# Merging Plans Efficiently*

Dana S. Nau, University of Maryland†
James Hendler, University of Maryland‡
Qiang Yang, University of Waterloo§

## Abstract

Generating action sequences to achieve a set of goals is a computationally difficult task. However, this paper demonstrates 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.

## 1 Introduction

Recent work has proved that the so-called "classical AI planning problem" – generating an action sequence to achieve some conjunction of goals – is computationally intractable [5, 4, 9, 10, 11]. 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 [5, 32, 37, 40, 42][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.

If the individual goals do not interact, then Korf [24] has shown that solving each one in turn (and essentially concatenating the results) will save an exponential amount of work. However, for most planning problems the goals are not independent (the most famous example of this is the "Sussman anomaly" [36], in which solving one goal undoes the independently derived solution to the other).

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—and then merge the individual plans, handling interactions while this merging is performed. We show that where certain restrictions hold, these plans can be merged in an optimal manner. We investigate a set of restrictions less severe 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. These restrictions 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.

*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 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.

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

[1] A review of this work is presented in [20].

For example, consider the following situation (based on [41]):

> 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: buy bread and buy milk.

The usual approach 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 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 statable in a clear and precise way (rather than simply referring to general knowledge about a particular domain of application);
2. the resulting classes of planning problems are large enough to be useful and interesting, but are "well-behaved" enough that planning may be done with a reasonable degree of efficiency.

In this paper, we identify a set of restrictions 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.

# 2 Related Work

One of the major problems with planning is how to handle interactions among goals or subgoals. One way to circumvent this problem is the condition of *linearity*, which is satisfied if the individual goals can be achieved sequentially in any arbitrary order.[2] Early planners, such as STRIPS [12], typically generated plans for the goals as if the planning problem were linear. 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.

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, [5, 27, 32, 37, 40, 42]) 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 [5].

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 [1, 16, 3], route planning [15], autonomous vehicle navigation [2, 25], and automated manufacturing [6, 7, 19, 29].

In domain-dependent planning, the issue of integrating the outputs of several planners has been considered important. Two major DARPA initiatives, the AirLand Battle Management program (cf. [1, 17]) and the Pilots' Associate program (cf. [35, 23]), 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 [21] for dealing with concurrent engineering systems.

A separate approach for handling multiple goals focuses on placing restrictions on goal and subgoal interactions. Perhaps the best known example of this is Vere's DEVISER [40] 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 [42] 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.

# 3 Problem Statement

A goal $G$ is 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,[3] such that if the actions are performed in any order consistent with the ordering constraints, $G$ will be achieved.

Each action $a$ has a cost, $\text{cost}(a)$. If $S$ is a plan or a set of actions, then $\text{cost}(S)$ is the sum of the costs of the individual actions.

[2] 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 [38].

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

Suppose $G$ is the conjunct of a number of other goals $G_1, G_2, \ldots, G_m$, and suppose that for each individual goal $G_i$, we are given a set of plans $\mathcal{P}_i$ such that each plan in $\mathcal{P}_i$ can achieve $G_i$. For example, in Section 1, (HAVE BREAD) and (HAVE MILK) are both goals for the conjunctive goal (AND (HAVE BREAD) (HAVE MILK)); and 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.

The interactions among the individual goals in $G$ 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. We consider the following kinds of merging and precedence interactions:

1. 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 there is a *deleted-condition interaction* in which $b$ removes one of the preconditions necessary for $a$, and if in addition, no other action can be inserted after $b$ to restore this precondition.

   Much previous work in planning has dealt with deleted-condition interactions. In general, deleted-condition interactions are more difficult to deal with than action-precedence interactions—but it appears that there are significant classes of problems where action-precedence interactions are the only form of deleted-condition interactions that occur.[4] Examples appear later in this section.

2. 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.

3. 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 $A$, such that $\text{cost}(M(A)) < \text{cost}(A)$. In this case we say that an *action-merging interaction* occurs.

   Action-merging interactions are a special case of the *enabling-condition interactions*, in which the actions performed to achieve one goal also make it easier to achieve other goals. Although enabling-condition interactions have been largely unexplored in the AI planning literature until recently, recent studies [18, 9] show that they can greatly affect the difficulty of planning. Later in this section we describe several examples of situations in which action-merging interactions are important.

   Even though a set of actions may be mergeable, it may not always be possible to merge them 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.

4. Sometimes, two different actions must occur at the same time. 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 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. 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.

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 $G$?

