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

Bernhard Nebel[a,b,1] Jana Koehler[b]

[a] *Universität Ulm, Fakultät für Informatik, Oberer Eselsberg, D-89069 Ulm, Germany*

[b] *German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany*

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 some situations, 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 two different plan reuse systems, pointing out possible pitfalls one should be aware of when attempting to employ reuse methods.

* 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 published in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, September 1993.

*** To appear in *Artificial Intelligence*.

[1] Corresponding author. E-mail: nebel@informatik.uni-ulm.de.

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 [18,33], integrate domain-dependent heuristics [21] or investigate reuse in the general context of deductive planning [3,5].

Some experiments give evidence that planning based on second principles might indeed be more efficient than planning from scratch [19,20,23,25,33]. However, it is by no means clear how far these results generalize. Addressing this problem, we analyze the computational problems of plan modification from an analytical and empirical point of view in order to identify possible pitfalls one should be aware of when employing reuse techniques.

Using a propositional planning framework, we show that modifying a plan is not easier than planning from scratch. Moreover, there exist special cases when modifying a plan *conservatively* [25, p. 196] can be harder than generating a plan from scratch, even if we assume that the old and the new instance are similar. From that we conclude that *conservative* plan modification—which is in fact an extremely dogmatic view of plan reuse—runs counter to the idea of increasing efficiency by plan reuse. For this reason, a conservative modification strategy should only be employed in a *replanning* context—when it is crucial to retain as many steps as possible—but not in a *plan reuse* context. In fact, all existing plan reuse system do not use a conservative modification strategy. Instead, plan modification is considered as a heuristic technique, which recycles as much of the old plan as the particular planning algorithm can *probably* use.

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 some situations 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 on the performance of two different plan-reuse systems, namely, SPA [19,20] and MRL [29]. The aim of this analysis is to identify factors influencing the efficiency gains of plan-reuse techniques. Although we used only a very narrow class of test cases, the experiments provide nevertheless a qualitative indication of the performance of reuse techniques and an idea of how different factors can influence the relative efficiency of reuse techniques.

The paper is organized as follows. In Section 2, we define the notion of propositional STRIPS planning following Bylander [6] and introduce a formal model of plan modification following Kambhampati and Hendler [25]. 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 [1,2,6,10,11,14,15,17]. 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 [6]. As Bylander [6] 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 [6], 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 $\perp$.[2]

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.[3]

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 $\mathit{Result}(\mathcal{I}, \Delta)$ satisfies the goal

[2] This is a slight deviation from Bylander's [6] 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.
[3] We assume that the reader is familiar with the basic notions of complexity theory as presented, for instance, by Garey and Johnson [16].

specification. PLANMIN [8] 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 [6,8] 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:

(i) *Identifying* an appropriate reuse candidate from a plan library.
(ii) *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 [23,25] and Hanks and Weld [19,20]. In this case, which we consider in Section 4, the costs of reuse may also be influenced by the retrieval problem.

Kambhampati and Hendler [25, 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 [25, 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 [33, p. 133]. The reason for this *conservative* approach to modification is twofold [25, p. 194–195]. Firstly, in a *plan-reuse* context, it is expected that the additional planning effort necessary to generate the new plan is minimized if the reused part of the old plan is maximized. Secondly, in a *replanning context*, i.e., when a plan has to be modified because of user-initiated specification changes or execution failures, one may want to respect as many previous commitments as possible.

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., [19, p. 96]).

Finally, it should be noted that although the framework we have defined above deals only with total-order plans, it can be easily modified to apply to partial-order planning, as well. Furthermore, all hardness results will apply to partial-order planning since total-order plans are simply special cases of partial-order 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 (but not all) turn out to be computationally harder than deduction [12,31].[4] A similar result

[4] 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., [16,22]), this implies that revising a propositional theory is harder than deduction, which is Π_1^p- or co-NP-

[32,13] 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 plans conservatively, for arbitrary and similar planning instances. After that, we consider the possible efficiency gain of less restricted modification strategies.

