Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-93-33

Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis

Bernhard Nebel, Jana Koehler

June 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Friedrich J. Wendl
Director

Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis

Bernhard Nebel, Jana Koehler

DFKI-RR-93-33

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-8901 8 and ITW 9000 8).

This report is a substantially revised and extended version of a paper that will appear in Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, September 1993.

Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis*†

Bernhard Nebel Jana Koehler
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
phone: +49 (681) 302-5254/-5259
e-mail: {nebel|koehler}@dfki.uni-sb.de

June 15, 1993

Abstract

The ability of a planner to reuse parts of old plans is hypothesized to be a valuable tool for improving efficiency of planning by avoiding the repetition of the same planning effort. We test this hypothesis from an analytical and empirical point of view. A comparative worst-case complexity analysis of generation and reuse under different assumptions reveals that it is not possible to achieve a provable efficiency gain of reuse over generation. Further, assuming "conservative" plan modification, plan reuse can actually be strictly more difficult than plan generation. While these results do not imply that there won't be an efficiency gain in the "average case," retrieval of a good plan may present a serious bottleneck for plan reuse systems, as we will show. Finally, we present the results of an empirical study of three different plan reuse systems, which leads us to the conclusion that the utility of plan-reuse techniques is limited and that these limits have not been determined yet.

*This work was supported by the German Ministry for Research and Technology (BMFT) under contracts ITW 8901 8 and ITW 9000 8 as part of the WIP project and the PHI project.

†This is a substantially revised and extended version of a paper that will appear in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, September 1993.

Contents

1 Introduction

Plan generation in complex domains is normally a resource and time consuming process. One way to improve the efficiency of planning systems is to avoid the repetition of planning effort whenever possible. For instance, in situations when the goal specification is changed during plan execution or when execution time failures happen, it seems more reasonable to *modify* the existing plan than to plan from scratch again. In the extreme, one might go as far as basing the entire planning process on plan modification, a method that could be called *planning from second principles.*

Instead of generating a plan from scratch, that method tries to exploit knowledge stored in previously generated plans. The current problem instance is used to find a plan in a plan library that—perhaps after some modifications—can be (re-)used to solve the problem instance at hand. Current approaches try to integrate methods from analogical or case-based reasoning to achieve a higher efficiency [22, 39], integrate domain-dependent heuristics [25] or investigate reuse in the general context of deductive planning [8, 33].

Some experiments give evidence that planning based on second principles might indeed be more efficient than planning from scratch [23, 24, 29, 31, 39]. However, it is by no means clear how far these results generalize. In fact, it is not obvious that *modifying* an existing plan is computationally as easy as *generating* one from scratch, in particular, if we adopt the principle of *conservatism* [29, 31], that is to try to recycle "as much of the old solution as possible" [39, p. 133] or to "produce a plan ... by minimally modifying [the original plan]" [31, p. 196].

Using a propositional planning framework, we show that modifying a plan is not easier than planning from scratch. On the positive side modification does not add any complexity to planning if we consider the *general case.* However, there exist special cases when modifying a plan *conservatively*, i.e., *by using as much of the old plan as possible*, can be harder than creating one from scratch. This means that conservative plan modification *is not uniformly as easy as plan generation.* Further, we show that these results also hold if we assume that the old and the new instance are similar. From that we conclude that conservative plan modification runs counter to the idea of increasing efficiency by plan reuse. A conservative strategy should only be employed in a replanning context when it is crucial to retain as many steps as possible.

Although it is impossible to prove that reusing plans leads to a speedup in terms of worst-case complexity, it seems intuitively plausible that in the average case plan reuse is more efficient than planning from scratch. However, finding a good reuse candidate in a plan library may be already very expensive, leading to more computational costs than can be saved by reusing

the candidate. We show that the problem of matching planning instances is NP-hard in the general case. We also consider some special cases that lead to a simplification of this problem.

Finally, we present empirical results of the performance of three different plan-reuse systems, namely, PRIAR [29, 31], SPA [23, 24], and MRL [34]. Contrary to the results reported by Kambhampati and Hendler [31], we observe that in a large number of cases plan reuse leads to an increase in runtime. The main reason for this fact seems to be that the planning systems underlying SPA and MRL use domain heuristics that lead to a highly efficient plan generation process. As a matter of fact, it seems to be the case that plan-reuse techniques are of limited utility, and these limits have not yet been determined.

The paper is organized as follows. In Section 2, we define the notion of propositional STRIPS planning following Bylander [9] and introduce a formal model of plan modification. In Section 3, we analyze the computational complexity of different modification problems relative to their corresponding planning problems. In Section 4, we consider one of the possible bottlenecks of plan-reuse techniques, namely, the retrieval and matching problem. Finally, in Section 5, we present our empirical findings and relate them to our analytical results.

2 Plan Modification in a Propositional Framework

The computational complexity of different forms of planning has been recently analyzed by a number of authors [4, 5, 9, 10, 13, 14, 18, 19, 21]. However, the computational complexity of plan modification has not been investigated yet. We will analyze this problem in the formal framework of *propositional STRIPS planning* as defined by Bylander [9, 10]. As Bylander [9] notes, this model of planning is "impoverished compared to working planners" and is only intended to be a "tool for theoretical analysis." However, since we are mainly interested in *comparing* plan generation with plan modification from a *complexity-theoretic perspective*, this framework is appropriate for our purposes.

2.1 Propositional STRIPS Planning

Like Bylander [9], we define an instance of propositional planning as a tuple $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where:

- $\mathcal{P}$ is a finite set of ground atomic formulae, the *conditions*,

- $\mathcal{O}$ is a finite set of *operators*, where each operator $o \in \mathcal{O}$ has the form $o^+, o^- \Rightarrow o_+, o_-$, where
  - $o^+ \subseteq \mathcal{P}$ are the *positive preconditions*,
  - $o^- \subseteq \mathcal{P}$ are the *negative preconditions*,
  - $o_+ \subseteq \mathcal{P}$ are the *positive postconditions* (add list), and
  - $o_- \subseteq \mathcal{P}$ are the *negative postconditions* (delete list).
- $\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*, and
- $\mathcal{G} = \langle \mathcal{G}_+, \mathcal{G}_- \rangle$ is the *goal specification* with $\mathcal{G}_+ \subseteq \mathcal{P}$ the positive goals and $\mathcal{G}_- \subseteq \mathcal{P}$ the negative goals.

$\mathcal{P}$ is the set of relevant conditions. A *state* is either *undefined*, written $\perp$, or a subset $S \subseteq \mathcal{P}$ with the intended meaning that $p \in \mathcal{P}$ is true in state S if $p \in S$, false otherwise. $\mathcal{O}$ is the set of *operators* that can change states. $\mathcal{I}$ is the initial state, and $\mathcal{G}$ is the goal state specification, with the intended meaning that all conditions $p \in \mathcal{G}_+$ must be true and all conditions $p \in \mathcal{G}_-$ must be false. A plan Δ is a finite sequence $\langle o_1, \ldots, o_n \rangle$ of *plan steps* $o_i \in \mathcal{O}$. An operator may occur more than once in a plan. A plan Δ *solves* an instance Π of the planning problem iff the *result* of the application of Δ to $\mathcal{I}$ leads to a state S that satisfies the goal specification $\mathcal{G}$, where the result of applying $\Delta = \langle o_1, \ldots, o_n \rangle$ to a state S is defined by the following function:

$$
\begin{aligned}
\mathit{Result}\colon (2^{\mathcal{P}} \cup \perp) \times \mathcal{O}^* &\rightarrow 2^{\mathcal{P}} \cup \perp \\
\mathit{Result}(S, \langle\rangle) &= S \\
\mathit{Result}(S, \langle o \rangle) &= \begin{cases} (S \cup o_+) - o_- & \text{if } o^+ \subseteq S \wedge o^- \cap S = \emptyset \\ \perp & \text{otherwise} \end{cases} \\
\mathit{Result}(S, \langle o_1, o_2, \ldots, o_n \rangle) &= \mathit{Result}(\mathit{Result}(S, \langle o_1 \rangle), \langle o_2, \ldots, o_n \rangle)
\end{aligned}
$$

In other words, if the precondition of an operator is satisfied by a state, the positive postconditions are added and the negative postconditions are deleted. Otherwise, the state becomes *undefined*, denoted by $\perp$.[1]

As usual, we consider *decision problems* in order to analyze the computational complexity of planning. This move is justified by the fact that all decision problems are at least as hard as the corresponding *search problems*, i.e, the problem of generating a plan.[2]

[1] This is a slight deviation from Bylander's [9] definition that does not affect the complexity of planning. This deviation is necessary, however, to allow for a meaningful definition of the plan modification problem.

[2] We assume that the reader is familiar with the basic notions of complexity theory as presented, for instance, by Garey and Johnson [20].

PLANSAT is defined to be the *decision problem* of determining whether an instance $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ of propositional STRIPS planning has a solution, i.e., whether there exists a plan Δ such that $Result(\mathcal{I}, \Delta)$ satisfies the goal specification. PLANMIN [12] is defined to be the problem of determining whether there exists a solution of length n or less, i.e., it is the decision problem corresponding to the *search problem* of generating plans with minimal length.

Based on this framework, Bylander [9, 10, 12] analyzed the computational complexity of the general propositional planning problem and a number of generalizations and restricted problems. In its most general form, both PLANSAT and PLANMIN are PSPACE-complete. Severe restrictions on the form of the operators are necessary to guarantee polynomial time or even NP-completeness.

2.2 Plan Reuse and Modification

As described in the Introduction, planning from second principles consists of two steps:

1. *Identifying* an appropriate reuse candidate from a plan library.
2. *Modifying* this plan candidate so that it solves the new problem instance.

