From: AAAI-93 Proceedings. Copyright © 1993, AAAI (www.aaai.org). All rights reserved.

Relative Utility of EBG based Plan Reuse in Partial Ordering vs. Total Ordering Planning

Subbarao Kambhampati* **and** **Jengchin Chen**
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406
Email: rao@asuvax.asu.edu

Abstract

This paper provides a systematic analysis of the relative utility of basing EBG based plan reuse techniques in partial ordering vs. total ordering planning frameworks. We separate the potential advantages into those related to storage compaction, and those related to the ability to exploit stored plans. We observe that the storage compactions provided by partially ordered partially instantiated plans can, to a large extent, be exploited regardless of the underlying planner. We argue that it is in the ability to exploit stored plans during planning that partial ordering planners have some distinct advantages. In particular, to be able to flexibly reuse and extend the retrieved plans, a planner needs the ability to arbitrarily and efficiently "splice in" new steps and sub-plans into the retrieved plan. This is where partial ordering planners, with their least-commitment strategy, and flexible plan representations, score significantly over state-based planners as well as planners that search in the space of totally ordered plans. We will clarify and support this hypothesis through an empirical study of three planners and two reuse strategies.

1. Introduction

Most work in learning to improve planning performance through EBG (explanation based generalization) based plan reuse has concentrated almost exclusively on state-based planners (i.e., planners which search in the space of world states, and produce totally ordered plans;[3, 14, 11, 20].) In contrast, the common wisdom in the planning community (vindicated to a large extent by the recent formal and empirical evaluations [1, 12, 15]), has held that search in the space of plans, especially in the space of partially ordered plans provides a more flexible and efficient means of plan generation. It is natural to enquire, therefore, whether partial order (PO) planning retains its advantages in the context of EBG based plan reuse.

In our previous work [8], we have shown that the explanation-based generalization techniques can indeed be extended in a systematic fashion to partially ordered partially instantiated plans, to give rise to a spectrum of generalization strategies. In this paper, we will address the issue of comparative advantages of doing EBG based plan reuse in a partial order planning framework. We will do this by separating two related but distinct considerations: the advantages of *storing* plans as partially ordered and partially instantiated generalizations and the advantages of *using* the stored generalizations in the context of a PO planning framework.

Storing plans in a partially ordered and partially instantiated form allows for compactness of storage, as well as more flexible editing operations at retrieval time. We will point out, however, that these advantages can be exploited whether the underlying planner is a PO planner or a total ordering planner. We will argue that it is in the ability to *use* the generalized plans during planning, that partial ordering planners have some distinct advantages over total ordering planners. In particular, to be able to flexibly reuse and extend the retrieved plans (when they are only partially relevant in the new problem situation), the planner needs to be able to arbitrarily and efficiently "splice in" new steps and sub-plans into the retrieved macro (and *vice versa*). Partial ordering planners, with their least-commitment strategy, and flexible plan representations, are more efficient than state-based planners as well as planners that search in the space of totally ordered plans, in doing this splicing. We argue that in many plan reuse situations, this capability significantly enhances their ability exploit stored plans to improve planning performance. We will support our arguments through focused experimentation with three different planners and two different reuse strategies.

The rest of the paper is organized as follows: the next section provides a brief characterization of different planners in terms of how they refine plans during search. Section 3. uses this background to characterize the advantages of partial order planners in exploiting stored plan generalizations to improve performance. Section 4.1. describes an empirical study to evaluate the hypotheses regarding the comparative advantages of PO planning. Section 4.2. presents and analyzes the results of the study. Section 5. argues in favor of basing plan reuse and other speedup learning research in partial order planning, and clears some misconceptions regarding PO planning which seem to have inhibited this in the past. Section 6. concludes with a summary of contributions. All through this paper, we shall refer to stored plan generalizations as *macros*, regardless of whether they get reused as macro-operators, or serve as a basis for library-based (case-based) plan reuse. We also use the terms "efficiency" and "performance" interchangeably to refer to the speed with which a planner solves a problem.

2. A Characterization of Planners in terms of allowable plan refinements

Whatever the exact nature of the planner, the ultimate aim of planning is to find a *ground operator sequence*, which is a *solution* to the given problem (i.e., when executed in a given initial state will take the agent to a desired goal state). From a first principles perspective, the objective of planning is to navigate this space, armed with the problem specification, and find the operator sequences that are solutions for the given problem. Suppose the domain contains three ground actions $a1$, $a2$ and $a3$. The regular expression $\{a1|a2|a3\}^*$ describes the potential solution space for this domain. If we are interested in *refinement planners* (i.e., planners which add but do not retract operators and constraints from a partial plan during planning) which most planners are, then the planner's navigation through the space of potential solutions can be enumerated as a directed acyclic graph (DAG), as illustrated in Figure 1.