3.1 Modifying Arbitrary Plans Conservatively

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 [8] or more global properties [2]). If PLANSAT_ρ is a restricted planning problem, then MODSAT_ρ shall denote the corresponding modification problem with the same restrictions.

Proposition 1 PLANSAT_ρ *transforms polynomially to* MODSAT_ρ *for all restrictions* ρ.[5]

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

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

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

Theorem 3 *If* PLANSAT_ρ *is* PSPACE*-complete or* NP*-complete, then* MODSAT_ρ *is a* PSPACE*-complete or* NP*-complete problem, respectively.*

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_ρ *problem such that the corresponding* MODDEL_ρ *and* MODDELINS_ρ *problems are* NP-*complete.*

[5] Proofs of theorems and propositions can be found in the appendix.

complete.[6]

This means that it can be harder to modify a plan than generating it from scratch. The reason for this fact is that the conservativity requirement introduces an additional source of computational complexity. This source of complexity is not visible when planning is NP-hard, because it requires simply another nondeterministic choice. However, it shows up in the case when planning itself is easy. Hence, the expectation that *conservatism* leads to increased efficiency does not seem to be justified.

3.2 Modifying Plans Conservatively 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* $\mathrm{PLANSAT}_\rho$ *problem such that the corresponding* $\mathrm{MODDEL1G}_\rho$ *and* $\mathrm{MODDELINS1G}_\rho$ *problems are* NP-*complete.*

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

[6] We were not able to identify a polynomial planning problem $\mathrm{PLANSAT}_\rho$ such that the corresponding MODMIX_ρ problem becomes NP-complete.

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 planning problem instance—and ignore the size of the plan to be modified.[7] Under these assumptions, it is possible to specify a Turing reduction from PLANSAT$_\rho$ to MODGEN1G$_\rho$.

Theorem 6 *If* PLANSAT$_\rho$ *is* PSPACE*-hard or* NP*-hard, then the corresponding* MODGEN1G$_\rho$ *problem is* PSPACE*-hard or* NP*-hard, respectively, in the size of the planning problem instance.*

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 additional complexity 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.*

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

[7] 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.

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* [25, 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. [8]

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 [19].

While this seems to be indeed a reasonable way to go, it is (of course) not a guaranteed cure for intractability. As the proof of Theorem 6 indicates, modifying a plan cannot be easier than generating one, even if we allow for arbitrary modification strategies.

Theorem 8 *If* PLANSAT$_\rho$ *is* PSPACE*-hard or* NP*-hard, then the corresponding* MODGEN1G$_\rho$ *problem is* PSPACE*-hard or* NP*-hard, respectively, in the size of the planning problem instance, even if we do not require to reuse a maximal subplan.*

As demonstrated by Theorem 8, we cannot hope for a provable speedup by plan-reuse techniques in terms of computational complexity. Nevertheless, one

[8] Kambhampati makes the same distinction in a later paper [24]. 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.

would expect a speedup in some cases. In fact, Bylander [7] 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. He assumes a number of operators that is exponential in the average size of the pre- and postconditions. While this appears to be an unrealistic assumption, Bylander's result is some indication on the analytical side that plan modification could be sometimes more efficient than planning from scratch.

One of the interesting and challenging problems in the research on plan reuse seems to be the identification of conditions under which plan reuse leads to a provable speedup. A possible candidate has been pointed out to us by one of the anonymous reviewers. *Variant process planning* [9], which is used in commercial manufacturing industries for generating process plans for a given product design, is based on (manual) modification of plans for similar product designs. Since in this case a similar design implies a similar plan, reusing old plans leads indeed to significant efficiency gains.

4 Plan Retrieval and Matching

Experiments in the blocks-world domain [19,20,23,25] 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.

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_j^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

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

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*[9]

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

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.,

[9] A function f is *injective* if it is "invertible," i.e., if $x \neq y$ implies $f(x) \neq f(y)$ for all x and y in the domain of f.

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

If there exists a *bijective* [10] 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 reused for solving Π' since Π' and Π are identical to within a renaming of constant symbols, i.e., $\mu(\Delta)$ 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 [20] and Kambhampati and Hendler [23,25], 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" [20, 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

[10] A function is *bijective* if it is injective and onto.

overall system performance. Unfortunately, the matching problem is an NP-hard problem.

