Plan-space vs. State-space Planning in Reuse and Replay

Laurie Ihrig & Subbarao Kambhampati
Department of Computer Science and Engineering,
Arizona State University, Tempe, AZ
85287-5406, USA
Email: {ihrig, rao}@asu.edu **Phone:** (602) 965-0113
FAX: (602) 965-2751

December 20, 1996

Key Words: Planning, Machine Learning, Case Based Reasoning

Subject Categories: Problem Solving/Planning, Learning

Abstract

The aim of case-based planning (CBP) is to improve the efficiency of plan generation by taking advantage of previous problem-solving experience. This paper explores the relative utility of placing CBP within plan-space vs. state-space planning. We argue that it is in the ability to adapt the previous case to the new situation that plan-space planning proves to be more efficient. To be able to extend a previous episode to solve the current problem situation, a planner needs the ability to *splice in* new steps into an existing sequence. The plan-space planner, which decouples the order of derivation of plan steps from their execution order, provides this capability. We will present controlled empirical studies that support our hypothesis regarding the relative advantage of plan-space planning in CBP. Our experiments demonstrate that this advantage holds whether we employ either of two CBP methods, which we call *plan reuse* and *derivation replay*.

Contents

List of Figures

List of Tables

1 Introduction

The aim of *Case-Based Planning* (CBP) [8, 15, 13] is to improve the efficiency of plan generation by exploiting the similarity between a new problem and previous problem-solving situations. CBP involves storing information in a *case library* about each planning episode as it is encountered. Each time that a problem is attempted, the library is consulted and a previous case judged to be similar to the new problem is retrieved. The earlier case is then adapted to meet the needs of the current situation.

When a retrieved case only partly solves the new problem, it is the task of the underlying planner to engage in further problem-solving effort to achieve any extra goals that are not covered. Since it is likely that further planning is needed, the efficiency of CBP will depend to a large extent on the ability of the planner to extend a case to solve a new problem. This paper is focused on the question: *In what way is the effectiveness of CBP influenced by the nature of the underlying planning strategy?* We consider two broad classes of domain-independent generative planning techniques: *state-space planners* which search in the space of world states, and *plan-space* planners which search in the space of partly-constructed plans. We systematically evaluate the tradeoffs in basing CBP in plan space vs. state space. The strategies are tested in two different CBP frameworks, representing alternative methods for exploiting previous planning experience. We call these frameworks *plan reuse* and *eager derivation replay* respectively. The two differ on what is actually stored in a case, as well as how the case is *fitted* to adapt to a new problem. In both methods, fitting produces a *skeletal plan* which contains all of the previous plan constraints which are relevant to the new situation. When the new problem contains extra goals not achieved within the skeletal

plan, then the underlying planner is employed to further refine this plan into a solution.

We will demonstrate through focused experimental studies in both reuse and replay that plan-space planners which make a weak commitment as to the ordering of steps have a greater ability to extend the skeletal plan. This has already been shown to be an advantage in generative planning [28, 1]. However, we find that the ability to *splice in* new steps and subplans into an existing sequence is of even greater benefit in exploiting previous experience through CBP. In our experiments plan-space and states-space planners show consistently wider performance differentials with replay than they do in from-scratch planning. We will provide an explanation as to why plan-space planning has an advantage in CBP over and above the potential benefits in generative planning.

Although we have conducted experiments to compare plan-space and state-space planning within two CBP frameworks, there are a number of different CBP systems which differ in the way that they adapt a previous case [8, 9, 24, 25, 33]. There is therefore a question as to whether our conclusions as to the benefits of plan-space CBP apply to these frameworks as well. Accordingly, in Section 6 we will also consider the applicability of our hypotheses to a variety of other CBP frameworks.

The rest of this paper is organized as follows: Section 2 describes the plan-space and state-space planning strategies and provides an overview of our approach to plan reuse and derivation replay. Section 3 presents some testable hypotheses concerning the advantages of plan-space planning in reuse and replay. Section 4 describes empirical studies that validate the hypotheses developed in the previous section. Section 5 discusses the cost benefits provided by reuse and replay. Section 6 discusses the ramifications of the experimen-

tal results in more sophisticated CBP frameworks. Section 7 describes some related work and Section 8 summarizes the contributions of this paper.

2 Background

In this section, we will briefly review the two classes of domain-independent planning strategies (section 2.1) as well as our reuse and replay CBP frameworks (section 2.2).

2.1 Domain Independent Planning

As we mentioned earlier, domain-independent planners come mainly in two varieties: state-space and plan-space [35]. Both navigate a space of states, armed with the problem specification. The problem is defined as a 3-tuple, $\langle I, G, \mathcal{A} \rangle$, made up of the initial state, I, the goal state, G, each represented as a conjunction of first-order literals, and a set, $\mathcal{A}$, of domain operators in STRIPS representation [6]. The state-space planner traverses a space of world states. Starting with the initial world state, each transition through this space represents the action accomplished by a single operator. With each application of the operator, the changes prescribed in the operator's add and delete lists are applied to the current position to produce the new current world state. The plan gradually forms as a side effect of this process; the path leading from the initial state is extended incrementally with each transition. The action that is associated with each step is appended to the end of the plan as the step is taken.

Because the ultimate aim of planning is to find a *ground operator sequence* which is a solution to the given problem (one which when executed from the initial state arrives at the goal state), another way of viewing the planning

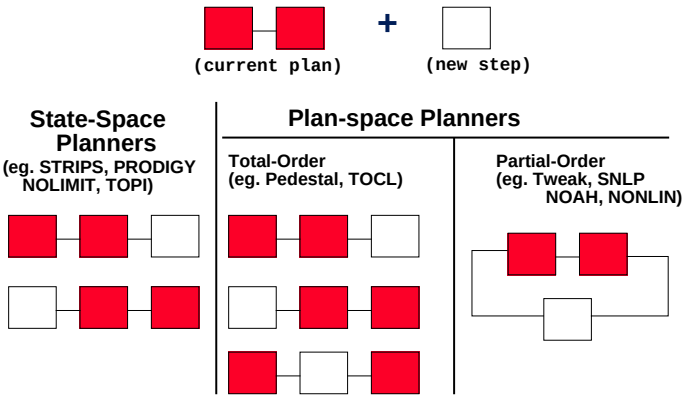
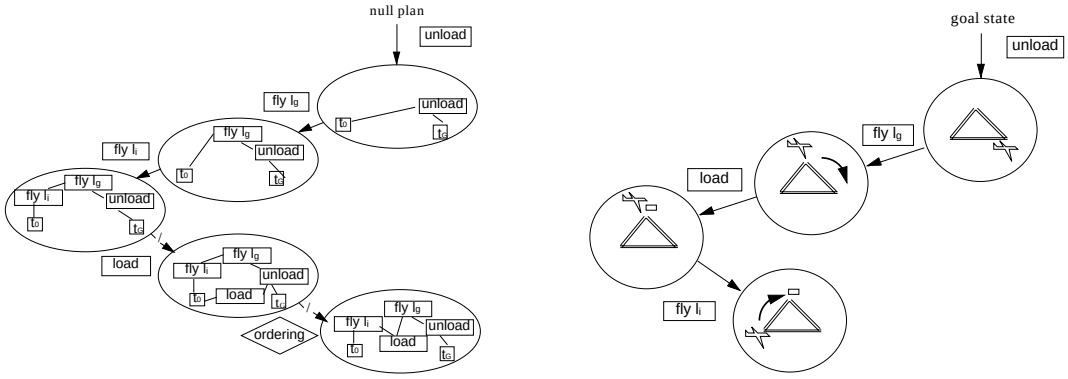


Figure 1: Step addition refinement in state-space and plan-space planners. In state-space planners, a new step is always added either to the beginning (in the case of forward state-space planning) or to the end (in the case of backward state-space planning) of the current step sequence. In plan-space planning, the new steps may be added anywhere in the current plan. The difference between the total-order and partial-order plan-space planners is that the former commit to a specific ordering right away, while the latter leave the new step unordered with respect to the existing steps, and order it only in response to future interactions.



(a) Plan Derivation in Plan-space Planning (SNLP)

(b) Plan Derivation in State-space Planning(TOPI)

Figure 2: A logistics transportation domain example showing the differences in plan derivation provided by state-space and plan-space planning. In both cases, a logistics plan for transporting one package from a location l_i to a location l_g is being derived.

action	(Load-Truck ?O ?T ?L)	action	(Load-Plane ?O ?P ?L)
precond	(at-object ?O ?L)	precond	(at-object ?O ?L)
	(at-truck ?T ?L)		(at-plane ?P ?L)
add	(inside-truck ?O ?T)	add	(inside-plane ?O ?P)
delete	(at-object ?O ?L)	delete	(at-object ?O ?L)
action	(Unload-Truck ?O ?T ?L)	action	(Unload-Plane ?O ?P ?Li)
precond	(inside-truck ?O ?T)	precond	(inside-airplane ?O ?A)
	(at-truck ?T ?L)		(at-plane ?P ?Li)
add	(at-object ?O ?L)	add	(at-object ?O ?Li)
delete	(inside-truck ?O ?T)	delete	(inside-plane ?O ?A)
action	(Drive-Truck ?T ?Li ?Lg)	action	(Fly-Plane ?P ?Li ?Lg)
precond	(at-truck ?T ?Li)	precond	(is-a AIRPORT ?Lg)
	(same-city ?Li ?Lg)		(at-plane ?P ?Li))
add	(at-truck ?T ?Lg)	add	(at-plane ?P ?Lg)
delete	(at-truck ?T ?Li)	delete	(at-plane ?P ?Li)
equals	(not (?Li ?Lg))	equals	(not (?Li ?Lg))

Figure 3: The specification of the Logistics Transportation Domain adapted for our experiments

process is as a search through a space of partly-constructed plans. This is the perspective taken by the plan-space planner. The action sequence is then not a side effect of the planning process; it is the object of the search. The set of ground operator sequences constitute a space of potential solutions. The planner begins with the null plan as the current active plan. Each transition to an adjoining operator sequence represents the addition of a constraint (step and/or step ordering) to the existing plan. The objective is to search this space by continually refining the plan until it lands on a step sequence that is a solution for the given problem.

It is interesting to note that state-space planning may also be seen as a process of constraint addition which can be considered a search in the space of plans [16]. However, from this viewpoint, the state-space planner has a restricted set of allowable refinements to the existing plan (see Figure 1). Since it determines which step to add on the basis of which operators may be applied in the current world state, steps may be added only to the end of the partial solution. Most planners used in learning research to date fall roughly in this category, in that they add steps contiguous to the end of a plan sequence, and advance the current state accordingly. These include forward-chaining means-ends analysis planners such as STRIPS [6], and PRODIGY [4] [1] There exist another class of planners which do backward search in the space of world states, starting with the goal state (for example, TOPI [1]). These planners add new actions only to the beginning of the partial solution during refinement. The current goal state is obtained by regressing the goal

[1]Notice that the *linearity assumption*, which specifies whether the planner manages its list of outstanding goals as a stack or an arbitrary list, has no effect on this. In particular, both PRODIGY, which makes the linearity assumption, and its extension NOLIMIT which doesn't (and thus allows interleaving of subgoals), both refine a partial plan by adding operators to the end of the current plan[33].

through the new step, and is used to determine the next action to be applied.

The perspective of plan-space planning (as opposed to state-space) provides more options as to the nature of the plan refinements that may be adopted (see Figure 1). Instead of considering the current world state, the plan-space planner looks into the sequence of actions which is formulated so far, and makes this the basis of its decisions. Some of the aspects of the partial plan which may be relevant are: step preconditions which are subgoals that are yet unsolved, whether there exist steps that could be made to achieve these goals given additional binding constraints and/or step orderings, and whether these goals would be achieved had it not been for other conflicting steps. Planners which make their choices as to additional constraints based on the current plan (as opposed to the state reached by executing the plan) may insert new steps into the plan. The new step may be interleaved anywhere in the partly-constructed solution, including in the middle of an existing step sequence. This adds to the flexibility of the planning process. It increases the likelihood of finding a solution on any given branch in the search tree.

A secondary source of flexibility for plan-space planners is their policy of weak commitment in ordering steps. Unlike state-space planners which maintain contiguous sequences of operators during their search (so that the current world state can be uniquely identified), some plan-space planners can search in the space of partial-order plans (see Figure 1). Many current-day planners such as NOAH [31], NONLIN [32], and SIPE [36] belong to the latter class, called partial-order (or PO) planners[2]. With a PO strategy, new steps

[2]Partial-order planners have also been called nonlinear planners. We prefer the former term since the latter gives the misleading impression that partial-order planning is related to the linearity assumption. In fact, as we mentioned earlier, the linearity assumption is concerned with the order in which different goals are attacked, and can be used in *any*

may be added in parallel to an existing step sequence. Later, as conflicts are detected between parallel operators, a step may be interleaved into the existing plan segment. This approach avoids commitment to arbitrary inter-operator orderings, thereby decreasing the likelihood of having to backtrack over these orderings [1, 28].

