be:

`(GO HOME BAKERY), (BUY BREAD), (GO BAKERY HOME)`

To achieve the `(HAVE MILK)` goal, a plan could be:

`(GO HOME DAIRY), (BUY MILK), (GO DAIRY HOME).`

If it takes less time to go between the bakery and the dairy than to go back from the bakery and then to the dairy, then the action `(GO BAKERY HOME)` can be merged with the actions in the second plan. The result is a cheaper overall plan:

```
(GO HOME BAKERY), (BUY BREAD), (GO BAKERY DAIRY),
(BUY MILK), (GO DAIRY HOME).
```

Example 2. Consider the automated manufacturing problem of drilling holes in a metal block. Several different kinds of hole-creation operations are available (twist-drilling, spade-drilling, gun-drilling, etc.), as well as several different kinds of hole-improvement operations (reaming, boring, grinding, etc.). Each time one switches to a different kind of operation or to a hole of a different diameter, one must put a different cutting tool into the drill. Suppose it is possible to order the operations so that one can work on holes of the same diameter at the same time using the same operation. Then these operations can be merged by omitting the task of changing the cutting tool. This and other similar manufacturing problems are of practical significance (see [20, 9]) and, in fact, much of the work in this paper derives from our ongoing work in developing of a computer system for solving such problems.

Suppose hole h_1 can be made by the plan

P_1: spade-drill h_1, then bore h_1;

and hole h_2 can be made by either of the plans

P_2: twist-drill h_2, then bore h_2;
P_2': spade-drill h_2, then bore h_2;

with $\text{cost}(P_2) < \text{cost}(P_2')$. If h_1 and h_2 have different diameters, then the least costly global plan will be to combine P_1 and P_2. However, if they have the same diameter, then a less costly global plan can be found by combining P_1 and P_2', merging the two spade-drilling operations, and merging the two boring operations.

Example 3. In a machine shop, consider the problem of finding a minimum-time schedule for satisfying some set of orders for products that can be produced in the shop. For each order, there may be a set of alternative schedules for producing it, and each such schedule consists of a set of operations to be performed on various machines.

An operation in a schedule is usually associated with a machine for carrying it out. If two or more operations require the same type of set-up, then doing them on the same machine may reduce the total time required—and thus reduce the total time required to complete all the schedules. In this case, we consider these operations as mergeable.

4. Solving the Problem

In this paper we consider two different cases of the multiple-goal plan optimization problem. The first case is where a single plan is generated for each goal. In this case, there is a set of restrictions which defines a class of problems that is reasonably large and interesting, but which can be solved in low-order polynomial time.

The second case is where more than one plan may be generated for each goal—necessitating choosing among the plans available for each goal in order to find an optimal global plan. This case is NP-hard, but there is a heuristic approach which works well in practice on this problem.

4.1. One Plan for Each Goal

Planning is often so difficult that most planning systems stop once they have found a single plan for each goal, without trying to find other plans as well. This section discusses the multiple-goal plan optimization problem in the case where only one plan is available for each goal.

4.1.1. Complexity

In analyzing the computational complexity of the problem, if we can show that a special case is NP-hard, it follows that the general case is also NP-hard. To do this, we consider the special case in which the following conditions hold: for each goal G_i, the plan P_i for G_i is a linear sequence of actions, each action has a cost of 1, and there are no interactions among the plans except for action-merging interactions.

In this special case, all interactions that might prevent the plans from being combined into a global plan have been disallowed. Thus, at least one global plan is guaranteed to exist: $\Pi = \bigcup_{i=1}^{g} P_i$. However, Π may not be an optimal plan, because it may be possible to merge some of its actions. In fact, Π may contain several different sets of mergeable actions, and merging some of them may preclude merging others. Different choices of which sets to merge may result in plans of different cost. For example, Fig. 2 shows two plans P_1 and P_2, and the results of merging them in different ways.

Since this special case requires that each P_i be a linear sequence of actions and each action have the same cost, the problem of finding the best global plan is equivalent to the problem of finding the shortest common supersequence (SCS) of n sequences. This problem has been shown to be NP-hard [18]. Since the special case is NP-hard, it follows that the multiple-goal plan optimization problem is NP-hard.

4.1.2. Plan Existence

The last section showed that the multiple-goal plan optimization problem with one plan per goal is NP-hard, by showing that a special case of that problem is also NP-hard. We now return to the general case, in which all of the interactions defined in Section 3 are allowed.

One way of handling an NP-hard problem is to simplify it either by relaxing the criteria for what constitutes a valid solution, or by imposing restrictions on what problem instances will be considered. For the current problem, one way to do this is to look not for the optimal global plan, but for any global plan that works—in other words, to solve the multiple-goal plan existence problem rather than the multiple-goal plan optimization problem. Whether a global plan exists is independent of whether there are any action-merging interactions, so for the multiple-goal plan existence problem we can ignore all action-merging interactions completely.

In particular, suppose that we are given the following:

1. A set of plans $S = P_1, P_2, \ldots, P_g$ containing one plan P_i for each goal G_i. Let n be the total number of actions in S.