Theorem 9 PMATCH *is* NP-*hard, even if the initial states are empty.*

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 criterion 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 [28]. 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.

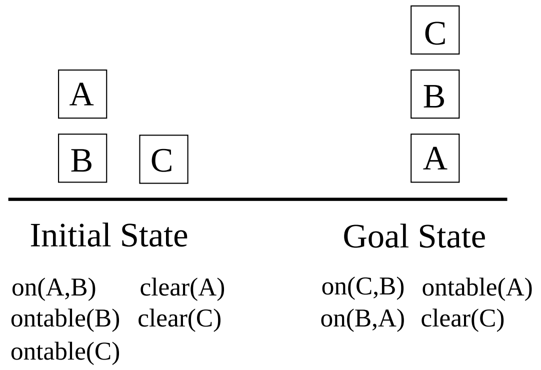


Fig. 1. A Blocks-World Example

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 `ontable`$(\cdot)$ and `clear`$(\cdot)$, then there are at most two mappings with a maximal number of goal atoms in common. It is then easy to identify the mapping that maximizes the match between the initial states.

Proposition 10 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 11 PMATCH *restricted to blocks-world planning instances, where the goal is a complete description of a set of stacks, is* NP*-hard.*

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 11 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.

Interestingly, (non-optimal) planning in the blocks-world is polynomial, even if there are many goal stacks [17]. In other words, in case of a special-purpose blocks-world planning system one better does not use a retrieval algorithm that identifies the optimal reuse candidate, but one that also accepts candidates that are less than perfect. Otherwise retrieval may become more expensive than plan generation.

5 Empirical Results

In order to complement our analytical results on the relationship between plan reuse and plan generation, we conducted some experiments to gain insight into the performance of reuse techniques under varying conditions. We were particularly interested in how the following conditions influence the efficiency gains of 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 planning domain:* properties of the planning domain can probably render matching and modification more or less difficult.

5.1 Plan-Reuse Systems

In our experiments, we used the plan-reuse systems SPA [19,20] and MRL [27,29].

SPA is based on a lifted version of McAllester and Rosenblitt's [30] systematic partial-order 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). 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*, 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 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.

The other plan reuse system we consider is MRL, which is based on the deductive (total-order) planner PHI [3,5]. The underlying logic of this planning system is the interval-based modal logic LLP [4]. It should be noted that in using this logic in a planning system it becomes possible to specify intermediate 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 [26]).

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 SPA, 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 – because systematic planners usually require exponential time.

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.

The systems use different planning formalisms, are implemented in different programming languages, and run on different machines. Therefore, a runtime performance comparison between the systems does not appear to be meaningful. Instead, we are interested in the *relative* efficiency caused by plan reuse when the above mentioned conditioned are varied. Although, we used only a quite narrow class of test cases, we still believe that our results provide at least a qualitative indication of the relative efficiency of plan-reuse techniques under varying conditions.

5.2 Test Cases

For our experiments, we considered test cases from two different domains. The first domain is a particular subset of blocks-world planning instances that has been used to explore the performance of PRIAR and SPA [23,25,20].

The blocks-world planning instances we 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 have 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 2 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" [17] 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 test case collection. Most probably, this property simplifies the generation and modification of plans. For this reason and because of the fact that optimal plans can be found in polynomial time for all blocks-world problem instances containing only one stack in their goal description [17], we believe that the claim [25, p. 198] that "experiments in the blocks-world certainly bear out the flexibility and efficiency of the incre-