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, suppose we are given the following plans, and that each action's cost is 2:

$$\begin{array}{lll} P_1: & A1 \rightarrow B1 & \text{(cost=4)} \\ P_2: & C1 \rightarrow D1 & \text{(cost=4)} \\ P_2': & A2 \rightarrow B2 \rightarrow D2 & \text{(cost=6)} \end{array}$$

Then the result of merging $P_1$ with $P_2$ is

$$\begin{array}{ll} A1 \rightarrow B1 & \\ C1 \rightarrow D1 & \text{(cost=8)} \end{array}$$

[4] In fact, the classic "Blocks World" can be reformulated in this way; see [18].

If each of the sets $\{A1, A2\}$ and $\{B1, B2\}$ is mergeable and the cost of each merged action is 3, then merging $P_1$ and $P_2'$ produces a less costly plan:

$$M(\{A1, A2\}) \rightarrow M(\{B1, B2\}) \rightarrow D2 \quad \text{(cost=7)}$$

Problems involving the optimization of multiple-goal plans occur in a number of problem domains. The general class of *all* such problems clearly will not fit within the confines of the restrictions specified in this paper (for example, we have not yet extended our approach to deal with scheduling deadlines), but significant and useful classes of problems can be found that do satisfy our restrictions. Here are some examples:

**Example 1: Machining-Operation Sequencing.** 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 [6, 19])—and one motivation for the current paper has been our work on such problems [29, 30].

**Example 2: Logistics Planning.** 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.** Consider the problem of finding a minimum-time schedule for satisfying some set of orders for products that can be produced in a machine 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. We consider such operations as mergeable.

**Example 4: Multiple Database-Query Optimization.** Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ be a set of database queries. 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 [33] 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 [34], 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 restriction 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 [45], 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

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 $I$ of interactions among the actions (these 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 $I$ (note that $i = O(n^2)$).

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 in $I$. Unless the remaining 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.

1. For each plan $P$ in $S$, create a graph representing $P$ as a Hasse diagram.[5] 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 $I$, modify the graph by creating a precedence arc between the two actions. For each simultaneous-action interaction in $I$, create a simultaneous-action arc between the two actions. For each identical-action interaction in $I$, combine the two actions into a single action. This step 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 $O(n^2)$). 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 Finding Near-Optimal Plans

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 appears in [45]. Two standard ways to address an NP-hard problem are (1) to look for restricted versions of the problem that are easier to solve, and (2) to look for near-optimal solutions rather than solutions that are strictly optimal. In this section, we employ both of these techniques: we place a restriction on the problem, and with this restriction we present an algorithm that finds near-optimal merged plans.

**Mergeability Restriction.** 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.*[6]

[5] This is a standard representation of a partially ordered set (e.g., see [31]). 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.

[6] Previously [43, 44, 22], we discussed how to get optimal solutions for problems satisfying both this restriction and another, more limiting, restriction. The current paper generalizes this earlier work, by obtaining near-optimal results even without the second restriction. For details, see [45].

We believe that this restriction holds for many interesting problems in domains of interest to AI planning researchers. For example, in most of the examples discussed in Section 3, there are large subclasses that easily satisfy this restriction (for further discussion of this, see [45]).

In [13], an algorithm is presented to find merged plans in the case where the only allowable interactions are action-merging interactions. Results presented in [13] show that this algorithm is guaranteed to find near-optimal plans. Algorithm 2, shown below, generalizes this algorithm to handle the case where there can be not only action-merging interactions, but also simultaneous action interactions, identical action interactions, and action-precedence interactions. Algorithm 2 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. If $\Sigma$ is a set of actions in $S$, then remove($\Sigma, S$) is the plan with all actions in $\Sigma$ removed, and all precedence relations relevant to actions in $\Sigma$ removed.
3. From the mergeability restriction, the actions in $P$ can be partitioned into $k$ mergeability classes. If action $\alpha$ is in mergeability class $i$, we define Type($\alpha$) $= i$.

Algorithm 2 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 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. $\Sigma :=$ Start($S$); $\quad R := \emptyset$; $\quad T := \{\text{Type}(\alpha) \mid \alpha \in \Sigma\}$.
3. Until $T$ is empty, do
   (a) Partition $\Sigma$ into $k$ classes, such that each class $\Sigma_i$ contains actions that are mergeable. Let $\Sigma_l$ be the subclass with merged action $\mu$, such that cost($\Sigma_l$) − cost(merge($\Sigma_l$)) is maximal.
   (b) $S :=$ remove($\Sigma_l, S$); $\quad R := R \cup \{\Sigma_l\}$; $T := T -$ Type($\mu$).
   (c) $\Sigma := \{\alpha \mid \alpha \in \text{Start}(S)$ *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 $O(kn/m)$. 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 $O((kn/m) + n^3)$.