2. A list of interactions among the actions in the plans (for example, members of this list might be statements such as "action a in plan P_i must precede

action b in plan P_j"). Let i be the total number of interactions in this list (note that $i = O(n^2)$).

Unless the interactions prevent the plans in S from being merged into a global plan, the global plan is just the set of individual plans in S, with additional ordering constraints imposed upon the actions in these plans in order to handle the interactions. This *combined* plan is called combine(S), and the following algorithm will produce it.

Note that combine(S), if it exists, is a unique, but it is not a totally ordered plan. Every valid embedding of combine(S) within a total ordering is guaranteed to be a valid plan, and Step 3 of the algorithm can easily be modified to produce all of these embeddings.

Algorithm 1.

1. For each plan P in S, create a graph representing P as a Hasse diagram.[3] Also, create a sorted linear index of the actions in the plans. This step can be done in time $O(n^2)$.

2. For each action-precedence interaction in the interaction list, modify the graph by creating a precedence arc between the actions named in the interaction. For each simultaneous-action interaction in the interaction list, create a simultaneous-action arc between the actions named in the interaction. For each identical-action interaction in the interaction list, combine the actions named in the interaction into a single action. If this step is done by sorting the interaction list and then checking it against the index of actions, it can be done in time $O(i \log i + (i+n)n) = O(n^3)$.

3. Check to see whether the graph still represents a consistent partial ordering (this can be done in time $O(n^2)$ using a topological sorting algorithm [13], with a straightforward extension to handle the simultaneous-action interactions). If it does not, then exit with failure (no global plan exists for this problem).

Algorithm 1 produces the combined plan combine(S) if it exists, in the case where there is one plan for each goal G_i. The total time required is $O(n^3)$, where n is the total number of actions in the plans.

[3]This is a standard representation of a partially ordered set (e.g., see [23]).

4.1.3. Plan Optimality

In order to avoid the NP-hardness of the general problem, Section 4.1.2 attacked the simpler problem of trying to find any plan that will work, rather than an optimal plan. If we want an optimal plan, then the task is much more difficult. This section considers some restrictions that make it feasible to look for an optimal global plan, rather than just a consistent one.

Restriction 1. If S is a set of plans, then the set of all actions in S may be partitioned into equivalence classes of actions $E_1, E_2, \ldots, E_p$, such that sets of actions A and B are mergeable if and only if A and B are subsets of the same equivalence class.

Restriction 2. If combine(S) exists, then it defines a partial order over the equivalence classes defined in Restriction 1; i.e., if E_i and E_j are two distinct equivalence classes and if combine(S) requires that some action in E_i occur before some action in E_j, then combine(S) cannot require that some action in E_j occur before some action in E_i. (This does not rule out the possibility of an action in E_i occurring immediately before another action in E_i; in such a case, the two actions can be merged.)

Restriction 1 is reasonable for a number of problems (for example, it is already satisfied in the Examples 1, 2 and 3 discussed previously).

Intuitively, Restriction 2 requires that merging one set of actions in an equivalence class does not preclude the possibility of merging other actions in the plans. So, for example, the plans in Fig. 2 do not satisfy Restriction 2, since merging actions $B1$ and $B2$ will preclude merging actions $A1$ and $A3$. Although this restriction is more limiting, it still allows many interesting problems. For instance, Restriction 2 is trivially satisfied in Example 1 since there is only one possible merge. In Example 2 it is satisfied in a more interesting way, since there exists a common sense ordering of the machining operations (e.g., don't twist-drill a hole after it has been bored, or the class of milling operations always precedes the class of drilling operations) which is quite natural to use for this problem. In Example 3, it may or may not be satisfied, depending on the particular scheduling problem being considered. For example, if the problem is to schedule setups of machinable parts on various machine tools for various machining operations, Example 3 satisfies Restriction 2 for the same kind of reason as Example 2.

Suppose the above restrictions are satisfied, and suppose we are given a set of plans S and a list of interactions, as was done in Section 4.1.2. If a global plan exists, then Algorithm 1 will produce the global plan combine(S). However, by merging some of the actions in combine(S), it may be possible to find other less costly plans. The following algorithm will find a least costly plan.

Algorithm 2.

1. Use Algorithm 1 to produce a digraph representing the combined plan combine(S). This can be done in time $O(n^3)$. If Algorithm 1 succeeds, then continue; otherwise, exit with failure.

2. For each equivalence class E_i of actions in combine(S), merge all of the actions in E_i. This produces a digraph in which each class of action occurs only once (e.g., see Fig. 3). From Restriction 2, it follows that this is a consistent plan; we call this plan merge(combine(S)). From Restriction 1 and the definition of mergeability, it can be proved by induction that this is the least costly plan which can be found by combining and merging actions in S. Merging the classes will, at worst, require redirecting all of the arcs in the digraph—and this can be done in time $O(n^2)$.

In the case where there is one plan for each goal G_i, Algorithm 2 produces an optimal way to combine and merge these plans if it is possible to combine them at all. The total time required is $O(n^3)$, where n is the total number of actions in the plans.