2.1.1 An Example

Figure 2 illustrates the behavior of plan-space and state-space planners in solving a simple problem taken from the logistics transportation domain described in [33] and adapted for our experiments as shown in Figure 3. Figure 2 shows the derivation of a solution to a problem in which a single package has to be transported between locations. It serves to illustrate the PO planners' commitment strategy when it comes to step orderings. SNLP adds orderings as required by the subgoaling structure, since steps that contribute conditions must precede the steps that require these conditions. Step orderings are also added to resolve conflicts between steps, for example, when one step is deleting the contribution of another. In the current example, this means that the LOAD action is first added in parallel to an existing step sequence. Further step orderings are then added to accomplish the interleaving of the new action into the existing segment.

Contrast this with TOPI's derivation of a solution to the same problem, also displayed graphically in Figure 2. Since this planner keeps its plans as total-order sequences, and extends a plan by adding steps to the beginning of the plan, the plan-step ordering is determined by the order in which goals

planner. The linearity assumption causes incompleteness in planners that search in the space of world states (such as STRIPS), but does not affect completeness in any way in planners that search in the space of plans.

are attempted. If this step order turns out to be unsuccessful, then the planner will have to backtrack to reach a solution. For example, if it has the plane leave the package location before it loads the package it will have to backtrack. For the plan-space planner, which has the ability to interleave steps into the plan, planning is complete without having to backtrack on goal order. As we will see later, it is this ability to interleave new steps into the plan which we believe gives the plan-space planner an advantage in plan adaptation.

2.2 The Reuse and Replay Frameworks

In this section, we describe our approach to plan reuse and derivational replay. These frameworks differ in two important phases: *storing* a case corresponding to the current planning episode in the library, and *fitting* a previous case to apply it to a new problem (see Figure 4). In plan reuse we store the final plan which is the *product of the planning episode*. In derivational replay, the *planning decisions* made during problem-solving are retained. The derivation trace that is stored in the library is a trace of the choices that lie along the derivation path leading from the root of the search tree to the final plan in the leaf node. These decisions then become instructions for a future search process.

Although both reuse and replay utilize a common retrieval strategy (see Section 2.2.4), the methods used to fit the case to the new situation differ. In the fitting phase, both reuse and replay take the retrieved case, and attempt to derive from it a *skeletal plan* which contains only the plan constraints that are judged to be *applicable* to the new problem situation. The specific strategy used to construct the skeletal plan is, however, different in reuse and replay. In reuse, the irrelevant parts of the plan are *retracted* to produce a

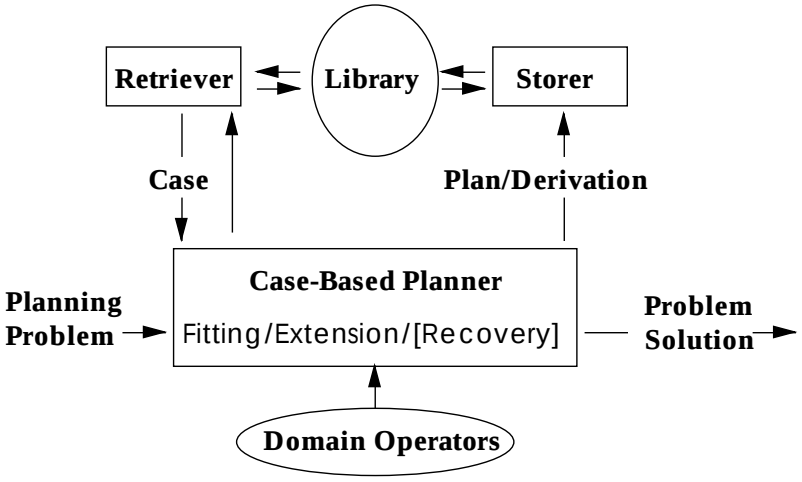
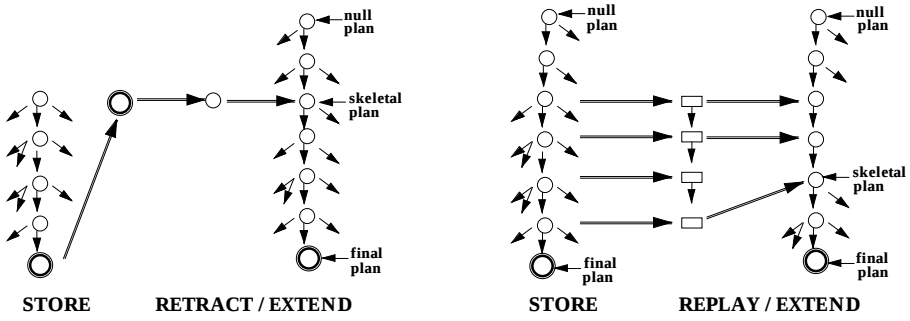


Figure 4: A schematic diagram illustrating the approach of case-based planning. In plan reuse, the final *plan* that is produced by each planning episode is stored in the case library. When a case is retrieved it is *fitted* to the new problem-solving situation by *retracting* the inapplicable parts of the plan. In derivational replay, a trace of the *derivation* of each solution is stored. A previous case is fitted to a new problem situation by *replaying* the decisions in the trace. In both, the plan that is produced is extended to achieve any extra goals of the new problem. When this extension fails a recovery phase is initiated.



(a) Plan Reuse

(b) Eager Derivation Replay

Figure 5: The differences in the approach of plan reuse and eager derivation replay. In reuse, inapplicable parts of the plan are retracted before the plan is incorporated into a node in the new search tree. In replay, the trace of the previous derivation is used as guidance to the new search process.

plan fragment which is then incorporated into the current active node of the new search process (see Figure 5). In eager derivation replay, the previous decisions made in deriving the plan are validated in sequence in the context of the new problem-solving context, and valid decisions are *replayed* to produce the skeletal plan. In the following sections we provide more details on these alternative fitting methods.

2.2.1 Case Fitting in Reuse

In plan reuse, fitting involves retracting the irrelevant plan segments [13, 15]. The algorithm takes as input the new problem specification, $\langle I', G', \mathcal{A} \rangle$, the previous plan, P_{old}, as well as an explanation for the correctness of the plan called its *validation structure* or *causal structure*. The validation structure is used to identify the parts of the plan which are irrelevant or inconsistent

with the new problem. This structure is made up of a number of *causal links* of the form $\langle s, p, s' \rangle$. A causal link contains the information that a step s is the *source* of an effect which is contributed to satisfy a precondition, p, of the destination step s'. Since a case may be considered as a candidate for retrieval if only a subset of the goals that it achieves are open conditions in the new context, there may be steps in the old plan which which are not relevant in the new planning situation. These may be identified through the validation structure by tracing the chains of causal links leading to the absent goals. The corresponding steps are then retracted from the old plan to produce the skeletal plan.

2.2.2 Case Fitting in Replay

In our replay strategy, fitting is accomplished through replay of the previous decisions recorded in the derivation trace. Since invalid decisions are skipped over, replay functions like retraction in plan reuse.

In order to understand the replay process, it is useful to think of planning decisions themselves as operators acting on partial plans. DerSNLP represents a partial plan as a collection of plan constraints [17]. In particular, a partial plan is represented as a 6-tuple, $\langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}, \mathcal{E}, \mathcal{C} \rangle$, where

1. $\mathcal{S}$ is the set of actions (step-names) in the plan, each of which is mapped onto an operator in the domain theory. $\mathcal{S}$ contains two dummy steps: t_I whose effects are the initial state conditions, and t_G whose preconditions are the input goals, G.

2. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and post-conditions of the operators which are represented in the plan

Type : ESTABLISHMENT	Type : ESTABLISHMENT
Kind : NEW STEP	Kind : NEW LINK
Preconditions :	Preconditions :
$\langle p, s\rangle \in \texttt{OpenConds}$	$\langle p, s\rangle \in \texttt{OpenConds}$
Effects :	Effects :
$\texttt{Steps}' = \texttt{Steps} \cup \{s'\}$	$\texttt{Links}' = \texttt{Links} \cup \{\langle s', p, s\rangle\}$
$\texttt{Links}' = \texttt{Links} \cup \{\langle s', p, s\rangle\}$	$\texttt{Orderings}' = \texttt{Orderings} \cup \{\langle s' \prec s\rangle\}$
$\texttt{Orderings}' = \texttt{Orderings} \cup \{\langle s' \prec s\rangle\}$	$\texttt{OpenConds}' = \texttt{OpenConds} - \{\langle p, s\rangle\}$
$\texttt{OpenConds}' = \texttt{OpenConds} - \{\langle p, s\rangle\}$	$\cup$ *preconditions*(s')
$\cup$ *preconditions*(s')	

Figure 6: Planning decisions have preconditions which are based on the current active plan and have effects which alter the constraints so as to produce the new current active plan.

steps, $\mathcal{S}$.

3. $\mathcal{O}$ is a partial ordering relation on S, representing the ordering constraints over the steps in $\mathcal{S}$.

4. $\mathcal{L}$ is a set of causal links of the form $\langle s, p, s'\rangle$ where $s, s' \in \mathcal{S}$.

5. $\mathcal{E}$ contains step effects, represented as $\langle s, e\rangle$, where $s \in \mathcal{S}$.

6. $\mathcal{C}$ is a set of open conditions of the partial plan, each of which is a tuple $\langle p, s\rangle$ such that p is a precondition of step s and there is no link supporting p at s in $\mathcal{L}$.

For DerSNLP, planning starts with the "null plan" (denoted by $\mathcal{P}_\emptyset$). Two choices made by SNLP are shown in Figure 6. The preconditions of these choices are specified in terms of the existence of an unachieved subgoal, p. The effects are the constraints that are *added* to the partial plan to achieve this open condition.

In replay, each of the previous decisions contained in the derivation trace is first compared with the current active plan to determine whether its precondition holds in the new context. Invalid decisions, those whose preconditions don't match, are skipped over. Previous choices which are judged to represent valid plan refinements are used as guidance to direct the new search process. Replaying a valid decision involves selecting a match for that decision from the children of the current active plan, and making this child the next plan refinement. When all of the previous decisions in the trace have been visited, the product of replay is a plan which contains all of the constraints prescribed by the case which are relevant to the new problem-solving context. Since invalid instructions have been skipped, the skeletal plan which is the end result of replay is comparable to the product of the retraction phase in plan reuse.

2.2.3 Case Extension and Recovery

In both reuse and replay, the constraints that are retracted/skipped are only those that are known a priori to be irrelevant. This does not mean that the constraints that are left in the skeletal plan are consistent with a solution. Whenever the skeletal plan is not a solution [3] the underlying planning strategy is employed to *extend* the skeletal plan to achieve a full solution to the new problem. There is always a possibility that this effort may fail in that the skeletal plan will not be able to be refined into a solution [30]. In these instances, completeness is preserved through the initiation of a *recovery* phase. In plan reuse, recovery is accomplished by starting the search process over from the beginning, starting at the null plan. In replay, recovery

[3] That is, whenever it contains open conditions corresponding to extra input goals or unsatisfied initial state conditions.

means backtracking over the skeletal plan and continuing the search process by exploring the siblings of the replayed path.

In replay, the fitting, extension and recovery phases do not represent an alteration in the underlying planning strategy. Each phase merely provides search control, directing the search process as to which node in the search tree to visit next. For example, case fitting is accomplished by directing the search down the previous derivation path. Case extension corresponds to exploring the subtree underneath the replayed path. Recovery involves backtracking over this path, and expanding its siblings.

Since eager derivation replay does not alter the underlying planning strategy, but merely provides search control, replay inherits all of the properties of the underlying planner. For example, when based within SNLP, it is complete, sound and systematic [27, 1]. It also means that case adaptation through replay is never strictly more difficult than from-scratch plan generation, whereas the same cannot be said for plan reuse [30].

In the next section, we will describe the retrieval strategy employed in both reuse and replay.

2.2.4 A Common Retrieval Strategy

Realistic domains like the logistics transportation domain described earlier add to the complexity of the retrieval process. There may be many domain objects of the same type, and an individual planning episode references a particular set of these objects. For a case to be widely applicable, the object constants must either be variablized as in [18], or alternatively, at the time of retrieval, we may create an object mapping which can be used to revise the old object names [13]. Here we adopt the latter strategy. We store cases representing individual planning episodes instantiated with the original

object constants. An object mapping is constructed when a case is considered as a candidate for retrieval. It is formed as a side effect of the process of mapping the old case onto the new problem specification. Candidate cases are then judged on their similarity to the new problem, given this mapping.