Assuming that the identification of a candidate is based on a (polynomial-time) heuristic evaluation function, the modification problem clearly determines the complexity. However, even if we assume that the plan retrieval process is supposed to identify the *optimal* candidate, this optimal candidate can be found easily. One can tentatively modify each plan in the library and select the plan that can be modified optimally. Since this amounts to "only" linearly many plan modification operations in the number of plans stored in the library, the computational complexity of modification determines the complexity of reuse. Note, however, that this does not hold any longer if we also consider (possibly exponentially many) *mappings* between propositions of the new problem instance and of the reuse candidate, as described by Kambhampati and Hendler [29, 31] and Hanks and Weld [23, 24]. In this case, which we consider in Section 4, the costs of reuse may also be influenced by the retrieval problem.

Kambhampati and Hendler [31, p. 196] define the *plan modification problem* as follows (adapted to our framework of propositional STRIPS planning):

> Given an instance of the planning problem $\Pi' = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}', \mathcal{G}' \rangle$ and a plan Δ that solves the instance $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, *produce* a plan Δ' that solves Π' by *minimally modifying* Δ.

We will call this problem MODGEN.

By "minimal modification of a plan" Kambhampati and Hendler [31, p. 195] mean to "salvage as much of the old plan as possible." Other authors are less explicit about what they mean by modifying a plan, but the idea to use as much of the old plan as possible for solving the new problem instance seems to be customary in order to minimize the additional planning effort [39, p. 133]. Of course, the part of the old plan that has been salvaged should be *executable*, i.e., the preconditions of all operators should be satisfied in the final plan. In order to guarantee this, we require that all operators are executable (see the definition of the function *Result*).

Another conceivable interpretation of "minimal modification," namely, of additionally adding as few plan steps as possible, is usually not considered. The reason for not imposing this constraint is obvious. This requirement would make modification as hard as finding an optimal plan, i.e., as hard as PLANMIN, because in this case PLANMIN reduces to modification for the limiting case of an empty modification candidate. Since most plan-reuse systems are only aimed at arbitrary instead of optimal solutions, such a requirement would in fact run counter to the idea of reducing planning effort.

Turning the above specified search problem into a decision problem leads to what we will call the MODSAT problem:

> An instance of the MODSAT problem is given by $\Pi' = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}', \mathcal{G}' \rangle$, a plan Δ that solves $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, and an integer $k \leq |\Delta|$. The question is whether there exists a plan Δ' that solves Π' and *contains a subplan of* Δ *of at least length* k?

In order to fully specify MODSAT, we have to define the meaning of the phrase "Δ' contains a subplan of Δ of length k." For this purpose, we define the notion of a *plan skeleton*, a sequence of operators and "wildcards," denoted by "$*$." The length of a plan skeleton is the number of operators, i.e., we ignore the wildcards. A plan skeleton can be *derived* from a plan according to a *modification strategy* $\mathcal{M}$ by deleting and rearranging plan steps and adding wildcards. A plan skeleton can be *extended* to a plan by replacing each wildcard by a possibly empty sequence of operators. Now we say that *plan* Δ' *contains a subplan of* Δ *of length* k *according to a modification strategy* $\mathcal{M}$ iff a skeleton Γ of length k can be derived from Δ according to $\mathcal{M}$ and Γ can be extended to Δ'. In general, we will consider only *polynomial-time* modification strategies, i.e., strategies such that verifying that the skeleton Γ can be derived from the plan Δ is a polynomial-time problem. In the following, we will consider three different plan modification strategies that satisfy this constraint.

The first alternative we consider is to allow for deletions in the original plan and additions before and after the original plan. Supposing the plan

$$\Delta = \langle o_1, \ldots, o_i, o_{i+1}, \ldots, o_{j-1}, o_j, \ldots, o_n \rangle,$$

the following plan skeleton could be derived from Δ, for instance:

$$\Gamma = \langle *, o_1, \ldots, o_i, o_j, \ldots, o_n, * \rangle,$$

where Γ has length $i + n - j + 1$. The corresponding modification problem will be called MODDEL.

The second alternative is to allow for deletion of plan steps in the old plan and additions before, after, and in the middle of the old plan. Assuming the same plan Δ as above, the following skeleton plan of length $i + n - j + 1$ could be derived:

$$\Gamma = \langle *, o_1, \ldots, o_i, *, o_j, \ldots, o_n, * \rangle.$$

The corresponding modification problem is called MODDELINS.

The final alternative is to count the number of plan steps in the plan skeleton Γ that also appear in the old plan Δ without considering the order. In other words, we view Δ and Γ as multisets and take the cardinality of the intersection as the number of old plan steps that appear in the new plan. The corresponding modification problem is called MODMIX. Although this model of modification may seem to give away too much of the structure of the old plan, "changing step order" is considered to be a reasonable modification operation (see, e.g., [23, p. 96]).

Finally, it should be noted that although the framework we have defined above deals only with linear plans, it can be easily modified to apply to nonlinear planning, as well. Furthermore, all hardness results will apply to nonlinear planning since linear plans are simply special cases of nonlinear ones.

3 The Complexity of Plan Modification

First of all, there is the question of whether modifying a plan can lead to a provable efficiency gain over generation in terms of computational complexity. Not very surprisingly, this is not the case when there are no restrictions on the original instance. However, it does not seem to be impossible to achieve an efficiency gain if we require the old and new problem instance to be similar.

Second, one may ask the question whether plan modification is always as easy as planning from scratch. This question comes up because of the minimality requirement in the definition of the plan modification problem. This requirement makes plan modification very similar to the *belief revision*

problem, i.e., the problem of changing a logical theory minimally in order to accommodate a new information. As is well-known, most revision schemata turn out to be computationally harder than deduction [16, 36].[3] A similar result [37, 17] holds for abduction, which may be viewed as "minimally modifying the assumptions in a proof."

In the following, we provide answers to both questions, addressing first the problem of modifying arbitrary plans and then the problem of modifying plans for similar instances.

3.1 Modifying Arbitrary Plans

One almost immediate consequence of the definitions above is that plan modification cannot be easier than plan generation. This even holds for all restrictions of the PLANSAT problem (concerning, e.g., the form of the operators [12] or more global properties [5]). If PLANSAT$_\rho$ is a restricted planning problem, then MODSAT$_\rho$ shall denote the corresponding modification problem with the same restrictions.

Proposition 1 PLANSAT$_\rho$ *transforms polynomially to* MODSAT$_\rho$ *for all restrictions* ρ.

Proof. The restriction of MODSAT$_\rho$ to empty old plans and $k = 0$ is identical to PLANSAT$_\rho$. ∎

However, plan modification is also not harder than plan generation in the general case.

Proposition 2 MODSAT *is* PSPACE*-complete.*

Proof. Because of Proposition 1 and the fact that PLANSAT is PSPACE-complete [9, Theorem 1], MODSAT is PSPACE-hard.

MODSAT is in NPSPACE because (1) guessing a skeleton Γ of length k and verifying that it can be derived from the old plan Δ and (2) guessing step by step (with a maximum of $2^{|\mathcal{P}|}$ steps) a new plan Δ' and verifying that it solves the instance Π' and extends Γ can be obviously done in polynomial space. Since NPSPACE = PSPACE, it follows that MODSAT $\in$ PSPACE. ∎

This proposition could be taken as evidence that plan modification is not harder than plan generation. However, it should be noted that the proposition is only about the general problem. So, it may be the case that there exist special cases such that plan modification is harder than generation. Such a

[3] More precisely, revision is in most cases Π_2^p-complete. Assuming, as is customary, that the polynomial hierarchy does not collapse (see, e.g., [20, 28]), this implies that revising a propositional theory is harder than doing deduction, which is Π_1^p- or co-NP-complete.

case will not be found among the PSPACE- and NP-complete planning problems, however.

Theorem 3 *If* PLANSAT$_\rho$ *is a restricted planning problem that is* PSPACE*-complete or* NP*-complete, then* MODSAT$_\rho$ *is* PSPACE*-complete or* NP*-complete problem, respectively.*

Proof. PSPACE-hardness and NP-hardness, respectively, are obvious because of Proposition 1. Membership follows in case of PSPACE by Proposition 2. In case of NP, we initially guess (1) n ($0 \leq n \leq |\Delta| + 2$) possibly empty plans Δ_i such that $|\Delta_i| \leq |\Delta|$, (2) $2n$ states $S_1, \ldots, S_{2n}$, and (3) n polynomially bounded proofs that there exists plans from each state S_{2i} to state S_{2i+1} for $1 \leq i \leq n-1$. Since PLANSAT$_\rho$ is in NP, such proofs exist (in most cases, these proofs will be plans). Then we verify in polynomial time (1) that $S_1 = \mathcal{I}$ and S_{2n} satisfies the goal specification $\mathcal{G}$, (2) that $\mathit{Result}(S_{2i-1}, \Delta_i) = S_{2i}$, (3) that the plan existence proofs are correct, and (4) that $\langle \Delta_1, *, \Delta_2, *, \ldots, \Delta_{n-1}, *, \Delta_n \rangle$ is a skeleton of length k that can be derived from Δ. This is obviously a nondeterministic algorithm that runs in polynomial time. ■

The converse of the above theorem does not hold, however. There exist cases when plan generation is a polynomial time problem while plan modification is NP-complete.

Theorem 4 *There exists a polynomial-time* PLANSAT$_\rho$ *problem such that the corresponding* MODDEL$_\rho$ *and* MODDELINS$_\rho$ *problems are* NP*-complete.*

Proof. The planning problem PLANSAT$_1^+$ defined by restricting operators to have only positive preconditions and only one postcondition can be solved in polynomial time [9, Theorem 7]. Let PLANSAT$_1^{+,\overline{post}}$ be the planning problem defined by restricting operators to have (1) only one postcondition p, (2) the negated condition $\overline{p}$ as a precondition, and (3) any number of additional positive preconditions. From the specification of the algorithm Bylander [9] gives for PLANSAT$_1^+$, it is evident that PLANSAT$_1^{+,\overline{post}}$ can also be solved in polynomial time (see also [5]). We will show that the corresponding modification problems MODDEL$_1^{+,\overline{post}}$ and MODDELINS$_1^{+,\overline{post}}$ are NP-complete.