When a refinement planner reaches an operator sequence that is not a solution, it will try to refine the sequence further (by adding more operators) in the hopes of making it a solution. Different types of planners allow different types of transitions in the DAG. For example, planners such as STRIPS and PRODIGY that do *forward search in the space of world states*, will only add operators to the end of the partial solution during refinement, and thus only transition via the solid lines in Figure 1.[1] Most planners used in learning research to-date fall in this category. On the other hand, planners

*This research is supported by National Science Foundation under grant IRI-9210997.

[1]Notice that the so called *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 linearity assumption, and its extension NOLIMIT[19] doesn't (and thus allows interleaving of subgoals), are both capable only of refining a partial plan by adding operators to the end of the current plan.

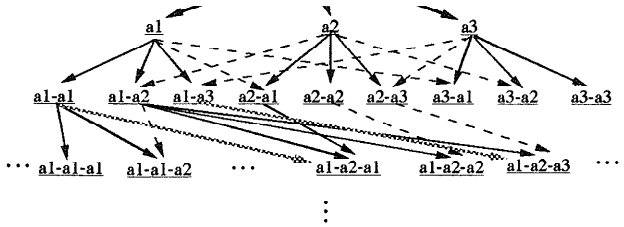

Figure 1: Characterization of refinements allowed by various planning strategies (see text)

which do backward search in the space of world states, will only add new operators to the beginning of the partial solution during refinement, and thus allow the transitions shown in dashed lines. Finally, planners which *search in the space of plans*, allow new operators to be added *anywhere* in the partial solution, *including in the middle of the existing plan*, and thus allow all of the refinements shown in the figure.

All the planners we discussed above can be called total ordering planners in that the partial plans they maintain during their search is always a totally ordered sequence of operators. However, planners searching in the space of plans have the option to either search in the space of totally ordered plans, or in the space of partially ordered (PO) plans. Many current-day planners such as NOAH, NONLIN, SIPE belong to the latter class, called partial order or PO planners[2] are generally more efficient as they avoid premature commitment to inter-operator orderings, there by improving efficiency over corresponding planners that search in the space of totally ordered plans [1, 15].

3. Advantages of Partial Order planning in plan reuse

3.1. Storage Compaction

A PO plan provides a compact representation for the possibly exponential number of its underlying linearizations by specifying just the steps, the partial ordering between steps and the codesignation and non-codesignation constrains between the variables. This flexible plan representation allows for a spectrum of order, precondition and structure generalizations. Our previous work [8] provides a systematic basis for generating these generalizations. Storing plans in PO form also allows for more sophisticated *editing* operations at retrieval time, when the macro is only partly applicable. Specifically, any irrelevant steps and constraints of the plan can be edited out by retracting the corresponding planning decisions. The retraction itself can be facilitated by justifying individual planning decisions in terms of the plan causal structures. Once such a justification framework is in place, the retraction of irrelevant constraints can be accomplished with the help of a polynomial time greedy algorithm (c.f. [5, 8]).

However, all the advantages of storage compaction and plan editing will hold whether the underlying planner generates a totally ordered or partially ordered (PO) plans. For example, generalization techniques described in our previous work on EBG for PO plans [8] can be used whether the plan was initially produced by a partial ordering or a total ordering planner. Similarly, even in reuse frameworks based on total ordering planners (e.g. [20, 18]), order generalization has been used as a way to separate independent parts of the plan and store them separately, thereby containing the proliferation of macros by reducing the redundancy among them. In other words, although storage considerations motivate the use of PO plan representation during plan reuse, they do not necessarily argue for the use of PO planning.

[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 linearity assumption. In fact, as we mentioned earlier linearity assumption is concerned with order in which different goals are attacked, and can be used in *any planner*. Linearity assumption causes incompleteness in planners that search in the space of world states (such as STRIPS and PRODIGY), but does not affect completeness in any way in planners that search in the space of plans.

ART-MD ($D^m S^1$): (A_i $\underline{prec}$: I_i $\underline{add}$: G_i $\underline{del}$: $\{I_j \mid j < i\}$)
ART-MD-NS ($D^m S^2$):
(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 2: The specification of Weld et. al.'s Synthetic Domains

3.2. Ability to exploit stored plans during plan reuse

In this section, we argue that the real utility of using partial order planning when doing EBG based plan reuse is that it provides a flexible and efficient ability to interleave the stored plans with new operators, thereby significantly increasing the planner's ability to exploit stored plans. To understand this, we need to look at the various possible ways in which a stored plan can be extended during planning.