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) → (buy Bread) →
(go Bakery Home);

$P_2$: (go Home Dairy) → (buy Milk) →
(go Dairy Home).

The action (go Home Bakery) of $P_1$ can be merged with the action (go Dairy Home) of $P_2$, producing a merged action (go Dairy Bakery). Likewise, the action (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 2 to this problem, we obtain the following trace:

1. Initially, $T = \{1, 2\}$, and

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

   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 2 picks up the second action in $P_2$, with an increment

   $$R := R \cup \{\{\text{(go Dairy Home)}\}\}.$$

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

   (go Home Bakery) → (buy Bread) →
   (go Bakery Dairy) → (buy Milk) →
   (go Dairy Home).

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

## 4.3 Analysis (One Plan per Goal)

Korf [24] 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 [24], 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 as follows, where $d_{\max} = \max_i d_i$:

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

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 $O((mb)^{d_1 + d_2 + \ldots + d_m})$.

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 [24]).

### 4.3.2 Dependent Goals

In Section 4.3.1, we assumed that all goals were independent. Now we generalize this result. 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 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, 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 $O((mb)^{d_1 + d_2 + \ldots + d_m})$.

This analysis shows 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[7] 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.

Under our mergeability restriction, we are able to develop an efficient algorithm for producing a near-optimal merged plan. We have argued that several domains currently being explored by planning researchers admit interesting cases for which this restriction holds.

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.

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

# 5 Multiple Plans per 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 [8]. 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 selection 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 [29]). 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: buy bread and buy milk.

This time, however, let us add the fact (based on [41]) 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

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 if there is more than one plan for each goal, it is NP-hard (a proof appears in [45]).

## 5.2 Finding Near-Optimal Plans

Since the merged plan existence problem with more than one plan per goal is NP-hard even when the mergeability restriction holds, the same is true for finding optimal and near-optimal merged plans. However, there is a heuristic approach that performs well in practice when the mergeability restriction holds. As we describe below, this approach involves formulating the problem as a state-space search problem, 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 $G = \{G_1, G_2, \ldots, G_m\}$.

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.[8]

**Algorithm 3.**
$A := \{\emptyset\}$ *(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 $A := A \cup \{\text{the children of } S\}$
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 3 will be guaranteed to return the optimal solution.

Let $S$ be any state at level $i$ in the state space, $T$ be any child of $S$, and $N(P, S)$ be the set of all actions in $P$ that cannot be merged with actions in $S$. 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 a particular function to use for $L$, namely the following function $L_3$:

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

In [45], we show that $L_3$ is a lower bound on the cost of any descendant of $S$ that is a goal state.

There are two kinds of cases that arise in computing $L_3(S)$. We discuss them in detail in [45], but here is a brief summary:

1. In some cases, we can quickly compute exact values for $L_3(S)$. In these cases, if we use Algorithm 3 with $L = L_3$, we will be guaranteed to find an optimal goal.
2. In other cases it is difficult to compute cost($N(P, S)$) quickly, but we can quickly compute a good approximation of it using Algorithm 2, and use this approximation to compute approximate values for $L_3(S)$. Since this approximation may be greater than $L_3(S)$, it will not always be a lower bound on the costs of the descendants of $S$ that are goals—and thus if we use this approximation, then the solutions returned by Algorithm 3 will not always be optimal. However, as discussed in [45], they are near-optimal, and we can give a bound on how far they are from the optimal solutions.

We now demonstrate Algorithm 3 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) → (buy Bread) →
(go Bakery Home);

$P_{12}$: (go Home Grocery) → (buy Bread) →
(go Grocery Home).

The plans for the goal (HAVE MILK) are

$P_{21}$: (go Home Dairy) → (buy Milk) →
(go Dairy Home);

$P_{22}$: (go Home Grocery) → (buy Milk) →
(go Grocery Home).

We now trace the operation of Algorithm 3. At level 1, there are two states,

$$S_1 = \{P_{11}\}; \quad S_2 = \{P_{12}\}.$$

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

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

The heuristic function values are

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

[8] The relationship between best-first branch and bound and the A* algorithm is well known [28]. The quantities $L(S)$, cost($S$), and $L(S)$ − cost($S$) used above are analogous to the quantities $f(S)$, $g(S)$, and $h(S)$ used in A*.