For the hardness part we use a reduction from SAT, the problem of satisfying a boolean formula in conjunctive normal form. Let $V = \{v_1, \ldots, v_m\}$ be the set of boolean variables and let $C = \{c_1, \ldots, c_n\}$ be the set of clauses. Now we construct a MODDEL$_1^{+,\overline{post}}$ problem that can be satisfied iff there exists a satisfying truth assignment for the SAT problem.

The set of conditions $\mathcal{P}$ contains the following ground atoms:

$$\begin{array}{lll}
T_i, & 1 \leq i \leq m, & v_i = \text{true has been selected} \\
F_i, & 1 \leq i \leq m, & v_i = \text{false has been selected} \\
S_i, & 1 \leq i \leq m, & \text{the truth value for } v_i \text{ has been selected} \\
E_i, & 0 \leq i \leq m, & \text{enable evaluation} \\
C_j, & 1 \leq n \leq n, & c_j \text{ evaluates to true.}
\end{array}$$

Further, we assume the following set of operators $\mathcal{O}$:

$$\begin{array}{lllllll}
 & & o^+, & o^- & \Rightarrow & o_+, & o_- \\
t_i & \equiv & \{T_i\}, & \emptyset & \Rightarrow & \emptyset, & \{T_i\} \\
f_i & \equiv & \{F_i\}, & \emptyset & \Rightarrow & \emptyset, & \{F_i\} \\
st_i & \equiv & \{T_i, E_0, \ldots, E_m\}, & \{S_i\} & \Rightarrow & \{S_i\}, & \emptyset \\
sf_i & \equiv & \{F_i, E_0, \ldots, E_m\}, & \{S_i\} & \Rightarrow & \{S_i\}, & \emptyset \\
e_i & \equiv & \emptyset, & \{E_i\} & \Rightarrow & \{E_i\}, & \emptyset \\
pos_{i,j} & \equiv & \{T_i, E_0, \ldots, E_m\}, & \{C_j\} & \Rightarrow & \{C_j\}, & \emptyset \quad \text{if } v_i \in c_j \\
neg_{i,j} & \equiv & \{F_i, E_0, \ldots, E_m\}, & \{C_j\} & \Rightarrow & \{C_j\}, & \emptyset \quad \text{if } \overline{v_i} \in c_j.
\end{array}$$

Assume the following initial and goal state:

$$\begin{aligned}
\mathcal{I} &= \{T_1, \ldots, T_m, F_1, \ldots, F_m\} \\
\mathcal{G}_+ &= \{E_0, \ldots, E_m\} \\
\mathcal{G}_- &= \{T_1, \ldots, T_m, F_1, \ldots, F_m\}.
\end{aligned}$$

The instance $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is, for example, solved by the following plan Δ:

$$\Delta = \langle t_1, \ldots, t_m, f_1, \ldots, f_m, e_0, \ldots, e_m \rangle.$$

Now consider the instance $\Pi' = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}', \mathcal{G}' \rangle$ such that

$$\begin{aligned}
\mathcal{I}' &= \mathcal{I} \\
\mathcal{G}'_+ &= \{E_0, \ldots, E_m, S_1, \ldots, S_m, C_1, \ldots, C_n\} \\
\mathcal{G}'_- &= \emptyset.
\end{aligned}$$

We claim that the SAT formula is satisfiable if, and only if, the plan Δ can be modified by deleting at most m operators and adding some operators before and after the resulting skeleton Γ in order to achieve a new plan Δ' that solves Π'.

First, the operators st_i and sf_i can only be added after the original plan because there are $m + 1$ operators e_i at the end of Δ that produce the preconditions for the above operators. Second, in order to achieve the part of the goal specification that requires S_i to hold for each i means that from each pair $\{t_i, f_i\}$ one operator in Δ must be deleted.

Now assume that the SAT formula is satisfiable. In this case, we can delete m of the t_i and f_i operators such that the T_i's and F_i's correspond to a satisfying truth assignment. Then it is trivial to construct a sequence of $pos_{i,j}$'s and $neg_{i,j}$'s that can be added in the end in order to achieve the goal specification requiring C_j, for all $1 \leq j \leq n$, to hold. Conversely, if such a sequence can be found, then the values of T_i and F_i give a satisfying truth assignment for the SAT formula.

Since st_i, sf_i, $pos_{i,j}$, and $neg_{i,j}$ cannot be added before any of the e_i operators, the reduction applies to MODDELINS$_1^{+,\overline{post}}$, as well.

Membership in NP follows since PLANSAT$_1^{+,\overline{post}}$ is in NP. Using the same algorithm as described in the proof of Theorem 3 leads to a nondeterministic polynomial-time algorithm for MODDEL$_1^{+,\overline{post}}$ and MODDELINS$_1^{+,\overline{post}}$. ■

We were not able to identify a polynomial planning problem PLANSAT$_\rho$ such that the corresponding MODMIX$_\rho$ problem becomes NP-complete. The reason for that is that all known polynomial-time planning problems have a particular simple structure. For all instances of these problems, it is possible to find plans that have the following property. Assuming there is a set of operators $\{o_i\}$ such that each operator could be individually added to the plan Δ and (with some additional additions and deletions, which could be determined in polynomial time) the resulting plans Δ_i still solve the planning instance, then it is possible to add all operators from $\{o_i\}$ collectively to Δ and extend this plan (in polynomial time) to a plan Δ' that still solves the instance. Hence, for all known polynomial-time planning problems PLANSAT$_\rho$, an algorithm for MODMIX$_\rho$ would first generate a plan to solve the planning problem instance and then try to extend this plan by as many operators from the old plan as possible, resulting in a polynomial algorithm that solves the MODMIX$_\rho$ problem.

3.2 Modifying Plans When the Planning Instances are Similar

The results above could be considered as being not relevant for plan modification in real applications because we made no assumption about the similarity between old and new planning instances. The efficiency gain expected from plan reuse, on the other hand, is based on the assumption that the new instance is *sufficiently close* to the old one—which supposedly permits an easy adaptation of the old plan to the new situation. Besides the fact that this looks like a good heuristic guidance, there is the question whether small differences between the old and the new instance lead to a provable efficiency gain in terms of computational complexity. So it might be perhaps the case that modification is easier than planning if the goal specifications differ only

on a constant or logarithmic number of atoms. Although this seems to be possible, there is the conflicting intuition that small changes in the planning instance could lead to drastic (and hard to compute) changes in the plans.

As it turns out, restricting the number of differing atoms does not lead to a different picture than the one presented in the previous subsection. First of all, Theorem 4 still holds for the restricted versions of the modification problems MODDEL and MODDELINS, where we require the old and new initial states to be identical and the old and new goal specification to differ only on one atom. We call these restricted versions of the modification problem MODDEL1G and MODDELINS1G, respectively.

Theorem 5 *There exists a polynomial-time* PLANSAT$_\rho$ *problem such that the corresponding* MODDEL1G$_\rho$ *and* MODDELINS1G$_\rho$ *problems are* NP*-complete.*

Proof. The transformation used in the proof of Theorem 4 is modified as follows. A new atom B is added, which is assumed to be false in the initial state $\mathcal{I}$ and not mentioned in the old goal specification $\mathcal{G}$. The new goal specification $\mathcal{G}'$ is:

$$\begin{aligned} \mathcal{G}'_+ &= \mathcal{G}_+ \cup \{B\} \\ \mathcal{G}'_- &= \mathcal{G}_-. \end{aligned}$$

Finally, the following operator is added:

$$\{E_0, \ldots, E_m, S_1, \ldots, S_m, C_1, \ldots, C_n\},\ \{B\} \Rightarrow \emptyset,\ \{B\}$$

The MODDEL$_\rho$ and MODDELINS$_\rho$ problems generated by this modified transformation obviously satisfy the constraint that the goal specifications differ only on one atom. Further, the modified transformation has obviously the same property as the original one, i.e., the generated MODSAT problems can be used to solve the satisfiability problem.

Membership in NP is again obvious. ∎

Although this theorem confirms the intuition that small changes in the goal specification can lead to drastic changes in the plan, it does not rule out the possibility that there are some hard planning problems such that the corresponding modification problems are easy if the goal specification is only changed marginally. In order to rule out this possibility, we would need something similar to Proposition 1. However, there appears to be no general way to reduce PLANSAT$_\rho$ problems to MODSAT1G$_\rho$ problems. For this reason, we will settle for something slightly less general. We will show that *generating* a plan by modifying a plan for a similar goal specification is at least as hard as the corresponding PLANSAT problem. Hence, instead of the decision problem MODSAT1G, we consider the search problem MODGEN1G.

Further, in order to allow for a "fair" comparison between PLANSAT and MODGEN1G, we measure the resource restrictions of MODGEN1G in terms of the size of the new planning problem instance—and ignore the size of the old plan to be reused.[4] Under these assumptions, it is possible to specify a Turing reduction from PLANSAT$_\rho$ to MODGEN1G$_\rho$.

Theorem 6 *If* PLANSAT$_\rho$ *is a restricted planning problem that is* PSPACE*-hard or* NP*-hard, then the corresponding* MODGEN1G$_\rho$ *problem is* PSPACE*-hard or* NP*-hard, respectively, even if we do not require to reuse a maximal subplan.*[5]

Proof. Using an oracle for MODGEN1G$_\rho$, we can *generate a plan* by modifying it iteratively, starting with the empty plan and empty goal specification and continuing by adding step by step one goal atom. Since the size of the goal specification is linearly bounded by the problem instance, we would need only linearly many calls. Supposing that the theorem does not hold would imply that generating a plan under restrictions ρ is easier than PLANSAT$_\rho$, which is impossible by definition.

In the above reduction, we did not rely on any particular property of the MODGEN1G$_\rho$ oracle. In particular, we did not make the assumption that the oracle has to recycle a maximal reusable plan skeleton. Hence, the result holds for arbitrary modification strategies, even those that are not required to use a maximal subplan. ■

It should be noted that the above theorems apply also to the modification problems that are restricted to have a one-atom-difference between the initial states.

3.3 Conservative versus Arbitrary Modifications

The hope that recycling maximal subplans increases the efficiency of plan reuse turns out to be unfounded, as the above results demonstrate. Our results imply that *conservative* plan modification introduces some combinatorics into the planning and reuse process. In particular, as a Corollary of Proposition 2 it follows that is not possible to determine efficiently (i.e., in polynomial time) a maximal reusable plan skeleton *before* plan generation starts to extend the skeleton.