4.2. More than One Plan for Each Goal

For some multi-goal planning problems, it is reasonable to expect that more than one plan may be found for each goal. (For example, this is done by the SIPS planning system for the manufacturing problem discussed in Example 2 [20]). Finding more than one plan for each goal is more complex computationally than finding just one plan for each goal, but it is useful because it can lead to better global plans.

To see this, consider once again the planning situation described in Section 1:

> John lives one mile from a bakery and one mile from a dairy. The two stores are 1.5 miles apart. John has two goals: to buy bread and to buy milk.

This time, however, let us add the fact that

John lives 1.25 miles from a large grocery store (based on [30]):

The best plans for the individual goals involve two separate trips: one to the store and one to the dairy. Given these plans, the approach described in the previous section would merge them so as to allow John to go directly from one store to the other. The best global plan, however, is to use the second-best plan for each goal (going to the grocery store), since this allows greater merging. If the planners for the individual goals deliver more than one solution for each goal, this better plan may be found.

4.2.1. Complexity

If more than one plan is available for each G_i, then there may be several different possible identities for the set S discussed in Section 4.1, and it may be necessary to try several different possibilities for S in order to find one for which combine(S) exists. This problem is NP-hard, even with Restrictions 1 and 2; this is proved in the Appendix by reducing CNF-satisfiability to it.

Since the multiple-goal plan existence problem is a special case of the multiple-goal plan optimization problem, the above result means that the multiple-goal plan optimization problem is also NP-hard.

Polynomial-time solutions do exist for several special cases of the multiple-goal plan existence and optimization problems. For plan existence, one such special case is the one discussed in Section 4.1, in which the number of plans for each goal was taken to be 1. For plan optimization, such a special case occurs if the number of different equivalence classes of actions is less than 3, and each action has the same cost. In this case, if no conflicting constraints are allowed to exist, the multiple-goal plan optimization problem can be solved in polynomial time, even if Restriction 2 is lifted. For example, this would occur in Example 2 if there were only two different kinds of machining procedures to be considered.

4.2.2. A Heuristic Algorithm

Although the general case of the multiple-goal plan optimization problem is NP-hard, there is a heuristic approach that performs well in practice when Restrictions 1 and 2 are satisfied. The approach is to formulate the problem as a state-space search and solve it using a best-first branch-and-bound algorithm.

Suppose that we are given the following: (1) for each goal G_i, a set of plans T_i containing one or more plans for G_i, and (2) a list of the interactions among the actions in all of the plans. In the state-space search, the state space is a tree. Each state is a set of plans; it contains one plan for each of the first i goals for some i. The initial state is the empty set (i.e., $i = 0$). If S is a state containing plans for the goals $G_1, G_2, \ldots, G_i$, then an immediate successor of S is any set $S \cup \{P\}$ such that P is a plan for G_{i+1}. A goal state is any state in which plans have been chosen for all of the goals $G_1, G_2, \ldots, G_g$. The cost of a state S is the cost of the plan obtained by applying Algorithm 2 to S; i.e.,

$$\text{cost}(S) = \text{cost}(\text{merge}(\text{combine}(S))).$$

Fig. 4 displays part of an example state space.

The search algorithm appears below. This algorithm is a best-first branch-and-bound search, which uses a lower bound function L to order the members of the list of alternatives being considered. Except for the use of U for pruning, this algorithm can also be thought of as a version of the A* search procedure, with $h(S) = L(S) - \text{cost}(S)$ as the heuristic function.

Algorithm 3.

```
  $A := (\emptyset)$          *(A is the branch-and-bound active list)*
  $U :=$ upper bound, computed as described in the text
  **loop**
        $S := \text{pop}(A)$          *(remove the first element of the list)*
        **if** $S$ is a goal state **then return** $S$
        **if** $L(S) \leq U$ **then begin**
           $B :=$ the successors of $S$, in order of least $L$-value first
           $A :=$ the result of merging the list $A$ with the list $B$
              *(Here we are doing merging of lists, not merging of actions.*
              *This is to maintain A in order of least L-value first.)*
        **end**
  **repeat**
```

In the search algorithm, pruning may be done by computing an upper bound on the cost of the best global plan. For each G_i, let $\text{best}(G_i)$ be the plan for G_i of least cost. The plans $\text{best}(G_i)$, $i = 1, 2, \ldots, g$, may or may not be able to be combined, depending on the interactions among them. If they cannot be combined, then U can be initialized to ∞. However, if $\text{combine}(\{\text{best}(G_1), \text{best}(G_2), \ldots, \text{best}(G_g)\})$ exists, then the upper bound is

$$U = \text{cost}(\text{merge}(\text{combine}(\{\text{best}(G_1), \text{best}(G_2), \ldots, \text{best}(G_g)\}))). \qquad (4.1)$$

Whether U exists—and if so, what its value is—can be determined in time $O(m^3)$ using Algorithm 2, where m is the total number of actions in $\{\text{best}(G_1), \text{best}(G_2), \ldots, \text{best}(G_g)\}$. During the search, any state S such that $L(S)$ is greater than U is discarded.