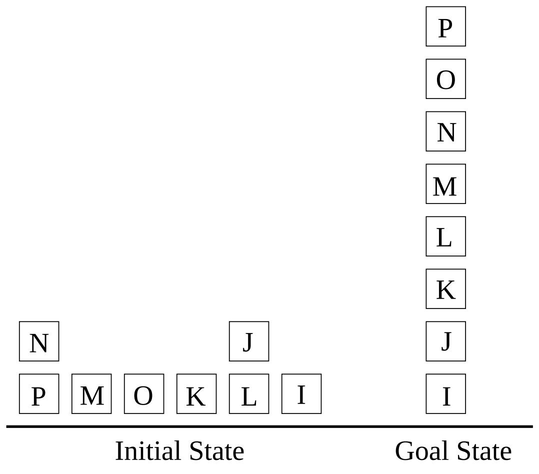


Fig. 2. The 8bs1 Example

mental plan modification ...over a variety of specification changes" is at least arguable. Nevertheless, this set of test cases appears to be useful for getting an idea how the relative efficiency differs, because these test cases have been used in evaluating different plan-reuse systems.

In order to analyze the effect different domains can have on the efficiency of the plan-reuse process, we considered also another domain. The second domain is the UNIX mail domain, which we used only in connection with the MRL system. In the mail domain, objects of different sort like *messages* and *mailboxes* are manipulated by actions like *read*, *delete*, and *save*. This domain differs from the blocks-world mainly in that the objects are all of different type.

5.3 Experimental Results

We ran different test samples on the two plan-reuse systems in order to get an idea how the performance of the reuse system vary under different conditions.

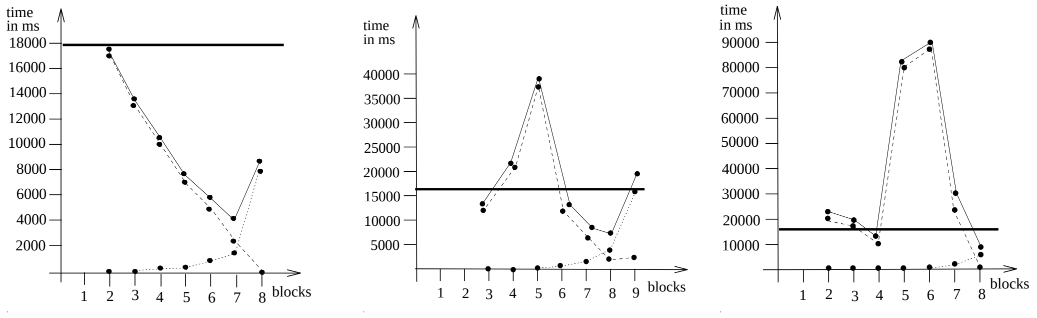
Influence of Similarity of Planning Instances

In the first experiment, we investigate 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 that are predefined in the plan library of SPA. 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 adaptation becomes more difficult moving from the first kind of tasks to the latter kind of tasks.

In Figure 3,[11] 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.[12] 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.

Figure 3a gives the performance data for the easiest modification problem, where the initial and the goal states differ only by the number of blocks used, in which case the total modification effort never exceeds the plan generation effort. If a linear 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

Fig. 3. Matching and modification costs in SPA. The 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. In particular, it happened that the modification and matching effort is higher than the effort of generating a plan from scratch.[13] For the reuse of $n$bs1 problems to solve the 8bs1 problem, the

[11] Each data point represents the average of 20 runs on a freshly initialized system. The deviation of a single run never exceeded 10%.

[12] SPA also provides an application-dependent matching algorithm which is linear but restricted to blocks-world instances with 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

[13] 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-

performance of plan modification becomes worse. There are more cases when plan reuse is less efficient than plan generation and plan modification can take more than 5 times as much time as generation.

Comparing these results with the empirical data on the performance of the PRIAR plan-reuse system reported by Kambhampati and Hendler [25], one notes that instead of a speedup in all cases, there are a number of cases when plan modification is actually more expensive. The reasons for these results are manifold. First of all, we did not employ the domain-dependent *control functions* SPA offers for the blocks-world domain.[14] Secondly, as already noted by Hanks and Weld [20], PRIAR's generative performance degrades much more quickly than its modification performance, leading to impressive savings for large instances.