Corollary 7 *It is* PSPACE*-hard to compute a maximal plan skeleton for* MODSAT *instances.*

[4]This is necessary to rule out such pathological situations as the one where modifying an *exponentially* long plan appears to be polynomial while generating it is exponential.

[5]Note that the proof applies to all complexity classes closed under polynomial Turing reductions. Hence, it also applies to the planning problems identified by Erol *et al* [18]—a fact pointed out to us by Tom Bylander.

In other words, plan generation and plan modification cannot be separated. For this reason, the planning process becomes actually more involved when recycling as much of the old plan as possible. Instead of searching for an arbitrary solution, a plan that contains a maximal subplan of the old plan has to be sought.

Having a closer look at Kambhampati and Hendler's PRIAR framework (which is described as addressing the plan modification problem by minimally modifying plans) reveals that plan skeletons are derived in *polynomial time* [31, p. 197] by a process called "annotation verification." Hence, by Corollary 7, this process cannot by any means derive maximal applicable plan skeletons. Further, the authors do not give any arguments that they approximate such skeletons. In fact, the skeletons derived by PRIAR are not even guaranteed to be applicable. So, PRIAR does not seem to address the problem of "minimally modifying plans," contrary to what the authors claim.

In fact, *maximal* reuse of an old plan only seems to make sense in a replanning context if costs are charged for *not executing already planned steps.* So, it seems to be the case that the two motivations for plan modification, namely, *replanning* and *reuse* may not be as similar as one might think. While in plan reuse the *efficiency* of the planning process is the most important factor, in *replanning* the minimal disturbance of the old plan may be more important, leading to a more involved planning process.[6]

Plan modification in the PRIAR framework—and in other plan-reuse systems—seems not to be a *computational problem* that has to be addressed, but rather a *solution*, a heuristic technique. The "plan skeleton" that is reused is not the maximal applicable one, but the one that *the particular planning algorithm perhaps can exploit in generating a solution.* In other words, the old plan is used as an "entry point" into the search space of possible plans, as made explicit by Hanks and Weld [23].

4 Plan Retrieval and Matching

As demonstrated by Theorem 6, we cannot hope for a provable speedup by plan-reuse techniques in terms of computational complexity. Nevertheless, one would expect a speedup in some cases. In fact, Bylander [11] shows that plan modification for similar planning instances is in some sense more efficient in the average case. The distributional assumptions Bylander makes are questionable, however. Further, Bylander assumes a number of operators that is exponential in the average size of the pre- and postconditions. Never-

[6]Kambhampati makes the same distinction in a later paper [30]. Based on arguments concerning the search process of a planner, he also argues that *guaranteeing* that every step that could be reused is reused could be computationally expensive—a conjecture confirmed by Theorem 4.

theless, Bylander's result is some indication on the analytical side that plan modification could be sometimes more efficient than planning from scratch.

On the empirical side, experiments in the blocks-world domain [23, 24, 29, 31] demonstrate that reusing a plan that solves an instance similar to the one under consideration leads indeed to an efficiency gain in many cases (see also Section 5). It should be noted, however, that in those experiments, the reuse candidate was supplied manually. In order to apply the reuse technique in the general case, it is necessary to provide a *plan library* from which a "sufficiently similar" reuse candidate can be chosen. "Sufficiently similar" could in this case mean that the reuse candidate has a large number of goal atoms and atoms in the initial state in common with the new instance. However, one may also want to consider reuse candidates that are similar to the new instance after the atoms in the reuse candidate have been systematically *renamed*. As a matter of fact, every plan reuse systems contains a *matching component* that tries to find a mapping between the objects of the reuse candidate and the objects of the new instance such that the number of common goal atoms is maximized and the additional planning effort to achieve the initial state of the library plan is minimized (see also Section 5). In the following, we will have a closer look at this matching problem.

4.1 Matching Planning Instances

In order to analyze the matching problem, we assume that the set of conditions $\mathcal{P}$ has some particular structure. Let $\mathbf{O}$ be a set of constants c_i, with the understanding that distinct constants denote distinct objects, and let $\mathbf{P}$ be a set of predicate symbols P_i^n of arity n, then $\mathcal{P}(\mathbf{O}, \mathbf{P})$ is the set of all ground atomic formulae over this signature. In domains, where there are different types of constants, it can be useful to employ a *many-sorted logic* instead of the unsorted logic we consider here. However, we will abstract from this issue and consider only problems such that all constants have the same type. As an example for such a domain, where an unsorted logic is sufficient, consider the blocks-world where we have only blocks (of the same size) and the predicates are universally applicable to all of these blocks.

We assume further that the operators are closed under substitution of constants by constants, i.e., we require that if there exists an operator o_k mentioning the constants $\{c_1, \ldots, c_n\} \subseteq \mathbf{O}$, then there exists also an operator o_l over the arbitrary set of constants $\{d_1, \ldots, d_n\} \subseteq \mathbf{O}$ such that o_l becomes identical to o_k if the d_i's are replaced by c_i's. In other words, we assume that the operators could be represented as ordinary STRIPS operators using variables.

If there are two instances

$$\Pi \;=\; \langle \mathcal{P}(\mathbf{O}, \mathbf{P}), \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$$

$$\Pi' = \langle \mathcal{P}(\mathbf{O}', \mathbf{P}'), \mathcal{O}', \mathcal{I}', \mathcal{G}' \rangle$$

such that (without loss of generality)

$$\begin{aligned} \mathbf{O} &\subseteq \mathbf{O}' \\ \mathbf{P} &= \mathbf{P}' \\ \mathcal{O} &\subseteq \mathcal{O}', \end{aligned}$$

then a *mapping* μ from Π to Π' is an *injective function*

$$\mu\colon \mathbf{O} \rightarrow \mathbf{O}'.$$

Although injectivity might not always be required, it is a safe condition. It guarantees that distinct constants are mapped to distinct constants. The mapping μ is extended to ground atomic formulae and sets of such formulae in the canonical way, i.e.,

$$\begin{aligned} \mu(P_i^n(c_1, \ldots, c_n)) &= P_i^n(\mu(c_1), \ldots, \mu(c_n)) \\ \mu(\{P_1(\ldots), \ldots, P_m(\ldots)\}) &= \{\mu(P_1(\ldots)), \ldots, \mu(P_m(\ldots))\}. \end{aligned}$$

If there exists a *bijective* mapping μ from Π to Π' such that all goal and initial-state atoms are matched, then it is obvious that a plan Δ for Π can be directly be reused for solving Π': $\mu(\Delta)$ solves Π'. In the case that μ is not a bijection or does not match all goal and initial-state atoms, $\mu(\Delta)$ can still be used as a starting point for searching for a plan that solves Π'.

Following Hanks and Weld [24] and Kambhampati and Hendler [29, 31], we define a *match* of a reuse candidate Π with a new instance Π' as a mapping μ from Π to Π' that maximizes first the cardinality of $(\mu(\mathcal{G}_+) \cap \mathcal{G}'_+) \cup (\mu(\mathcal{G}_-) \cap \mathcal{G}'_-)$ and second the cardinality of $\mu(\mathcal{I}) \cap \mathcal{I}'$. It should be noted that in SPA and PRIAR the conditions for the initial-state match are slightly more complicated. In SPA, the number of "open conditions" is minimized, i.e., violations of preconditions in the library plan are minimized. In PRIAR, the number of "inconsistencies in the validation structure" of the library plan is minimized. Since the absence of one atom in the initial state may lead to several "open conditions" or "inconsistencies in the validation structure," our measure is slightly different from the ones used in SPA and PRIAR. Nevertheless, it is certainly also a reasonable approximation of "the amount of planning work necessary to get the input initial world state to the state expected by the library plan" [24, p. 25]. While our purely syntactic criterion is certainly inferior in predictive power, it is probably easier to compute than the measures used in SPA and PRIAR because in our case it is not necessary to consider the structure of the library plan.

The optimization problem defined above corresponds to the following decision problem, which we call PMATCH:

Given two planning instances, Π and Π', and two natural numbers k and n, decide whether there exists a mapping μ from Π to Π' such that $|(\mu(\mathcal{G}_+) \cap \mathcal{G}'_+) \cup (\mu(\mathcal{G}_-) \cap \mathcal{G}'_-)| = k$, $|\mu(\mathcal{I}) \cap \mathcal{I}'| \geq n$ and there is no mapping μ' with $|(\mu'(\mathcal{G}_+) \cap \mathcal{G}'_+) \cup (\mu'(\mathcal{G}_-) \cap \mathcal{G}'_-)| > k$.

It should be noted that in order to select an optimal reuse candidate from the plan library, this matching problem has to be solved for each potentially relevant candidate in the plan library. Of course, one may use structuring and indexing techniques in order to avoid considering all plans in the library. Nevertheless, it seems unavoidable to solve this problem a considerable number of times before an appropriate reuse candidate is identified. For this reason, the efficiency of the matching component is most probably crucial for the overall system performance. Unfortunately, the matching problem is an NP-hard problem.

Theorem 8 PMATCH *is* NP*-hard, even if the initial states are empty.*

Proof. NP-hardness is proved by a polynomial transformation from the *subgraph isomorphism* problem for directed graphs [20, p. 202], which is NP-complete. This problem is defined as follows:

Given two digraphs $G = (V_1, A_1), H = (V_2, A_2)$, does G contain a subgraph isomorphic to H, i.e., do there exist subsets $V \subseteq V_1$ and $A \subseteq A_1$ such that $|V| = |V_2|$ and $|A| = |A_2|$, and there exists a ono-to-one function $f: V_2 \to V$ satisfying $(u, v) \in A_2$ if and only if $(f(u), f(v)) \in A$?