When a macro is retrieved to be reused in a new problem situation, it will only be a partial match for the problem under consideration: (*i*) The macro may contain extraneous goals/constraints that are not relevant to the problem at hand. (*ii*) There may be some outstanding goals of the problem that the retrieved macro does not match. The first situation can be handled largely through the editing operations described earlier. In the second case, the planner may have to do some further planning work even after the macro is incorporated into the current plan. The way a planner extends the macro during planning critically affects its ability to reuse stored plans in new situations. This, in turn, depends upon whether the planner searches in the space of world-states or plans (Section 2.).

Suppose a planner is solving a problem involving a set G of goals, and retrieves a macro M which promises to achieve a subset G' of these goals. Let $g \in (G - G')$ be an outstanding goal of the problem. We will say that M is **sequenceable** with respect to the outstanding goal g if and only if there exists a subplan P for achieving g such that $M.P$ or $P.M$ (where "." is the sequencing operator) will be a correct plan for achieving the set of goals $G \cup \{g\}$. M is said to be **interleavable**[3] with respect to g, if and only if there exists a sub-plan P for achieving g such that P can be merged with M without retracting any steps, ordering constraints or binding constraints in M or P. In particular, if M corresponds to a plan $\langle T_M, O_M, \mathcal{B}_M \rangle$ (where T_M is the set of steps, O_M is the partial ordering on the steps and $\mathcal{B}_M$ is the binding constraints on the variables), and P corresponds to a plan $\langle T_P, O_P, \mathcal{B}_P \rangle$, then there exists a plan $P' : \langle T_M \cup T_P, O_M \cup O_P \cup O', \mathcal{B}_M \cup \mathcal{B}_P \cup \mathcal{B}' \rangle$ which achieves the set of goals $G \cup \{g\}$.[4]

Clearly, interleavability is more general than sequenceability. There are many situations when the macros are not sequenceable but only interleavable with respect to the outstanding goals of the planner. Consider the simple artificial domains, ART-IND, ART-MD and ART-MD-NS (originally described in [1]) shown in Figure 2. These domains differ in terms of the serializability [10] of the goals in the domain. ART-IND contains only independent goals (notice that none of the actions have delete lists). The goals in ART-MD are interacting but *serializable* while those in ART-MD-NS are *non-serializable*.[5] In particular, in the latter domain, macros will be

[3]Note that interleavability here refers to the ability to interleave plan steps. This is distinct from the ability to interleave subgoals. In particular state-based planners that don't use linearity assumption can interleave subgoals, but cannot interleave plan steps

[4]Interleavability of macros, as defined here differs from *modifiability* (c.f. [5, 6]) in that the latter also allows retraction of steps and/or constraints from the macro, once it is introduced into the plan. While modifiability is not the focus of the current work, in our previous work [5], we have shown that PO planners do provide a flexible framework for plan modification. More recently, we have been investigating the utility tradeoffs involved in incorporating a flexible modification capability in plan reuse [4].

[5]From the domain descriptions, it can be seen that a conjunctive goal

	SNLP			TOCL			TOPI	
	scratch	+SEBG	+IEBG	scratch	+SEBG	+IEBG	scratch	+SEBG
ART-MD								
% Solved	100%	100%	100%	100%	100%	100%	100%	100%
Cum. time	80	306	136	92	177	2250	1843	3281
% Macro usage	-	20%	100%	-	20%	100%	-	6%
ART-MD-NS								
% Solved	40%	40%	100%	30%	26%	60%	40%	40%
cum. time	19228	21030	4942	22892	23853	14243	20975	23342
% Macro usage	-	0%	100%	-	0%	100%	-	0%

Table 1: Performance statistics in ART-MD and ART-MD-NS domains.

interleavable, but not sequenceable with respect to any outstanding goals of the planner. 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). Clearly, the only way a macro can be reused in this domain is by interleaving it with new operators (unless of course it is an exact match for the problem).

Even 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 sequenceable with respect to the outstanding goals. For example, suppose the planner is trying to solve a problem with goals $G_1 \wedge G_2 \wedge G_3$ from ART-MD domain, and retrieves a macro which solves the goals $G_1 \wedge G_3$: $A_1 \rightarrow A_3$. Clearly, the outstanding goal, G_2 is not sequenceable with respect to this macro, 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.

The foregoing shows that any planner that is capable of using macros only when they are sequenceable with respect to the outstanding goals is less capable of exploiting its stored plans than a planner that can use macros also in situations where they are only interleavable. From our discussion in Section 2., it should be clear that planners that search in the space of world states, such as STRIPS, PRODIGY, and NOLIMIT [19], which refine plans only by adding steps to the beginning (or end, in the case of backward search in the space of states) of the plan, can reuse macros only when they are sequenceable with respect to the outstanding goals. In contrast, planners that search in the space of plans can refine partial plans by introducing steps anywhere in the middle of the plan, and thus can reuse macros even when they are only interleavable with respect to the outstanding goals. Of these latter, partial order planners, which eliminate premature commitment to step ordering through a more flexible plan representation, can be more efficient than total order planners. Based on this, we hypothesize that partial order planners not only will be able to reuse both sequenceable and interleavable macrops, but will also be able to do it more efficiently. This, we believe is the most important advantage of partial ordering planning during reuse.