All in all, the experiments indicate that there is a certain danger that the modification effort may be in fact higher than the generation effort if the reuse candidate is not structurally similar to the new problem instance. Hence, we have an interesting tradeoff for the plan retrieval component. If we try to retrieve the reuse candidate with a best match, we may have a good chance that the plan can be easily modified, but the retrieval itself can be costly. On the other hand, if the retrieval component only performs an approximate match, matching might be inexpensive, but the modification effort can be quite high.

Influence of Planning Domain

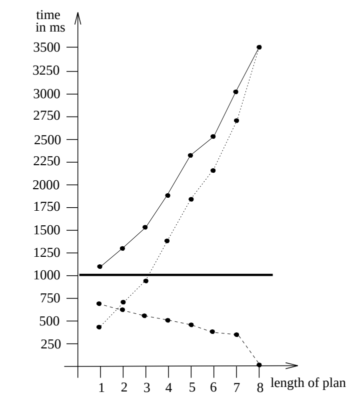
With our second experiment, we want to highlight the influence of the application domain on the performance of plan-reuse techniques. In the experiment, we considered in addition to the blocks-world the UNIX mail domain, which is quite different from the blocks-world. Typical planning instances in the 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).

Running MRL on $n$bs $\rightarrow$ 8bs instances and on mail domain instances, we obtained the runtime behavior displayed in Figure 4.[15] It should be noted that we used the same proof tactics and order-sorted unification algorithm for

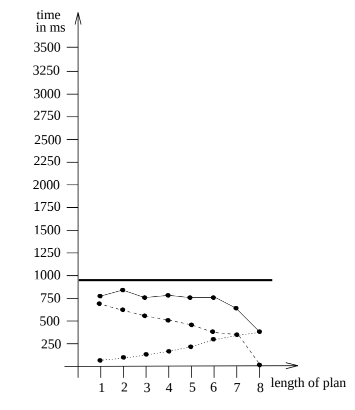
dependent features but by the plan-modification process.

[14] The reader should note that the use of control functions leads to a much better performance of the system as reported in [20].

[15] As above, the data points represent the average of 20 runs on a freshly initialized system.



a) MRL: blocks world



b) MRL: mail domain

Fig. 4. Matching and modification costs in MRL. As above, the 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.

both example sets.

The experiments demonstrate that the effort for planning from scratch 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 for 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—as it occurs in planning from second principles and case-based planning—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 the same combinatorial cliffs that exist for planning from scratch also exist for plan reuse. Hence, case-based planners do not offer a guaranteed cure for intractability. In fact, plan modification can even be harder than planning from scratch—if we require the modification to be *conservative.* While *conservatism* has been discussed in the literature as a possible way to increase the efficiency, it turns out that this hope is not justified.

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.

The identification of sources of computational complexity raises the question of how implemented systems cope with the combinatorial cliffs. This motivated experiments with existing plan-reuse systems in order to identify possible pitfalls for reuse techniques. Summarizing our empirical results, we noted that

- if the underlying planning system is already very efficient (for a given domain), the costs for matching and modification can easily be higher than the costs for generating a plan from scratch;
- if the reuse candidate is not structurally similar to the new instance, the modification effort can be much higher as in the case when the candidate is similar;
- if a domain-independent optimal match between candidate and new instance is sought, the retrieval costs can be quite high;
- if the planning domain is heterogeneous (i.e., different objects have different types), matching becomes much more efficient.

Our future research will concentrate on further theoretical and empirical investigations of plan-reuse techniques. We are particularly interested in identifying conditions under which plan reuse is provably more efficient than plan generation. Further, we plan to analyze the empirical performance of different plan-reuse systems on more complex real-world domains in order to characterize the range of applicability of particular reuse techniques.

Acknowledgements

We would like to thank Christer Bäckström, Tom Bylander, Subbarao Kambhampati, and the two anonymous referees, who provided helpful comments and suggestions 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.