Given an instance of the subgraph isomorphism problem, we construct an instance of PMATCH as follows.

$$\begin{array}{lllll} \mathbf{O} & = & \mathbf{O}' & = & V_1 \cup V_2 \\ \mathbf{P} & = & \mathbf{P}' & = & \{P\} \\ \mathcal{I} & = & \mathcal{I}' & = & \emptyset \\ \mathcal{G}_- & = & \mathcal{G}'_- & = & \emptyset \\ & & \mathcal{G}_+ & = & \{P(v,w) \mid (v,w) \in A_2\} \\ & & \mathcal{G}'_+ & = & \{P(v,w) \mid (v,w) \in A_1\}. \end{array}$$

Now it is obvious that G contains a subgraph isomorphic to H iff there exists a mapping μ such that $|\mu(\mathcal{G}_+) \cap \mathcal{G}'_+| = |A_2|$. ∎

It should be noted that NP-hardness of PMATCH holds even if we do not require an optimal match of the initial state. Hence, the hardness result applies immediately to the matching criterions used in SPA and PRIAR.

This NP-hardness result implies that matching may be indeed a bottleneck for plan reuse systems. In fact, it seems to be the case that planning

instances with complex goal or initial-state descriptions may not benefit from plan-reuse techniques because matching and retrieval is too expensive.

One promising avenue of further research may be to look for good polynomial approximation algorithms for the matching problem. Another way out may be to characterize those planning instances for which matching can be performed in reasonable time. For instance, one way to reduce the matching costs is to introduce sorts in order to limit the number of possible matches.

In the following we will have a closer look at the matching problem in the blocks-world domain. This domain is interesting for two reasons. First, the instances are relatively simple, and may thus permit efficient matching. Second, the blocks-world domain has been used extensively to illustrate the benefits of plan reuse.

4.2 Matching Blocks-World Planning Instances

In general, a blocks-world planning instance consists of

- a set of blocks $\mathbf{O} = \{b_1, \ldots, b_n\}$,
- the set of predicates $\mathbf{P} = \{\texttt{ontable}(\cdot), \texttt{clear}(\cdot), \texttt{on}(\cdot,\cdot)\}$,
- operators $\texttt{Move}(x, y, z)$ (move block x from y to z), $\texttt{Stack}(x, y)$ (pick up block x from the table and stack it on block y), and $\texttt{Unstack}(x, y)$ (unstack x from y),
- the initial state that should be complete (i.e., mention every true atomic ground formula corresponding to the initial physical configuration of blocks) and consistent (i.e, describing one possible physical configuration of the blocks), and
- the goal state that specifies a set of ground atomic formulae to be achieved.

Provided, the goal state is also a complete description of a physical configuration, it is possible to visualize the initial state and goal state as in Figure 1.

Most of the planning instances that have been used to demonstrate the benefits of plan reuse techniques all have a particular simple structure. The goal state is always one stack of blocks. As is easy to see, the matching problem for such instances can be solved in polynomial time. In order to maximize goal matching, the blocks in the smaller stack must be mapped to the blocks in the larger stack respecting the order of the blocks. Obviously, there are only linearly many such mappings. In fact, if the goal description also contains atoms of the form $\texttt{ontable}(\cdot)$ and $\texttt{clear}(\cdot)$, then there are at most two mappings with a maximal number of goal atoms in common. It

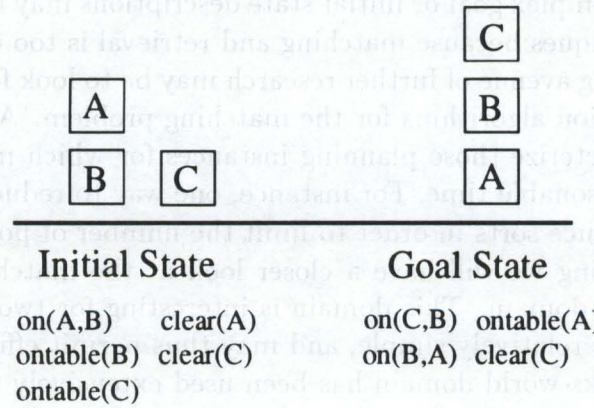


Figure 1: A Blocks-World Example

is then easy to identify the mapping that maximizes the match between the initial states.

Proposition 9 PMATCH *restricted to blocks-world planning instances, where the goal is a complete description of one stack, is a polynomial-time problem.*

However, this positive result does not generalize. If we drop the restriction that the goal is one stack, the matching problem becomes again NP-hard.

Theorem 10 PMATCH *restricted to blocks-world planning instances, where the goal is a complete description of a set of stacks, is* NP*-hard.*

Proof. In order to prove NP-hardness, we use a polynomial transformation from the NP-complete problem of *3-dimensional matching* (3DM), which is defined as follows [20, p. 221]:

> Given a set $M \subseteq W \times X \times Y$, where W, X, and Y are disjoint sets having the same number q of elements, decide whether M contains a matching, i.e., a subset $N \subseteq M$ such that $|N| = q$ and no two elements of N agree in any coordinate.

For convenience, we assume that there exists a function g that assigns a unique index to all elements in $W \cup X \cup Y$ such that

$$\begin{aligned} 1 \leq g(w) \leq q & \quad \text{for all } w \in W, \\ 1+q \leq g(x) \leq 2q & \quad \text{for all } x \in X, \\ 1+2q \leq g(y) \leq 3q & \quad \text{for all } y \in Y. \end{aligned}$$

Given an instance of 3DM, we construct two planning instances

$$\Pi = \langle \mathcal{P}(\mathbf{O}, \mathbf{P}), \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$$
$$\Pi' = \langle \mathcal{P}(\mathbf{O}', \mathbf{P}'), \mathcal{O}', \mathcal{I}', \mathcal{G}' \rangle$$

in the following way (see also Figure 2):

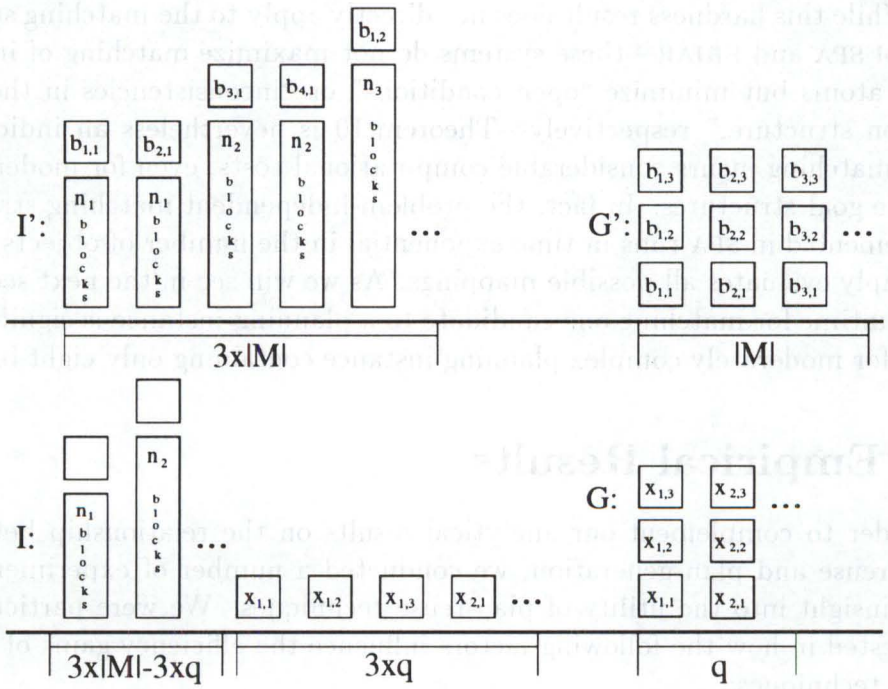


Figure 2: Reduction used in the proof of Theorem 10

1. For each triple $\langle m_{i,1}, m_{i,2}, m_{i,3} \rangle \in M$, $1 \leq i \leq |M|$, we set up a stack of three blocks $b_{i,1}, b_{i,2}, b_{i,3}$ in the goal description $\mathcal{G}'$, i.e., we add the ground atomic formulae $\texttt{ontable}(b_{i,1})$, $\texttt{on}(b_{i,2}, b_{i,1})$, $\texttt{on}(b_{i,3}, b_{i,2})$, $\texttt{clear}(b_{i,3})$ to $\mathcal{G}'_+$.

2. For each block $b_{i,j}$ appearing in the goal state $\mathcal{G}'$, we add a stack of $g(m_{i,j}) + 1$ blocks to the initial state description $\mathcal{I}'$, where $b_{i,j}$ is the top block of this stack.

3. We set up q stacks of three blocks $x_{j,1}, x_{j,2}, x_{j,3}$, $1 \leq j \leq q$, in the goal state $\mathcal{G}$, where $x_{j,1}$ is the bottom block and $x_{j,3}$ is the top block.

4. For each block $x_{j,k}$ appearing in the goal state description $\mathcal{G}$, a stack of height 1 consisting of the block $x_{j,k}$ is added to the initial state description $\mathcal{I}$.

5. For each set S_h of stacks with the same height h in the initial state description $\mathcal{I}'$, we add $|S_h| - 1$ stacks of height h to the initial state $\mathcal{I}$.

Now it is obvious that there exists a mapping μ from Π to Π' that matches $|\mathcal{G}_+|$ goal atoms and $|\mathcal{I}| - q$ initial-state atoms iff there exists a 3-dimensional matching. ∎

While this hardness result does not directly apply to the matching strategies of SPA and PRIAR—these systems do not maximize matching of initial-state atoms but minimize "open conditions" or "inconsistencies in the validation structure," respectively—Theorem 10 is nevertheless an indication that matching incurs considerable computational costs, even for moderately simple goal structures. In fact, the problem-independent matching strategy implemented in SPA runs in time exponential in the number of objects since it simply evaluates all possible mappings. As we will see in the next section, the runtime for matching one candidate to a planning instance is significant, even for moderately complex planning instance containing only eight blocks.