The algorithm used to determine whether a particular case is retrieved for reuse takes as input the new problem, $\langle I', G', \mathcal{A} \rangle$, the problem that was previously solved, $\langle I, G, \mathcal{A} \rangle$, the old plan for this problem, P_{old}, and the plan's validation structure. Determining the relevance of a case starts with the initial construction of a mapping of the goal contained in G onto G'. This mapping is done with the restriction that predicates map onto themselves, and object constants map onto themselves or other objects of the same immediate type.

Once a case is found which solves a subset of the new goals, then the case is further judged on the basis of the conditions satisfied through initial state effects. These are readily identified by the validation structure of the old plan. Each is represented as the condition p in a validation, $\langle t_i, p, s' \rangle$, in which the source is the dummy step, t_i. Since it is unlikely that a case can be found which is an exact match in all of these relevant initial state conditions, it is usual to judge the applicability of a case on the basis of a *similarity metric* which considers a case to be applicable if a set number of these conditions are consistent with the new initial state [13, 33]. Basing retrieval on a partial match increases the probability that a case will be found. However, it also means that the skeletal plan will contain action chains which are not executable in the new situation. The underlying refinement planner may then be used to extend these chains to reachieve some of the unsatisfied initial conditions. Moreover, since cases are retrieved which solve only a subset of the new problem goals, there also may be extra input goals present

in the new problem which were not covered by the previous case. Further planning effort is also needed to achieve these extra goals that were not attempted earlier. This means that the success with which a planner extends a partial solution into a complete plan will have an effect on the efficiency of CBP. The relative support provided by state-space and plan-space planners for this extension phase is the subject of the next section.

3 Analysis of Plan-Space and State-Space Case Extension Strategies

In this section, we analyze the advantages of plan-space planning in extending the skeletal plan which is obtained from either reuse, or, alternatively, from replay.

3.1 Case Extension in Plan Reuse

To understand the advantage of plan-space planning in reuse, we need to consider how the skeletal plan derived from the retrieved case may be extended into a solution for the new problem. Suppose a planner is attempting a new problem $\langle I', G', \mathcal{A} \rangle$, by extending a skeletal plan that is a ground operator sequence (or macro), M, which solves $\langle I, G, \mathcal{A} \rangle$, where $G \subset G'$.

Definition 1 (Step-Sequencable Macros) *We will say that M is step-sequencable with respect to the new problem $\langle I', G', \mathcal{A} \rangle$, if and only if there exists subplans N and N' such that $N \bullet M \bullet N'$ (where "$\bullet$" is the step-sequencing operator) will be a correct plan for solving the problem $\langle I', G', \mathcal{A} \rangle$.*

If the retrieved plan is a partial-order plan then the plan corresponds to a set of ground operator sequences. A step-sequencable partial-order plan is

defined as follows:

Definition 2 (Step-Sequencable Partial-Order Plans) *A partial-order plan consists of a collection of constraints P which define a set of ground operator sequences (macros). We will say that P is step-sequencable with respect to a new problem $\langle I', G', \mathcal{A} \rangle$, if and only if there exists a macro consistent with P which is step-sequencable with respect to $\langle I', G', \mathcal{A} \rangle$*

For some problems, those in which step order is critical, a retrieved macro may not be step-sequencable. This does not mean that the macro is inconsistent with any extension into a solution. It may be that a plan that solves the full set of goals contains all of the steps provided by the macro. It's just that a solution to the full problem may only be found by interleaving these steps with additional ones.

Definition 3 (Step-Interleavable Macros) *A macro M is said to be step-interleavable with respect to a new problem $\langle I', G', \mathcal{A} \rangle$, if and only if there exists a subplan M' such that M' may be merged with M to achieve $\langle I', G', \mathcal{A} \rangle$ without retracting any steps, step orderings or binding constraints in M, and without altering the ordering of the steps in M.*

Finally, M is said to be **step-modifiable** with a new problem if and only if there exists a subplan which can be merged with M to achieve a solution after retracting one or more steps or binding constraints in M.

Definition 4 (Step-Modifiable Macros) *A macro M is said to be step-modifiable with respect to $\langle I', G', \mathcal{A} \rangle$, if and only if there exists a subplan M' such that M' may be merged with M to achieve $\langle I', G', \mathcal{A} \rangle$ after retracting one or more steps or binding constraints in M, or after altering the ordering of the steps in M.*

Macro	State-space	Plan-space
step-sequencable	extendable	extendable
only step-interleavable	non-extendable	extendable
only step-modifiable	non-extendable	non-extendable

Table 1: Relationship between the step-sequencability of a macro with respect to the new problem to be solved and the ability of different refinement planners to extend the macro into a solution.

A step-interleavable partial-order plan is defined as follows:

Definition 5 (Step-Interleavable (Step-Modifiable) Partial-Order Plans) *A partial-order plan consisting of a collection of constraints P is step-interleavable (step-modifiable) with respect to a new problem $\langle I', G', \mathcal{A} \rangle$, if and only if there exists a macro consistent with P which is step-interleavable (step-modifiable) with respect to $\langle I', G', \mathcal{A} \rangle$*

Interleavability of macros, as defined here, differs from modifiability in that the latter also allows retraction of steps, binding constraints and step orderings from the macro. Clearly, interleavability is more general than sequencability, as modifiability is more general than interleavability.

From the definitions above, it is easy to see that if a retrieved plan P is only step-modifiable with respect to the new problem, then it contains constraints which are inconsistent with any extension of the problem. Ideally we want CBP to be able to extend a plan P as long as it is either step-sequencable or step-interleavable with respect to the additional goals. We note that since state-space planners may add steps only either to the beginning or to the end of the current step sequence, they will not be able to extend plans which are only step-interleavable. Plan-space planners, which

Type of Subgoals	Step Sequencability
independent	sequencable
serializable	*may be* sequencable
nonserializable	*not* sequencable

Table 2: Relationship between the serializability of problem goals for a state-space planner and the sequencability of a macro which solves a proper subset of those goals.

may add steps anywhere in the partial plan, may refine a previous plan also in the instances where steps have to be interleaved. Table 1 summarizes these differences.

3.1.1 When do we encounter interleavable macros?

In the foregoing, we noted that state-space planners cannot support adaptation of macros which are only step-interleavable, while plan-space planners may extend these macros. For this to make a difference in terms of performance, there must be many situations where a case-based planner retrieves macros which are only step-interleavable. In the following, we will provide examples of planning domains in which subplans that have to be interleaved are common, and characterize them in terms of the serializability of the subgoals in those domains.

Table 2 shows the relation between the types of goals present in the domain and the expected types of macro sequencability a case-based planner is likely to encounter. To understand this relationship, consider the simple artificial domains, ART-IND, ART-MD and ART-MD-NS originally described in [1] and shown in Figure 7. These domains differ in terms of the serializability of the goals in the domain [26, 1]. From the domain descriptions, it

can be seen that a conjunctive goal $G_i \wedge G_j$ (where $i < j$) in the ART-IND domain may be achieved by a state-space planner by attempting the two goals in any order, giving rise to two plans $A_i \rightarrow A_j$ and $A_j \rightarrow A_i$. All problems from this domain are made up of goals which are *independent* [26] and may be solved by a state-space planner by attempting goals in succession, and concatenating plans for individual goals [1]. In the case of the ART-MD domain, only the first of the two plans above will be a correct plan for solving $G_i \wedge G_j$. This is because the delete literals in the actions demand that G_i be achieved before G_j. Conjunctive goals in ART-MD are thus not independent. They are serializable, however, since a state-space planner is able to find the correct order in which to attempt its goals and achieve a solution by concatenating subplans for individual goals.

It is clear that in the ART-IND domain, which contains only independent subgoals, every plan will be step-sequencable with a new problem containing extra input goals. When the goals are serializable, as is the case in ART-MD, the distribution of stored macros may be such that the retrieved macro is not sequencable with respect to a new problem (see Table 2). For example, suppose the planner is trying to solve a problem with goals $G_1 \wedge G_2 \wedge G_3$ from the ART-MD domain, and retrieves a macro which solves the goals $G_1 \wedge G_3$: $A_1 \rightarrow A_3$. Clearly, the macro is not step-sequencable with respect to the new problem, since the only way of achieving $G_1 \wedge G_2 \wedge G_3$ will be by the plan $A_1 \rightarrow \underline{A_2} \rightarrow A_3$, which involves interleaving a new step into the retrieved macro. To summarize, in domains with serializable, but nonindependent subgoals, there are some instances where old plans are only step-interleavable with respect to a new goal, and others where they are also step-sequencable.

In the case of the ART-MD-NS domain, all the subgoals are non-serializable. To see this, consider that subplans for G_i and G_j have to be interleaved to give

ART-IND (D^0S^1)**:** $(A_i\ \underline{prec}:\ I_i\ \underline{add}:\ G_i\ )$
ART-MD (D^mS^1)**:** $(A_i\ \underline{prec}:\ I_i\ \underline{add}:\ G_i\ \underline{del}:\ \{I_j \mid j < i\})$
ART-MD-NS (D^mS^2)**:**
$(A_i^1\ \underline{prec}:\ I_i\ \underline{add}:\ P_i\ \underline{del}:\ \{I_j \mid j < i\})$
$(A_i^2\ \underline{prec}:\ P_i\ \underline{add}:\ G_i\ \underline{del}:\ \{I_j \mid \forall j\} \cup \{P_j \mid j < i\})$

Figure 7: The specification of Barrett and Weld's Synthetic Domains

the plan $A_i^1 \rightarrow A_j^1 \rightarrow A_i^2 \rightarrow A_j^2$. This means that a state-space planner cannot achieve a correct solution by attempting goals in succession, and concatenating subplans for individual goals. It may only solve all of its goals by interleaving attempts on subgoals related to individual goal conjuncts. Any planner which refines plans by adding steps only to the end of the existing step sequence will not be able to incorporate a macro unless that macro solves the full problem. To illustrate, consider the macro for solving a problem with conjunctive goal $G_1 \wedge G_2$ in ART-MD-NS, which will be: $A_1^1 \rightarrow A_2^1 \rightarrow A_1^2 \rightarrow A_2^2$. Now, if we add G_3 to the goal list, the plan for solving the new conjunctive goal $G_1 \wedge G_2 \wedge G_3$ will be $A_1^1 \rightarrow A_2^1 \rightarrow \underline{A_3^1} \rightarrow A_1^2 \rightarrow A_2^2 \rightarrow \underline{A_3^2}$ (where the underlined actions are the new actions added to the plan to achieve G_3). The only way a macro may be reused in this domain is by interleaving it with new steps, unless of course it is an exact match for the problem. Table 2 summarizes these correlations between subgoal types and the sequencability of a macro.

In some domains, for example the logistics transportation domain discussed earlier, problems may be solved by concatenating subplans for individual goals, but optimal plans may only be found by interleaving these subplans. As an example, consider the problem where two packages have to be transported between the same locations. We can solve each goal individ-

ually and concatenate subplans for transporting each package, but the result is clearly not optimal in terms of number of steps. A shorter solution would involve loading both packages in succession and then transporting them together. Clearly, in such reuse situations the plan-space planner will have an advantage from the point of view of plan quality[4].

The foregoing discussion may be summarized in terms of the following hypothesis:

Hypothesis 1 *In domains which contain problems that are conjunctions of goals and plan-step order is critical to the achievement of all the goals, reuse based on plan-space planning frameworks will be more effective than reuse based on state-space planning frameworks. This is because in such domains effective reuse may require refining old plans that are only step-interleavable with respect to the new goals (which is not supported by state-space planners).*

We will empirically evaluate this hypothesis in Section 4.

3.2 Case Extension in Eager Derivation Replay

In this section we will do a parallel analysis of the relative advantages of planning strategies when CBP employs replay instead of reuse to support adaptation. As we noted earlier, given a new problem, $\langle I', G', \mathcal{A} \rangle$, and a retrieved case which solves an old problem, $\langle I, G, \mathcal{A} \rangle$, eager derivation replay proceeds by replaying all of the applicable decisions of the case. We call the partial plan resulting from this process the skeletal plan, since, like the skeletal plan of plan reuse, it contains all of the constraints which have been

[4]If the planner is interested in guaranteeing the quality of the plans produced, and automatically prunes *bad* plans, we will indeed find that the retrieved plans will be non-sequencable with new goals.

judged to be applicable to the new problem. It is then the task of the underlying planner to achieve any extra goals which are left open in this plan. Here we consider replay to be *sequenced* if the skeletal plan that is produced through replay is refined into a full solution. In these instances, replay is sequenced with further refinement choices to arrive at a solution, since the series of decisions that were obtained through replay and which make up the replayed segment of the path are concatenated with further choices to reach the final solution.

A goal that is left open in the skeletal plan may be a top level goal in $(G' - G)$. It also may be a condition in $(I - I')$ which was relevant to the achievement of one of the goals in G but is not present in the new initial state, I'. We will distinguish three different ways a search path may be extended to solve an extra goal. First, a path which is a sequence of decisions may be concatenated with further choices.