4. Empirical Evaluation

The discussion in the previous section leads to two plausible hypotheses regarding the utility of PO planning in plan reuse frameworks. (*i*) PO planners are more efficient in exploiting interleavable macros than planners that search in the space of totally ordered plans, as well as state-based planners and (*ii*) This capability significantly enhances their ability to exploit stored macros to improve performance in many situations, especially in domains containing non-serializable sub-goals. We have tested these hypotheses by comparing the performance of three planners -- a partial ordering planner, a total ordering planner, both of which search in the space of plans; and a state-based planner -- in conjunction with two different reuse strategies.[6] In the following two subsections, we describe our experimental setup and discuss the results of the empirical study.

4.1. Experimental Setup

Performance Systems: Our performance systems consisted of three simple planners implemented by Barrett and Weld [1]: SNLP, TOCL and TOPI. SNLP is a causal-link based partial ordering planner, which can arbitrarily interleave subplans. TOCL is a causal link based total ordering planner, which like SNLP can insert a new step anywhere in the plan, but unlike SNLP, searches in the space of totally ordered plans[7]. SNLP, by virtue of its least commitment strategy, is more flexible in its ability to interleave operators than is TOCL. The third planner, TOPI carries out a backward-chaining world-state search. TOPI only adds steps to the beginning of the plan. Thus, unlike SNLP and TOCL, but like planners doing search in the space of plan states, such as STRIPS and PRODIGY, TOPI is unable to interleave new steps into the existing plan. All three planners are *sound* and *complete*. The three planners share many key routines (such as unification, operator selection, and search routines), making it possible to do a fair empirical comparison between them.

Reuse modes: To compare the ability of each planner to exploit the stored plan generalizations in solving new problems, the planners were run in three different modes in the testing phase: **Scratch mode**, **SEBG (or sequenceable EBG) mode** and **IEBG (or interleavable EBG) mode**. In the **scratch** mode, the planner starts with a null plan and refines it by adding steps, orderings and bindings until it becomes a complete plan. In the **SEBG** mode, the planner first retrieves a stored plan generalization that best matches the new problem (see below for the details of the retrieval strategy). The retrieved plan is treated as an opaque macro operator, and is added to the list of operator templates available to the planner. The planner is then called to solve the new problem, with this augmented set of operators. The **IEBG** mode is similar to the SEBG mode, except that it allows new steps and constraints to be introduced between the constituents of the instantiated macro and the rest of the plan, as the planning progresses. To facilitate this, whenever the planner selects a macro to establish a precondition, it consider the macro as a transparent plan fragment, and adds it to the existing partial plan. This operation involves updating the steps, orderings and causal links of the current partial plan with those of the macro. In the case of SNLP, the exact ordering of the steps of the macro with respect to the current plan can be left partially specified (e.g., by specifying the predecessor of the first step of the macro, and the successor of the last step of the macro), while in the case of TOCL, partial plans need to be generated for *each* possible totally ordered interleaving of the steps of the macro with respect to the steps of the current partial

$G_i \wedge G_j$ (where $i < j$) can be achieved in ART-IND domain by achieving the two goals in any order, giving rise to two plans $A_i \rightarrow A_j$ and $A_j \rightarrow A_i$. Only the first of these two plans will be a correct plan in ART-MD domain, since the delete literals in the actions demand that G_i be achieved before G_j. Finally, in ART-MD-NS domain, the subplans for G_i and G_j have to be interleaved to give the plan $A_i^1 \rightarrow A_j^1 \rightarrow A_i^2 \rightarrow A_j^2$.

[6] Code and test data for replicating our experiments can be acquired by sending mail to *rao@asuvax.asu.edu*

[7] Each partially ordered plan produced by SNLP corresponds to a set of totally ordered plans. TOCL generates these totally ordered plans whenever SNLP generates the corresponding partially ordered plans.

