Merging Separately Generated Plans with Restricted Interactions*

Qiang Yang
University of Waterloo[†]

Dana S. Nau
University of Maryland[‡]

James Hendler
University of Maryland[§]

Major Classification: Problem solving.
Minor Classification: Planning.

*This work was supported in part by an NSERC operating grant to Dr. Yang, by an NSF Presidential Young Investigator award for Dr. Nau, NSF Equipment grant CDA-8811952 for Dr. Nau, NSF Grant NSFD CDR-88003012 to the University of Maryland Systems Research Center, NSF grant IRI-8907890 for Dr. Nau and Dr. Hendler, and ONR grant N00014-91-J-1451 for Dr. Hendler.

[†]Computer Science, Waterloo, Ont. N2L 3G1, Canada. Email: qyang@watdragon.waterloo.edu. Tel: 519-888-4716.

[‡]Computer Science Department, Systems Research Center, and Institute for Advanced Computer Studies. College Park, MD. 20742, USA. Email: nau@cs.umd.edu. Tel: 301-405-2684.

[§]Computer Science Department, Systems Research Center, and Institute for Advanced Computer Studies. College Park, MD. 20742, USA. Email: hendler@cs.umd.edu. Tel: 301-454-4148.

Abstract

Generating action sequences to achieve a set of goals is a computationally difficult task. When multiple goals are present, the problem is even worse. Although many solutions to this problem have been discussed in the literature, practical solutions focus on the use of restricted mechanisms for planning or the application of domain dependent heuristics for providing rapid solutions (i.e. domain-dependent planning). One previously proposed technique for handling multiple goals efficiently is to design a planner or even a set of planners (usually domain-dependent) that can be used to generate separate plans for each goal. The outputs are typically either restricted to be independent and then concatenated into a single global plan, or else they are merged together using complex heuristic techniques. In this paper we explore a set of limitations, less restrictive than the assumption of independence, that still allow for the efficient merging of separate plans using straightforward algorithmic techniques.

In particular, we demonstrate that for cases where separate plans can be individually generated, we can define a set of limitations on the allowable interactions between goals that allow efficient plan merging to occur. We propose a set of restrictions that are satisfied across a significant class of planning domains. We present algorithms that are efficient for special cases of multiple plan merging, propose a heuristic search algorithm that performs well in a more general case (where alternative partially-ordered plans have been generated for each goal), and describe an empirical study that demonstrates the efficiency of this search algorithm.

Keywords: Planning, Automated Reasoning, Problem Solving, Problem Decomposition, Search Control.

1 Introduction

Recent work has proved what many researchers have suspected for a long time: that the so-called "classical AI planning problem" – generating an action sequence to achieve some conjunction of goals – is computationally intractable [Chapman, 1987, Bylander, 1991, Erol et al., 1991, Erol et al., 1992a, Erol et al., 1992b]. A number of ways around this problem are currently being explored, ranging from approaches that use highly restricted mechanisms for planning to the application of domain-dependent heuristics for providing rapid solutions. This latter approach, often referred to as domain-dependent planning, is often successful at producing plans quickly for the limited domains of applicability, at the cost of a large knowledge acquisition and engineering effort. However, given the importance of the planning problem to many commercial and military applications, the cost has often been deemed justifiable.

The inefficiency of planning becomes even more problematic in the presence of a set of goals needing to be jointly solved. Most domain-specific planners are too brittle to handle the numerous interactions that may arise between the actions in the goal conjunction, and thus an assumption of independence is needed to generate a set of plans from a single domain-specific system. While much of modern planning research has focused on the issues of dealing with such interactions in domain-independent ways (for example [Chapman, 1987, Sacerdoti, 1977, Tate, 1977, Vere, 1983, Wilkins, 1984][1]), little work has focused on how the outcomes of either separate domain-dependent planners, or multiple runs of the same planner, could be combined into a single global plan.

One piece of work that does relate to this area is Korf's [Korf, 1987a] analysis of the decomposition of plans into subgoals in search related planning systems. He shows that if the subgoals are independent, then solving each one in turn (and essentially concatenating the results) will divide both the base and the exponent of the complexity function by the number of subgoals. Where the results are "serializable" –that is, where previous goals can be protected during the execution of other goals – Korf shows that such clear-cut results cannot be obtained. However, he goes on to point out that by performing the decomposition of serializable goals, the branching factor is often reduced significantly since many operations may be ruled out in protecting goals that have already been achieved.

The major limitation with decomposition, however, is that for most planning problems the goals and subgoals often interact or conflict with each other in non-serializable ways[2].

[1] A review of this work is presented in [Hendler et al. 1990].

[2] The most famous example of this is the "Sussman anomaly," in which solving one goal undoes the

Unfortunately, it appears impossible to achieve both efficiency and generality in handling goal/subgoal interactions. Domain-independent planners attempt to handle interactions that can occur in many possible forms, and thus they sacrifice the gains in efficiency that might possibly be achieved if some of these forms were disallowed. Domain-dependent planners can often do better at dealing with goal/subgoal interactions 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 discuss an approach to multiple-goal planning that falls somewhere in the middle of this trade-off. The approach is to generate plans for each goal individually, ignoring how each plan might affect the other goals. The individual plans are then merged together, handling interactions while this merging is performed. We show that where certain restrictions hold, these plans can be merged in an optimal manner. In particular, we try to abstract out the kinds of goal and subgoal interactions that occur in a set of problem domains, and develop planning techniques capable of performing well in all domains in which no other kinds of interactions occur. We investigate a set of limitations less restrictive than either independence or linearity assumptions, although they are not as general as the sorts of interactions handled in the larger class of "non-linear" planners.

The restrictions that 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 yield a better combined plan. For example, consider the following situation (based on [Wilensky, 1983]):

> 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.

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

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

and to solve the conjunction taking interactions into account. However, supposing that we have some sort of simple, possibly domain-specific, "trip" planner that can efficiently generate plans for the individual goals. This planner would develop 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). If concatenated together, these plans would solve the conjoined goal, but they'd be inefficient – we'd make two separate trips. However, the two

independently derived solution to the other.

trips can be merged into a single trip by replacing the "come home" subaction of the first trip and the "drive to the dairy" subaction of the second trip with "drive from the dairy to the bakery."

As mentioned above, the restrictions required for our approach to be applicable are limiting, but less so than some of the restrictions proposed in the literature. 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.

In this paper, we identify a set of restrictions that satisfying the above criteria. We also discuss the complexity of the resultant planning problems, and demonstrate that the merging of multiple plans can be performed efficiently under these restrictions. We present algorithms that are efficient for special cases of plan merging, propose a heuristic search algorithm that performs well in a more general case, and describe an empirical study that demonstrates the efficiency of this search algorithm.

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 that has been used to circumvent this problem is the condition of *linearity.* This condition is satisfied in a planning problem if the individual goals can be achieved sequentially in any arbitrary order[3]. Early planners typically generated plans for the goals as if the planning problem were linear. As an example, the STRIPS planner [Fikes and Nilsson, 1971] handled compound goals that were conjuncts of component goals in this manner. Unfortunately, this often led to redundant actions in the plans generated by STRIPS, and could occasionally get the planner into an endless cycle of re-introducing and trying to satisfy the same goal over and over again. Thus, to use such a planner in domains where subgoals or goals interact strongly, it was necessary to add ways to detect and resolve the conflicts.

[3]The literature sometimes uses the term "linear" to describe the situation where "some" rather than "any" ordering will work. A discussion of planning terminology is provided in [Tate et al. 1990].

Later planners typically were based on the 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 a precondition of another goal or subgoal). The plan thus developed is a partially ordered set of actions. Most of the best-known planning systems (for example, [Chapman, 1987, McDermott, 1977, Sacerdoti, 1977, Tate, 1977, Vere, 1983, Wilkins, 1984]) generate plans using this technique.

Although these "least commitment" 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 [Chapman, 1987]. Such extensive computation is not feasible for planning in many real-world domains. This exponentiality is of particular difficulty in 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 examples include military command and control planning applications [Baker and Greenwood, 1987, Glasson and Pomarede, 1987, Brown and Gaucus, 1987], route planning [Garvey and Wesley, 1987], autonomous vehicle navigation [Berlin et al. 1987, Linden and Owre, 1987], and automated manufacturing [Chang and Wysk, 1985, Cutkoski and Tenenbaum, 1987, Hayes, 1987, Nau, 1987].

