To appear in Proc. 12th Natl. Conf. on Artificial Intelligence (AAAI-94), August, 1994

Derivation Replay for Partial-Order Planning

Laurie H. Ihrig & Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406
email: laurie.ihrig@asu.edu rao@asu.edu

Abstract

Derivation replay was first proposed by Carbonell as a method of transferring guidance from a previous problem-solving episode to a new one. Subsequent implementations have used state-space planning as the underlying methodology. This paper is motivated by the acknowledged superiority of partial-order (PO) planners in plan generation, and is an attempt to bring derivation replay into the realm of partial-order planning. Here we develop DerSNLP, a framework for doing replay in SNLP, a partial-order plan-space planner, and analyze its relative effectiveness. We will argue that the decoupling of planning (derivation) order and the execution order of plan steps, provided by partial-order planners, enables DerSNLP to exploit the guidance of previous cases in a more efficient and straightforward fashion. We validate our hypothesis through empirical comparisons between DerSNLP and two replay systems based on state-space planners.

Introduction

Case Based Planning involves storing individual instances of planning episodes and using them to tackle new situations. One method of reusing an earlier planning episode is through *derivational analogy* (DA) (Carbonell 1986; Veloso 1992). By this method, a trace of a previous search process is retrieved and replayed in solving a new problem. The decisions that led to a successful solution in the prior case are used to guide the new search process. After its first proposal in (Carbonell 1986), DA has subsequently been found to be of use in many areas, including planning, problem solving, design and automatic programming (Veloso 1992; Mostow 1989; Blumenthal and Porter 1994; Bhansali and Harandi 1991). Much of this work has been done in state-space problem solvers. The aim of the current work is to adapt derivational analogy to partial-order planning. We are motivated by a desire to see whether the known advantages of PO planning over state-space planning in plan generation, (Barrett and Weld 1994; Minton *et al.* 1992), also make the former a more efficient substrate for replay.

*This research is supported in part by National Science Foundation under grant IRI-9210997, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039. Thanks to Manuela Veloso for helpful clarifications regarding Prodigy/Analogy, and to Suresh Katukam and Ed Smith for their comments.

Derivational analogy includes all of the following elements (Veloso 1992): a facility within the base-level planner to generate a trace of the derivation of a problem solution, the indexing and storage of the solution trace in a library of cases, the retrieval of a case in preparation for solving a new problem, and finally, a replay mechanism by which the planner can utilize a previous derivation in a new search process. The storage and retrieval aspects of DA remain the same whether we use plan-space or state-space planners. In particular, solutions to the storage problem such as those proposed in (Veloso 1992) and (Kambhampati 1994) can be used for this purpose. Only the contents of the trace and the details of the replay component depend on the underlying planner. Thus, in the current work, we focus on the automatic generation and replay of the solution trace.

We will start by describing DerSNLP, an implementation of derivation replay within SNLP, a PO planner (McAllester and Rosenblitt 1991; Barrett and Weld 1994). We will then use DerSNLP as a case-study to explore the relative advantages of doing replay within plan-space vs. state-space planners. One of the difficult decisions faced by the replay systems is that of deciding when and where to interleave from-scratch effort with derivation replay (c.f. (Blumenthal and Porter 1994)). In general, there are no domain-independent grounds for making this decision. This makes *eager replay*, i.e., replaying the entire trace before returning to from-scratch planning, the most straightforward strategy. We will show that, for replay systems based on state-space planners, eager replay inhibits effective transfer in problems where the plan-step ordering is critical to the solution. We will argue that the decoupling of planning (derivation) order from execution order of steps, provided by the plan-space planners, allows effective transfer to occur with eager replay in more situations, and thus provides for a more efficient and straightforward replay framework. We will validate this hypothesis by comparing DerSNLP to replay systems implemented on two different state-space planners: NOLIMIT (which was recently the basis for a comprehensive DA implementation in (Veloso 1992)), and TOPI (Barrett and Weld 1994).

DerSNLP: Derivation Replay for SNLP

As we mentioned earlier, derivation replay involves storing traces of previous problem-solving decisions and replaying them to solve similar problems more efficiently. The problem-

solving trace that is retained for future replay consists of a sequence of instructions that describe the series of choices made in the original planning episode. Choice points correspond to backtracking points in whatever planning algorithm has been selected. The content of each choice therefore reflects the underlying methodology. For example, a state-space means-ends analysis (MEA) planner such as NOLIMIT (Veloso 1992) makes decisions as to which subgoal to accomplish next, which step to use to achieve the subgoal, as well as when to add an applicable step to the plan.