5 Empirical Results

In order to complement our analytical results on the relationship between plan reuse and plan generation, we conducted a number of experiments to gain insight into the utility of plan-reuse techniques. We were particularly interested in how the following factors influence the efficiency gains of plan-reuse techniques:

- the underlying planning system: efficiency gains of plan reuse are measured relatively to the effort spent on solving the same problem by planning from scratch, i.e, the efficiency of the underlying planning system influences the savings we can expect to obtain by plan-reuse techniques.
- similarity between the planning instances: the effort spent on matching and plan modification depends supposedly at least partially on the structural similarity between the reuse candidate and the new instance.
- the application domain: properties of the application domain can probably render matching and modification more or less difficult.

5.1 Plan Modification Systems

We considered the following three plan-reuse systems:

- PRIAR [29, 31],

- SPA [23, 24], and
- MRL [33, 34].

PRIAR is, as all other plan plan-reuse systems we consider in this section, based on a plan generation system that has been extended to cope with the plan modification problem. PRIAR's generative part is derived from the hierarchical, nonlinear planner NONLIN [38]. The key idea in extending the generation part in order to deal with plan modification is to store the internal causal dependency structure used during plan generation and to exploit this structure, also called *validation structure*, when a plan has to be modified.

After a *match* between the reuse candidate (Π, Δ) and the new planning instance Π' has been computed, a process called *annotation verification* computes something similar to what we called *plan skeleton*. The computation of this skeleton proceeds by removing "inconsistencies" in the validation structure. This skeleton is then expanded by a process called *refitting* in order to solve Π'.

Two points may be worth noting about PRIAR with respect to its performance. First, the matching process is only briefly sketched by Kambhampati and Hendler [29, 31] and the available empirical data on PRIAR's performance [29, 31] does not include the matching costs. Second, according to the description of PRIAR, the refitting process first tries to expand the computed plan skeleton and retracts steps from the skeleton only if this expansion fails. In other words, PRIAR is an "optimistic" system, relying on the assumption that the plan skeleton can be expanded to a plan with high probability. As a matter of fact, all experiments described by Kambhampati and Hendler have the property that the skeleton can be expanded.[7]

SPA is based on a lifted version of McAllester and Rosenblitt's [35] systematic nonlinear planning algorithm. In this framework, the planning process is viewed as a search through a tree of partial plans. *Plan generation* starts at the root of the tree (corresponding to the empty plan) and adds plan steps and constraints, while *plan modification* starts at an arbitrary place in the tree and can either add (going down in the tree) or delete constraints and steps (going up in the tree). As in PRIAR, plan modification in SPA has three different phases. In the first phase, a reuse candidate is *matched* against the new planning instance. In the second phase, which is called *fitting* (this should not be confused with the *refitting* process in PRIAR!), a plan skeleton is computed. In the third phase, called *adaptation*, the skeleton is used to find a plan to solve the new instance.

As described in the preceding section, *plan matching* in SPA is based on finding a mapping between the objects of the reuse candidate and the new planning instance that maximizes the number of common goal atoms. If

[7] More generally, the blocks-world domain has this property.

several mappings lead to a best match, the initial preconditions from the reuse candidate and the current plan specification are matched against each other and a mapping that leads to a minimal number of unsatisfied preconditions of operators in the reuse candidate is chosen.

Plan fitting modifies the reuse candidate in order to create a plan skeleton by removing superfluous causal dependencies and marking all unsatisfied conditions. Finally, the *plan adaptation* process tries to find a solution for the new planning instance by *extending* the skeleton, i.e., adding new constraints or plan steps, and *reduction*, i.e., removing constraints or plan steps. In other words, SPA is less optimistic than PRIAR about the probability that an extension of the plan skeleton leads to a solution anbd considers retractions from the skeleton right from the beginning.

Finally, the third plan reuse system we consider is MRL, which is based on the deductive (linear) planner PHI [6, 8] (implemented in SICSTUS PROLOG). The underlying logic of this planning system is the interval-based modal logic LLP [7]. It should be noted that in using this logic in a planning system it becomes possible to specify temporary goals, i.e., goals that have to be achieved at some point and not necessarily in the end, something which could not be done in the usual STRIPS or TWEAK type planning systems (see also [32]).

Plan generation in PHI is performed by constructing proofs for plan specifications in a sequent calculus. During the proof, a plan (formula) is constructed satisfying the formal plan specification. The proofs are guided by *tactics*, which support the declarative representation of control knowledge and make deductive planning quite efficient. The search space considered during the proof can be kept to a manageable size and only those deduction steps are performed which seem to be promising. Contrary to the two systems mentioned above, PHI is not a "complete" planner in the sense that it will (eventually) find a plan if one exists. However, the currently implemented *tactics* are sufficient for generating all "easy to find" plans. As a matter of fact, it was possible to adapt the blocks-world planning instances without changing or adding tactics. While the "incompleteness" of PHI may seem to be a disadvantage, the guarantee that a "complete" planner will eventually find a plan if one exists is only of limited value, since finding this plan may simply take too much time (see for instance Figure 4).

The application domain of PHI is the UNIX *mail domain*, where objects like *messages* and *mailboxes* are manipulated by actions like *read*, *delete*, and *save*.

Plan reuse by the MRL system is based on a logical formalization of the reuse process including the modification, representation and retrieval of plans. The system is able to automatically reuse and modify sequential, conditional, and iterative plans.

Plan modification in MRL proceeds in two phases: During the *plan interpretation* phase the current planning instance and the specification of the reuse candidate are semantically compared. This process is implemented as a theorem proving attempt. The result of the plan interpretation phase is a proof stating that the plan belonging to the reuse candidate can be reused without modification, or a failed proof from which refitting information can be extracted. *Plan refitting* starts with constructing a *plan skeleton* from the reused plan according to the result of the proof attempt using the modification strategy MODDELINS. The plan skeleton is *extended* to a correct plan by a constructive proof of the plan specification formula which was instantiated with this skeleton.

5.2 Application Domains

For our experiments, we considered two different domains. The first application domain is the blocks-world, or more precisely, a particular class of blocks-world planning instances that has been used to explore the performance of PRIAR and SPA. The second domain is the UNIX mail domain, which we used only in connection with the MRL system, though.

Since there is a large collection of empirical data for the modification of plans in the blocks-world domain available for the PRIAR system [29, 31], it seems to be a good idea to test other systems on the same examples. As a matter of fact, a subset of Kambhampati and Hendler's examples has been used in the empirical evaluation of SPA [24].

The blocks-world planning instances used can be roughly categorized as falling into two classes named "$n$bs" and "$n$bs1," where n is an integer parameter denoting the number of blocks which are involved:

- $n$bs instances involve an initial state in which all blocks are clear and on the table and a goal state with one stack that contains all blocks mentioned in the description of the initial state.
- $n$bs1 instances have the same goal state as $n$bs instances, but in the initial state some of the blocks are stacked on others.

Figure 3 gives as an example the configuration of blocks in the 8bs1 blocks-world planning instance.

Considering the 8bs1 instance in more detail, it becomes obvious that there are no "deadlocks" [21] during plan generation. In other words, one can easily generate an *optimal* plan by simply building up the goal stack starting at the bottom block and it is never necessary to put a block temporarily on the table before moving it to its final position. Further, this property holds for all $n$bs1 instances contained in PRIAR's planning instance collection. Most probably, this property simplifies the generation and modification of plans

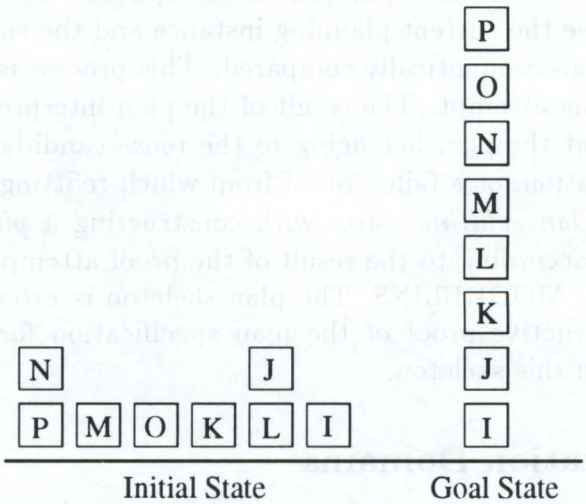


Figure 3: The 8bs1 Example

since, e.g., a plan solving the 8bs instance also solves the corresponding 8bs1 instance. For this reason and because of the fact that optimal plans can be found in polynomial time [21] for all blocks-world problem instances containing only one stack in their goal description, we believe that the claim [31, p. 198] that the "experiments in the blocksworld certainly bear out the flexibility and efficiency of the incremental plan modification ... over a variety of specification changes" is at least arguable.

In order to evaluate the effect different application domains can have on plan reuse performance, we considered in addition to the blocks-world the UNIX mail domain, which is quite different from the blocks-world. Typical planning instances in the blocks-world incorporate a large number of objects of the same type (blocks) but only a small number of different operators. Typical planning instances in the mail domain involve few objects which are of different type (e.g., mails and mailboxes) but a large number of different operators (e.g., open or close a mailbox, read, save, and delete messages).

5.3 Experiments

We start with a brief review of PRIAR's performance data [29, 31]. Most of the experiments are of the kind $n$bs $\rightarrow$ $k$bs1, i.e., a plan solving an $n$bs problem is reused to solve a $k$bs1 problem. Comparing the modification effort with the effort spent on solving the same problem by planning from scratch, very drastic and impressive savings were obtained as Figure 5 for the 7bs1, 8bs1, and 12bs1 instances indicates.

In all examples considered by Kambhampati and Hendler, plan reuse by

the PRIAR system leads to savings between 30 and 90 % compared to planning from scratch. Running other plan-reuse systems on the same examples led to less drastic improvements. Sometimes the reuse effort turned out to be even higher than the generation costs (see below).

Explanations for these quite positive results of plan-reuse techniques in the PRIAR system could be that

- in measuring the plan-reuse costs the time for matching a reuse candidate to a new planning instance has not been considered;
- PRIAR is an "optimistic" system, i.e., it is based on the assumption that the computed plan skeleton can be used in the final plan with high probability (for all examples considered for the PRIAR system, this probability is identical to one);
- PRIAR's generative capabilities degrade much more quickly than its capabilities of modifying a plan (see Figures 4 and 5), an observation also already made by Hanks and Weld [24].