Within the domain-dependent planning world, the issue of integrating the outputs of several planners has been considered an important one. Two major DARPA initiatives, the AirLand Battle Management program (cf. [Baker and Greenwood, 1987, Greenwood et al. 1987]) and the Pilots' Associate program (cf. [Smith, 1987, Key, 1987]), for example, were centered around the notion of a set of different domain-specific planners generating separate plans for aspects of a mission with a central coordinator (generally viewed as itself some sort of domain-dependent expert system) that could integrate the outputs. More recently, a similar approach was proposed by Kambhampati and Tenenbaum [Kambhampati and Tenenbaum, 1990] for dealing with concurrent engineering systems. This work differed from the earlier work in that it allowed other entities than planners to be included and used a planning framework for the integration, as opposed to a domain-specific expert.

A separate approach to dealing with inefficiency in the handling of multiple goals focuses on placing appropriate restrictions on goal and subgoal interactions. Perhaps the best known

example of this is Vere's DEVISER [Vere, 1983] system which approached the 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 of requisite actions. Wilkins' SIPE system [Wilkins, 1984] provided a general mechanism for handling multiple goals, but also allowed for the integration of specific rules for limiting the set of interactions to be considered at various times in the planning, and to allow human operator interaction in eliminating possibilities and making decisions.

In this paper we consider an approach that is less domain-specific than the rule-based combination of plans generated by separate domain-dependent planners, but is not quite as general as the solutions envisioned within the fully domain-independent, conjoined goal planning system framework. Suppose we are given a planning system, comprised of one or more planners, that can generate (partially-ordered) plans for each of the individual goals. We make no particular assumptions about the types of planners involved in the system—they may be domain-dependent or independent as necessary. We examine a set of goal/subgoal interaction restrictions that, where they apply, allow the efficient merging of these plans into an optimal global plan. We examine two situations: where a single separate plan is generated for each goal, and where more than one plan might be generated per goal. 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

We consider a goal G to be a collection of predicates describing some desired state of the world. A plan P for G is a set of actions $\mathcal{A}(P)$, together with a partial ordering on the order in which these actions must be performed,[4] such that if the actions are performed in any order consistent with the ordering constraints, G will be achieved.

We consider each action a to have a cost, denoted by $\text{cost}(a)$. We define the cost of a plan P or a set of actions A to be the sum of the costs of the individual actions; i.e., $\text{cost}(A) = \sum_{a \in A} \text{cost}(a)$ and $\text{cost}(P) = \sum_{a \in \mathcal{A}(P)} \text{cost}(a)$.

Suppose the goal G is the conjunct of a number of other goals $G_1, G_2, \ldots, G_m$. For the example given in Section 1, `(HAVE BREAD)` and `(HAVE MILK)` are both goals for the conjunctive goal `(AND (HAVE BREAD) (HAVE MILK))`. Suppose that for each individual goal

[4] In addition to the usual kind of partial ordering constraint having the form "action a must be done before action b," we also allow constraints specifying that two actions must be performed at the same time.

G_i, we are given a set of plans $\mathcal{P}_i$ such that each plan in $\mathcal{P}_i$ can achieve G_i. For example, we might be able to achieve (**HAVE BREAD**) either by going to the bakery or by going to the supermarket. One way to try to achieve G would be to select a plan P_i from each set $\mathcal{P}_i$, and try to combine the plans $P_1, P_2, \ldots, P_m$ into a "global plan" for G. Depending on what kinds of interactions occur among the actions in these plans, it might or might not be possible to combine them successfully.

In the literature, the interactions that occur among the individual goals in a goal conjunction typically are of two main types: *precedence* interactions, in which the specific order between the two goals is critical, and *merging* interactions, in which resources or actions may be shared between the goals. Although much of the work in the planning field has discussed the former, the latter are also important in the combination of separate plans. Thus, in this paper, we consider both merging and precedence interactions, specifically:

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 useful effects of all actions in A, while leaving the resultant plan correct. 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 $\text{cost}(M(A)) \geq \text{cost}(A)$, we can ignore it for the purposes of planning. We consider A to be mergeable if $\text{cost}(M(A)) < \text{cost}(A)$; 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 that 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 this would require b to precede itself.

2. An *action-precedence* interaction is an interaction that 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 that can be inserted after b to restore this precondition.

Much previous work in planning has dealt with *deleted-condition interactions.* Although 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. Thus, in general, they are more difficult to deal with than action-precedence interactions. This is a primary limitation with our approach as compared to traditional domain-independent systems. However, it appears that there are significant classes of problems where action-precedence interactions are the only form of deleted-condition interactions that occur.[5] Examples of such problems appear later in this section.

3. Plans for different goals may sometimes contain some of the same actions. An *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, and we call such an interaction a *simultaneous-action interaction.* An example would be two robotic hands working together in order to pick up an object. This kind of interaction is different from an action-merging interaction, because it says that two actions *must* be merged to result in a correct plan, whereas in action-merging interactions, the actions do not have to be merged. It is also different from an identical-action interaction, because these simultaneous actions are not identical.

How to construct a list of interactions for a given set of plans is a problem-dependent task. As described above, action-merging interactions can be detected by looking for actions that contain subactions that cancel each other out. Such situations often occur among actions that require common resources. It is also possible to detect action-precedence interactions by matching the preconditions and effects of the operators in a plan. However, to find the set of identical-action and simultaneous-action interactions, one needs more information than is contained in the common STRIPS operator representation. The additional information requires time points or intervals in action definitions [Allen, 1983], or other such temporal representations. For the purpose of this paper, we will assume that the list of interactions can be constructed using a combination of the domain knowledge expressed in the operators, and the plans generated for the goals.

[5] In fact, the classic "Blocks World" can be reformulated in this way; see [Gupta and Nau, 1992].

Depending on what interactions appear in a given planning problem, it may or may not be possible to find plans for the individual goals that can be combined into a global plan. For example, if P_1 is the sequence of actions (a_1, a_2), P_2 is the sequence of actions (b_1, b_2), and if a_2 must precede b_1 and b_2 must precede a_1, then there is no way to combine P_1 and P_2. We define the *merged plan existence problem* to be the following problem:

> Do there exist plans $P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2, \ldots, P_m \in \mathcal{P}_m$ that can be merged into a "global plan" for the conjoined goal G?

If there is a global plan, then there may be more than one global plan, and different global plans may have different costs. For example, in the shopping example given in Section 1, we discussed two global plans:

1. drive to the dairy, buy milk, come home, drive to the bakery, buy bread, and come home;
2. drive to the dairy, buy milk, drive from the dairy to the bakery, buy bread, and come home.

We define the *optimal merged plan problem* to be the following problem:

> What is the optimal (i.e., least-cost) plan P that can be found by selecting plans $P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2, \ldots, P_m \in \mathcal{P}_m$ and merging them into global plans for the conjoined goal G?[6]

For each i, the least costly plan in $\mathcal{P}_i$ is not necessarily part of the optimal plan P, because a more costly plan in $\mathcal{P}_i$ may be mergeable in a better way with the plans for the other goals. For example, Figure 1 shows the results of merging a plan P_1 with two different plans P_2 and P_2'.

Problems involving the optimization of multiple-goal plans occur in a number of problem domains currently being explored by the planning community, such as manufacturing planning, logistics planning, 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

[6]Depending on what plans are in the sets $\mathcal{P}_i$, P may or may not be "globally optimal", i.e., optimal over all possible plans for the conjoined goal G. It is easy to specify conditions on the sets $\mathcal{P}_i$ sufficient to guarantee that P is globally optimal—but with some planning systems (particularly some kinds of domain-dependent planners) it may not be feasible to test whether these conditions are satisfied.

Figure 1: Merging P_1 and P_2' results in a plan with fewer steps than merging P_1 and P_2.

with scheduling deadlines), but significant and useful classes of problems can be found that satisfy these restrictions. Several specific examples are given below.

Example 1. During the last several years, much work has been done in the area of process planning for manufacturing. As an example, consider the problem of using machining operations to make 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 mount a different cutting tool on the machine tool. If the same machining operation is to be performed on two different holes of the same diameter, then these two operations can be merged by omitting the task of changing the cutting tool. This and several other similar manufacturing problems are of practical significance (see [Chang and Wysk, 1985, Hayes, 1987]), and in fact, much of the work in this paper derives from our ongoing work in developing a computer system for solving such problems [Nau, 1987, Nau et al. 1988].

Example 2. An important class of problem currently being considered by the planning community is that of logistics planning, particularly the planning of a set of deliveries. In this case, different plans containing deliveries to the same place may be combined into a single trip, saving considerable expense. In addition, an inexpensive delivery

technique needed for one delivery can be subsumed by one with a greater cost in a second plan, while still allowing for an overall savings. For example, if a cargo of food could be delivered to a location via a parachute drop, but some other piece of equipment going to the same site requires a landing, then both items can be delivered on the same plane (the one that lands) and the extra steps required for the parachute drop can be deleted.

Example 3. Scheduling is another area in which the planning technology is currently being exploited. As an example, 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.