If $L(S)$ is a lower bound on the costs of all successors of S that are goal states, then L is admissible, in the sense that Algorithm 3 will be guaranteed to return the optimal solution. We now discuss various possible functions to use for L. To do this, we temporarily assume the following property: that merging plans for two different goals always results in a plan at least as expensive as either of the two original plans. In other words, if P and Q are plans for two distinct goals, then

$$\text{cost}(\text{merge}(\text{combine}(P, Q))) \geq \max(\text{cost}(\text{merge}(P)), \text{cost}(\text{merge}(Q))). \quad (4.2)$$

We will later discuss what happens when this property is not satisfied.

If Eq. (4.2) is satisfied, then clearly $L_0(S) = \text{cost}(S)$ is a lower bound on the cost of any successor of S (this would correspond to using $h \equiv 0$ in the A* search algorithm). However, a better lower bound can be found as follows. Suppose S contains plans for $G_1, \ldots, G_i$. For each $j > i$, let $P^*(S, j)$ be the plan P for G_j which minimizes $\text{cost}(\text{merge}(\text{combine}(S \cup \{P\})))$. Let

$$L_1(S) = \max_{j>i} \text{cost}(\text{merge}(\text{combine}(S \cup \{P^*(S,j)\}))). \quad (4.3)$$

To show that L_1 is admissible, let S be any state, and let S' be any descendant of S that is a goal state. We must show that $L_1(S) \leq \text{cost}(S')$. From Eq. (4.3), there is some $j > i$ such that

$$L_1(S) = \text{cost}(\text{merge}(\text{combine}(S \cup \{P^*(S,j\}))). \quad (4.4)$$

But S' contains some plan P_j for the goal G_j, and from the definition of $P^*(S, j)$,

$$\text{cost}(\text{merge}(\text{combine}(S \cup \{P^*(S,j\}))) \leq \text{cost}(\text{merge}(\text{combine}(S \cup \{P_j\}))). \quad (4.5)$$

Thus, by repeated application of Eq. (4.2),

$$L_1(S) \leq \text{cost}(\text{merge}(\text{combine}(S'))), \quad (4.6)$$

so L_1 is admissible.

One way to find $P^*(S, j)$ is to compute $\text{merge}(\text{combine}(S \cup \{P\}))$ for each plan $P \in T_j$ using Algorithm 2, and then select the plan P that yields the minimum cost. If P is a plan, let $a(P)$ be the set of actions in P. If the above approach is used to compute $L_1(S)$, the time required is

$$\sum_{j=i+1}^{g} \sum_{P \in T_j} O((|a(S)| + |a(P)|)^3). \tag{4.7}$$

During this computation, it may be realized that some P cannot be combined with S, due to conflicting constraints. In that case, the plan P can be removed from further consideration, simplifying the computation of L_1 on successors of S. Also, if none of the plans in T_j can be combined with S, then there is no possible way that S can be extended into a complete global plan, so search can be discontinued at S.

By sacrificing the quality of the lower bound somewhat, we can compute it more efficiently. We associate with each state S some sets $H_1(S), H_2(S), \ldots, H_g(S)$, which are computed as follows. For the initial state $(S = \emptyset)$, for $j = 1, 2, \ldots, g$,

$$H_j(S) = \{a(P) | P \text{ is a plan for } G_j\}. \tag{4.8}$$

Let S be any state at level $i - 1$, and let S' be the state formed by including a plan P_i for the goal G_i. Then, for $j = i + 1, \ldots, g$,

$$H_j(S') = \{Q' | Q \in H_j(S)\}, \tag{4.9}$$

where Q' is Q minus each action which falls into the same equivalence class as some action in P_i. We now define

$$L_2(S') = \text{cost}(\text{merge}(\text{combine}(S')) + \max_{j=i+1}^{g} \min\{\text{cost}(Q) | Q \in H_j(S')\}, \tag{4.10}$$

where cost(Q) is the sum of the costs of the actions in Q, and where the min of an empty set is taken to be 0. It is left as an exercise for the reader to show that L_2 is admissible.

Computing $L_2(S)$ takes time

$$\sum_{j=i+1}^{g} \sum_{V \in H_j(S)} O(|V| + |a(P_i)|) \tag{4.11}$$

This is much less than the time required for computing $L_1(S)$, for several reasons:

1. Comparing the expression $|V| + |a(P_i)|$ in Eq. (4.11) to the expression $|a(S)| + |a(P)|$ in Eq. (4.7), $|a(P_i)|$ is about the same as $|a(P)|$ assuming that the size of the plans are not very different, but $|V|$ is much smaller than $|a(S)|$. Furthermore, the expression $|a(S)| + |a(P)|$ is cubed in Eq. (4.7), and the expression $|V| + |a(P_i)|$ is not raised to a power in Eq. (4.11).

2. Initially, T_j and H_j have the same number of elements, which is the number of alternate plans for goal G_j. For increasing values of i, the size of $H_j(S)$ decreases, so that less elements are summed in Eq. (4.11). But in Eq. (4.7), the size of T_j is independent of i. Therefore, $|H_j|$ is always less than or equal to $|T_j|$.

L_2 is usually less informed than L_1, because it only takes into account the actions which are not in the same equivalence classes as any actions in S'. However, if all actions have the same cost, then $L_2 = L_1$.