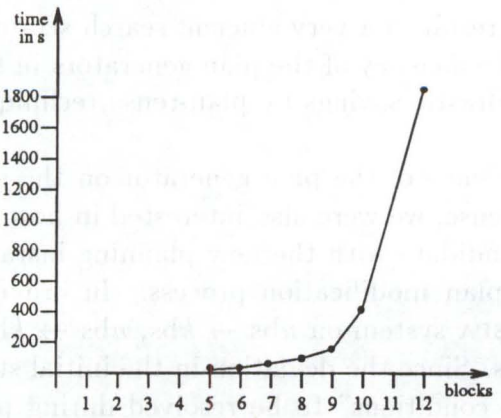


Figure 4: Planning in PRIAR (measured on an EXPLORER-II)

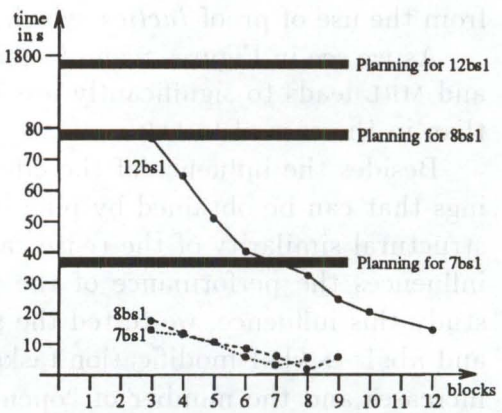


Figure 5: Savings with PRIAR: $n$bs $\rightarrow$ $k$bs1

In contrast to PRIAR, planning from scratch is much more efficient in SPA and PHI as Figures 6 and 7 indicate, even when abstracting from the difference induced by the different platforms. PRIAR simply degrades very quickly, while SPA and PHI show a more graceful degradation of performace with the size of the problem instance. In particular, it is interesting to note that SPA's performance is identical for solving $n$bs and $n$bs1 instances, which might be explained by our observation that a plan that solves an $n$bs instance also solves the corresponding $n$bs1 instance.

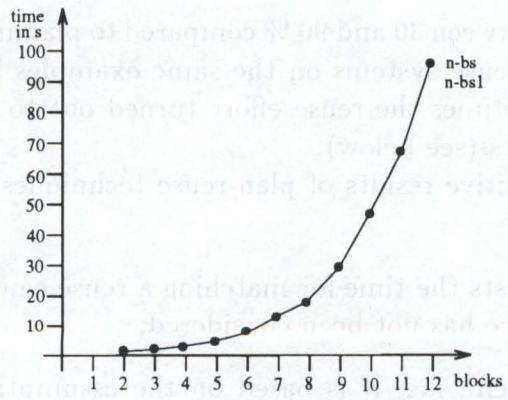


Figure 6: Planning in SPA
(measured on a SOLBOURNE 602/128)

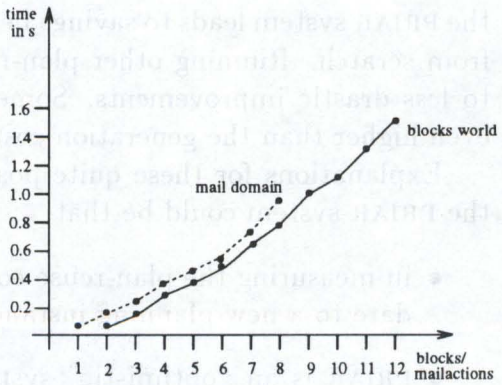


Figure 7: Planning in PHI
(measured on a SOLBOURNE 602/128)

Figure 7 displays the performance of PHI for the blocks world instances $n$bs as well as for the mail domain. As one might expect, planning in the mail domain is more expensive since this domain contains more operators than the blocks-world domain. As mentioned above, the efficiency of PHI results from the use of proof *tactics*, which result in a very efficient search strategy.

As we see in Figures 8 and 9, the efficiency of the plan generators in SPA and MRL leads to significantly less drastic savings by plan-reuse techniques than in the case of PRIAR.

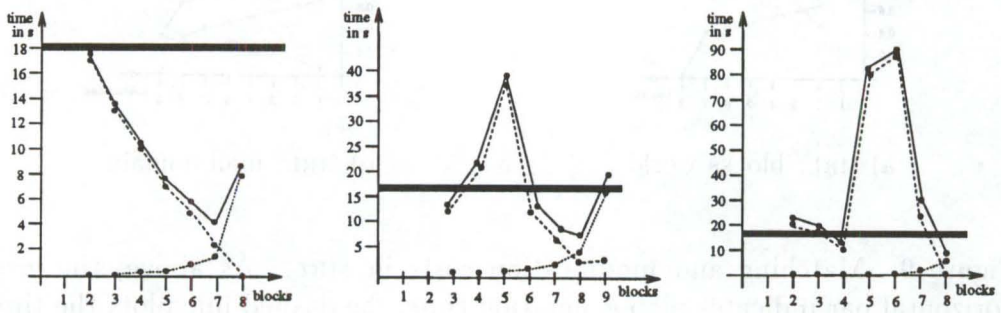
Besides the influence of the efficiency of the plan generator on the savings that can be obtained by plan-reuse, we were also interested in how the structural similarity of the reuse candidate with the new planning instance influences the performance of the plan modification process. In order to study this influence, we tested the SPA system on $n$bs $\rightarrow$ $k$bs, $n$bs $\rightarrow$ $k$bs1, and $n$bs1 $\rightarrow$ $k$bs1 modification tasks. Since the deviation in the initial state increases and the number of "open conditions" to be resolved during plan adaptation increases, we expected that plan adaption becomes more difficult moving from the first kind of tasks to the latter kind of tasks.

In Figure 8, we give the results of the experiments described above for the case $k = 8$. We also performed the same experiments with $k = 7$ and $k = 12$, which led to a similar picture.

In all examples, matching shows an exponential run time behavior for the domain-independent matching algorithm we used.[8] As a matter of fact, even for a moderately sized domain containing only eight blocks, the matching costs are already significant. For the 9bs $\rightarrow$ 8bs1 example, the time of matching is already significantly higher than the plan modification time.

[8] SPA also provides an application-dependent matching algorithm which is linear but restricted to the blocks-world domain, where there is only one goal stack. Instead of this more efficient method, we used the general matching algorithm in order to get an idea about the matching costs in SPA in the general case

Figure 8a gives the performance for the easiest modification problem, where the initial and the goal states differ only by the number of blocks used. Here, SPA shows a performance similar to PRIAR. However, the savings are less drastic, but the total modification effort never exceeds the plan generation effort. If a non-exponential matching algorithm would be used, the modification effort would linearly decrease as the reused plan becomes more and more similar to the desired solution.



a) SPA: $n$bs $\rightarrow$ 8bs b) SPA: $n$bs $\rightarrow$ 8bs1 c) SPA: $n$bs1 $\rightarrow$ 8bs1

Figure 8: Matching and modification costs in SPA. The grey horizontal bar gives the time for generating a plan for 8bs or 8bs1 from scratch. The dashed line plots the time for plan modification and the dotted line plots the time for matching using a problem-independent strategy. The solid line plots the resulting time for matching and modification.

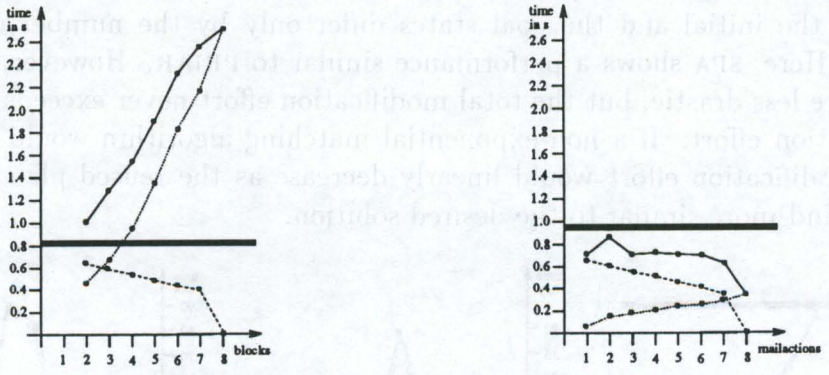
When the modification tasks become more difficult, since the reuse candidate and the new planning instance are structurally less similar, the savings of plan modification become less predictable. A phenomenon which we observed is the occurrence of peaks in the plan modification effort.[9]

For the reuse of $n$bs1 problems to solve the 8bs1 problem the performance of plan modification becomes worse. The peak is higher and the phenomenon occurs for more reused planning instances. We have no explanation for this phenomenon and furthermore it does not coincide with performance measures reported in [24].

With our last experiment, we want to highlight the influence of the application domain on the performance of plan-reuse techniques. Running MRL on $n$bs $\rightarrow$ 8bs instances and on mail domain instances, we obtained the runtime behavior displayed in Figure 9. It should be noted that we used the same proof tactics and order-sorted unification algorithm for both example sets.

The experiments demonstrate that the effort for planning from scratch

[9]The observed runtime behavior correlates linearly with the number of considered partial plans. In other words, the runtime peaks are not caused by any machine-dependent features but by the plan-modification process.



a) MRL: blocks world

b) MRL: mail domain

Figure 9: Matching and modification costs in MRL. As above, the grey horizontal bar indicates plan generation time, the dashed line plots the time for plan modification, and the dotted line plots the time for matching. The solid line plots the resulting time for matching and modification.

and for plan modification is almost the same for both problems, but they differ significantly in the effort which has to be spent on matching. In the blocks world, matching is much more expensive because the goal state description is very homogeneous, i.e., all objects are of the same sort. This leads to many different matching possibilities. In the mail domain we have fewer objects and they are of different sorts, which makes matching less expensive since the unification algorithm can benefit from the sort information—an observation supporting our conjecture that many-sorted logics in heterogeneous domains can lead to a significant efficiency gain for the matching problem (see Section 4.1).