Example 4. Another important area of problems related to planning is that of multiple-query optimization in database systems. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ be a set of queries to be processed. Associated with each query Q_i is a set of alternate access plans $\{P_{i1}, P_{i2}, \ldots, P_{ik_i}\}$. Each plan is a set of partially ordered tasks that produces the answer to Q. For example, one task might be to find all employees in some department whose ages are less than 30, and whose salaries are over $50,000. Each task has a cost, and the cost of a plan is the sum of the costs of its tasks. Two tasks can be merged if they are the same, or if the result of evaluating one task reduces the cost of evaluating the other. The multiple-query optimization problem [Sellis, 1988] is to find a global access plan by selecting and merging the individual plans so that the cost of the global plan is minimized. As described in [Shim et al. 1991], the plan merging techniques described in this paper provide significantly improved performance in solving this problem.

In this paper we consider two different cases of the optimal merged plan problem. The first case, discussed in Section 4, is where a single plan is generated for each goal (i.e., each $\mathcal{P}_i$ contains exactly one plan). In this case, there is a set of restrictions that defines a class of problems that is reasonably large and interesting, but that can be solved in low-order

polynomial time.

The second case is where more than one plan may be generated for each goal (i.e., each $\mathcal{P}_i$ may contain more than one plan). This necessitates choosing among the plans available for each goal in order to find an optimal global plan. As discussed in Section 5, this case is NP-hard, but we define a heuristic search technique that works quite well in practice.

4 One Plan for Each Goal

Most planning systems, both domain-independent and domain-dependent, plan only until they find some set of actions that are expected to satisfy the goal when applied in the initial situation. Since planning is often extremely difficult or inefficient, most planning systems stop once they have found a single plan for each goal, without trying to find other plans as well. In this section we examine the merging of plans that are created by such planners.

4.1 Plan Existence with One Plan per Goal

We start out by looking at the merged plan existence problem with one plan per goal. In Section 3, we pointed out that whether or not there exists a global plan is independent of whether there are any action-merging interactions. Thus, for the merged plan existence problem we can ignore all action-merging interactions completely. Thus, suppose we are given the following:

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

2. A set of interactions, I, among the actions in the plans (for example, members of this set could 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 set (note that $i = O(n^2)$).[7]

Unless the interactions prevent the plans in S from being merged into a global plan, the global plan is just the union of the 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.

Algorithm 1.

[7] In practice, of course, there may be more than one list of interactions—but in this case we simply take the union of the sets.

1. For each plan P in S, create a graph representing P as a Hasse diagram.[8] Also, create a sorted linear index, L, 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 I, 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 L, 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 [Knuth, 1968], 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. Note that combine(S), if it exists, is unique, but is only partially ordered rather than totally ordered. Of course, every valid embedding of combine(S) within a total ordering is guaranteed to be a valid plan, and Step 3 of Algorithm 1 can easily be modified to produce all of these embeddings if so desired.

4.2 Plan Optimality with One Plan per Goal

4.2.1 Finding Optimal Plans (with Two Restrictions)

In Section 4.1, we showed that if there is just one plan per goal, it is easy to find a merged plan if one exists. However, if we want instead to find the optimal merged plan, the problem becomes NP-hard. A proof of this, using a reduction of the shortest common subsequence (SCS of n sequences) is shown in Appendix A. One standard way to address an NP-hard problem is to look for restricted versions of the problem that are easier to solve. This section considers two restrictions that make it feasible to look for an optimal merged plan, rather

[8] A Hasse diagram is a standard representation of a partially ordered set (e.g., see [Preparata and Yeh, 1973]). We actually need a slight generalization of a Hasse diagram here, since we also have the case where two non-identical actions are constrained to occur at the same time.

than just a consistent one. (In the following section we relax the more limiting of these restrictions and show it is still possible to generate near-optimal plans.)

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 for every set of actions A, the actions in A are mergeable if and only if they are in the same equivalence class. We call these equivalence classes *mergeability classes*.

Restriction 2. If combine(S) exists, then it defines a partial order over the mergeability classes defined in Restriction 1; i.e., if E_i and E_j are two distinct mergeability 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.) Intuitively, Restriction 2 requires that merging one set of actions in a mergeability class does not preclude the possibility of merging other actions in the plans. So, for example, the plans in Figure 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.

Figure 2: Merging actions B_1 and B_2 precludes merging A_1 and A_3.

Although these restrictions look quite formidable, we believe that they will hold for many interesting problems in domains of interest to AI planning researchers. In fact, we believe that in most domains in which the interactions are limited as described in this paper, there exist potentially useful subsets for which these restrictions hold. Revisiting the examples from Section 3 we give some examples:

1. Restrictions 1 and 2 both already hold in the automated manufacturing domain described in Example 1 of Section 3. In this case, there is 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) that is quite natural to use for this problem. For a more detailed discussion of this problem, see [Karinthi et al., 1992].

2. Although many logistics planning problems do not have both restrictions holding, there are interesting problems that do. For example, the "delivery scheduler" of Section 3 clearly has mergable actions for means of "loading," "transporting," and "delivering" that would fall into mergeability classes – for example, loading a plane cannot be merged with delivering a payload. In the special case where all of the possible merges of "loads" must occur prior to any merges of "transports," which in turn must occur prior to any merges of "delivers," Restrictions 1 and 2 apply. (The more general case (in which we do not require "loads" before "transports" before "delivers") is handled in Section 4.2.2).

3. A job-shop scheduling problem involves selecting a sequence of operations (i.e. a process routing) whose execution results in the completion of an order, or several orders. An order is a description of a product to be made, including its due-date, quality, and material. A scheduling algorithm assigns start and end times as well as resources (machines, human operators, etc.) to each operation.

   When several orders are submitted, a global schedule for all orders is required to satisfy various constraints on time and resouces. Each schedule for an individual order might include operations for retrieving all the raw stocks, followed by the machining (see example 1), followed by the distribution of parts through the warehouse and to distribution points. When several schedules are combined, there are many opportunities for merging similar operations together to reduce the total time, and improve the quality, of the global schedule. For example, if machine set-up costs are high, then doing the same type of operation on the same machine can help save the set-up costs, and reduce the *maximal lateness* of the global schedule. While a better schedule might allow some of these operations to be performed in parallel, making this assumption might allow us to recognize a lot of important merging opportunities, with low computational costs.

4. Finally, restrictions 1 and 2 also hold in the multiple-query optimization problem discussed in Example 3 of Section 3. To see this, let P and P' be plans for processing queries Q and Q', respectively. Let s and t be any two steps of P, and s' and t' be any two steps of P'. Then s, t, s', and t' are database operations (such as retrieval or join operations). Suppose it is necessary for s to precede t. Then it must be the case that the output of s is needed to produce the input of t. Now, suppose s' is mergeable with s, and t' is mergeable with t. Given the nature of the problem, it follows that the output of t' is not needed to produce the input of s', so it is not necessary for t' to precede s'. Thus, the mergeability classes form a partial order, in which the class that contains s and s' precedes the class that contains t and t'. For a more detailed discussion of the multiple-query optimization problem and how plan-merging applies to it, see [Shim et al. 1991].

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. 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 an optimal (i.e., 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 mergeability 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 Figure 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 that 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.

Figure 3: Plans P_1 and P_2 are merged using Algorithm 2, producing P_3.

4.2.2 Finding Near-Optimal Plans (without Restriction 2)

Algorithm 2 returns an optimal merged plan for any set of plans that can be combined, as long as the mergeability classes satisfy the partial order restriction (Restriction 2) While we have presented examples of problems that satisfy Restriction 2, there are others do not fall in this class. In the "bread and milk" example in Section 1, the actions *drive from bakery to home* and *drive from home to the dairy* belong to one mergeability class, while the actions *drive from dairy to home* and *drive from home to the bakery* belong to the another class. However, merging two actions in one class precludes merging the actions in the other class. Thus, Restriction 2 is not satisfied in this example. Similarly, logistics planning problems, such as those described in Example 2 of Section 3, could include numerous deliveries to a number of different destinations—in which case some "load" steps could occur before some "deliver" steps, but after others. In this case, restriction 2 would clearly be violated.

In this section, we consider how to merge plans when Restriction 2 is not satisfied. As we demonstrated earlier, if Restriction 2 is not satisfied then it is NP-hard to find an optimal merged plan. An alternative option is to find an algorithm that produces near-optimal plans. In [Foulser et al. 1992], an algorithm is presented to find merged plans in the case where the only allowable interactions are action-merging interactions. Results presented in [Foulser et al. 1992] show that their algorithm is guaranteed to find near-optimal plans. Algorithm 3, shown below, generalizes their algorithm to handle the case where there can be not only action-merging interactions, but also simultaneous action interactions, identi-

cal action interactions, and action-precedence interactions. Algorithm 3 makes use of the following functions:

1. Given a set of plans S, one can find a set of actions with no other actions before them. This set of actions is only preceded by the initial state. We denote this set by Start(S).