For SNLP, a search node corresponds to a partly-constructed partially-ordered plan. SNLP makes two types of decisions. An *establishment* decision is a choice as to the method of achieving an open subgoal. A new step may be added to contribute an open condition and a *causal link* formed that links the new contributor to the step that consumes the condition. Alternatively, a causal link may be formed from an existing step.

The second type of decision is a choice as to the method of *resolution* of a *threat* to a causal link. When a causal link is threatened by a possibly intervening step, the conflict may be resolved either by adding a step-ordering [1] or by adding variable binding constraints that remove the threat. Plan refinement proceeds as follows:

1. If there exist conflicts in the current active plan, **Act**, then choose a threat and handle the selected threat by nondeterministically choosing a resolution: either promotion of the step that is threatening the link, demotion of that step, or the addition of variable binding constraints to resolve the conflict, else
2. If there are no threats then choose a condition among **Act**'s open conditions, and nondeterministically select a refinement of **Act** that handles the condition by adding a causal link from an (existing/new) step.

This process terminates when there are no more open conditions and no more threats to the existing causal links.

DerSNLP extends SNLP by including a replay facility. Its output is a trace of the decision process that led to its final solution. Figure 1 contains an example trace produced by DerSNLP while attempting a problem from the logistics transportation domain of (Veloso 1992). This domain involves the movement of packages across locations by various transport devices. The trace corresponds to a simple problem which contains the goal of getting a single package, OB2, to a designated airport, AP1.

The solution trace is annotated with the choices that were made along the path from the root of the search tree to the final plan in the leaf node. Each decision becomes an instruction for a future search process. 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

[1] For an example, consider that the effect of the step, S_k, is clobbering the contribution of P by another step, S_i, as described by a causal link, $S_i \xrightarrow{P} S_j$, and may possibly be ordered in between S_i and S_j. The conflict may then be resolved by *promoting* S_k so that it comes after the threatened link, in other words, by adding a step ordering $S_j < S_k$.

Goal : (AT-OB OB2 AP1)
Initial : ((IS-A AIRPORT AP1) (IS-A AIRPORT AP2))
(IS-A AIRPORT AP3) (AT-PL PL1 AP3)
(AT-OB OB2 AP2) ...

Name : G1 Type : START-NODE
Name : G2 Type : ESTABLISHMENT Kind : NEW-STEP New Step: (UNLOAD-PL OB2 ?P1 AP1) New Link: (1 (AT-OB OB2 AP1) GOAL) Open Cond: ((AT-OB OB2 AP1) GOAL)
Name : G3 Type : ESTABLISHMENT Kind : NEW-STEP New Step: (FLY-PL ?P1 ?A2 AP1) New Link: (2 (AT-PL ?P1 AP1) 1) Open Cond: ((AT-PL ?P1 AP1) 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 AP3) 3) Open Cond: ((AT-PL ?P1 ?A3) 3)
Name : G6 Type : ESTABLISHMENT Kind : NEW-LINK New Link: (0 (IS-A AIRPORT AP2) 3) Open Cond: ((IS-A AIRPORT ?A2) 3)

Name : G7 Type : ESTABLISHMENT Kind : NEW-LINK New Link: (0 (IS-A AIRPORT AP1) 2) Open Cond: ((IS-A AIRPORT AP1) 2)
Name : G8 Type : ESTABLISHMENT Kind : NEW-STEP New Step: (LOAD-PL OB2 PL1 ?A4) New Link: (4 (INSIDE-PL OB2 PL1) 1) Open Cond: ((INSIDE-PL OB2 PL1) 1)
Name : G9 Type : ESTABLISHMENT Kind : NEW-LINK New Link: (3 (AT-PL PL1 AP2) 4) Open Cond: ((AT-PL PL1 ?A4) 4)
Name : G10 Type : RESOLUTION Kind : PROMOTION Unsafe-link : ((3 (AT-PL PL1 AP2) 4) 2 (AT-PL PL1 AP2))
Name : G11 Type : ESTABLISHMENT Kind : NEW-LINK New Link: (0 (AT-OB OB2 AP2) 4) Open Cond: ((AT-OB OB2 AP2) 4)

Key to Abbreviations:
PL = PLANE
AP = AIRPORT
OB = OBJECT

Final Plan: (FLY-PL PL1 AP3 AP2) Created 3
(LOAD-PL OB2 PL1 AP2) Created 4
(FLY-PL PL1 AP2 AP1) Created 2
(UNLOAD-PL OB2 PL1 AP1) Created 1
Ordering of Steps: ((4 < 2) (3 < 4) (4 < 1) (3 < 2) (2 < 1))

Figure 1: An Example Solution Trace for DerSNLP

is valid in the context of the new active plan if the condition that it previously achieved is also an open condition in the new plan and, secondly, if the action that was taken earlier in achieving this condition is consistent with the new plan's constraints on variable bindings. A threat resolution choice is justified if steps added through replay result in a similar threat in the new active plan, and if the prescribed method of resolution (promotion/demotion/separation) is consistent with that plan.