A Proofs

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$_\rho$. $\Box$

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

Proof. Because of Proposition 1 and the fact that PLANSAT is PSPACE-complete [6, 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. $\Box$

Theorem 3 *If* PLANSAT$_\rho$ *is* PSPACE*-complete or* NP*-complete, then* MODSAT$_\rho$ *is a* 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. $\Box$

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 [6, 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 [6] gives for PLANSAT$_1^+$, it is evident that PLANSAT$_1^{+,\overline{post}}$ can also be solved in polynomial time (see also [2]). 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:

T_i, $1 \leq i \leq m$, $v_i =$ true has been selected

F_i, $1 \leq i \leq m$, $v_i =$ false has been selected

S_i, $1 \leq i \leq m$, the truth value for v_i has been selected

E_i, $0 \leq i \leq m$, enable evaluation

C_j, $1 \leq j \leq n$, c_j evaluates to true.

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

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

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 $\text{MODDELINS}_1^{+,\overline{post}}$, as well.

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

Theorem 5 *There exists a polynomial-time* PLANSAT_ρ *problem such that the corresponding* MODDEL1G_ρ *and* MODDELINS1G_ρ *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:

$$\mathcal{G}'_+ = \mathcal{G}_+ \cup \{B\}$$
$$\mathcal{G}'_- = \mathcal{G}_-.$$

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_ρ and MODDELINS_ρ 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. □

Theorem 6 *If* PLANSAT_ρ *is* PSPACE-*hard or* NP-*hard, then the corresponding* MODGEN1G_ρ *problem is* PSPACE-*hard or* NP-*hard, respectively, in the size of the planning problem instance.*

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. □

Theorem 8 *If* PLANSAT$_\rho$ *is* PSPACE*-hard or* NP*-hard, then the corresponding* MODGEN1G$_\rho$ *problem is* PSPACE*-hard or* NP*-hard, respectively, in the size of the planning problem instance, even if we do not require to reuse a maximal subplan.*

Proof. In the reduction used in the proof of Theorem 6, 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. □

Theorem 9 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 [16, 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 one-to-one function $f\colon V_2 \to V$ 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.

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

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

Theorem 11 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 [16, 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

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

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

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

in the following way (see also Figure A.1):

(i) 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 `ontable`$(b_{i,1})$, `on`$(b_{i,2}, b_{i,1})$, `on`$(b_{i,3}, b_{i,2})$, `clear`$(b_{i,3})$ to $\mathcal{G}'_+$.

(ii) 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.

(iii) 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.

(iv) 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 $\mathcal{I}$.

(v) 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 $\mathcal{I}$.