2. Let Σ be a set of actions in S. Then remove(Σ, S) is the plan with all actions in Σ removed, and with all the precedence relations relevant to actions in Σ removed.

3. According to Restriction 1, actions in P can be partitioned into k mergeability classes. For an action α in mergeability class i, we define Type(α) $= i$.

Intuitively, Algorithm 3 operates in a manner similar to Algorithm 2. It first invokes Algorithm 1 to handle all the simultaneous action interactions, identical action interactions and action-precedence interactions. If the plans can be combined, then it merges the actions in the resultant plans from start to end. In each iteration of the merging process, actions in Start(S) that belong to the same mergeability class are merged into a final plan. The algorithm appears below.

Algorithm 3.

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. let $\Sigma :=$ Start(S), $R := \emptyset$ and $T := \{\text{Type}(\alpha) \mid \alpha \in \Sigma\}$.

3. Until T is empty, do

3a. Partition Σ into k classes, such that each class Σ_i contains actions that are mergeable. Let Σ_l be the subclass with merged action μ, such that $\text{cost}(\Sigma_l) - \text{cost}(\text{merge}(\Sigma_l))$ is maximal.

3b. $S :=$ remove(Σ_l, S), $R := R\bigcup\{\Sigma_l\}$, $T := T - \text{Type}(\mu)$.

3c. $\Sigma := \{\alpha \mid \alpha \in \text{Start}(S) \textit{ and } \text{Type}(\alpha) \in T\}$.

4. If S is not empty, goto 2.

5. In the set of original plans S, merge the actions in each set as given by R.

If there are n actions in the plans, with k mergeability classes and m plans, then step 1 of this algorithm takes time $O(n^3)$, and steps 2 through 5 take time $O(kn)$. Step 6 redirects all of the arcs in the digraph representing S, which can be done in time $O(n^2)$ in the worst case. Therefore, the total time complexity is $O((kn) + n^3) = O(n^3)$.

Consider the "bread and milk" example of Section 1. The two plans for the goals `(HAVE BREAD)` and `(HAVE MILK)` are listed below:

P_1: `(go Home Bakery)` then `(buy Bread)` then `(go Bakery Home)`;

P_2: `(go Home Dairy)` then `(buy Milk)` then `(go Dairy Home)`.

The action `(go Home Bakery)` of P_1 can be merged with `(go Dairy Home)` of P_2, with a merged action `(go Dairy Bakery)`, since this action is cheaper than the two original actions, and since the merged plan is still correct. Likewise, `(go Bakery Home)` of P_1 can be merged with `(go Home Dairy)` of P_2, yielding a merged action `(go Bakery Dairy)`. Thus, the number of mergeability classes is $k = 2$.

Applying Algorithm 3 to this problem, we obtain the following trace:

1. Initially, Start(S) = {`(go Home Bakery)`, `(go Home Dairy)`}, and $T = \{1, 2\}$. Since the two actions belong to different mergeability classes, the algorithm arbitrarily chooses an action and put it in set R. Thus, R = {{`(go Home Bakery)`}}. After this step, $T = \{2\}$.

2. With the first action of P_1 removed, the algorithm then recomputes Start(S):

   Start(S) = {`(go Bakery Home)`, `(go Home Dairy)`}

   Since both actions belong to the same mergeability class, they are merged, yielding

   R = {{`(go Home Bakery)`}, {`(go Bakery Home)`, `(go Home Dairy)`}}

   and $T = \emptyset$.

3. The last iteration of Algorithm 3 picks up the second action in P_2, with an increment $R := R \bigcup$ {{`(go Dairy Home)`}}.

4. Step 6 of the algorithm merges all action sets in R, resulting in a merged plan:

   `(go Home Bakery)` then `(buy Bread)` then
   `(go Bakery Dairy)` then `(buy Milk)` then `(go Dairy Home)`.

In this case, this plan is the optimal merged plan.

4.3 Analysis: One plan per goal

Korf [Korf, 1987a] has pointed out that given certain assumptions, one can reduce exponentially the time required to solve a conjoined-goal planning problem if the individual goals are independent. Below, we generalize Korf's result. Instead of requiring that the individual goals be completely independent, we relax this requirement by allowing the interactions defined in Section 3—provided, of course, that they do not prevent plans for the individual goals from being combined into a plan for the conjoined goal. We show that in this case, one can still reduce the time required for planning exponentially.

4.3.1 Independent Goals

In our conjunctive goal $G = \{G_1, G_2, \ldots, G_m\}$, suppose that for each i, the set of actions A_i relevant for achieving the goal G_i is completely separate from the set of actions A_j relevant for achieving any other goal G_j. Then if we decompose the initial state I into substates $I_1, I_2, \ldots, I_m$, and plan for each individual goal G_i separately using only the actions in A_i, we can concatenate the plans for all the G_i's to produce a plan for the conjoined goal G. In [Korf, 1987a], Korf discusses this approach in the context of a particular example, namely the 8-puzzle. Below, we restate Korf's assumptions and results in a more general abstract manner, allowing us (in Section 4.3.2) to use them to derive a similar result for the case of (restricted) dependent goals.

One way to try to find a plan for G_i is by doing a state-space search, starting at I_i and performing actions in A_i as they become applicable. Let b_i be the average branching factor for the state space (i.e., the average number of actions applicable at each node in the space), and d_i be the length of the shortest plan for G_i. For the analysis in this section, we make the following assumptions: (1) in order to find a plan for G_i, our planner takes worst-case time $O(b_i^{d_i})$, (2) the planner finds a plan of length $O(d_i)$, and (3) there is a $b > 1$ such that $b_1 = b_2 = \ldots = b_m = b$. Then Korf's reasoning gives us the following results:

1. Suppose that in order to find a plan for G, we plan for the individual goals separately and concatenate the resulting plans. The time needed by our planner to find a plan for G_i is $O(b_i^{d_i})$, and the number of actions in the resulting plan for G_i is d_i. Concatenating the plans for all the G_i's can be done in time $O(d_1 + d_2 + \ldots + d_m)$. Thus, the time required to plan for G in this way is

$$O(b_1^{d_1} + b_2^{d_2} + \ldots + b_m^{d_m} + d_1 + d_2 + \ldots + d_m) = O(b^{d_{\max}}),$$

   where $d_{\max} = \max_i d_i$.

2. If we do not decompose the goal G into the individual goals, then each state s in the state space for G represents a combination of problem states $s_1, s_2, \ldots, s_m$ for the individual goals $G_1, G_2, \ldots, G_m$. Thus, the operators applicable to s include all the operators applicable to $s_1, s_2, \ldots, s_m$. Therefore, the branching factor for this space is $b_1 + b_2 + \ldots + b_m = mb$. Furthermore, since the conjoined goal can be achieved only by achieving all of the individual goals, the length of the shortest plan for G is $O(d_1 + d_2 + \ldots + d_m)$. Thus, if we assume (as Korf did) that assumptions (1) and (2) also apply to the conjoined problem, then the time required to find a plan for G without goal decomposition will be $O((mb)^{d_1+d_2+\cdots+d_m})$.

From the above analysis, it follows that in the worst case, planning for G by decomposing it into the individual goals will achieve an exponential amount of reduction in the time required to solve the problem, provided that the individual goals are independent (the same result that Korf derived for the 8-puzzle in [Korf, 1987a]).

4.3.2 Dependent Goals

In Section 4.3.1, we assumed that all goals were independent. In this section we generalize this result. In particular, instead of requiring that the operators for each goal G_i be completely independent of the operators for any other goal G_j, suppose instead that we allow the kinds of interactions described in Section 3, provided that these interactions do not prevent us from combining the plans for the individual goals into a plan for G. Let us leave Korf's other assumptions unchanged. Then we get the following results:

1. Suppose that in order to find a plan for G, we plan for the individual goals separately and combine the resulting plans. As before, the time needed to plan for a single individual goal G_i is $O(b_i^{d_i})$, and the length of the resulting plan is $O(d_i)$. We can no longer simply concatenate the resulting plans, but we can combine them using Algorithm 2. This can be done in time $O((d_1 + d_2 + \ldots + d_m)^3)$. Thus, the time required to plan for G in this way is

$$O(b_1^{d_1} + b_2^{d_2} + \ldots + b_m^{d_m} + (d_1 + d_2 + \ldots + d_m)^3) = O(b^{d_{\max}}),$$

which is the same as before.