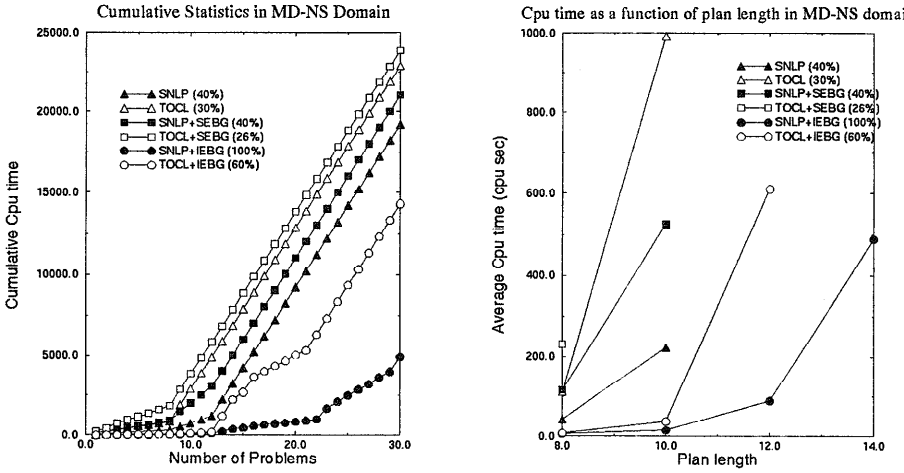

Figure 3: Performance across the three reuse modes in the ART-MD-NS domain

plan. SNLP is thus more efficient and least committed than TOCL in interleaving macros.

It is easy to see that the SEBG strategy can reuse a macro if and only if it is sequenceable with the other outstanding goals of the plan, while the more general IEBG strategy can also reuse a macro whenever it is interleavable with other outstanding goals of the plan. From the description of the three planners above, it should also be clear that only SNLP and TOCL can support IEBG mode. TOPI, like other state-based planners such as STRIPS, PRODIGY and NOLIMIT, cannot support IEBG mode.

Our SEBG and IEBG strategies differ from usual macro operator based reuse strategies in that rather than use the entire plan library as macro-operators, they first retrieve the best matching plan from the library and use that as the macro-operator. The primary motivation for this difference is to avoid the high cost of going through the entire plan library during each operator selection cycle. (This cost increase is due to both the cost of operator matching and instantiation, and the increased branching factor). Instead of a single best match plan, the strategies can be very easily extended to start with two or more best matching plans, which between them cover complementary goal sets of the new problem. This would thus allow for transfer from multiple-plans. Here again, the ability to interleave plans will be crucial to exploit multiple macros.

Storage and Retrieval Strategies: To control for the factors of storage compaction, and flexible plan editing, no specialized storage or editing strategies are employed in either of the planners. The retrieval itself is done by a simple (if unsophisticated) strategy involving matching of the goals of the new problem with those of the macros, and selecting the one matching the maximum number of goals. Although, there is much scope for improvement in these phases (for example, retrieval could have been done with a more sophisticated causal-structure based similarity metric, such as the one described in [6]), our choices do ensure a fair comparison between the various planners in terms of their ability to exploit stored plans.

Evaluation strategy: As noted earlier, sequenceability and interleavability of the stored macros with respect to the goals of the encountered problems can be varied by varying the ratio of independent vs. serializable vs. non-serializable goals in the problems. The artificial domains described in Figure 2, and Section 3.2. make ideal testbeds for varying the latter parameter, and were thus used as the test domains in our study. Our experimental strategy involved training all three planners on a set of 50 randomly generated problems from each of these domains. The training problems all have between 0 and 3 goals. During the training phase, each planner generalizes the learned plans using EBG techniques and stores them. In the testing phase, a set of 30 randomly generated problems, that have between 4 and 7 goals (thus are larger than those used in the training phase) are used to test the extent to which the planners are able to exploit the stored plans in the three different modes. A limit of 1000 cpu sec. per problem is enforced on all the planners, and any problem not solved in this time is considered unsolved (This limit includes both the time taken for retrieval and the time taken for planning). To eliminate any bias introduced by the time bound (c.f. [16]), we used the maximally conservative statistical tests for censored data, described by Etzioni and Etzioni in [2], to assess the significance of all speedups. All experiments were performed in interpreted Lucid Commonlisp running on a Sun Sparc-II.

4.2. Experimental Results

Table 1 shows the cumulative statistics for solving the 30 test problems from each domain for all three planners and all three reuse modes. For each domain, the first row shows the percentage of domain test problems that were correctly solved by each strategy within the 1000 cpu. sec. time bound. The second row shows the cumulative cpu. time. for running through all the test problems (as mentioned, if a problem is not solved in 1000 cpu. seconds, we consider it unsolved and add 1000 cpu. sec. to the cumulative time). The third row shows the percentage of the solved problems whose solutions incorporated retrieved library macros.

In the ART-MD domain, which has subgoals that are easily serializable, none of the planners show much improvement through reuse (although SNLP does perform significantly faster than TOCL in the interleaving EBG mode). All three planners are able to solve all the test problems from scratch within the 1000 cpu sec. time limit. The addition of SEBG and IEBG strategies does not change the solvability horizon. More importantly, the cumulative time taken is slightly worse in both SEBG and IEBG strategies as compared to from scratch planning. This is understandable given that the problems in this domain are easy to solve to begin with, and any improvements provided by reuse strategy are offset by the retrieval costs.