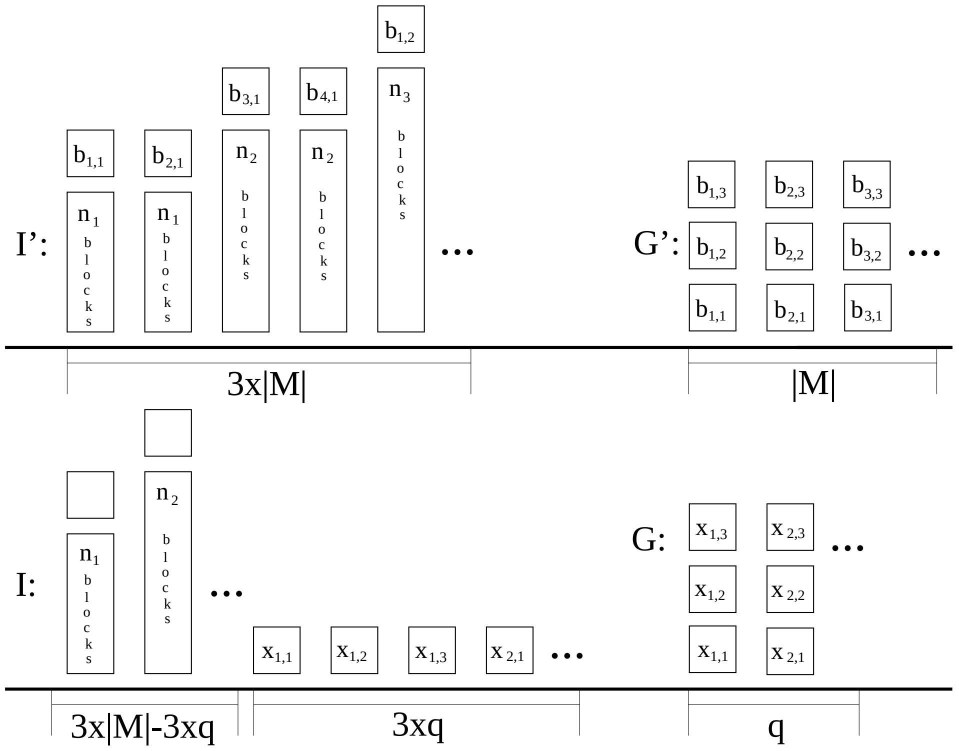


Fig. A.1. Reduction used in the proof of Theorem 11

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

References

[1] C. Bäckström and I. Klein. Parallel non-binary planning in polynomial time. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 268–273, Sydney, Australia, Aug. 1991. Morgan Kaufmann.

[2] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1430–1435, Chambery, France, Aug. 1993. A long version of this paper appeared as Research Report LiTH-IDA-R-93-34, Computer and Information Science, Linköping University, Sweden.

[3] M. Bauer, S. Biundo, D. Dengler, J. Koehler, and G. Paul. PHI—a logic-based tool for intelligent help systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 460–466, Chambery, France, Aug. 1993.

[4] S. Biundo and D. Dengler. The logical language for planning LLP. DFKI Research Report, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, 1994. To appear.

[5] S. Biundo, D. Dengler, and J. Koehler. Deductive planning and plan reuse in a command language environment. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 628–632, Vienna, Austria, Aug. 1992. Wiley.

[6] T. Bylander. Complexity results for planning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 274–279, Sydney, Australia, Aug. 1991. Morgan Kaufmann.

[7] T. Bylander. An average case analysis of planning. In *Proceedings of the 11th National Conference of the American Association for Artificial Intelligence*, pages 480–485, Washington, DC, July 1993. MIT Press.

[8] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

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

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

[11] S. V. Chenoweth. On the NP-hardness of blocks world. In *Proceedings of the 9th National Conference of the American Association for Artificial Intelligence*, pages 623–628, Anaheim, CA, July 1991. MIT Press.

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

[13] 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.

[14] 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.

[15] K. Erol, D. S. Nau, and V. S. Subrahmanian. When is planning decidable? In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pages 222–227, Washington, D.C., 1992. Morgan Kaufmann.

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

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

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

[19] S. Hanks and D. S. Weld. Systematic adaptation for case-based planning. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pages 96–105, Washington, D.C., 1992. Morgan Kaufmann.

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

[21] A. E. Howe. Failure recovery analysis as a tool for plan debugging. In *Working Notes of the AAAI Spring Symposium "Computational Considerations in Supporting Incremental Modification and Reuse"*, pages 25–30, Stanford University, Mar. 1992.

[22] 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.

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

[24] S. Kambhampati. Utility tradeoffs in incremental plan modification and reuse. In *Working Notes of the AAAI Spring Symposium "Computational Considerations in Supporting Incremental Modification and Reuse"*, pages 36–41, Stanford University, Mar. 1992.

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

[26] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363, Vienna, Austria, Aug. 1992. Wiley.

[27] J. Koehler. Towards a logical treatment of plan reuse. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pages 285–286, Washington, D.C., 1992. Morgan Kaufmann.

[28] J. Koehler. An application of terminological logics to case-based reasoning. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference*, Bonn, Germany, May 1994. Morgan Kaufmann. To appear.

[29] J. Koehler. Flexible plan reuse in a formal framework. In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning*, pages 171–184, Amsterdam, The Netherlands, 1994. IOS Press.

[30] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference of the American Association for Artificial Intelligence*, pages 634–639, Anaheim, CA, July 1991. MIT Press.

[31] 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.

[32] 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.

[33] M. M. Veloso. Automatic storage, retrieval, and replay of multiple cases using derivational analogy in PRODIGY. In *Working Notes of the AAAI Spring Symposium "Computational Considerations in Supporting Incremental Modification and Reuse"*, pages 131–136, Stanford University, Mar. 1992.