DerSNLP also contains a mechanism for replay of a previous derivation. Along with the new problem description, it receives as input a small set of cases. It is assumed that these cases are retrieved from the case library, and correspond to previously solved problems that have goals in common with the new problem. Replay is called from the `PlanRefinement` procedure at the point where an open condition is to be handled (See Figure 2).The set of guiding cases (**GCs**) is searched for one that contains that condition as a top level goal. If a candidate is found its solution trace is passed to the `Replay` procedure, outlined in Figure 2.The previous trace may correspond to more than one open condition.

DerSNLP's default replay strategy is *eager* in that the full trace is visited during a single call to `Replay`. This avoids the decision of how to interleave the replay of multiple cases, as well as how to interleave replay with from-scratch planning, both decisions that must be faced by NOLIMIT (Veloso 1992). It is our contention that interleaving is not as important for replay in a plan-space framework. Each instruction in the trace is therefore visited in succession.

The replay process must validate each instruction contained

PROCEDURE `PlanRefinement` **(GCs, Act): Plans**
1.IF there is an unsafe link in **Act**, THEN
  choose *t* from **Act**'s unsafe links, and
  RETURN `HandleUnsafe` (*t*, **Act**)
2. ELSE, pick a subgoal *o* from the list of open conditions
3. IF `GetCase` (*o*,**GCs**,**Act**) returns a candidate *c*, THEN
4. IF `Replay` (*c*, **Act**) returns a plan, return this plan only
5. ELSE, RETURN `HandleOpen` (*o*, **Act**)

PROCEDURE `Replay` **(Instrn, Act):Plan**
1.IF null Instrn, THEN
  RETURN **Act**, ELSE
2. SelectedRefinement:= `ValidateInstrn` (Instrn, **Act**)
3.IF SelectedRefinement, THEN
  RETURN `Replay` (Next(Instrn), SelectedRefinement)
4. ELSE, RETURN `Replay` (Next(Instrn), **Act**)

PROCEDURE `ValidateInstrn` **(Instrn, Act):Plan**
1.IF Instrn is of type Establishment, THEN
2. IF OpenCondOf (Instrn) ∈ OpenCondsOf (**Act**), THEN
  ActRefinements:=`HandleOpen`(**Act**, OpenCond)
3. ELSE, IF Instrn is of type ThreatResolution, THEN
4. IF UnsafeLinkOf (Instrn) ∈ UnsafeLinksOf(**Act**), THEN
  ActRefinements:=`HandleUnsafe` (**Act**, UnsafeLink)
5.SelectedRefinement:=
  `FindMatch` (DecisionOf (Instrn), ActRefinements)
6.IF SelectedRefinement, THEN
  push its siblings onto SearchQueue, and
  RETURN SelectedRefinement
7. ELSE RETURN FALSE

Figure 2: Outline of DerSNLP's replay procedures

in the old trace before transferring its guidance. Replay calls `ValidateInstrn` to justify an instruction in the context of the current active plan, **Act** (See Figure 2). Consider as an example the trace contained in Figure 1 and suppose that this trace is being replayed for a second problem in which there is an additional package, OB3, which is also to be transported to the same airport. At the beginning of the replay episode, **Act** contains two open conditions: (AT-OB OB2 AP1) and (AT-OB OB3 AP1). 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** [2]. Since (AT-OB OB2 AP1) 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 OB2 ?P1 AP1), to one of these plan refinements. `ValidateInstrn` returns the plan refinement that corresponds to the prescribed choice. Siblings of this plan are added to the open list of the new search process.

In our eager replay strategy, control is shifted to the sequence of instructions in the previous trace. It is this sequence that determines the order in which conditions are solved and threats are resolved. Decisions in the trace that are not justified are skipped. The output of the `Replay` procedure is a 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.

The plan that is returned by the `Replay` procedure may still contain open conditions, either because a prior decision that was taken in solving a subgoal is not valid in the new context, or because there are top level goals that are not covered by the previous trace. For example, if the trace in Figure 1 is replayed for a problem in which the initial location of the airplane is changed, instruction G5 is not valid and must be skipped, and the condition it establishes is left open. Further planning effort is then needed to achieve this unestablished goal. Moreover, if the trace is replayed for a problem in which there are extra top level goals that are not in the previous problem, then these goals will also be left open.