Definition 6 (Decision-Sequencable Search Path) *A search path which contains a sequence of decisions D is decision-sequencable with respect to a new problem, $\langle I', G', \mathcal{A} \rangle$, if and only if there exists two decision sequences E and E' such that $E \bullet D \bullet E'$ (where "$\bullet$" is the decision sequencing operator) will produce a plan which is correct for $\langle I', G', \mathcal{A} \rangle$.*

Often a decision sequence may only be extended to achieve an extra goal if it is interleaved with additional choices. In this instance, we say that the decision sequence is interleavable with respect to the new problem.

Definition 7 (Decision-Interleavable Search Path) *A search path which contains a sequence of decisions D is decision-interleavable with respect to a new problem, $\langle I', G', \mathcal{A} \rangle$, if and only if there exists a sequence of decisions*

D' such that D may be merged with D' to produce a plan which achieves $\langle I', G', \mathcal{A} \rangle$ without retracting any decisions in D.

Finally, the path may have to be modified if it is to be extended to solve an extra goal.

Definition 8 (Decision-Modifiable Search Path) *A search path which contains a sequence of decisions D is decision-modifiable with respect to a new problem, $\langle I', G', \mathcal{A} \rangle$, if and only if there exists a sequence of decisions D' such that D may be merged with D' to produce a plan which achieves $\langle I', G', \mathcal{A} \rangle$ after retracting one or more decisions in D.*

For replay to be sequenced the replayed path must be decision-sequencable with respect to the new problem. It remains for us to gain an understanding of situations where decision sequencability is guaranteed.

3.2.1 When is a replayed path decision-sequencable?

Whether the replayed path is decision-sequencable will depend on the underlying planner. For a state-space planner *the decision sequence obtained through the guidance of replay is decision-sequencable with respect to a new problem only if the replayed path leads to a skeletal plan which is step-sequencable with respect to the new problem.* This is because a state-space strategy does not allow the interleaving of additional steps. Since the planner is continually progressing the current world state as steps are added, they may only be added to the end of the sequence. This means that any extra input goals have to be achieved by extensions to either the head or the tail of the plan. The relationship between step-sequencability of the skeletal plan and decision-sequencability is shown in Table 3.

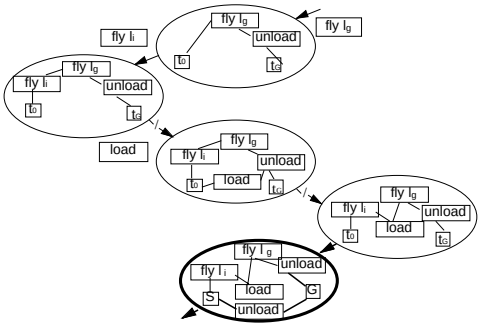
Skeletal Plan	**State-space**	**Plan-space**
step-sequencable	decision-sequencable	decision-sequencable
only step-interleavable	only decision-interleavable	decision-sequencable
only step-modifiable	only decision-modifiable	only decision-modifiable

Table 3: Relationship between the step-sequencability of the skeletal plan with the extra goals and the decision-sequencability of the replayed path that produced the skeletal plan

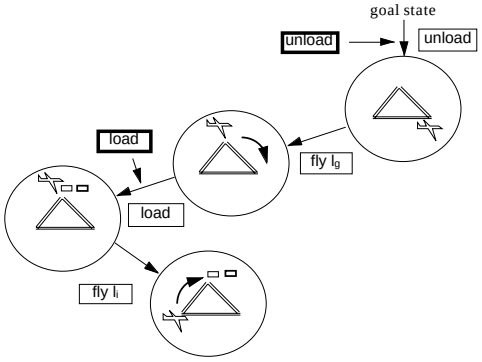
Consider the example from the logistics transportation domain contained in Figure 3.2.1. This figure shows a derivation that corresponds to transporting a single package to a designated airport, and illustrates the situation where this derivation is replayed for the problem that requires the additional goal of transporting a second package to the same airport. Additional planning effort is required to derive new steps to load and unload the extra package.

If the state-space planner, TOPI, attempts to refine the skeletal plan to add these new steps, the steps will be added to the beginning of the existing sequence, and the final plan so produced will have the plane retrace its route after depositing one package to retrieve the second. Replay based on a state-space planner is therefore more likely to produce a solution which contains more steps than the problem requires. As illustrated in Figure 3.2.1, if replay is based within the partial-order planner, SNLP, then the derivation which accomplished the same plan may be extended to achieve the extra goals. The added steps are first placed in parallel to the existing sequence, and the plan so produced is further refined by adding step orderings so that the new package is loaded and unloaded at the correct point in the sequence.

This shorter plan may be produced through replay by a state-space plan-



(a) Replay in Plan-space Planning (SNLP)



(b) Replay in State-space Planning (TOPI)

Figure 8: A transportation domain example showing the differences in support provided by state-space and plan-space planning to replay. In both cases, a logistics plan for transporting one package from a location l_i to a location l_g is being replayed to solve a new problem where two packages need to be transported from l_i to l_g. The replayed search path is extended (extension is shown in bold) to add extra steps to load and unload the additional package. A state-space planner must alternate replay with from-scratch planning in order to interleave these steps at the optimal point in the plan, whereas a plan-space planner may extend the replayed path to reach the same solution for the new problem.

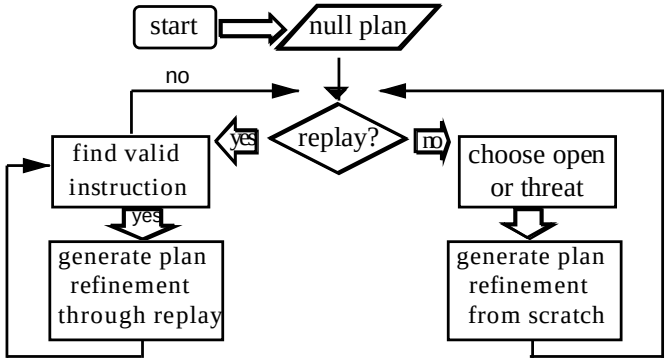
ner if replay is alternated with from-scratch planning for the extra goals (See Figure 3.2.1). Such an alternating strategy is shown in Figure 9. To distinguish our replay strategy from this alternating one, we refer to the former as *eager derivation replay*. The problem with such replay-alternating strategies is that there is no domain and problem-independent way of predicting when one should choose to replay vs. do from-scratch effort. If a wrong decision is made in this regard, the planner will be forced to backtrack over the plan resulting from replay. Such backtracking could be costly in terms of performance. Indeed, past work on replay that was based in state-space planning systems [3] has spent considerable effort towards finding best heuristic strategies to decide how to interleave replay with from-scratch planning to minimize such backtracking.

The plan-space planner does not have to alternate replay with further planning effort in order to interleave steps into a previous plan. In all instances when the new problem may be solved by interleaving new steps, the replayed path may be extended to reach a solution. It is only when the previous choices have to be modified, that the replayed path has to be backtracked over. This discussion suggests a testable hypothesis regarding the relative efficiency of replay in state-space vs. plan-space planning frameworks:

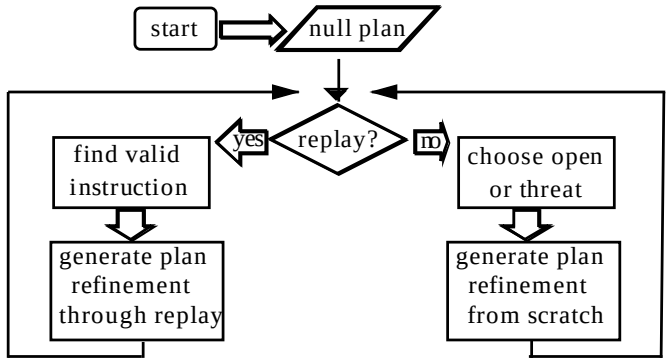
Hypothesis 2 *Plan-space planners will show greater improvement with eager derivation replay over state-space planners whenever step order is critical to a solution, that is, whenever new steps have to be interleaved into the previous plan in order to solve the new problem.*

We will empirically evaluate this hypothesis in the next section.

Figure 9: Schematic characterization of eager replay vs an alternating replay strategy. In eager replay, all the applicable instructions from the previous trace are replayed before control passes to from-scratch refinement. In an alternating replay strategy, the planner has a choice of replaying an instruction or doing a from-scratch refinement. Note that in either replay strategy, there is complete flexibility as to when, and how many times, the replay process should be initiated during planning.



(a) Eager Replay



(b) Alternating Replay

4 Empirical Evaluation of the Advantages of Plan-Space Planning

An empirical analysis was conducted to test our hypotheses regarding the relative effectiveness of reuse and replay for plan-space vs state-space planners. To do this we chose two state-space planners, TOPI [1] and NOLIMIT [33], in addition to the PO planner, SNLP. We implemented reuse and replay on these planners and compared their performance. The experimental strategy consisted of training all three planners with a set of randomly generated problems, and testing them on another set of randomly generated (but larger) problems. The planners solved problems and stored plans in the training phase. During the testing phase, the planners used these stored plans to solve new problems. Two types of CBP strategies were tested— one corresponding to the plan reuse strategy described in Section 2.2 and the other corresponding to the eager derivation replay strategy described in the same section. In the following, we briefly describe the planners, the reuse and replay strategies and the test domains.

4.1 Performance Systems

Our performance systems included two planners implemented by Barrett and Weld [1]. SNLP (POCL) is a causal-link partial-order planner, which may arbitrarily interleave subplans. The second planner, TOPI, carries out a backward-chaining world-state search. TOPI adds steps only to the beginning of the plan. Thus, unlike SNLP, but like planners doing search in the space of world states, such as STRIPS and PRODIGY, TOPI is unable to interleave new steps into the existing plan. SNLP and TOPI share many key routines (such as unification, operator selection, and search routines),

making it possible to do a fair empirical comparison between them.

NOLIMIT was chosen as the third planner to be tested. NOLIMIT is a version of PRODIGY which was the basis of the derivational replay system reported in [33]. Like PRODIGY and STRIPS, it uses means-ends analysis, first subgoaling to establish a set of potential operators, then forward chaining from the current world state, by applying potential operators to the plan. Applicable operators (operators whose preconditions are true in the current state) are added to the end of the plan and the current state is advanced appropriately. Unlike STRIPS, NOLIMIT can defer step addition in favor of further subgoaling. This means that in contrast to STRIPS, NOLIMIT can solve problems that are nonserializable, such as those in the ART-MD-NS domain, which cannot be solved by concatenating subplans for individual goal conjuncts. However, since it adds steps only to the end of the existing plan sequence, NOLIMIT is therefore also unable to interleave steps into the current plan. To facilitate fair comparisons, NOLIMIT was reimplemented on the same substrate as the other two planners. This was done following the algorithm contained in [33].

We have created a framework for accomplishing reuse and replay within each planning strategy. These are described below.

4.2 Implementation of Plan Reuse

Recall that in reuse, a previous case consists of a plan which is first fitted to adapt to the new problem-solving situation. As described in Section 2.2, fitting involves deleting irrelevant chains of steps leading to goals that are not present in the new problem. For our reuse experiments, this was not actually necessary, since cases were chosen which solved a subset of the new problem goals. Fitting in reuse involved the insertion of the old plan into

the current active node in the new search process. Care was taken to update the open conditions of the plan. For TOPI, the new goal state had to be regressed through the existing step sequence to create the current set of open conditions.

4.3 Implementation of Eager Derivation Replay

In replay, a previous case consists of a trace of the planning decisions which led to a successful solution in the earlier episode (see Section 2.2). Each decision corresponds to a choice among the alternatives at one of the backtracking points in the underlying planning algorithm. The content of each choice therefore changes with the planning methodology.

We have extended each planning strategy to include a replay facility. For example, DerSNLP (Derivational SNLP) extends SNLP by a capability for producing traces of successful plan derivations and replaying previous traces. A sample trace of SNLP's decision process is shown in Figure 10. The trace corresponds to a simple problem from the logistics transportation domain of [33] which contains the goal of getting a single package, OB1, to a designated airport, l_g. The derivation trace contains the choices that were made along the path from the root of the search tree to the final plan in the leaf node. Instructions contain a high level description of both the decision taken and its basis for justification in the future context. For example, a step addition is applicable in the context of the new search process if the condition that it previously achieved is also an open condition in the new active plan. A threat resolution choice is justified if steps added through replay result in a similar threat in the new active plan.

Consider as an example the trace contained in Figure 10 and suppose that it is being replayed for a second problem in which there is an additional

Goal : (AT-OB OB1 l_g)
Initial : ((IS-A AIRPORT l_g) (IS-A AIRPORT l_i))
(IS-A AIRPORT l_p) (AT-PL PL1 l_p)
(AT-OB OB1 l_i) ...