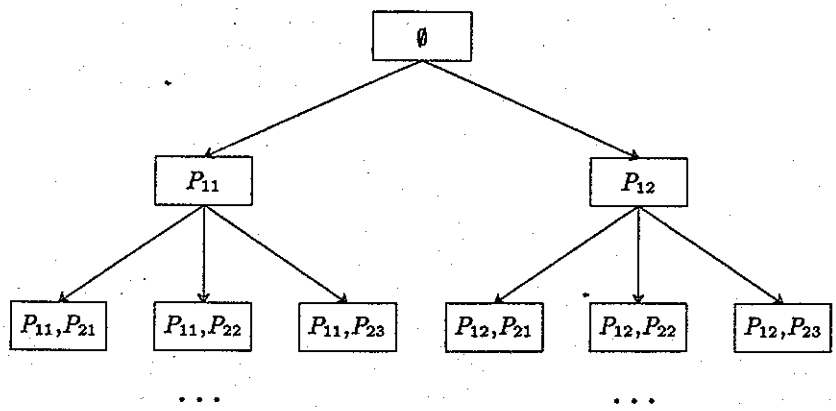


Figure 1: An example state space. Here $P_{ij}$ is the $j$'th alternate plan for goal $G_i$.

$L_3(S_2) = 2.5 + \min\{\text{cost}(P_{21}), \text{cost}(\{\texttt{(buy Milk)}\})\} = 2.5$. Thus, Algorithm 3 will expand $S_1$ next. $S_1$ has two successors:

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

As a result of applying Algorithm 2, 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) → (buy Milk and Bread) →
(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 3 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 3'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 mergeability restriction holds, but isn't overly constraining.

As an example of such a domain, we have conducted experiments with the algorithm using the EFHA process selection system [39], a domain-dependent planner based on the earlier SIPS process selection system [29]. 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 sequence of machining operations for making several holes in a piece of metal stock (similar to the problem described in Example 2 of Section 3). As we discuss in [45], this is a problem for which Algorithm 2 is guaranteed to find the least costly plan—and thus, so is Algorithm 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. many 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 other, then it is unclear (without lots of searching) which of $P$ and $Q$ will result in the best set of merges.

Table 1: Results for Algorithm 3 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 |

This test problem is significantly more difficult than the kind of problem that would arise in real-world process selection. In designing a real part containing a large number of holes, a designer would normally specify the holes in a much more regular manner than our random choice of holes. In particular, there would usually be only a few different sizes, shapes, and tolerance specifications for the holes, making the merging task much easier.

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 3 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.

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 selection. 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 sequences, 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 the mergeability restriction holds. However, we have developed a heuristic search algorithm to find near-optimal solutions when this restriction holds. In some cases (such as the machining example illustrated above), the algorithm is guaranteed to find optimal merged 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.

## 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.

Although it is clear that this approach 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 approach 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 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) [14, 26]. 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 back-

tracking 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 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 [24] 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. Finding the optimal merged plan is NP-hard.
2. An efficient algorithm is presented to generate a combined plan from the individual goals.
3. Under a further restriction (the mergeability restriction) that appears reasonable for many realistic problem domains, an efficient algorithm for generating the near-optimal combined plan is presented.
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 least-cost individual plans, because higher-cost plans may allow more merging. Where multiple plans are generated, the primary results include:

1. Even determining whether a merged plan exists is NP-hard.
2. A branch-and-bound heuristic search algorithm is demonstrated for finding optimal and near-optimal conjoined 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 [29, 30]. 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


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

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

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

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

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

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

[7] 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.

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

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

[10] K. Erol, D. Nau, and V. S. Subrahmanian. When is planning decidable? In *Proc. First Internat. Conf. AI Planning Systems*, pages 222–227, June 1992.

[11] K. Erol, D. Nau, and V. S. Subrahmanian. On the complexity of domain-independent planning. In *Proc. AAAI-92*, pages 381–386, July 1992.

[12] 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.

[13] 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.

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

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

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

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

[18] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, August 1992.

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

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

[21] 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.

[22] 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.

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

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

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

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

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

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

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

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

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

[38] 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.

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

[40] 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.

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

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

[43] Q. Yang, D. Nau, and J. Hendler. An approach to multiple-goal planning with limited interactions. In *AAAI Spring Symposium*, Stanford, 1989.

[44] Q. Yang, D. S. Nau, and J. Hendler. Optimization of multiple-goal plans with limited interaction. In *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.

[45] Q. Yang, D. S. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 9(1), February 1993. To appear.