The situation in ART-MD-NS domain is quite different. We see that none of the planners are able to solve more than 40% of the problems in the from-scratch mode. Of the two reuse modes, SEBG remains at the same level of correctness as from-scratch. This is not surprising, since as discussed in Section 3.2., in ART-MD-NS domain, the stored plans are not sequenceable with respect to any remaining outstanding goals of the planner. Thus, unless the macro is an exact match (i.e., solves the full problem), it cannot be reused by an SEBG strategy.

The IEBG strategy on the other hand, dramatically improves the correctness rates of TOCL and SNLP from 40% to 60% and 100% respectively, while TOPI, which cannot support IEBG, stays unaffected. Moreover, as hypothesized, SNLP's improvement is

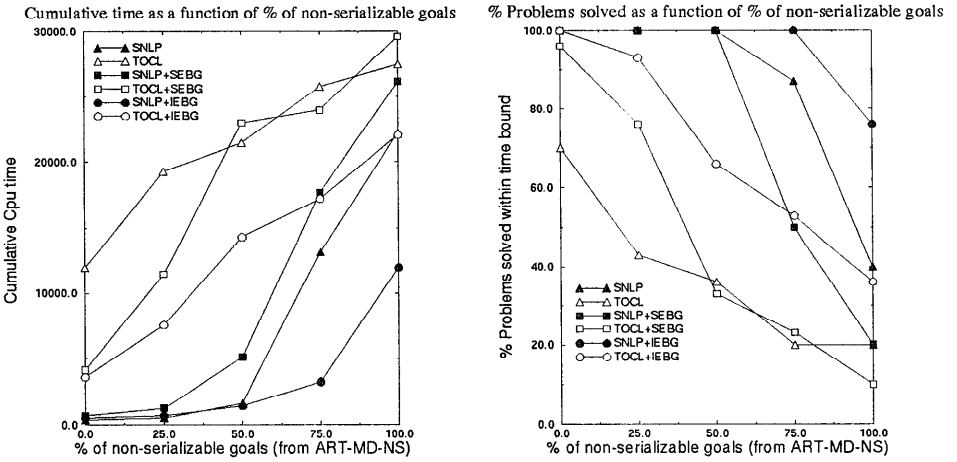

Figure 4: Cumulative performance as a function of % of non-serializable sub-goals

more significant than that of TOCL.[8] The plots in Figure 3 compare the performance of TOCL and SNLP for all three reuse strategies in this domain. The left one plots the cumulative planning time as a function of the problem number (with the problems sorted in the increasing order of difficulty). The right plot shows the average cpu time taken by each planner as a function of the plan length. We see that in IEBG mode SNLP significantly outperforms TOCL in the ability to exploit stored plans both in terms of cumulative time, and in terms of solvability horizon.

Experiments in Mixed Domains: The test domains ART-MD and ART-MD-NS above were somewhat extreme in the sense that the former only has serializable goals while the latter only has non-serializable goals. More typically, we would expect to see a mixture of independent, serializable and non-serializable goals in a problem distribution. To understand how the effectiveness of the various reuse strategies vary for such mixed problem distributions, we experimented with a mixed domain obtained by combining the actions of ART-IND (the domain with independent subgoals) and ART-MD-NS (the domain with non-serializable subgoals) domains (shown in Figure 2). Five different training and testing suites, each containing a different (pre-specified) percentage of non-serializable goals in the problem distribution, were generated. We experimented with problem sets containing 0, 25, 50, 75 and 100% non-serializable goals (where 0% corresponding to the problem set having goals drawn solely from ART-IND, and 100% corresponding to the problem set with goals drawn solely from ART-MD-NS). For each mixture, 50 training problems and 30 testing problems were generated randomly, as discussed before. Given the inability of TOPI to support IEBG, we concentrated on comparisons between SNLP and TOCL.

The plots in Figure 4 summarize the performance in each problem set as a function of the percentage of the non-serializable goals in the problem set. The plot on the left compares the cumulative time taken by each strategy for solving all the problems in the test suite of each of the 5 problem sets The plot on the right shows the percentage problems successfully solved within the time bound by each strategy for each problem set. Once again, we note that SNLP using IEBG shows the best performance in terms of both the cumulative time and the percentage problems solved. IEBG strategy is also the best strategy for TOCL, but turns out to be considerably less effective than the IEBG strategy for SNLP. More interestingly, we see that the performance of IEBG strategy compared to the base-level planner improves as the percentage of non-serializable goals in the problem set increases for both SNLP and TOCL. By the same token, we also note that the relative performance of SEBG strategy worsens with increased percentage of non-serializable goals for both SNLP and TOCL.