Some problems (e.g., Example 2) satisfy the restriction given in Eq. (4.2), and others do not. If Eq. (4.2) is not satisfied, then there will be some states S such that $L_0(S)$, $L_1(S)$, and $L_2(S)$ are not lower bounds on S. For most reasonable planning problems, S can be shown to be dominated by states on other paths in the search tree, in which case Algorithm 3 is still guaranteed to return a least-cost solution (for a mathematical analysis of such cases, see [21]). In other cases, such S might not be dominated by states on other paths, in which case Algorithm 3 might not return the optimal solution—but in this case, Algorithm 3 will still return results that are close to optimal (for a mathematical analysis of such cases, see [12]).

In the worst case, Algorithm 3 takes exponential time. Since the multiple-goal plan optimization problem is NP-hard, this is not surprising. What would be more interesting is how well Algorithm 3 does on the average. However, the structure of the multiple-goal plan optimization problem is complicated enough that it is not clear how to characterize what an "average case" should be; and there is evidence that the "average case" will be different for each application area. Therefore, we have restricted ourselves to doing empirical studies of Algorithm 3's performance on a class of problems that seemed to us to be "reasonable."

Fig. 5 shows experimental results of using this algorithm for planning in the automated manufacturing domain. The problem to be solved involved planning how to drill several holes in a piece of metal stock, as described in Example 2. For the test, the procedure was to create randomly generated sets of holes with varying machining requirements (such as depth, surface finish, etc.), and to generate a plan for each hole individually using EFHA [28], a successor to our SIPS process planning system [20]. For each set of holes, Algorithm 3 was invoked on the plans for these holes to produce a global plan which minimized the total time required for tool changes. As shown in Fig. 5, the number of possible states in the search space grew exponentially as a function of the number of holes in the set, but the number of states searched by Algorithm 3 with the heuristic function L_2 grew only linearly. Intuitively, this means that the portion of the search space explored by our search algorithm is a narrow region around the optimal path from the root of the search tree to the goal state.

5. Concluding Remarks

This paper has examined some ways in which planning can be done efficiently if the planning problem satisfies certain simplifying assumptions about the goal/subgoal interactions. A set of such assumptions has been developed for the the multiple-goal optimization problem. The main results are summarized below:

1. The multiple-goal plan optimization problem is NP-hard.
2. By imposing some restrictions that are reasonable for some problem domains, the problem can be made computationally easy when there is only one plan available for each goal.
3. Even with the restrictions, the problem is still NP-hard if each goal has multiple alternate plans, but in this case there is a good heuristic approach for solving the problem. A number of problems satisfy these restrictions, including some problems of practical importance [9, 20].

One major limitation of this paper is that it concentrates on how to combine plans which have already been developed for individual goals. In the application domains in which we have been working [20], we have developed domain-dependent techniques for developing plans for the individual goals—but an obvious question is whether there is a natural extension of our approach for creating plans rather than just optimizing existing plans. We are convinced that the answer to this question is "yes", and such an approach will probably not require any restrictions other than the ones discussed in this paper. In addition, it may be possible to develop similar techniques for use in planning or plan optimization in cases where the interactions satisfy other kinds of limitations instead of the specific ones described in this paper. We intend to explore this issue further.

Another question which remains to be answered is whether the particular restrictions discussed in this paper are too restrictive. For example, there may be reasonable ways to solve the multiple-goal plan optimization problem in the case where there are a limited number of violations of Restriction 2. In addition, relaxing these restrictions will not produce exponential behavior in every case. A further classification of these exceptions may lead to a less restrictive set of limitations. Moreover, in some problem domains, there may be reasonable ways to solve the multiple-goal plan optimization problem even when Restriction 2 is lifted. Some of these problem domains are of practical interest, and we are currently investigating how to characterize these domains.

Appendix: NP-Completeness of the Multiple-Goal Plan Existence Problem

The purpose of this appendix is to show that if there may be more than one plan for each goal, then the multiple-goal plan existence problem is NP-complete. To do this, we show that NP-completeness occurs in the special case where the only kind of goal interaction that occurs is the identical-action interaction.

It is easy to see that the problem is in NP, so the proof will be complete if the problem is shown to be NP-hard. We do this by reducing the CNF-satisfiability problem to it.

Given a set U of variables and a collection C of clauses over U, the CNF-satisfiability problem asks whether there is an assignment of truth values to the variables in U which satisfies every clause in C. To reduce this problem to the multiple-goal plan existence problem, we associate a goal G_i with each clause C_i of C. G is the conjunct of the individual goals G_i. For each literal $l_{ij} \in C_i$, we create a plan (a_{ij}, b_{ij}) for the goal G_i. If $l_{ij} = l_{kl}$, then we specify that a_{ij} and a_{kl} must be identical, and b_{ij} and b_{kl} must also be identical. If $l_{ij} = \neg l_{kl}$, then we specify that a_{ij} and b_{kl} must be identical, and b_{ij} and a_{kl} must also be identical.

It is easy to see that that this reduction can be computed in polynomial time. It remains to be shown that (1) if C is satisfiable, then there is a consistent global plan for G; and (2) if there is a consistent global plan for G, then C is satisfiable. These two statements are proved below.