Name : G1
Type : START-NODE

Name : G2
Type : ESTABLISHMENT
Kind : NEW STEP
New Step: (UNLOAD-PL OB1 ?P1 l_g)
New Link: (1 (AT-OB OB1 l_g) GOAL)
Open Cond: ((AT-OB OB1 l_g) GOAL)

Name : G3
Type : ESTABLISHMENT
Kind : NEW STEP
New Step: (FLY-PL ?P1 ?A2 l_g)
New Link: (2 (AT-PL ?P1 l_g) 1)
Open Cond: ((AT-PL ?P1 l_g) 1)

Name : G4
Type : ESTABLISHMENT
Kind : NEW STEP
New Step: (FLY-PL ?P1 ?A3 ?A2)
New Link: (3 (AT-PL ?P1 ?A2) 2)
Open Cond: ((AT-PL ?P1 ?A2) 2)

Name : G5
Type : ESTABLISHMENT
Kind : NEW LINK
New Link: (0 (AT-PL PL1 l_p) 3)
Open Cond: ((AT-PL ?P1 ?A3) 3)

Name : G6
Type : ESTABLISHMENT
Kind : NEW LINK
New Link: (0 (IS-A AIRPORT l_i) 3)
Open Cond: ((IS-A AIRPORT ?A2) 3)

Name : G7
Type : ESTABLISHMENT
Kind : NEW LINK
New Link: (0 (IS-A AIRPORT l_g) 2)
Open Cond: ((IS-A AIRPORT l_g) 2)

Name : G8
Type : ESTABLISHMENT
Kind : NEW STEP
New Step: (LOAD-PL OB1 PL1 ?A4)
New Link: (4 (INSIDE-PL OB1 PL1) 1)
Open Cond: ((INSIDE-PL OB1 PL1) 1)

Name : G9
Type : ESTABLISHMENT
Kind : NEW LINK
New Link: (3 (AT-PL PL1 l_i) 4)
Open Cond: ((AT-PL PL1 ?A4) 4)

Name : G10
Type : RESOLUTION
Kind : PROMOTION
Unsafe-link : ((3 (AT-PL PL1 l_i) 4)
2 ¬(AT-PL PL1 l_i))

Name : G11
Type : ESTABLISHMENT
Kind : NEW LINK
New Link: (0 (AT-OB OB1 l_i) 4)
Open Cond: ((AT-OB OB1 l_i) 4)

Key to Abbreviations:
PL = PLANE
OB = OBJECT

Final Plan: (FLY-PL PL1 l_p l_i) Created 3
(LOAD-PL OB1 PL1 l_i) Created 4
(FLY-PL PL1 l_i l_g) Created 2
(UNLOAD-PL OB1 PL1 l_g) Created 1
Ordering of Steps: ((4 ¡ 2) (3 ¡ 4) (4 ¡ 1) (3 ¡ 2) (2 ¡ 1))

Figure 10: An example solution trace for DerSNLP

package, OB2, which is also to be transported to the same airport. At the beginning of the replay episode, the null plan is the current active node, **Act**. It contains two open conditions: (AT-OB OB1 l_g) and (AT-OB OB2 l_g). The first instruction to be replayed is of type establishment. It prescribes a new action, and it is annotated with the subgoal that step achieves. The validation process starts by matching the open condition of the instruction against the open conditions of **Act** [5]. Since (AT-OB OB1 l_g) is also open in the current active plan, the validation procedure goes on to generate the children of **Act** which also establish that condition by adding a new step. It then attempts to match the high level description of the step contained in the instruction, i.e., (UNLOAD-PL OB1 ?P1 l_g), to one of these plan refinements. The validation procedure returns the plan refinement that corresponds to the prescribed choice.

Replay was similarly implemented within the state-space planners, TOPI and NOLIMIT. Although the content of the stored traces are different, since the actual choices made differ with each planner, a common replay strategy was employed for each planner. Replay is *eager* in that control is shifted to the sequence of instructions in the derivation trace. It is this sequence that determines the order in which conditions are solved and threats are resolved. The output of the replay procedure is a skeletal plan which incorporates all of the prescribed refinements that are valid in the new context. The search process continues with this plan as the new active node.

[5] Conditions are matched based on an object mapping formed during retrieval of a previous case. Objects in the old goal condition are mapped into their corresponding objects in the new similar goal.

4.4 Experiments in Reuse and Replay for Mixed Artificial Domains

The three planning strategies were first tested on a series of problem sets each of which contained mixed independent and nonserializable goals. We constructed a domain which combined the actions of ART-IND (the domain with independent subgoals) and ART-MD-NS (the domain with nonserializable subgoals) as shown in Figure 7. Five different experiments were run on problem sets containing 0, 25, 50, 75 and 100% nonserializable goals. In each experiment, 30 four-goal problems were randomly generated according to the pre-specified mix, and the planning strategies were tested both in from-scratch mode, through reuse of an old plan which solved a subproblem containing three of the four goals, and through replay of a derivation trace corresponding to that subproblem.

The plots in Figures 11 and 12 summarize the performance of each planning strategy as a function of the percentage of the nonserializable goals. The plots on the left compares the number of plan refinements taken in solving all the 30 problems in each of the five problem sets. The plots on the right shows the total CPU time for each problem set.

DerSNLP in reuse mode exhibited the best performance both in terms of the cumulative time and the number of plans explored. Similar results were obtained for replay (see Figure 12). More importantly, notice also that for the state-space planners, DerTOPI and DerNOLIMIT, the improvements provided by reuse and replay decrease with increased percentage of nonserializable goals. The performance of the state-space planners was poorer with replay as opposed to generative planning when problems were 100% nonserializable (see Figure 12). These results are in agreement with our hypothesis

Figure 11: Reuse performance as a function of % of non-serializable sub-goals

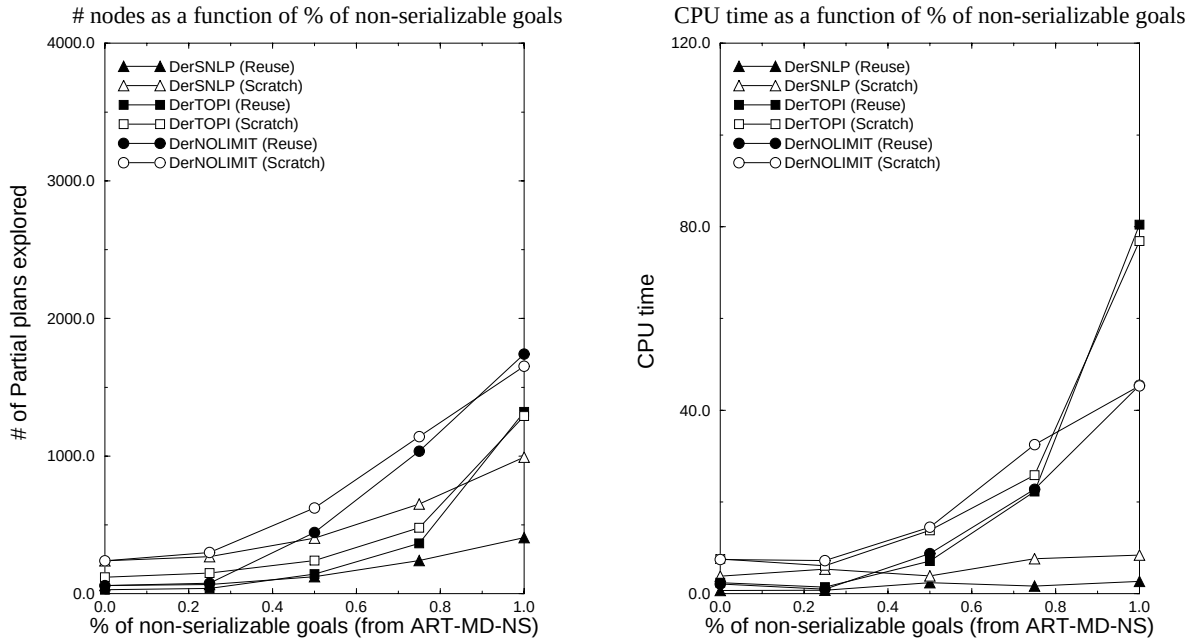
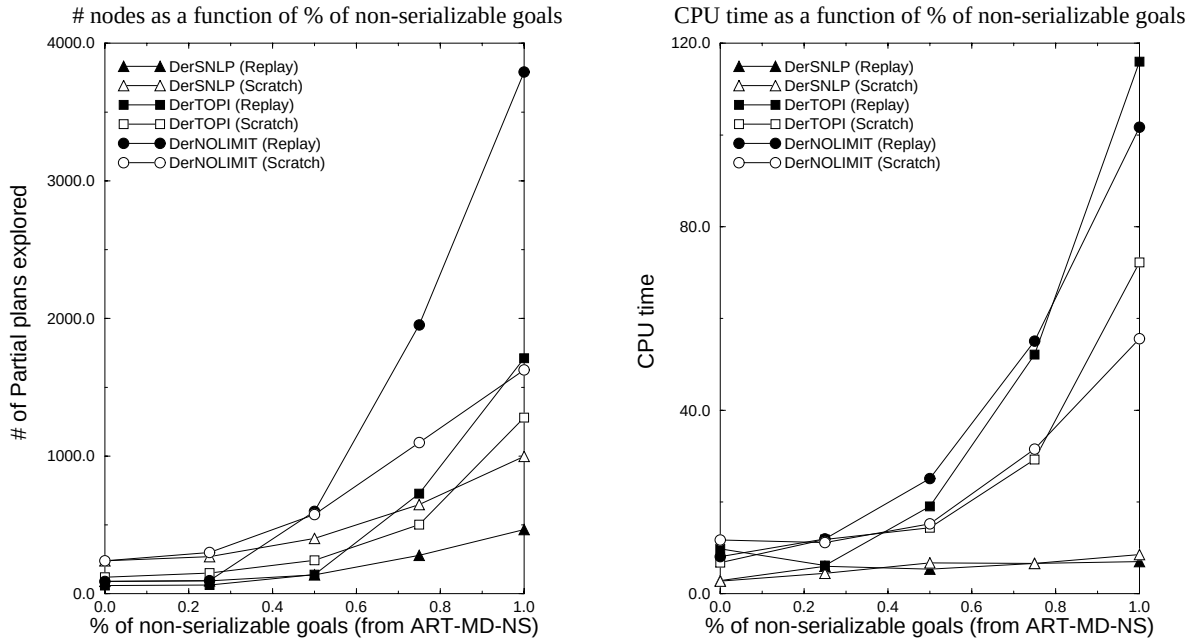


Figure 12: Replay performance as a function of % of non-serializable sub-goals



that when step order is critical, and new steps have to interleaved into the existing plan, that a plan-space planner will display greater performance improvements with reuse and replay.

4.5 Replay Experiments in the Logistics Transportation Domain

Our previous experiments demonstrate that plan-space planners are better at exploiting a previous case as the step order decisions become critical. To further support this conclusion, we experimented with the the more realistic logistics transportation domain of [33]. Problems with nonserializable goals were randomly generated, and the initial conditions of each problem were varied. These represented the location of various transport devices (one airplane and four trucks) over four cities, each city containing an airport and a post office. Four packages were randomly distributed over airports. So as to make step order critical, problems were chosen from this domain to contain subgoals that interact. Problems represent the task of getting one or more packages to a single designated airport. Each time that a problem was solved, a trace of the derivation of the solution was stored in the case library. Before each new problem was attempted, the library was searched for a previous case that solved a subset of the goals. No attempt was made to match the initial state conditions. Each planning strategy was tested both on solving the problem from scratch, and from replay of the previous case.

For the sake of comparison, we also ran the same experiments on problems randomly generated from the artificial domain, ART-MD-NS. Whereas each problem in ART-MD-NS has a unique solution, in the transportation domain there are many possible solutions varying in length. However, *optimal (shortest) solutions can only be found by interleaving plans for individual*

goals. This difference has an important ramification on the way eager replay misleads state-space planners in these domains. Specifically, in the ART-MD-NS domain, state-space planners will have to necessarily backtrack from the path prescribed by eager replay to find a solution. In the transportation domain, they can sometimes avoid backtracking by continuing in the replayed path, but will produce plans which are longer in terms of the number of steps.

4.5.1 Experimental Setup

Each experiment consisted of a single run in which problems were attempted in four phases. Goals were randomly selected for each problem, and, in the case of the transportation domain, the initial state was also randomly varied between problems. Each phase corresponded to a set of 30 problems. Problem size was increased by one goal for each phase. All the planners used a depth-first strategy in ART-MD-NS. To decrease overall running times in the transportation domain, the planners used a best-first search which was biased to extend the skeletal plan prior to the siblings of the replayed path. The replay strategy that we employed for the current investigation was *eager* in that the full trace was visited during a single call to the replay procedure.