Summary: The results in our empirical studies are consistent with our hypothesis regarding the superiority of PO planners in exploiting stored macros. First, we showed that TOPI fails to improve its performance when faced with interleavable macros, while SNLP and TOCL can both exploit them. Next we showed that SNLP is more efficient in exploiting stored macros than TOCL. In particular, the strategy of using SNLP planner with IEBG reuse strategy significantly outperforms all the other strategies including TOCL+IEBG, in most cases. The higher cost of TOCL+IEBG strategy can itself be explained by the fact that TOCL generates partial plans corresponding to each possible interleaving of the macro with the new steps, while SNLP can maintain partially ordered plan and interleave the steps as necessary.

5. Related Work

Starting with STRIPS, stored plans have traditionally been reused as opaque macro operators that cannot be interleaved during planning. We believe that this was largely due to the limitations imposed by the underlying state-based planners. It is of course possible to get the effect of reuse of interleavable macros within state-based planning indirectly through the use of single operator macros (aka *preference search control rules* [14]). However, it is not clear what are the advantages of starting with a simplistic planner and get interleavability indirectly, when more sophisticated planners, such as PO planners, allow interleavability naturally. More generally, we believe that eager compilation strategies such as search control rules are complementary to rather than competing with more lazy learning strategies such as plan reuse. In some cases, the planner is better served by the lazy strategy of retrieving and modifying a large plan, rather than the eager strategy of compiling each incoming plan into search control rules. In the former, the ability to interleave is essential, and planners like STRIPS would be at a natural disadvantage. Interestingly enough, this was one of the original reasons for the shift from state based planning of STRIPS to plan-space based partial-order planning of NOAH, within the planning community. As McDermott [13, p. 413] remarks, if you want the ability improve performance by piecing large canned plans together, postponing decisions about how these plans will interact, then partial order planning is in some sense the inevitable choice.

In [19, 20], Veloso et. al. advocate basing learning techniques within state-based (total ordering) planning without linearity assumption, rather than within partial order planning. They justify this

[8] Using the statistical tests for censored data advocated by Etzioni in [2], we find that the hypothesis that SNLP+IEBG is faster than SNLP as well as the hypothesis that SNLP+IEBG is faster than TOCL+IEBG are both supported with very high significance levels by our experimental data. The *p-value* is bounded above by 0.000 for both the signed test, and the more conservative censored signed-rank test. The hypothesis that TOCL+IEBG is faster than TOCL is however supported with a much lower significance level (with a *p-value* of .13 for sign test and .89 for the censored signed-rank test).

and also scale up better to more expressive action representations because checking necessary truth of a proposition becomes NP-hard for partially ordered plans containing such actions. To begin with, as we discussed in Section 2., the inability to interleave macros is due to the limited refinement strategies allowed by a state-based planner, and has little to do with whether or not the planner makes linearity assumption. Secondly, the argument regarding the relative difficulty of scaling up the partial ordering planners to more expressive action representations is based on the (mistaken) premise that a PO planner has to interpret the full (necessary and sufficient) modal truth criterion for PO plans during each search iteration. Recent work [7, 12, 15] amply demonstrates that sound and complete partial ordering planners do not necessarily have to interpret the full-blown modal truth criterion at each iteration (since they only need completeness in the space of ground operator sequences rather than in the space of PO plans), and thus can retain their relative advantages over total ordering planners even with more expressive action representations. This, coupled with our experiments showing the superiority of PO planners in exploiting stored plans, make PO planning an attractive framework for doing plan reuse.

The concentration on state-based planning has also been true of much of the speedup learning work within planning, including search control rules and precondition abstraction. In [14, p. 83] Minton et. al. seem to justify this by the observation: "*[..] the more clever the underlying problem solver is, the more difficult the job will be for the learning component*". Preferring a state-based planning strategy only to make learning easier seems to be some what unsatisfactory, especially given that there exist more sophisticated planning strategies that avoid many of the inefficiencies of the state-based planners More over, we believe that the shift to more sophisticated planning strategies is not just an *implementation issue*; it may also lead to qualitatively different priorities in techniques as well as target concepts worth learning. As an example, one source of the inefficiency of STRIPS and other state-based planners stems from the fact that they confound the execution order (i.e., the order in which the goals are achieved during execution) with the planning order (i.e., the order in which the goals are attacked during planning). As we shift to planners that search in the space of plans, such as PO planners, planning order is cleanly separated from execution order; and many of the inefficiencies of state-based planners are naturally avoided. Thus, learning strategies and target concepts that are designed with state-based planners in mind may be of limited utility when we shift to more flexible planners. As an example, "*goal order preference rules*" of the type "work on $on(y, z)$ before $on(x, y)$", learned by EBL strategies in blocks world, are not that useful for partial order planners, which avoid premature step orderings to begin with. Similarly, in [17] Smith and Peot argue that may of the static abstractions generated with state-based planners in mind (e.g. [9]), impose unnecessary and sometimes detrimental ordering constraints, when used in conjunction with the more flexible and efficient partial order planning strategies. All of this, in our view, argues in favor of situating research on learning to improve planning performance within the context of more flexible and efficient planning paradigms such as partial order planning.