1. Suppose there is an assignment of truth values to the variables in U which satisfies C. Then we construct a set S of plans, one for each goal G_i. For each i, the clause C_i in C contains some literal l_i^* in C_i whose value is TRUE;

we let S contain the corresponding plan (a_{ij}, b_{ij}). Suppose that the plans in S cannot be combined into a consistent global plan. Then there are two plans $p_i = (a_{ij}, b_{ij})$ and $p_k = (a_{kl}, b_{kl})$ such that a_{ij} and b_{kl} are constrained to be identical, and b_{ij} and a_{kl} are constrained to be identical. But this means that $l_i^* = \neg l_k^*$, violating our requirement that both l_i^* and l_k^* have the value TRUE. Thus, the plans in S can be combined into a consistent global plan.

2. Conversely, suppose there is a set of plans S which can be combined into a consistent global plan. Then we assign truth values to the variables in U as follows: for each variable $v \in U$, if its corresponding plan is in S, then assign it the value TRUE; otherwise, assign it the value FALSE. Since S can be combined into a consistent global plan, this means that no variable can receive both the values TRUE and FALSE. Furthermore, since S must contain at least one plan for each goal G_i, at least one literal in each clause will receive the value TRUE. Thus, this assignment of truth values satisfies C.

References

[1] T.C. Baker, J.R. Greenwood "Star: an environment for development and execution of knowledge-based planning applications" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[2] Berlin, M., Bogdanowicz, J. and Diamond, W. "Planning and control aspects of the scorpius vision system architecture" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[3] A. Brown and Gaucus, D. "Propsective Situation Assessment" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[4] D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence* (32), 1987, 333-377.

[5] T. C. Chang and R. A. Wysk, *An Introduction to Automated Process Planning Systems,* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[6] M. R. Cutkoski and J. M. Tenenbaum, "CAD/CAM Integration Through Concurrent Process and Product Design," *Proc. Symposium on Integrated*

and Intelligent Manufacturing at ASME Winter Annual Meeting, 1987, pp. 1-10.

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

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Bell Laboratories, Murray Hill, New Jersey, 1979.

[9] C. Hayes, "Using Goal Interactions to Guide Planning," *Proc. AAAI-87*, 1987, 224-228.

[10] Garvey, T. and Wesley, L. "Knowledge-based Helicopter Route Planning" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[11] D.P. Glasson, and J.L. Pomarede "Navigation Sensor Planning for Future Tactical Fighter Missions" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[12] L. R. Harris, "The Heuristic Search Under Conditions of Error," *Artificial Intelligence* (5), 1974, 217-234.

[13] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Weseley, Reading, Mass., 1968.

[14] Korf, R.E., "Planning as Search: A Quantitative Approach," *Artificial Intelligence* (33), 1987, 65-88.

[15] Korf, R.E. "Real-Time Path Planning" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[16] Linden, T., and Owre, S. "Transformational Synthesis Applied to ALV Mission Planning" *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[17] M. Luria, "Goal Conflict Concerns," *Proc. IJCAI*, 1987, 1025-1031.

[18] D. Maier, "The Complexity of Some Problems on Subsequences and Supersequences," *J. ACM* (25), 1978, 322-336.

[19] D. McDermott *Flexibility and Efficiency in a Computer Program for Designing Circuits*, AI Laboratory, Massachusetts Institute of Technology, Technical Report AI-TR-402, 1977.

[20] D. S. Nau, "Automated Process Planning Using Hierarchical Abstraction," Award winner, Texas Instruments 1987 Call for Papers on Industrial Automation, *Texas Instruments Technical Journal*, Winter 1987, 39-46.

[21] D. S. Nau, V. Kumar, and L. N. Kanal, paper in preparation.

[22] N. Nilsson, *Principles of Artificial Intelligence*, Chapters 7 and 8, Tioga Publishing Co., 1980.

[23] F. P. Preparata and R. T. Yeh, *Introduction to Discrete Structures*, Addison-Wesley, Reading, Mass., 1973.

[24] E. D. Sacerdoti, "A Structure of Plans and Behavior," *American Elsevier, New York,* 1977.

[25] M. Stefik, "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, (16), 1981, 111-140.

[26] G. Sussman, "A Computer Model of Skill Acquisition," *American Elsevier*, New York, 1982.

[27] A. Tate, "Generating Project Networks," *Proc. IJCAI*, 1977, 888-893.

[28] S. Thompson, "Environment for Hierarchical Abstraction: A User Guide," Tech. Report, Computer Science Department, University of Maryland, College Park, 1989.

[29] S. A. Vere, "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (PAMI-5:3), 1983, 246-247.

[30] R. Wilensky, *Planning and Understanding*, Addison-Wesley: Reading, Massachusetts, 1983.

[31] D. Wilkins, "Domain-independent Planning: Representation and Plan Generation," *Artificial Intelligence*, (22), 1984.