2. If we do not decompose the goal G into the individual goals, then each state s in the state space for G represents a combination of problem states $s_1, s_2, \ldots, s_m$ for the individual goals $G_1, G_2, \ldots, G_m$. Because of the interactions, not necessarily all of the

operators applicable to $s_1, s_2, \ldots, s_m$ will be applicable to s, but in the worst case, the branching factor will still be $O(mb)$. Furthermore, since we have assumed that the interactions do not prevent us from combining the plans for the individual goals into a plan for G, the length of the shortest plan for G is $O(d_1 + d_2 + \ldots + d_m)$. Thus, as before, the time required to find a plan for G without goal decomposition will be $O((mb)^{d_1+d_2+\cdots+d_m})$.

From the above analysis, it follows that in the worst case, planning for G by decomposing it into the individual goals will achieve an exponential amount of reduction in the time required to solve the problem even if the individual goals are not independent, provided that the dependencies do not prevent the plans for the individual goals from being combined.

4.4 Summary: One Plan per Goal

So far, we have examined the case in which a single plan[9] is generated for each goal, and these plans are then merged into a single plan for the conjoined goal. In this case, there is an efficient algorithm to find out whether a consistent merged plan exists—but the problem of generating the optimal such plan is NP-hard.

By adding further limitations, particularly the restrictions that the mergeable actions fall into a set of classes and that these classes can be partially ordered, we are able to develop an efficient algorithm for producing an optimal merged plan. The resulting plan is guaranteed to be the lowest-cost plan that can be produced by combining the separate plans and merging those actions that can be merged. We argued that several domains currently being explored by planning reseachers admit interesting cases for which these restrictions hold. Furthermore, in situations where the mergeability classes are not partially ordered, we have also presented an algorithm that produces near-optimal merged plans.

Finally, we have shown a mathematical analysis of the situation in which separately generated plans are merged into a conjoined plan. We derive a result identical to Korf's for multiple goal plans in which the separate plans are completely independent—an exponential improvement is possible. However, we also show that this same exponential improvement is possible even without the assumption of independence, provided that the kinds of interactions that occur are only those described in Section 3.

[9] Which, we remind the reader, may be a partial order. No assumption of total ordering was made.

5 More than One Plan for Each Goal

When generating plans, it is often possible within one run of the planner to find several possible methods for achieving a goal. As possibilities are explored and a plan is generated, adding extra backtracking (i.e. treating goal success as failure) may allow other possible action sequences to be found.

For example, one idea currently being explored is to let a planner continue working to improve a plan as long as time permits [Drummond and Bresina, 1990]. The planner can successively generate different (usually improving) plans until some time threshold is exceeded. A similar idea, although less time-dependent, appears in the SIPS process planning system, which generates sequences of machining operations for producing machinable parts (cf. Example 2 of Section 3). SIPS finds multiple plans for each part to be manufactured [Nau, 1987]). Although this system finds the lowest-cost plans first (thus additional plans may require more extensive machining) and although finding more than one plan for each goal is more complex computationally than finding just one plan for each goal, SIPS generates alternative plans precisely because they can lead to better overall plans using the merging techniques described in this paper.

To understand why generating multiple plans may lead to better results even if the least costly plan is generated first, 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 (based on [Wilensky, 1983]) that

> John lives 1.25 miles from a large grocery store that sells both bread and milk.

The best plans for the individual goals involve two separate trips: one to the store and one to the dairy. However, this would require making two 2-mile trips for a total of 4 miles. The approach described in the previous section would allow them to be merged so that John could go directly from one store to the other (for a total trip of $1 + 1.5 + 1 = 3.5$ miles). A better plan, however, is to use the second-best plan for each goal (going to the grocery store). Even though taken separately these would generate a worse plan (two 2.5-mile trips for a total of 5 miles), they permit more significant merging when combined together (a single trip of $1.25 + 1.25 = 2.5$ miles). Thus, if the planners for the individual trips delivered more than one solution for each goal, this better plan could be found.

5.1 Plan Existence with Multiple Plans per Goal

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 the merged plan existence problem—and in the case where there is more than one plan for each goal, it is NP-hard even when Restrictions 1 and 2 of Section 4.2.1 are satisfied. A proof of this using a reduction of CNF-satisfiability is provided in Appendix B.

Polynomial-time solutions do exist for several special cases of the merged plan existence and optimal merged plan problems. For plan existence, one obvious special case is the one discussed in Section 4, in which the number of plans for each goal was taken to be 1. For plan optimization, a special case occurs if the number of different mergeability classes is less than 3. In this case, if no conflicting constraints are allowed to exist, the optimal merged plan problem can be solved in polynomial time, even if Restriction 2 is lifted. (For example, this would occur in Example 2 of Section 3 if there were only two different kinds of machining procedures to be considered).

5.2 Plan Optimality with Multiple Plans per Goal

5.2.1 Finding Optimal Plans (with the Two Restrictions)

Since the merged plan existence problem with more than one plan per goal is NP-hard even when Restrictions 1 and 2 hold, the same is true of the optimal merged plan problem. However, there is a heuristic approach that performs well in practice when Restrictions 1 and 2 hold. As we describe below, this approach involves formulating the problem as a state-space search and solving 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 $\mathcal{P}_i$ containing one or more plans for G_i;

2. a list of the interactions among the actions in all of the plans.

Our state space is a tree in which each state at the i'th level is a plan for the goals $G_1, G_2, \ldots, G_i$. This tree is defined as follows (an example is shown in Figure 4):

1. the initial state is the empty set;

2. for each state S at level i of the tree, the children of S consist of all plans of the form merge(combine(S, P)) such that $P \in \mathcal{P}_{i+1}$ is a plan for G_{i+1}.

Every state at level m is a goal state, for these states are plans for the conjoined goal $G = \{G_1, G_2, \ldots, G_m\}$.

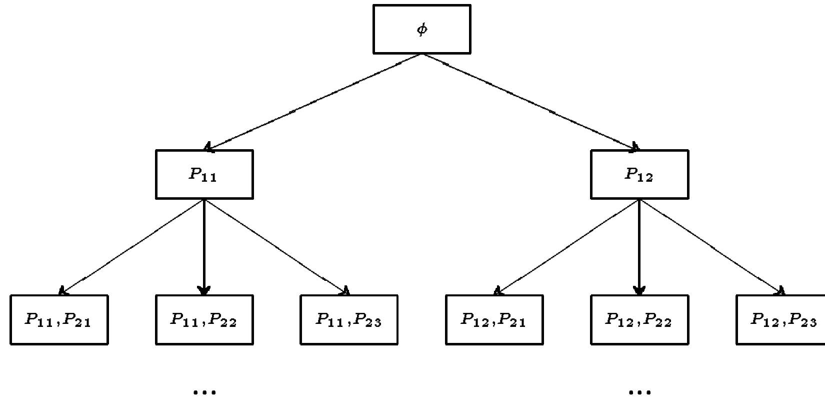

Figure 4: An example state space. Here P_{ij} is the jth alternate plan for goal G_i.

To search the state space, we use the best-first branch-and-bound algorithm shown below. This algorithm maintains an active (or open) set A that contains all states eligible for expansion. To choose which member of A to expand next, the algorithm makes use of a function $L(S)$ that returns a lower bound on the costs of all descendants of S that are goal states. It always chooses for expansion the state $S \in A$ for which $L(S)$ is smallest.[10]

Algorithm 4.

$A := \{\emptyset\}$ *(A is the branch-and-bound active set)*

loop

remove from A the state S for which $L(S)$ is smallest

if S is a goal state **then return** S

else $A := A \cup \{$the children of $S\}$

repeat

If $L(S)$ is a lower bound on the costs of all descendants of S that are goal states, then L is admissible, in the sense that Algorithm 4 will be guaranteed to return the optimal solution. Below we develop a lower bound function that is informed enough to reduce the search space dramatically in many cases.

[10] The relationship between best-first branch and bound and the A* algorithm is well known [Nau et al. 1984]. The quantities $L(S)$, cost(S), and $L(S) - \text{cost}(S)$ used above are analogous to the quantities $f(S)$, $g(S)$, and $h(S)$ used in A*.

Let S be any state at level i in the state space, and T be any child of S. Then T is the result of merging S with some plan $P \in \mathcal{P}_{i+1}$. Thus, if $N(P,S)$ is the set of all actions in P that cannot be merged with actions in S, then

$$\mathrm{cost}(T) \geq \min_{P \in \mathcal{P}_{i+1}} \mathrm{cost}(N(P,S)) + \mathrm{cost}(S).$$

Similarly, if U is any child of T, then

$$\mathrm{cost}(U) \geq \max(\min_{P \in \mathcal{P}_{i+1}} \mathrm{cost}(N(P,S)), \min_{P \in \mathcal{P}_{i+2}} \mathrm{cost}(N(P,S))) + \mathrm{cost}(S).$$

By applying this argument repeatedly, we get

$$\mathrm{cost}(V) \geq \mathrm{cost}(S) + \max_{j>i} \min_{P \in \mathcal{P}_j} \mathrm{cost}(N(P,S))$$

for every goal state V below S. Thus $L_1(S) = \mathrm{cost}(S)$ is an admissible lower bound function (this would correspond to using $h \equiv 0$ in the A* search algorithm). However, a better lower bound can be found from the same formula:

$$L_2(S) = \mathrm{cost}(S) + \max_{j>i} \min_{P \in \mathcal{P}_j} \mathrm{cost}(N(P,S))$$

L_2 is an admissible lower bound function that is better than L_1. However, by making more intelligent use of the information contained in the sets $N(P,S)$, we can compute an even better lower bound function: the function $L_3(S)$ described below.