6. Concluding Remarks

In this paper, we have addressed the issue of relative utility of basing EBG based plan reuse strategies in partial ordering vs. total ordering planning. We observed that while storage compactions resulting from the use of PO plans can be exploited irrespective of whether the underlying planner is a PO planner or a total ordering planner, the PO planners do have distinct advantages when it comes to the issue of effectively *reusing* the stored plans. In particular, we showed that PO planners are significantly more efficient in exploiting interleavable macros than state-based planners as well as planners that search in the space of totally ordered planners. We also showed that this capability substantially enhances the ability to exploit stored macros to improve performance in many situations, where the domain goals and problem distributions are such that a significant percentage of stored macros are only interleavable with respect to the outstanding goals of the planner. Although this can happen both in domains with serializable subgoals and domains with non-serializable subgoals, our experiments show that the effect is particularly strong in the latter.

When taken in the context of recent work on comparative advantages of PO planners in plan generation [1, 12, 15], our study strongly argues for situating EBG based plan reuse strategies within the context of PO planning framework. We believe that such a move would also benefit other learning strategies such as search control rules and abstractions [4], and are currently working towards verifying these intuitions.

Acknowledgements: We wish to thank Tony Barrett and Dan Weld for distributing the code for SNLP, TOCL and TOPI planners. We also thank Oren Etzioni, Laurie Ihrig, Smadar Kedar, Suresh Katukam, Prasad Tadepalli and (anonymous) reviewers of AAAI-93 and ML-93 for helpful comments on a previous draft.

References

[1] A. Barrett and D. Weld. Partial order planning: Evaluating possible efficiency gains. Technical Report 92-05-01, Department of Computer Science and Engineering, University of Washington, June 1992.

[2] O. Etzioni and R. Etzioni. Statistical methods for analyzing speedup learning experiments. *Machine Learning*. (To Appear).

[3] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251--288, 1972.

[4] S. Kambhampati. Utility tradeoffs in incremental plan modification and reuse. In *Proc. AAAI Spring Symp. on Computational Considerations in Supporting Incremental Modification and Reuse*, 1992

[5] S. Kambhampati and J.A. Hendler. A validation structure based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3), June 1992.

[6] S. Kambhampati. Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence* (To appear).

[7] S. Kambhampati and D.S. Nau. On the Nature and Role of Modal Truth Criteria in Planning. University of Maryland Inst. for Systems Res. Tech. Rep. ISR-TR-93-30, 1993.

[8] S. Kambhampati and S. Kedar. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. Technical Report (ASU-CS-TR 92-008), Arizona State University, 1992. (A preliminary version appears in Proc. 9th AAAI, 1991).

[9] Craig Knoblock. Learning abstraction hierarchies for problem solving. In *Proc. 8th AAAI*, pages 923--928, 1990.

[10] R. Korf. Planning as Search: A quantitative approach. *Artificial Intelligence*, 33(1), 1987.

[11] J. Allen and P. Langley and S. Matwin. Knowledge and Regularity in Planning. In *Proc. AAAI Symp. on Computational Consideration in Supporting Incremental Modification and Reuse*, 1992.

[12] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th AAAI*, 1991.

[13] D. McDermott. Regression Planning. *Intl. Jour. Intell. Systems*, 6:357-416, 1991.

[14] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni and Y. Gil. Explanation-based Learning: A problem solving perspective. *Artificial Intelligence*, vol 40, 1989.

[15] S. Minton, M. Drummond, J. Bresina, and A. Philips. Total order vs. partial order planning: Factors influencing performance. In *Proc. KR-92*

[16] A. Segre, C. Elkan, and A. Russell. A critical look at experimental evaluation of EBL. *Machine Learning*, 6(2), 1991.

[17] D.E. Smith and M.A. Peot. A critical look at Knoblock's hierarchy mechanism. In *Proc. 1st Intl. Conf. on AI Planning Systems*, 1992.

[18] P. Tadepalli and R. Isukapalli. Learning plan knowledge from simulators. In *Proc. workshop on Knowledge compilation and speedup learning.*

[19] M. M. Veloso, M. A. Perez, and J. G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proc. DARPA Planning workshop* pages 207--212, November 1990.

[20] M.M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, Carnegie-Mellon University, 1992.