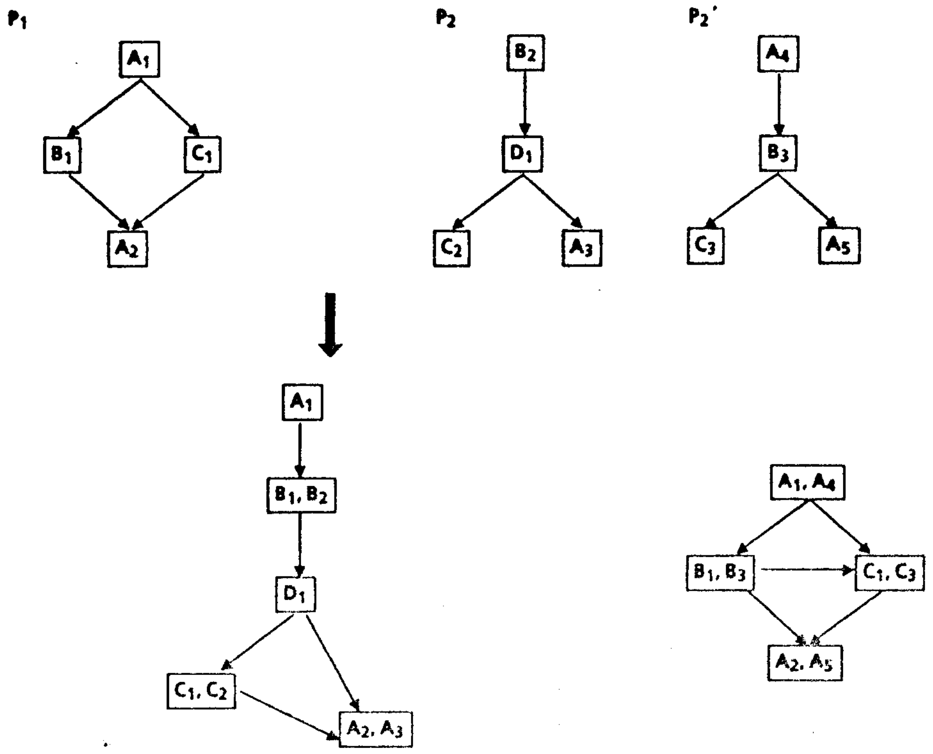

Figure 1: Results of merging a plan P_1 with each of two alternative plans P_2 and P_2'.

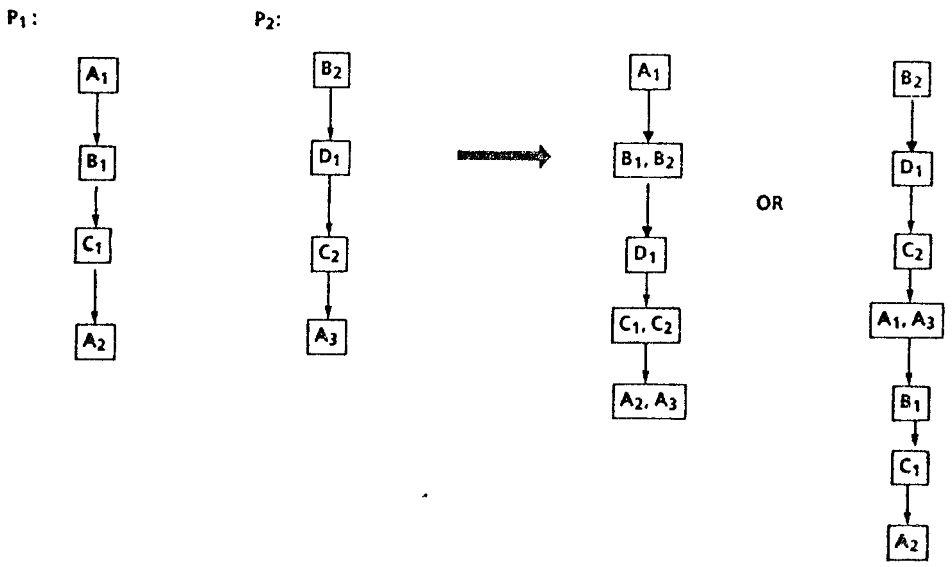

Figure 2: Two plans that can be merged in different ways. Actions are mergeable if their names begin with the same letter.

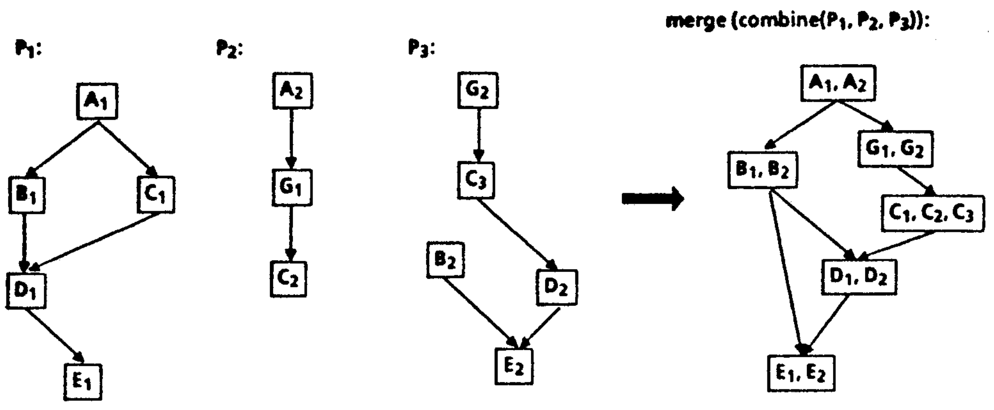

Figure 3: Finding an optimal global plan by merging plans for various goals.