4.5.2 Storage and Retrieval Strategies

A library of cases was formed over the entire run. Each time a problem was attempted, the library was searched for a previous case that was *similar*. If one was found, the new problem was run both in scratch and replay mode, and the problem became part of the 30 problem set for that phase. If there was no previous case that applied, the problem was merely added to the library.

Case retrieval was based on a primitive similarity metric. For one-goal

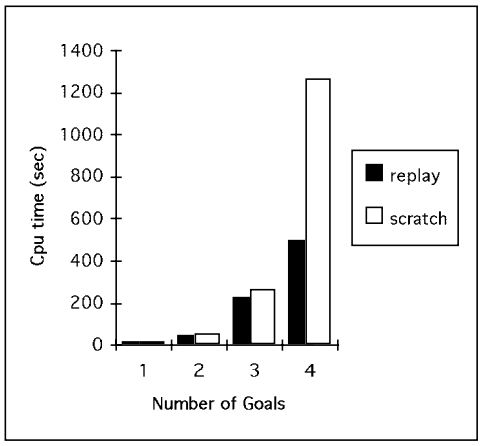
problems, a case was selected from the same set of one-goal problems that had accumulated during the first phase. A previous case was judged as sufficiently similar to the problem at hand to be considered for replay if the goals matched. For later phases, corresponding to multi-goal problems, cases were retrieved from the previous phase and were chosen so as to have all but one of the goals matching. The initial state conditions were randomly varied between problems, and retrieval was not based on similarity in these conditions. This meant that the plan that was returned by the replay procedure contained open conditions which include a goal corresponding to the extra top level goal, and also included initial state conditions that were changed in the new problem. Since the retrieved plans need to be extended to solve the new problems in all the multi-goal phases, we would expect the relative effects of replay on plan-space vs. state-space planning to be apparent in these phases.

4.5.3 Experimental Results

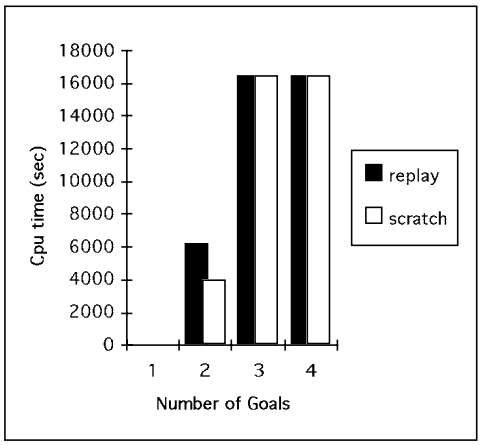
The results of the experiments are shown in Tables 4 and 5. Each table entry represents cumulative results obtained from the sequence of 30 problems corresponding to one phase of the run. The first row of Table 4 shows the percentage of problems correctly solved within the time limit (100 seconds for the ART-MD-NS domain, and 550 seconds for the logistics transportation domain). The average solution length is shown in parentheses for the transportation domain (solution length was omitted in ART-MD-NS since all the problems have unique solutions.) The subsequent rows of Table 4 contain the total number of search nodes visited for all of the 30 test problems, and the total CPU time. DerSNLP was able to solve as many or more of the multi-goal problems than the two state-space planners both in from-scratch and

Table 4: Performance statistics in ART-MD-NS and Logistics Transportation Domain. CPU time is in seconds on a Sun Sparc-II running CMU Commonlisp. Average solution length is shown in parentheses next to %Solved for the logistics domains only.

Phase	**ART-MD-NS** (*depth-first, CPU limit: 100sec*)						**Logistics** (*best-first, CPU limit: 550sec*)			
	DerSNLP		**DerTOPI**		**DerNOLIMIT**		**DerSNLP**		**DerTOPI**	
	replay	scratch	replay	scratch	replay	scratch	replay	scratch	replay	scratch
One Goal										
%Solved	100%	100%	100%	100%	100%	100%	100% (3.5)	100% (3.5)	100% (5.0)	100% (3.5)
nodes	30	90	30	60	30	120	617	946	46	507
time(sec)	.73	.68	.45	.47	.63	2.5	15	19	11	49
Two Goal										
% Solved	100%	100%	100%	100%	100%	100%	100% (5.8)	100% (5.8)	97% (6.2)	63% (5.6)
nodes	257	317	180	184	347	296	1571	2371	15824	8463
time(sec)	2	2	5	4	8	12	50	51	6216	3999
Three Goal										
% Solved	100%	100%	100%	100%	100%	100%	100% (7.9)	100% (7.9)	0%	0%
nodes	395	679	549	462	1132	662	6086	7400	-	-
time(sec)	7	4	16	11	34	34	230	262	-	-
Four Goal										
% Solved	100%	100%	100%	100%	100%	100%	100% (10.0)	100% (10.0)	0%	0%
nodes	577	1204	1715	1310	5324	1533	9864	24412	-	-
time(sec)	35	43	227	96	368	100	497	1264	-	-



(c) DerSNLP



(d) DerTOPI

Figure 13: Replay performance in the logistics transportation domain.

replay modes, and did so in less time. Our implementation of DerNOLIMIT was not able to solve any of the multi-goal problems in the transportation domain within the time limit, and this column is therefore omitted from the table.

In the ART-MD-NS domain, replay resulted in performance improvements for DerSNLP which increased with problem size (see Table 4). Comparative improvements with replay were not found for the two state-space planners in the multi-goal phases. In the logistics domain, not only did DerTOPI fail to improve performance through replay, it also experienced an increase in average solution length. In contrast, replay in DerSNLP led to performance improvements without increasing the solution length.

The graphs in Figure 13 show the performance of SNLP and TOPI in scratch and replay modes as a function of the problem size. The left axis

shows the total cpu time for the 30 problem set taken by each planner. We see that in scratch mode SNLP outperforms TOPI both in terms of cumulative time, and in terms of number of problems solved. Furthermore replay is beneficial to SNLP but not to TOPI. These results are consistent with our hypothesis that state-space planners will be misled by eager replay when step order is critical.

Table 5 reports three different measures that indicate the effectiveness of replay. The first is the percentage of *sequenced* replay. Recall that a sequence of decisions produced through replay is decision-sequencable with respect to the new problem if it may be extended by further refinement choices into a solution. Here we consider replay to be *sequenced* if the search path that is obtained through guidance from the previous trace is extended by further planning effort into a final plan. The percentage of sequenced replay therefore indicates the percentage of problems for which replay guides the new search directly down a path to a solution. Sequenced replay results in savings which are potentially exponential in the length of the portion of the solution path that is obtained through replay, as it avoids from-scratch search for that segment of the path (see Section 5). Nonsequenced replay is evident when some plan-refinements that were obtained through replay do not appear in the final solution path. When this happens, replay may actually exhibit a poorer performance in comparison to from-scratch search.

For the state-space planners, replay was entirely nonsequenced for multi-goal problems in either domain. Replay for the PO planner was entirely sequenced in the ART-MD-NS domain (see Table 5). In the logistics domain, the PO planner also experienced some nonsequenced replay, but less than the state-space planner in the multi-goal phase [6].

[6]There are two reasons for the non-sequenced replay shown by DerSNLP in the trans-

The efficiency of replay for the PO planner is also indicated by the two other measures contained in the final two rows of Table 5. These are the percentage of plan-refinements on the final solution path that were formed through guidance from replay (% Der), and the percentage of the total number of plans created through replay that remained in the final solution path (% Rep). The PO planner did better than the other two according to these measures in every phase in which multi-goal problems were solved, supporting our hypothesis regarding the relative effectiveness of eager replay in plan-space as opposed to state-space planning.

4.6 On the Effectiveness of Planning Strategies in Generative planning vs. Case-based planning

In this paper, we have argued that plan-space planners have some advantages over state-space planners in supporting case-based planning. Previous research that compared plan-space and state-space planners in generative planning have come to similar conclusions [28, 1]. Given this, it might seem that the effectiveness of a planning strategy in CBP is totally determined by its efficiency in generative planning. We will argue that this is not the case for the following reasons. Avoiding a strong commitment is most effective in generative planning when there are several alternative choices and many of these represent a wrong choice in that all attempts to extend the plan will fail. The advantage entailed by a weaker commitment is lessened by

portation domain. First, unlike ART-MD-NS, where all the problems had the same initial state, in the transportation domain, the initial conditions were randomly varied. Since our primitive similarity metric did not consider these initial state differences, this meant that replayed cases were less similar in this domain. Second, the best-first search strategy used in the transportation domain tends to compete against replay, directing the planner away from the replayed path when the rank of the replayed path is sufficiently high. This in turn reduces the percentage of sequenced replay.

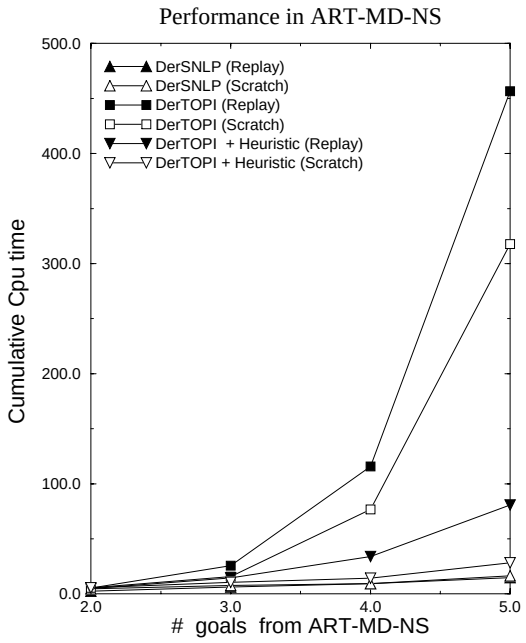


Figure 14: Comparison of from-scratch and replay performance in the ART-MD-NS domain, where TOPI was supplied with a domain-specific heuristic (cpu time in seconds on a Sparc-II running CMU Commonlisp)

	ART-MD-NS			Logistics	
Phase	**DerSNLP**	**DerTOPI**	**DerNOLIMIT**	**DerSNLP**	**DerTOPI**
One Goal					
% Seq	100%	100%	100%	93%	100%
% Der	100%	100%	100%	63%	99%
% Rep	100%	100%	100%	93%	100%
Two Goal					
% Seq	100%	0%	0%	83%	0%
% Der	32%	14%	13%	32%	24%
% Rep	100%	28%	25%	84%	37%
Three Goal					
% Seq	100%	0%	0%	47%	-
% Der	53%	14%	25%	34%	-
% Rep	100%	22%	38%	59%	-
Four Goal					
% Seq	100%	0%	0%	67%	-
% Der	65%	18%	31%	51%	-
% Rep	100%	24%	42%	78%	-

Table 5: Measures of effectiveness of replay

the adoption of a good heuristic for making the correct decision. However, often such a heuristic will give different advice over slight variations in the problem specification. For example, the presence of an extra goal in the new problem may mean that what was a correct choice in a previous case is a wrong choice in the new context. A planning strategy that makes strong commitments may therefore be a poor substrate for case-based planning, *even when it is relatively efficient in plan generation.* We therefore suggest that a weak commitment strategy may be advantageous in CBP over and above any benefits it provides for generative planning. Our empirical results corroborate this conclusion.

In particular, it is evident from the results in mixed domains that the advantage gained through replay is not a direct function of the efficiency of the

planning strategy employed (see Figure 12). This follows from the finding that as the percentage of nonserializable goals is increased, replay performance for the state-space planner degrades at a faster rate than performance in from-scratch mode. This result leads to the conclusion that when goals are nonserializable a plan-space strategy is even more important for a planner engaged in replay than it is for from-scratch planning.

In order to further demonstrate this finding, we re-ran DerTOPI in the ART-MD-NS domain but allowed it to employ a domain-specific heuristic which greatly improved its performance in from-scratch mode. When the heuristic was in effect, goals were selected so that steps were always added in the order found in a correct plan for the problem currently being attempted. As Figure 14 indicates, although DerTOPI+Heuristic showed much improved overall performance in from-scratch mode, in replay mode performance was worse than in from-scratch planning. The reason for this is evident when one considers that in replay, DerTOPI is directed to attempt its goals in the order prescribed by the previous case and therefore must abandon the heuristic choice when engaged in replay. Moreover, although the previous choice is a correct choice for that subproblem, it is not the correct choice for achieving the new problem. Replay therefore misleads the planner down the wrong path, a path which must be backtracked over to reach the new solution.

4.7 Summary and Conclusion

The experiments reported in this section concentrated on the relative effectiveness of state-space and plan-space planners in supporting plan reuse and derivation replay. Results indicate that when step order is critical to a solution, and new steps have to be interleaved into the old plan, the plan-space

planner has an advantage. The state-space planners tested failed to improve their performance through reuse or replay when step order was critical. SNLP, which avoids a strong step order commitment showed improvements with reuse and replay on the same problems. The results are thus consistent with our hypotheses regarding the advantage of plan-space planning in reuse and replay.