Let S be a state at level i in the state space, and let $j, k > i$. We say that $\mathcal{P}_j$ and $\mathcal{P}_k$ are S-*connected* if either of the following conditions holds: (1) there are plans $P \in \mathcal{P}_j$ and $Q \in \mathcal{P}_k$ such that $N(P,S)$ and $N(Q,S)$ contain some actions that are mergeable, or (2) there is a set $\mathcal{P}_h$ that is S-connected to both P_j and P_k. S-connectedness is an equivalence relation, so we let $C_1(S), C_2(S), \ldots$, be the equivalence classes (thus each class $C_k(S)$ contains one or more of the $\mathcal{P}_j$'s). We refer to these equivalence classes as *S-connectedness classes.* Having done this, we can now define

$$L_3(S) = \mathrm{cost}(S) + \sum_j \max_{\mathcal{P}_k \in C_j(S)} \min_{P \in \mathcal{P}_k} \mathrm{cost}(N(P,S)) \tag{1}$$

It can easily be shown that L_3 is a lower bound on the cost of any descendant of S that is a goal state.

In order to compute $L_3(S)$, we need efficient ways to compute the sets $N(P,S)$ and the sets $C_i(S)$. These sets can be computed in low-order polynomial time as described below:

1. If S is the initial state (i.e., $S = \emptyset$), then for every plan P in any of the sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, $N(P,S) = \mathcal{A}(P)$. If S is at level i in the state space and P is in one of $\mathcal{P}_{i+1}, \mathcal{P}_{i+2}, \ldots, \mathcal{P}_m$, then

$$N(P,S) = N(P,S') - \{\text{all actions in } P \text{ that can be merged with actions in } S'\},$$

where S' is the parent of S. Thus, by retaining the value computed for $N(P,S')$, we can compute the value of $N(P,S)$ in time $O(|S| + |P|)$, where $|S|$ and $|P|$ are the numbers of actions in S and P, respectively.

2. Whether or not $\mathcal{P}_j$ and $\mathcal{P}_k$ are S-connected depends on the actions in the sets $N(P,S)$ such that P is in $\mathcal{P}_j$ or $\mathcal{P}_k$. Since each set $N(P,S)$ is a subset of $N(P,S')$, this means that $\mathcal{P}_j$ and $\mathcal{P}_k$ cannot be S-connected unless they are S'-connected. Thus the S-connectedness classes can be computed by splitting the S'-connectedness classes into subclasses.

To demonstrate the application of Algorithm 4, consider the following example from a machining domain. Suppose that the goal is to drill two holes h_1 and h_2 on a piece of metal stock. To make hole h_1, the plan is

P_1 : `(spade-drill` h_1`)` then `(bore` h_1`)`.

To make hole h_2, however, there are two alternative plans:

$P_{21:}$ `(twist-drill` h_2`)` then `(bore` h_2`)`.

$P_{22:}$ `(spade-drill` h_2`)` then `(bore` h_2`)`.

Each time one switches to a different kind of machining operation, a different cutting tool must be mounted. Suppose that the relative costs are as follow: 1 for each tool change, 1 for each twist drilling operation, 1 for each boring operation, and 1.5 for each spade drilling operation. Then the costs of the individual plans are $\text{cost}(P_1) = \text{cost}(P_{21}) = 4$ and $\text{cost}(P_{22}) = 4.5$.

At level 0, the initial state $S_0 = \emptyset$, and the plan sets $\mathcal{P}_1 = \{P_1\}$ and $\mathcal{P}_2 = \{P_{21}, P_{22}\}$ are S-connected since they share tool-changing operations for boring and spade-drilling.

At level 1, the state-space has only one state, namely $S_1 = \{P_1\}$. $N(P_{21}, S_1) = \{\text{twist-drill}(h_2)\}$ and $N(P_{22}, S_1) = \emptyset$. Thus,

$$L_3(S_1) = \text{cost}(S_1) + \min\{\text{cost}(N(P_{21}, S_1), N(P_{22}, S_1))\} = 4.$$

At level 2, the two successor states of S_1 are:

$$S_{21} = \{\text{merge}(\text{combine}(P_1, P_{21}))\}, S_{22} = \{\text{merge}(\text{combine}(P_1, P_{22}))\}$$

Their heuristic estimates are $L_3(S_{21}) = 7$, $L_3(S_{22}) = 6.5$. Thus, the optimal goal state is

S_{22} = `(spade-drill` h_1`,` h_2`)` then `(bore` h_1 `and` h_2`)`.

5.2.2 Finding Near-Optimal Plans (without Restriction 2)

The admissibility of the heuristic functions L_1, L_2 and L_3 depends on the optimality of the cost computation of any given state S. When Restriction 2 is not satisfied, Algorithm 2 can no longer be applied for obtaining the optimal solution. But in this case, we can apply Algorithm 3 to compute the cost of a state, and then use this cost value for computing the lower bound functions L_i used in Algorithm 4. We will refer to this new algorithm as Algorithm 5.

Since Algorithm 5 uses Algorithm 3, and since the plans produced by Algorithm 3 are near-optimal rather than optimal, the solutions returned by Algorithm 5 will not always be optimal. However, we can give a bound on how far they are from the optimal solutions. Suppose the goal G is the conjunct of a number of other goals $G_1, G_2, \ldots, G_m$, and suppose that for each G_i we have a set of plans $\mathcal{P}_i$. Let $\mathcal{P}$ be the Cartesian product

$$\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2 \times \ldots \times \mathcal{P}_m.$$

Then each $T \in \mathcal{P}$ contains one plan for each goal. For each $T \in \mathcal{P}$, if it is possible to combine the plans in T, then Algorithm 3 will combine and merge them to produce a merged plan P_T. Since Algorithm 5 systematically enumerates the states in the search space, it will terminate with a P_T if there exists a set T of plans that can be combined. In addition, although P_T may not be the least costly plan that can be produced by combining and merging the plans in T, the heuristic functions L_i are still lower bounds on the costs of P_T's. As a result, Algorithm 5 will select the least costly plan in $\{P_T | T \in \mathcal{P}\}$. In other words, Algorithm 5 is optimal *with respect to* Algorithm 3.

We now demonstrate Algorithm 5 on the "bread and milk" example given in the beginning of Section 5. The plans for the goal `(HAVE BREAD)` are

P_{11}: `(go Home Bakery)` then `(buy Bread)` , then `(go Bakery Home)`;

P_{12}: `(go Home Grocery)` then `(buy Bread)` , then `(go Grocery Home)`.

The plans for the goal `(HAVE MILK)` are

P_{21}: `(go Home Dairy)` then `(buy Milk)` then `(go Dairy Home)`.

P_{22}: `(go Home Grocery)` then `(buy Milk)` then `(go Grocery Home)`.

We now trace the operation of Algorithm 5.

At level 1, there are two states,

$$S_1 = \{P_{11}\}, S_2 = \{P_{12}\}.$$

Taking the distance between any two locations as the cost of going from one to the other, we have

$$\text{cost}(S_1) = 2, \text{cost}(S_2) = 2.5.$$

The heuristic function values are

$$L_3(S_1) = 2 + \min\{\text{cost}(\{(\text{buy Milk})\}), \text{cost}(P_{22})\} = 2, \text{ and}$$

$$L_3(S_2) = 2.5 + \min\{\text{cost}(P_{21}), \text{cost}(\{(\text{buy Milk})\})\} = 2.5.$$

Thus, Algorithm 4 will expand S_1 next.

There are two successors of S_1.

$$T_1 = \{P_{11}, P_{21}\}; \qquad T_2 = \{P_{12}, P_{22}\}.$$

As a result of applying Algorithm 3, the merged plan T_1 corresponds to going to the dairy to buy milk, going from the dairy to the bakery to buy bread, and finally going home from the bakery. The cost of this plan is $\text{cost}(T_1) = 1 + 1.5 + 1 = 3.5$. T_2 corresponds to going to the bakery and the grocery store on separate trips, giving rise to a cost of 4.5. Since both are more costly than S_2, S_2 will be expanded next. One successor of S_2 combines and merges P_{12} with P_{22}, yielding the plan

`(go Home Grocery)` , then `(buy Milk and Bread)` then `(go Grocery Home)`.