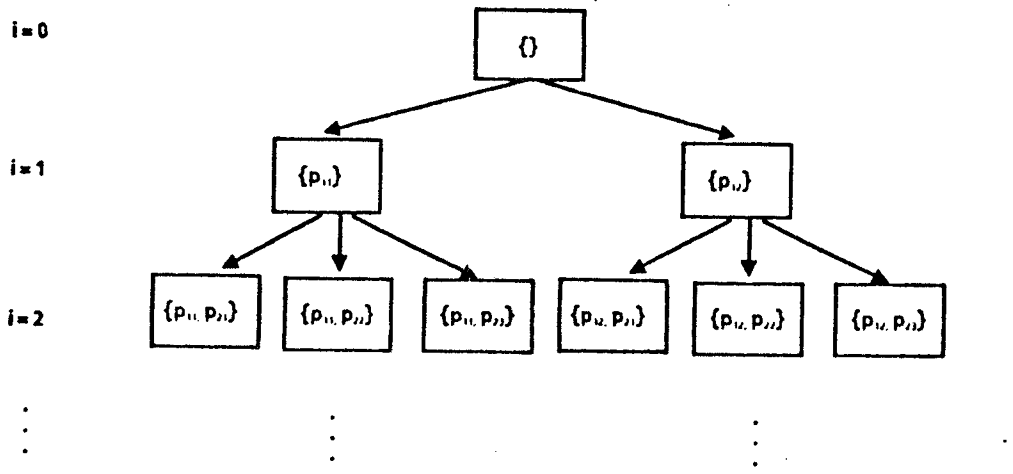

Figure 4: An example state space. Here p_{ij} is the jth alternative plan for the ith goal G_i.

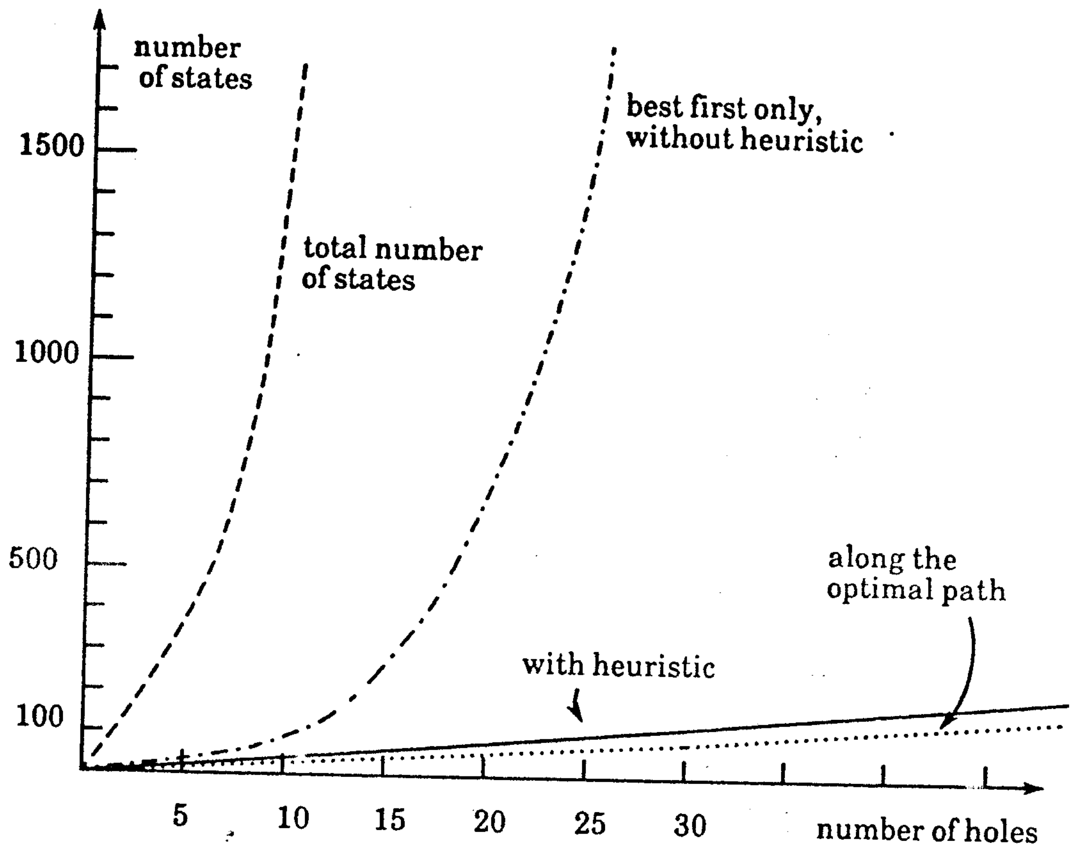

Figure 5: Experimental results of merging process plans for making machined holes.