The different matching costs lead to different relative performance figures by plan reuse in MRL: in the mail domain, solving the current problem by reusing a given plan leads to an efficiency gain, while solving the blocks world problem by plan reuse is always more expensive.

6 Conclusions

Improving the efficiency of planning systems by adding capabilities to modify existing plans has received some research interest recently. We considered the relationship between plan reuse and plan generation from an analytical and empirical point of view in this paper.

In analyzing the relative computational complexity of plan modification

versus plan generation, we showed that plan modification is as hard as plan generation and *sometimes modification is even harder than planning from scratch.* We also showed that these results hold under the restriction that the modification process has to account for only one changed atom in the goal specification. In particular, we proved that deriving the maximal reusable subplan is not easier than planning. Hence, we cannot hope for minimizing planning effort by first identifying the maximal applicable subplan which is then (minimally) extended by plan generation. In fact, in plan-reuse systems, plan modification is not attacked as a problem but considered as a heuristic technique. This means that instead of *using as much of the old plan as possible* these systems recycle *as much of the old plan as the particular planning algorithm will perhaps be able to use in solving the new problem instance.* Hence, adopting the principle of *conservatism* in plan modification only seems to make sense in a replanning context where one wants to minimize the perturbation of the original plan.

Although plan modification does not lead to a provable efficiency gain in terms of computational complexity, it seems intuitively plausible that reusing old plans can sometimes (perhaps in a significant number of cases) lead to an improvement in efficiency. However, in order to exploit plan-reuse techniques in the general case, it is necessary to select an appropriate reuse candidate from a plan library. The bottleneck in retrieving such a candidate from the library seems to be that the *matching* problem, the problem of matching the objects of the reuse candidate to the objects of the new planning situation, is already quite difficult. As we show, this problem is NP-hard in general. This holds even for moderately simple blocks-world planning instances. Only in the case that there is exactly one stack in the goal description, the matching problem is solvable in polynomial time.

Complementing our analytical results by experiments with existing plan-reuse systems, we noted that the relative efficiency gain of plan-reuse techniques depends crucially on the efficiency of the underlying planning system. In particular, we noted that for the systems SPA and MRL the relative savings by plan-reuse techniques were significantly less drastic than with the PRIAR system. One main reason seems to be that the plan generation systems used in SPA and MRL are much more efficient than the generative component of PRIAR. Furthermore, we noted that the structural similarity between the reuse candidate and the new planning instance can have a significant influence on the performance of the plan modification process. As a matter of fact, for a large number of structurally not too similar planning instances, the reuse costs were higher than the costs of planning from scratch. Further, we noted in this context that the costs of matching (only one candidate against the new planning instance) can already be significant. Finally, we compared the effect of different application domains on the reuse effort (in the MRL system). Interestingly, the plan modification effort did not change, but the

matching costs were much higher in the blocks-world domain (with a large number of objects of the same type) than in the mail domain (with fewer objects that are of different types).

Summarizing, we conclude that plan-reuse techniques may be of a more limited value than previously thought. First of all, it is not clear for which type of domains and/or problems plan-reuse techniques lead to a *predictable* efficiency improvement. As a matter of fact, in a large number of experiments we observed that plan-reuse can be more expensive than planning from scratch! Second, even if such domains and problems have been identified, there is still the problem of how to solve the retrieval and matching problem efficiently.

Acknowledgements

We would like to thank Christer Bäckström, Tom Bylander, and Subbarao Kambhampati, and two anonymous IJCAI referees, who provided helpful comments on an earlier version of this paper. In particular, Tom's remarks and questions heavily influenced the paper. We would also like to thank Steven Hanks and Daniel Weld, who made their SPA system available to us and answered our questions patiently.

References

[1] *Proceedings of the 9th National Conference of the American Association for Artificial Intelligence*, Anaheim, CA, July 1991. MIT Press.

[2] *Working Notes of the AAAI Spring Symposium "Computational Considerations in Supporting Incremental Modification and Reuse"*, Stanford University, Mar. 1992.

[3] *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, Washington, D.C., 1992. Morgan Kaufmann.

[4] C. Bäckström and I. Klein. Parallel non-binary planning in polynomial time. In IJCAI-91 [26], pages 268–273.

[5] C. Bäckström and B. Nebel. Complexity results for SAS^+ planning. In IJCAI-93 [27]. To appear.

[6] M. Bauer, S. Biundo, D. Dengler, J. Koehler, and G. Paul. PHI - a logic-based tool for intelligent help systems. In IJCAI-93 [27]. To appear.

[7] S. Biundo and D. Dengler. The logical language for planning LLP. DFKI Research Report, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, 1993. To appear.

[8] S. Biundo, D. Dengler, and J. Koehler. Deductive planning and plan reuse in a command language environment. In ECAI-92 [15], pages 628–632.

[9] T. Bylander. Complexity results for planning. In IJCAI-91 [26], pages 274–279.

[10] T. Bylander. Complexity results for extended planning. In AIPS-92 [3].

[11] T. Bylander. An average case analysis of planning. In *Proceedings of the 11th National Conference of the American Association for Artificial Intelligence*, Washington, DC, July 1993. MIT Press. To appear.

[12] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 1993. To appear.

[13] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, July 1987.

[14] S. V. Chenoweth. On the NP-hardness of blocks world. In AAAI-91 [1], pages 623–628.

[15] *Proceedings of the 10th European Conference on Artificial Intelligence*, Vienna, Austria, Aug. 1992. Wiley.

[16] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[17] T. Eiter and G. Gottlob. The complexity of logic-based abduction. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *Proceedings Tenth Symposium on Theoretical Aspects of Computing STACS-93*, pages 70–79, Würzburg, Germany, Feb. 1993. Springer-Verlag.

[18] K. Erol, D. S. Nau, and V. S. Subrahmanian. On the complexity of domain-independent planing. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence*, pages 381–386, San Jose, CA, July 1992. MIT Press.

[19] K. Erol, D. S. Nau, and V. S. Subrahmanian. When is planning decidable? In AIPS-92 [3], pages 222–227.

[20] M. R. Garey and D. S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979.

[21] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2):223–254, 1992.

[22] K. J. Hammond. Explaining and reparing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.

[23] S. Hanks and D. S. Weld. Systematic adaptation for case-based planning. In AIPS-92 [3], pages 96–105.

[24] S. Hanks and D. S. Weld. The systematic plan adaptor: A formal foundation for case-based reasoning. Technical Report TR 92-09-04, University of Washington, Department of Computer Science and Engineering, Seattle, WA, Sept. 1992.

[25] A. E. Howe. Failure recovery analysis as a tool for plan debugging. In AAAI Spring Symp. REUSE '92 [2], pages 25–30.

[26] *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, Aug. 1991. Morgan Kaufmann.

[27] *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, Aug. 1993.

[28] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A*, pages 67–161. MIT Press, 1990.

[29] S. Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach.* PhD thesis, University of Maryland, College Park, 1989.

[30] S. Kambhampati. Utility tradeoffs in incremental plan modification and reuse. In AAAI Spring Symp. REUSE '92 [2], pages 36–41.

[31] S. Kambhampati and J. A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[32] H. Kautz and B. Selman. Planning as satisfiability. In ECAI-92 [15], pages 359–363.

[33] J. Koehler. Towards a logical treatment of plan reuse. In AIPS-92 [3], pages 285–286.

[34] J. Koehler. Flexible plan reuse in a formal framework. DFKI Research Report, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, 1993. To appear.

[35] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In AAAI-91 [1], pages 634–639.

[36] B. Nebel. Belief revision and default reasoning: Syntax-based approaches. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference*, pages 417–428, Cambridge, MA, Apr. 1991. Morgan Kaufmann.

[37] B. Selman and H. Levesque. Abductive and default reasoning: A computational core. In *Proceedings of the 8th National Conference of the American Association for Artificial Intelligence*, pages 343–348, Boston, MA, Aug. 1990. MIT Press.

[38] A. Tate. Generating project networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 888–893, Cambridge, MA, Aug. 1977.

[39] M. M. Veloso. Automatic storage, retrieval, and replay of multiple cases using derivational analogy in PRODIGY. In AAAI Spring Symp. REUSE '92 [2], pages 131–136.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohui Wu, Ansgar Bernardi, Christoph Klauck: Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-30
Rolf Backofen, Gert Smolka:
A Complete and Recursive Feature Theory
32 pages

RR-92-31
Wolfgang Wahlster:
Automatic Design of Multimodal Presentations
17 pages

RR-92-33
Franz Baader: Unification Theory
22 pages

RR-92-34
Philipp Hanschke: Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer:
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke:
Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke: Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer:
An Alternative to H-Subsumption Based on Terminological Reasoning
9 pages

RR-92-40
Philipp Hanschke, Knut Hinkelmann: Combining Terminological and Rule-based Reasoning for Abstraction Processes
17 pages

RR-92-41
Andreas Lux: A Multi-Agent Approach towards Group Scheduling
32 pages

RR-92-42
John Nerbonne:
A Feature-Based Syntax/Semantics Interface
19 pages

RR-92-43
Christoph Klauck, Jakob Mauss: A Heuristic driven Parser for Attributed Node Labeled Graph Grammars and its Application to Feature Recognition in CIM
17 pages

RR-92-44
Thomas Rist, Elisabeth André: Incorporating Graphics Design and Realization into the Multimodal Presentation System WIP
15 pages

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster:
WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi:
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert:
Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta:
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen:
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-20
Franz Baader, Bernhard Hollunder:
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller:
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy:
Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel:
Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

DFKI Technical Memos

TM-91-13
Knut Hinkelmann: Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Busemann: Prototypical Concept Formation An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

DFKI Documents

D-92-14
Johannes Schwagereit: Integration von Graph-Grammatiken und Taxonomien zur Repräsentation von Features in CIM
98 Seiten

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.):
UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

D-92-22
Werner Stein: Indexing Principles for Relational Languages Applied to PROLOG Code Generation
80 pages

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.):
Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch:
Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth: Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz Saarbrücken 29.-30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch: Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis

Bernhard Nebel, Jana Koehler

RR-93-33
Research Report