Moreover, we have also provided evidence that the efficiency of a planning strategy in reuse and replay is somewhat independent of its effectiveness in generative planning. When the state-space planner, TOPI, employed a heuristic for choosing a goal from the agenda, its generative planning performance in ART-MD-NS was comparable to SNLP's. This indicates that a strong step order commitment is not a disadvantage if one has a surefire method of determining a priori the correct step order. However, TOPI's replay performance continued to be poor in comparison to from-scratch planning.

It is always possible to adopt domain-dependent heuristics which have the potential, at least in simple domains like ART-MD-NS, of giving good advice as to when and in what order to apply steps to the plan. However, making the correct choice is dependent on knowing the identity of all of the input goals in the problem at hand. A previous case which solves only a subset of these goals will mislead the planner into taking a wrong decision. Although strong commitment is not necessarily a bad strategy for generative planning, particularly if one has available good heuristics, it may not be a good strategy for CBP. When different problems require different step orderings, a weak commitment strategy as to step ordering is preferable for CBP.

5 Analyzing the Expected Benefit of Eager Derivation Replay

In previous sections we proposed, and validated, some carefully circumscribed hypotheses regarding the effectiveness of plan-space vs. state-space planning in supporting the *extension phase* of case-based planning strategies. In this section, we place these results in the broader context of the CBP architecture which has been our focus. An analysis into the various factors that determine the cost of plan reuse can be found in the literature (see, for example, [30]). Here we will look at the cost of eager derivation replay, with the aim of investigating how the ability of the underlying planner to extend a plan will affect replay cost.

5.1 An Analysis of the Cost Benefit of Eager Derivation Replay

Suppose we are interested in solving a planning problem $\langle I', G'\mathcal{A}\rangle$. If the effective branching factor of the underlying planner for this problem is b, and we explore the search tree to a depth limit d then the cost of generating the solution from scratch is b^d.

Now suppose we are solving the same problem through eager derivation replay and that a case is retrieved with a retrieval cost ρ. Suppose further that after the fitting phase of cost η we produce a skeletal plan P_{sk}. Three stages of replay contribute to its overall cost: retrieval, fitting, and extension/recovery. The cost R of CBP is therefore

$$R = \rho + \eta + R_{e/r} \tag{1}$$

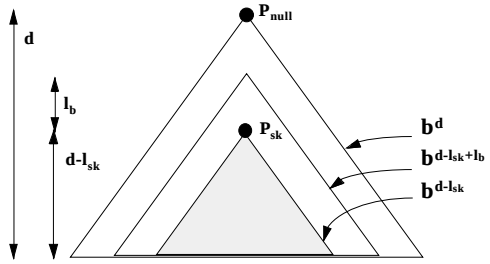


Figure 15: The cost of extension/recovery is $b^{d-l_{sk}+l_b}$ where l_b is the number of nodes on the replayed path that are backtracked over in order to reach a solution

where $R_{e/r}$ is the combined cost of extension and recovery. This last factor includes the cost of extending P_{sk} into a solution, and, if extension fails, recovering from the failure. In the next section we will look at the cost of extension/recovery.

5.1.1 The Cost of Extension/Recovery in the Replay Framework

With an *extension-first* replay strategy, the subtree underneath the skeletal plan is explored first. If the planner is successful in refining the skeletal plan into a solution within the depth limit d, then there is no recovery phase, and the cost of extension/recovery is $b^{d-l_{sk}}$ where l_{sk} is the length of P_{sk} in terms of the number of refinements needed to produce the plan (see Figure 15). If the skeletal plan cannot be refined into a solution within the depth limit, and the subtree underneath the skeletal plan has been explored to depth d, the planner then backtracks over the skeletal plan and continues solving the problem from scratch in the normal manner. Recovery involves expanding the siblings of the replayed path until a solution is reached. The total cost of extension/recovery is therefore b^d which is also the cost of solving the problem from scratch.

In order to analyze the advantage gained through replay, we must look at the *expected* cost given the probability of a successful extension of the skeletal plan. This cost is

$$E_{e/r} = pb^{d-l_{sk}} + (1-p)b^d \tag{2}$$

In general, the planner will not have to explore all of siblings of each node of the replayed path before reaching a solution. In particular, if a depth first search is employed, the planner recovers by exploring these siblings in order starting with the siblings of the skeletal plan. Suppose it is forced to backtrack over l_b decisions in the replayed path leading to P_{sk}. The value of l_b can vary between 0 and l_{sk} depending where the first wrong choice has been made. The cost of recovery is then $b^{d-l_{sk}+l_b}$ and the expected cost of extension/recovery is

$$E_{e/r} = pb^{d-l_{sk}} + (1-p)b^{d-l_{sk}+l_b} \tag{3}$$

It can be seen from 3 that the higher the probability of skeletal plan extension, the more effective replay will be. Moreover, when P_{sk} is not extendable, the cost of replay increased exponentially with l_b, the number of number of nodes in the replayed path that have to be backtracked over in order to reach a solution.

For replay to be effective over from-scratch planning the expression in 3 needs to be smaller than b^d. Replay will be effective when $p \approx 1$. This strong dependence of the efficiency of replay on a high probability of skeletal plan extension, together with the greater capability of plan-space planners to extend a skeletal plan, explains why plan-space planning has an advantage in replay.

5.2 Factors Influencing Overall Cost of Reuse/Replay

Let us now analyze the various factors that decide whether reuse/replay is effective:

- The cost of retrieval ρ. We obviously need sophisticated techniques that reduce the retrieval cost. Not surprisingly, a significant amount of work on plan reuse and case-based reasoning was devoted to this [13, 33, 7].

- The cost of *fitting* η. Since fitting takes polynomial time in the size of the retrieved case and the new problem this factor is dominated by the other factors.

- The length of the skeletal plan l_{sk}. The larger this value the greater the potential savings. The cost reduction increases exponentially with the size of the skeletal plan. This potential exponential reduction is what makes case-based planning so attractive. We hope that the retrieval and indexing are such that the retrieved plan will give rise to a larger skeletal plan (that covers more of the new problem).

- The probability p that the skeletal plan can be successfully extended. This probability determines whether a recovery phase will be needed to adapt the plan. Whenever a recovery phase is required, an exponential surcharge is added over and above the other costs. As noted earlier, if p is sufficiently low, case-based planning can in fact be costlier than planning from scratch.

- In replay, the length l_b of the number of refinements leading to P_{sk} that have to be backtracked over in order to reach a solution. If extension

fails this factor will determine the amount of effort that has to be put into recovery.

Of these factors, the retrieval cost ρ, the fitting cost η, as well as the length of the skeletal plan l_{sk} can be made reasonably independent of the underlying planner, assuming that the same retrieval and indexing strategies are used for all planners. In Section 4 we have demonstrated that the planner has an effect on the probability of successful extension, p. Given the same retrieval and fitting methods, the probability p of extending the skeletal plan into a solution is higher for reuse and replay based within plan-space planning. This explains the observed performance differentials.

6 Ramifications in More Sophisticated CBP Frameworks

In the previous sections, we have considered the potential benefits of plan-space planning in CBP. However, we have looked at only two methods of case adaptation. Both involve a refinement strategy which first fits the case to the new problem situation, then extends the case to achieve any extra goals before recovering by backtracking or returning to from-scratch planning. Other techniques which can be found in the literature employ alternative reuse and replay strategies. There is therefore a question as to whether our results apply to these methodologies as well. First we will consider some more sophisticated CBP techniques.

6.1 Effect of Sophisticated Backtracking and Adaptation Strategies

6.1.1 Backtracking Strategies

We noted that much of the inefficiency of state-space planners in supporting adaptation stems from their need to backtrack whenever new steps have to be interleaved into the current skeletal plan. It is possible to alleviate the effect of this inefficiency to some extent by resorting to dependency-directed backtracking algorithms. Even if we ignore the overhead associated with such strategies, it is still instructive to note that they only try to cure the symptom of the malady, rather than the malady itself. In similar situations, plan-space planners will not need to backtrack in the first place.

6.1.2 Multi-Case Adaptation

Some case-based planning systems, such as PRODIGY/ANALOGY [33] use multiple cases in guiding adaptation. Although our empirical study into replay involved only replay of a single case for each problem, we believe that the results can also be extended to multi-case replay. When we have multiple cases, in addition to the decision as to how to alternate replay with from-scratch planning, we also have the decision as to *how to alternate the replay of individual cases*. When step order is critical, the state-space planner will have to attempt both types to successfully exploit the guidance of previous cases. Once again, the plan-space planner will have less need to do either of these, and can thus not only replay the cases in any order, but also employ eager replay on individual cases. Similarly, the plan reuse strategy employed here can be very easily extended to start with multiple macros, which between them cover complementary subsets of the new problem goals. The ability to

interleave plans will again be advantageous in multi-plan reuse.

6.1.3 Multi-Pass Adaptation

Some implemented case-based planning systems, such as REMAID [3] and PRODIGY/ANALOGY [33] use multi-pass strategies in adaptation. For example, rather than abandon the case after it has been replayed once, such systems keep the case and replay it again if parts of the case that are not previously applicable, become applicable as the skeletal plan is being extended. Multi-pass replay can be thought of as the process of replaying multiple copies of the same case. Each time the case is visited, we replay all the applicable portions of the case.[7] Our eager replay strategy may be easily extended to include multiple passes on the same case. It is however not clear whether such multi-pass adaptation algorithms will have an overall positive impact on the effectiveness of case-based planning. Multi-pass algorithms are akin to using "macro operators" to satisfy the preconditions of other macro operators. Prior work in explanation based learning [29] shows that unconstrained macro-chaining could have an adverse effect on performance.

6.2 The Cost of Extension/Recovery in a Transformational Framework

Some CBP systems employ transformational as opposed to refinement methods [8, 9]. Their approach to the reuse of a plan may therefore involve simultaneous addition and retraction of plan constraints. In this section we will

[7] PRODIGY/Analogy [33] uses a slightly different strategy in this regard. Specifically, it uses a "pointer" to keep track of the part of the case that has already been replayed. Every time a case is visited, the replay is restarted at the position indicated by the pointer. PRODIGY/Analogy also uses this strategy to facilitate alternating replay (see Figure 9). Since PRODIGY/Analogy uses a state-space planner, such alternating replay is required to support interleaving of steps into skeletal plan.

analyze whether our conclusions as to the benefits of plan-space CBP apply to these frameworks as well. We will look at the cost benefits of plan-space planning in an alternative transformational planning framework.

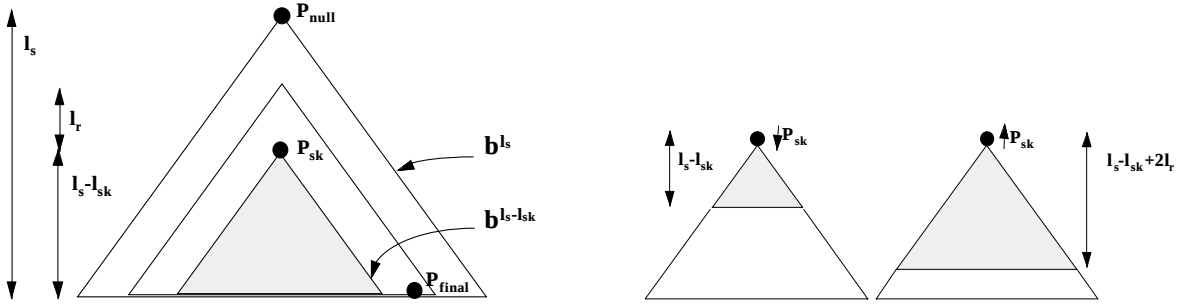
The SPA system of Hanks and Weld [9] attempts to directly adapt the skeletal plan resulting from the fitting stage by allowing retraction of plan constraints to be tried in parallel to skeletal plan refinement. At the start of planning, the search queue is initiated with two copies of the fitted library plan, one tagged DOWN for extension and the other tagged UP for retraction. In the downward direction planning proceeds in the same manner as generative planning. In the upward direction a plan is replaced on the queue by its parent (tagged UP) and siblings (tagged DOWN). The UP branch of the search may arrive at a solution by first retracting one or more of the skeletal plan constraints, and then adding new constraints to this reduced plan until a solution is reached. Both directions are pursued *simultaneously*. The derivation path leading to the new solution is thus typically made up of l_r constraint retractions, followed by $l_r + l_s - l_{sk}$ constraint additions. Since both UP and DOWN directions are pursued simultaneously, the search done by SPA is:

$$R = 2 \times b^{l_s - l_{sk} + 2l_r} \approx b^{l_s - l_{sk} + 2l_r} \tag{4}$$

Notice that for the special case of $l_r = 0$, this becomes the search done when the solution is found under the DOWN branch (i.e., through skeletal plan extension).