In this case, this plan is the optimal merged plan, with a cost of 2.5. However, as we mentioned previously, in other cases the merged plan found by the algorithm may not always be the optimal one.

5.3 Analysis: Multiple Plans per Goal

In the worst case, Algorithm 4 takes exponential time. Since the optimal merged plan problem is NP-hard, this is not surprising. A better analysis would be to describe how well the search algorithm does in the average case. However, the structure of the optimal merged

plan problem is complicated enough that it is not clear how to characterize what an "average case" should be. Furthermore, the "average case" may be different in different application areas. Therefore, the best analysis we can offer is an empirical study of Algorithm 4's performance on problems that appear to be typical of the class of problems in which plan merging looks to be most interesting: those where the restrictions described above hold, but aren't overly constraining.

As an example of such a domain, we have conducted experiments with the algorithm using the EFHA process planning system [Thompson, 1989], a domain-dependent planner based on the earlier SIPS process planner [Nau, 1987]. The decision to use EFHA was made for a largely pragmatic reason: as the developers of the code, we had complete access and could implement the algorithms in precise detail. In addition, we could vary the parameters involved in the generation of alternate plans, to make sure they would not be overly uniform. We attempted to design a problem for EFHA to solve that would be typical of the class of problems we would expect the merging techniques to solve, but that wouldn't be overly simple.

The problem we chose was to find a least-cost plan for making several holes in a piece of metal stock (similar to the problem described in Example 2 of Section 3). We generated specifications for 100 machined holes, randomly varying various hole characteristics such as depth, diameter, surface finish, locational tolerance, etc. We used these holes as input to the EFHA system, allowing it to produce at most 3 plans for each hole. EFHA found plans for 81 of the holes (for the other 19 the machining requirements were so stringent that EFHA could not produce any plans using the machining techniques in its knowledge base).

The distributions of the hole characteristics were chosen so that the plans generated for the holes would have the following characteristics:

1. a wide selection of plans, rather than lots of duplicate plans for different holes;

2. not very many holes having an "obviously best" plan (i.e., a plan that is a sub-plan of all the other plans for that hole);

3. lots of opportunities to merge actions in different plans;

4. a large number of "mergeability tradeoffs"' in choosing which plan to use for a goal. For example, the plan P for the goal G_i may merge well with the actions in some set of plans $\mathcal{P}$ for the other goals, and the plan Q may merge well with the actions in some set of plans $\mathcal{Q}$ for the other goals—but if neither $\mathcal{P}$ nor $\mathcal{Q}$ are subsets of each

Table 1: Experimental results for Algorithm 4 using L_3.

Number of holes n	Nodes in the search space	Nodes expanded
1	2	1
2	10	2
3	34	3
4	98	4
5	284	6
6	852	9
7	2372	12
8	6620	16
9	19480	22
10	54679	28
11	153467	38
12	437460	51
13	1268443	61
14	3555297	86
15	9655279	110
16	29600354	170
17	80748443	223
18	250592571	250

other, then it is unclear (without lots of searching) which of P and Q will result in the best set of merges.

The results of the experiments are shown in Table 1. Each entry in the table represents an average result over 450 trials. Each trial was generated by randomly choosing n of the 81 holes (duplicate choices were allowed), invoking Algorithm 4 on the plans for these holes using the lower bounding function L_3, and recording how many nodes it expanded in the search space. The total cost of each plan was taken to be the sum of the costs of the machining operations in the plan and the costs for changing tools.

Figure 5 plots the average number of nodes in the search space and the average number of nodes expanded by the algorithm (Columns 2 and 3 of Table 1, respectively) , as functions of the number of holes n (Column 1 of Table 1). As shown in this figure, these numbers closely match the functions $y = 1.3(2.9^n)$ and $y = 1.2(1.4^n)$.

Figure 5: Data and curve-fits for the performance of Algorithm 4.

We regard the performance of the algorithm as quite good—especially since the test problem was chosen to be significantly more difficult than the kind of problem that would arise in real-world process planning. In real designs, designers would normally specify holes in a much more regular manner than our random choice of holes, making the merging task much easier. For example, when merging real-world process plans, we doubt that there would be many of the mergeability tradeoffs mentioned earlier; and without such tradeoffs, the complexity of the algorithm is polynomial rather than exponential.

5.4 Summary: Multiple Plans per Goal

Section 5 has dealt with the case in which there may be more than one plan for each goal. In this case, the problem of generating the lowest-cost conjoined plan is NP-hard, even when Restrictions 1 and 2 hold. However, we have developed a heuristic search algorithm to

solve the problem when these restrictions hold, and an extension of the heuristic algorithm when Restriction 2 does not hold. Our heuristic function for this algorithm (the function L_3) is admissable, and thus the algorithm is guaranteed to find optimal merged plans, with respect to the plan merging algorithm used for merging one set of plans.

Since the problem is NP-hard, the worst-case time complexity of the search algorithm is exponential in the worst case. However, our empirical results show that the algorithm performs quite well for a relatively complex problem in the domain of process planning.

6 Future Work

One major limitation of the work described in this paper is that it only concentrates on how to combine plans that have already been developed for individual goals. In the application domains in which we have been working, particularly process planning, 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. One way to create plans is to partition a multiple goal into several subgoals to solve, apply an algorithm for solving each subgoal currently, and then apply Algorithm 2 for combining and merging the individual plans into a global plan. The procedure multi-goal-plan(G) below is one way to do this:

procedure multi-goal-plan(G).
  G is a set of goals $G = G_1, G_2, ...G_n$.
  for every G_i in G
    P_i := plan-for-goal(G_i)
    endfor;
    return merge(combine($P_1, P_2, ..., P_n$));
end multi-goal-plan.

procedure plan-for-goal(G_i)
  nondeterministically choose an action A capable of achieving G_i
  for every precondition H_j of A
    P_j = plan-for-goal(H_j)
      P_j is a partially ordered set of actions that achieves H_j
  endfor;
  P := combine($P_1, P_2, ..., P_n$)

$Q := P$ followed by A
return Q
end plan-for-goal.

The procedure **plan-for-goal** generates one plan for each goal and then merges them—but since it does this nondeterministically, it will find an optimal plan if one exists. If

1. there are only action-precedence, simultaneous-action, identical-action and action-merging interactions among the actions of different plans P_i,
2. the action-merging interactions satisfy Restrictions 1 and 2 of this paper, and
3. there exists at least one plan per goal,

then the nondeterministic procedure **plan-for-goal(G_i)** will halt in polynomial time if a plan exists. In this case the problem is at most NP-hard (as opposed to the traditional planning problem, which can be considerably worse [Erol et al., 1991, Erol et al., 1992a, Erol et al., 1992b].

Although it is clear that this algorithm will work, we classify this phase of our research as future work, since it is not clear as to either how general the result and how the algorithm would perform in practice. The key issues, which we plan to address in the future, are to characterize domains where these restrictions hold for plan creation, and to analyze the worst case and average case behavior of plan generation procedure in these domains. 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.

Another question that remains to be answered is whether the particular interactions discussed in this paper are too restrictive. For example, there may be reasonable ways to solve the optimal merged plan 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. A related problem is how to generalize the kind of interactions allowed. For example, if one allows arbitrary deleted-condition interactions, then a similar search algorithm could be used, except that the resulting search tree would have a greater branching factor. Thus, it would appear that in domains where the number of such conflicts is limited, our approach is still viable.

Finally, we believe that a parallel can be drawn between the optimal merged plan problems and constraint satisfaction problems (CSP's) [Freuder, 1982, Mackworth, 1981]. In CSP, there is a set of variables, each with a set of possible values to be assigned to it, and a set of consistency relations between the variables. A solution to a problem using constraint propagation is to find one or all consistent variable assignments. In optimal merged plan problems, each goal can be considered as a variable, and the set of alternate plans for a goal as values for that variable. The consistency relations between the variables are defined in terms of the action-precedence, identical-action and simultaneous-action interactions.

However, there are also major differences between CSP and optimal merged plan problems. A solution for an optimal merged plan problem has to be minimal in cost, while most well-known algorithms for CSP are based on backtracking algorithms that do not guarantee optimality—and action-merging interactions, which make it possible to reduce costs of combined plans in optimal merged plan problems, are not considered at all in existing CSP research. Thus, our approach can be considered as an extension of CSP research to include the task of achieving optimality. We are currently exploring whether this relationship between CSP and plan merging can be exploited either for generating faster solutions to merging problems (using variants of the CSP techniques) or for guaranteeing optimal solutions to CSP problems that admit merging interactions.

7 Conclusion

In this paper we have been exploring a technique for merging together sets of plans generated either by a single planner (used separately for each goal) or by a set of special purpose planners. Such a technique has been explored in the literature either in the context of search problems relating to planning [Korf, 1987a] or using complex mechanisms for integrating the outputs of a set of planners (as discussed in Section 2).

The approach taken in the paper has been to explore the merging of these plans in the context of a set of limitations on the interactions between plans. The interactions proposed, although by no means fully general, are less restrictive than those of "independence," "serializability," or "linearity" previously proposed in the literature.

We have explored two different variants of this problem. Where a single plan is generated for each goal the primary results include:

1. The optimal merged plan problem is NP-hard.

2. An efficient algorithm is presented to generate a combined plan from the individual

goals. Without further restrictions, the generated plan cannot be guaranteed to be the optimal combination.

3. By imposing two further restrictions, that we propose as reasonable for many realistic problem domains, an efficient algorithm for generating the optimal combined plan is presented. Furthermore, when the more limiting of the restrictions is not satisfied, we have also presented an algorithm that will find near-optimal plans.

4. An analysis is providing showing that where the interactions are limited as described, an exponential amount of savings over solving a conjoined goal is possible.

Where more than one plan may be generated for each goal, the best conjoined plan is often not simply the conjunct of the lowest cost individual plans – higher cost plans may allow more merging. Where multiple plans are generated, the primary results include:

1. Even with the restrictions used in the single goal case, the problem is still NP-hard.

2. A branch-and-bound heuristic search algorithm is demonstrated for finding conjoined plans. An admissable heuristic, and several variants, are proposed to show that this search can find optimal plans.

3. Empirical results are shown demonstrating that in an interesting class of automated manufacturing problems, the heuristic algorithm performs quite well, still growing exponentially but by a very small factor.

We regard this work as a first step, which demonstrates the potential improvements to planning that can be found by exploiting restrictions on allowable interactions. In the previous section, we have outlined several possible extensions of this work—but even without these, this approach is currently being used successfully in at least one application domain [Nau, 1987, Nau et al. 1988]. As we continue our research into more general forms of limited-interaction planning, we are convinced that this approach has potential for significantly improving the performance of planning systems across a number of additional domains.

References

[Allen, 1983] J.F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, 26, No. 11, pages 832-843, 1983.

[Baker and Greenwood, 1987] 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.

[Berlin et al. 1987] 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.

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

[Bylander, 1991] T. Bylander, "Complexity Results for Planning," *Proc. IJCAI-91*, 1991, pp. 274–279.

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

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

[Cutkoski and Tenenbaum, 1987] 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.

[Drummond and Bresina, 1990] M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," *Proc. AAAI-90,* 1990, 138-144.

[Erol et al., 1991] K. Erol, D. Nau, and V. S. Subrahmanian. "Complexity, Decidability and Undecidability Results for Domain-Independent Planning," submitted for publication, 1991.

[Erol et al., 1992a] K. Erol, D. Nau, and V. S. Subrahmanian, "When is Planning Decidable?", *Proc. First Internat. Conf. AI Planning Systems*, 1992, to appear.

[Erol et al., 1992b] K. Erol, D. Nau, and V. S. Subrahmanian, "On the Complexity of Domain-Independent Planning," *Proc. AAAI-92*, 1992, to appear.

[Fikes and Nilsson, 1971] 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.

[Foulser et al. 1992] D.E. Foulser, M. Li and Q. Yang, "Theory and Algorithms for Plan Merging," to appear in *Artificial Intelligence*, 1992. Also available as Research Report, CS-90-40, University of Waterloo, Waterloo, Ont. Canada.

[Freuder, 1982] E.C. Freuder, " A sufficient condition of backtrack-free search." *Journal of the ACM*, 29(1):23–32, 1982.

[Garey and Johnson, 1979] 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.

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

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

[Greenwood et al. 1987] J. Greenwood, G. Stachnick and H. Kay "A Procedural Reasoning System for Army Maneuver Planning," *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

[Gupta and Nau, 1991] N. Gupta and D. Nau, "Complexity Results for Blocks-World Planning," *Proc. AAAI-91*, 1991. Honorable mention for the best paper award.

[Gupta and Nau, 1992] N. Gupta and D. Nau, "On the Complexity of Blocks-World Planning," *Artificial Intelligence*, 1992, to appear.

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

[Hendler et al. 1990] J. Hendler, A. Tate, and M. Drummond "AI Planning: Systems and Techniques" *AI Magazine,* 11(2), May, 1990, 61-77.

[Kambhampati and Tenenbaum, 1990] S. Kambhampati and J.M. Tenenbaum, "Planning in Concurrent Domains," *Proc. of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and COntrol,* Nov. 1990.

[Karinthi et al., 1992] R. Karinthi, D. Nau, and Q. Yang. "Handling feature interactions in process planning," *Applied Artificial Intelligence*, special issue on AI for manufacturing, 1992, to appear.

[Key, 1987] C. Key "Cooperative Planning in the Pilot's Associate," *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

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

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

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

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

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

[Mackworth, 1981] A.K. Mackworth, " Consistency in networks of relations," In Webber and Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Morgan Kaufmann Publishers Inc., 1981.

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

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

[Nau, 1987] 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.

[Nau et al. 1984] D.S. Nau, V. Kumar and L. Kanal, "General Branch and Bound, and Its Relation to A* and AO*," *Artificial Intelligence*, (23), 1984, 29-58.

[Nau et al. 1988] D. S. Nau, R. Karinthi, G. Vanecek, and Q. Yang, "Integrating AI and Solid Modeling for Design and Process Planning," *Proc. Second IFIP Working Group 5.2 Workshop on Intelligent CAD*, Cambridge, England, Sept. 1988.

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

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

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

[Sellis, 1988] T. Sellis, "Multiple-Query Optimization," *ACM Transactions on Database Systems* (13:1), March 1988, 23–52.

[Shim et al. 1991] K. Shim, T. Sellis, and D. Nau, "Improvements of a Heuristic Algorithm for Multiple-Query Optimization," submitted for journal publication, 1991.

[Smith, 1987] Smith, "Plan Coordination in Support of Expert Systems Integration," *Proceedings DARPA Knowledge-based Planning Workshop*, Dec. 1987.

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

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

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

[Tate et al. 1990] A. Tate, J. Hendler, and M. Drummond, "A Review of AI Planning Techniques" *Readings in Planning*, Allen, J., Hendler, J., and Tate, A. (eds.) Morgan-Kaufmann: Palo Alto, California, 1990.

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

[Vere, 1983] 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.

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

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

[Yang and Tenenberg, 1990] Qiang Yang and Josh Tenenberg, "Abtweak: Abstracting a nonlinear, least commitment planner," Proceedings of the Eigth National Conference on Artificial Intelligence, Boston, August, 1990, pages 204-209.

A NP-Hardness of the Optimal Merged Plan Problem

In this appendix, we show that the optimal merged plan problem is NP-hard even if there is only one plan per goal.

The Shortest Common Supersequence (SCS) problem is as follows: given m sequences of characters $S_1, S_2, \ldots, S_m$, what is the shortest sequence S such that each S_i is a subsequence of S? This problem has been shown to be NP-hard [Maier, 1978].

Below, we sketch how the SCS problem can be reduced in polynomial time to the optimal merged plan problem with one plan per goal (the details of this reduction are left to the reader):

> For each sequence $S_i = (c_{i1}, c_{i2}, \ldots, c_{ik_i})$, we define a plan $P_i = (a_{i1}, a_{i2}, \ldots, a_{ik_i})$, where all actions have the same costs, and a goal G_i achieved by P_i. The only kind of interaction that occurs in an action-merging interaction: we define two actions a_{ij} and a_{kl} to be mergeable if and only if $c_{ij} = c_{kl}$. It follows that the plans $P_1, \ldots, P_m$ can be merged into a plan $P = (a_1, a_2, \ldots, a_k)$ iff the sequence $S = (c_1, c_2, \ldots, c_k)$ is a supersequence for $S_1, \ldots, S_m$.

From the above, it follows that the optimal merged plan problem with one plan per goal is NP-hard.

B NP-Completeness of the Merged 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 merged 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 that satisfies every clause in C. To reduce this problem to the merged 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 that 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 that 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.

List of Symbols

Symbol	**Meaning**
$\in$	element of
$\subseteq$	subset
combine	plan combination function
merge	plan merging function
best	best plan function
cost	cost of a plan
$\mathcal{P}$	set of plans
$\mathcal{A}$	set of actions

List of Tables

1 Experimental results for Algorithm 4 using L_3. 32

List of Figures

1 Merging Alternate Plans. 11
2 Example of Plan Merging. 15
3 Example of Algorithm 2. 18
4 Search Space 26
5 Performance of Algorithm 4. 33