With a reasonable retrieval strategy, we can expect that the average solution length, l_s, is no larger for solutions under the skeletal plan as opposed to solutions in other regions of the search tree. Under these conditions, reuse

Figure 16: Simultaneous extension/retraction involves a cost of $b^{l_s - l_{sk} + 2l_r}$ where l_r is the number of retractions that have to be performed on the skeletal plan in order to obtain a solution.



(a) Search Tree

(b) Parallel Extension/Retraction

is most effective when no retraction is required. Specifically, it can be seen from 4 that the cost of reuse increases exponentially with l_r, the number of retractions that have to be performed. It is therefore likely that plan-space planning will be more effective in this CBP framework as well.

7 Related Work and Discussion

7.1 Effect of Other Types of Commitment on the Effectiveness of Reuse

In this paper, we have provided evidence that the premature step-order commitment made by state-space planners makes them inferior to plan-space planners as substrates to case-based planning. Step ordering is not the only type of commitment that distinguishes planning systems. This brings up the question as to whether the hypotheses of this paper may also be applicable to other types of commitments. To address this, we start by looking at other types of strong commitments made by planners. These include:

Commitments to Causal Structures: To reduce redundancy in the search space, some plan-space planners such as SNLP commit to causal structures. In practice, what this means is that once a precondition c of a step s is established with the help of the effect of some step s', they protect this establishment by ensuring that no step s'' can come in between s' and s and violate c. Blythe and Veloso showed that this gives SNLP a disadvantage in certain domains [34]. However, there are other plan-space planners such as UA [28] and TWEAK [5] which do not commit to specific causal links, while MP and MP-I [14] allow for causal structures that strike a better balance between redundancy and commitment. Disjunctive planners such as GRAPHPLAN [2], SATPLAN

[23, 22], DESCARTES [12, 11] and UCPOP-D [21] avoid commitment by forming disjunctions over plan constraints. Although disjunctive planners at first glance seem very different from the traditional refinement planners, recent work [21] has shown that these planners can be understood in terms of how search is introduced into refinement planning. In particular, when a partial plan P is refined using some refinement strategy into a set of new plans $\{P_1, \cdots P_n\}$, traditional planners split the resultant plans into different search branches and handle them separately. An alternative is to keep plans together as a single entity which contains disjunctions over plan constraints. Disjunctive planners use this technique to eliminate search from plan refinement, but introduce it into a *solution extraction* stage which is used to find a solution within a disjunctive plan.

Commitments to Specific Ways of Achieving a Goal: We have discussed how planners such as SNLP, TOPI, NOLIMIT commit to a specific way of achieving the goal, whereas disjunctive planners make weaker commitments by forming disjunctions over plan constraints. Another way of introducing a weak commitment is through hierarchical task reduction. Planners such as NONLIN [32] and SIPE [36] first work out the details of a plan at abstract levels, and then refine it at more concrete levels. Systems such as PRIAR [15] use these hierarchical planners as the substrate for plan reuse.

We believe that any type of strong commitment has a deleterious effect in plan reuse and replay. We can rely on the semantics of refinement planning [19, 20] to understand this. From the perspective of refinement planning, the skeletal plan is a partial plan that can be seen as a shorthand notation

for the set of complete plans (operator sequences) that are consistent with the current constraints comprising the skeletal plan. This set is called the candidate set of the (skeletal) plan. When a refinement planner extends a skeletal plan by adding constraints to it, it is essentially narrowing down the candidate set. From this point of view, the skeletal plan can be extended into a complete solution to the new problem only when this solution belongs to the candidate set of the skeletal plan to begin with. The larger the candidate set of the original skeletal plan, the higher this probability.

It is easy to show that planners that make weak commitments to plan constraints have partial plans that have larger candidate sets. For example, consider the general scenario where we want to satisfy some precondition c of step s_2 by using the effects of a step s_1. We clearly need to order the steps such that s_1 comes before s_2. The state-space planners do this by enforcing a contiguity constraint (denoted by "$*$") between s_1 and s_2, thereby guaranteeing not only that s_1 comes before s_2, but that no other steps come in between. Plan space planners on the other hand add a partial ordering constraint (denoted by "$\prec$") between s_1 and s_2, which allows other steps to come between them. Given two partial plans "$s_1 * s_2$" and "$s_1 \prec s_2$," it is easy to see that more operator sequences will be consistent with the latter than the former. Thus the latter has a larger candidate set.

Similarly, a plan space planner such as SNLP also adds a causal link constraint to preserve c between s_1 and s_2, while planners such as UCPOP-D [21] form a disjunction over causal link constraints. Once again, the candidate set of the plan produced by SNLP will be smaller than that produced by UCPOP-D. Finally, in the case of task reduction planning, the step s_2 that is picked for making s_1 true is likely to be an *abstract* action which can be potentially realized in terms of many possible plan templates. Since at this

point we have not made any commitment to any one of those templates, the candidate set of the partial plan would be even larger.

In summary, planners that make weaker commitments tend to produce partial plans (and eventually skeletal plans) with larger candidate sets. Skeletal plans with larger candidate sets offer higher probability that the candidate set contains a solution for the new problem (which can then be found by refining the skeletal plan), thus reducing the need for a recovery phase, and increasing the effectiveness of CBP. This puts in a larger perspective the effect of step-order commitment which is the focus of this paper.

7.2 Future Work

7.2.1 Indexing

In Section 5, we argued that the effectiveness of case-based planning is undermined whenever the skeletal plan cannot be extended, and a recovery phase is necessitated. In this paper we have provided empirical evidence that has demonstrated that a plan-space planner has a better chance of extending a skeletal plan. However, plan-space planners don't *guarantee* that a case will be extendable all the time. In particular, if the skeletal plan is only step-modifiable with respect to the new goals, these planners will also be misled by the retrieved case, and need to recover. For reuse and replay to improve performance, it is necessary to select an appropriate case. If the wrong case is chosen, the cost of retrieving, fitting and modifying the case may exceed that of planning from scratch [30].

This suggests an approach for dynamically improving retrieval – specifically, we could consider a retrieved case to be providing wrong guidance whenever the skeletal plan derived from it could not be extended and a re-

covery phase was necessitated. Whenever this happens, we could analyze the search tree to understand the reasons as to why the skeletal plan could not be extended, and annotate the stored case with these reasons, so that such mis-retrieval will not occur in the future. Using explanation-based learning techniques [17] we have implemented this approach for improving retrieval in the presence of derivational replay [10], and have shown that this approach leads to substantial improvements in replay performance.

7.2.2 Plan Quality

Until now, we concentrated on the issue of extending the skeletal plan into *a* solution for the planning problem, and have largely ignored the issue of the quality of the plans. The results of this paper however do have some relevance to the issue of plan quality in CBP. In many domains, the problems may be such that step-sequencable reuse will produce a solution, but step-interleaving is needed if one wants an optimal solution. In other words, even in cases where decision-sequencable replay is feasible, the ability to support step-interleaving could improve the quality of the plan. The results of our experiments shown in Table 4, lend support to this hypothesis. In particular, the numbers in parentheses show the average lengths of plans produced by plan-space and state-space planners during replay. We note that plan-space planners do better than state-space planners in this regard. Our future work is aimed at further improving plan quality by identifying *bad plans* through an independent plan evaluation, and implementing a retrieval strategy which learns to avoid bad plans as well as plan failures.

7.2.3 CBP within Disjunctive Planning

Future work will investigate reuse and replay within disjunctive planners. These can be seen to fall into a generalized refinement planning template [21] that subsumes both the traditional, as well as newer planners that introduce full disjunction such as GRAPHPLAN [2] and SATPLAN [23, 22]. So far we have implemented replay only on planners which commit to particular plan constraints and split the search space. It would seem that planners which introduce full disjunction and thus form no commitments cannot benefit from reuse and replay. However, there exist planners such as UCPOP-D [21] and DESCARTES [12, 11] which fall within the extremes of full splitting and full disjunction. For example, UCPOP-D introduces disjunction only in simple establishments. Our future work will focus on the question as to whether the performance improvements gained through these newer techniques can be further improved through CBP.

8 Summary and Conclusion

In this paper we have presented two case-based planning frameworks which we call plan reuse and eager derivation replay. Both frameworks are built on top of an underlying generative planner which is used in extending a skeletal plan derived from a previous case into a solution for a new problem, and, when the skeletal plan cannot be refined into a solution, in continuing the search process by backtracking over this plan. As such, both frameworks preserve the soundness and completeness of the underlying planning strategy. In particular, eager derivation replay merely provides search control for the underlying planner, and therefore inherits all of its properties. For example, when based within SNLP, it is complete, sound and systematic.

Using these two CBP frameworks, we have addressed the issue of the relative utility of grounding case-based planning in plan-space vs. state-space planning strategies. We have argued that given the same retrieval and fitting strategies, a plan-space planner has a greater capability to extend a previous case into a full solution to a new problem. Our empirical studies have demonstrated that this quality of plan-space planners makes them substantially more effective in supporting reuse and replay over state-space planners in domains where step order is critical. We have provided empirical support for the hypothesis that the effectiveness of CBP depends on the underlying planning strategy. The results of these experiments, and their ramifications for more sophisticated CBP frameworks were discussed.

References

[1] A. Barrett and D. Weld. Partial order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67:71–112, 1994.

[2] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings IJCAI-95*, 1995.

[3] B. Blumenthal and B. Porter. Analysis and empirical studies of derivational analogy. *Artificial Intelligence*, 67:287–327, 1994.

[4] J. Carbonell, C. Knoblock, and S. Minton. Prodigy: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990.

[5] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–337, 1987.

[6] R. Fikes and N. Nilsson. A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[7] K. Hammond. Chef: A model of case-based planning. In *Proceedings AAAI-86*, pages 261–271. AAAI, 1986. Philadelphia, Pennsylvania.

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

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

[10] L. Ihrig and S. Kambhampati. Design and implementation of a replay framework based on a partial order planner. In *Proceedings AAAI-96*, 1996.

[11] D. Joslin. *Passive and active decision postponement in plan generation.* PhD thesis, Intelligent Systems Program, University of Pittsburgh, 1996.

[12] D. Joslin and M. Pollack. Is early commitment in plan generation ever a good idea. In *Proceedings-AAAI96*, pages 1188–1193, 1996.

[13] S. Kambhampati. Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, 10, 1994.

[14] S. Kambhampati. Multi-contributor causal structures for planning: A formalization and evaluation. *Artificial Intelligence*, 69, 1994.

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

[16] S. Kambhampati, L. Ihrig, and B. Srivastava. A candidate set based analysis of subgoal interactions in conjunctive goal planning. In *Proceedings of the 3rd Intl. Conf. on AI Planning Systems*, 1996.

[17] S. Kambhampati, S. Katukam, and Y. Qu. Failure driven dynamic search control for partial order planners: An explanation-based approach. *Artificial Intelligence*, 1996. To Appear.

[18] S. Kambhampati and S. Kedar. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence*, 67:29–70, 1994.

[19] S. Kambhampati, C. Knoblock, and Q. Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, 76:167–238, 1995. Special Issue on Planning and Scheduling.

[20] S. Kambhampati and B. Srivastava. Universal classical planner: An algorithm for unifying state-space and plan-space approaches. Technical Report ASU CSE TR 94-002, Arizona State University, January 1995 1995.

[21] S. Kambhampati and X. Yang. On the role of disjunctive representations and constraint propagation in refinement planning. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, 1996.

[22] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, 1996.

[23] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings AAAI-96*. AAAI, 1996. Portland, Oregon.

[24] J. Koehler. Avoiding pitfalls in case-based planning. In *Proceedings of the 2nd Intl. Conf. on AI Planning Systems*, pages 104–109, 1994.

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

[26] R. Korf. Planning as search: a qualitative approach. *Artificial Intelligence*, 33:65–68, 1987.

[27] D. McAllester and D Rosenblitt. Systematic nonlinear planning. In *Proceedings AAAI-91*, pages 634–639, 1991.

[28] S. Minton, J. Bresina, and M. Drummond. Total order and partial order planning: a comparative analysis. *Journal of Artificial Intelligence Research*, pages 227–262, 1994.

[29] R. Mooney. The effect of rule use on the utility of explanation-based learning. In *Proceedings IJCAI-89*, pages 725–730, 1989.

[30] B. Nebel and J. Koehler. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, pages 427–454, 1995.

[31] E. Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, New York, 1977.

[32] A. Tate. Generating project networks. In *Proceedings IJCAI-77*, pages 888–889, 1977. Cambridge, Massachusetts.

[33] M. Veloso. *Learning by analogical reasoning in general problem solving.* PhD thesis, Carnegie-Mellon University, 1992.

[34] M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the 2nd Intl. Conf. on AI Planning Systems*, pages 170–175, 1994.

[35] D. Weld. An introduction to partial-order planning. *AI Magazine*, 1994.

[36] D. E. Wilkins. Recovering from execution errors in sipe. *Computational Intelligence*, 1:33–45, 1985.