In contrast to state-space planners, PO planners with their least-commitment strategy are more flexible as to the order of derivation of plan steps. If extra goals require steps that have to be interleaved into the plan that is produced through a replay episode, these steps can be added after the full trace is replayed. As an example, suppose again that the trace contained in Figure 1 is replayed for the problem that requires the additional goal, (AT-OB OB3 AP1), and OB3 is initially on the old route taken by the airplane. DerSNLP can further refine the plan that results from replaying the trace by adding only the steps needed to load and unload the extra package. This is not possible if replay is based on a state-space planner. In the next section, we discuss this point in greater detail.

Advantages of Basing Replay on a PO Planner

In this section, we will compare the DerSNLP algorithm to replay implementations on state-space planners (and problem solvers) (Veloso 1992; Blumenthal and Porter 1994). State-space planners refine plans by adding steps either only to the end, or only to the beginning, of the already existing operator sequence. This means that when step order is critical to the success of the plan, steps must be added to the plan according to their execution order. When replay is based on a state-space planner and step order is critical, eager replay may not allow for effective transfer. Consider our simple problem from the logistics transportation domain contained in Figure 1. Suppose that the problem was previously solved by a state-space planner. Suppose further that this case is replayed in solving a problem of transporting an additional package that is located somewhere along the plane's route. Further planning effort is needed to add steps to load and unload the extra package. Since steps have to be added in their order of execution, replay will have to be interrupted at the right place in order to add these additional steps. However, the optimal point in the derivation for inserting the new steps depends on the problem description, since it will depend on where the new package is located on the old route. In general, there are no domain-independent grounds for deciding when and where the from-scratch problem-solving effort should be interleaved with replay. The more straightforward *eager* replay strategy, on the other hand, will tend to mislead the

[2] 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.

planner into wrong paths, eventually making it backtrack or find inoptimal plans. The early work on derivation replay that is rooted in state-space planning has therefore been forced to focus a good deal of attention on the problem of determining *when* to plan for additional goals (Blumenthal and Porter 1994). It is not an easy matter to determine at what point in the derivation to stop in order to insert further choices corresponding to extra goals.

For the PO planner, there is less need to make the difficult decision as to when to interrupt replay. In particular, since the PO planners decouple derivation (planning) order of plan steps from their execution order, an eager replay strategy will not mislead them as much. This makes for a more efficient and straightforward replay of the trace, since control can be solely in the hands of the previous search process for the duration of the replay episode.

To summarize, eager replay is the most straightforward way of combining replay and from-scratch efforts. State-space planners tend to be misled by eager replay in situations where step order is critical for a plan's success. DerSNLP, based on a partial-order planning framework, does not suffer from this problem. This leads us to the hypothesis that plan-space planners will exhibit greater performance improvements when using eager replay.[3] The next section provides an empirical evaluation of this hypothesis.

Empirical Evaluation

An empirical analysis was conducted in order to test our hypothesis regarding the relative effectiveness of eager replay for PO planners. To do this we chose two state-space planners, TOPI (Barrett and Weld 1994) and NOLIMIT (Veloso 1992). We implemented eager replay on these planners and compared their performance with DerSNLP. TOPI does simple backward search in the space of states, adding steps to the plan in reverse order of execution. NOLIMIT is a version of PRODIGY which was the basis of the DA system reported in (Veloso 1992). Like PRODIGY and STRIPS, it uses means-ends analysis, attempting goals by backward-chaining from the goal state. 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. It does this by adding relevant operators to a list of potential operators before actual placement in the plan.

To facilitate fair comparisons, the three planning methods, SNLP, TOPI, and NOLIMIT, were (re)implemented on the same substrate. A replay mechanism was added to each planner which follows an eager replay strategy. With this strategy the search process is interrupted to replay the entire derivation trace before returning to from-scratch planning.

[3]Even DerSNLP may be misled by eager replay in some cases. For example, the previous case may have achieved one of the goals using some step s_1 and the new problem contains a goal g_n which cannot be achieved in the presence of s_1. However, in such cases, state-space planners will also be misdirected. Thus our hypothesis is only that DerSNLP is *less likely* to be misled (and thus more likely to exploit the previous case) by eager replay.

Domains

ART-MD-NS domain: Experiments were run on problems drawn from two domains. The first was the artificial domain, ART-MD-NS, originally described in (Barrett and Weld 1994) and shown in the table below:

ART-MD-NS ($D^m S^2$):
A_i^1 $\underline{precond}$: I_i $\underline{add}$: P_i $\underline{delete}$: $\{I_j | j < i\}$)
A_i^2 $\underline{precond}$: P_i $\underline{add}$: G_i $\underline{delete}$: $\{I_j | \forall j\} \cup \{P_j | j < i\}$)

Conjunctive goals from this domain are *nonserializable* in that they cannot be achieved without interleaving subplans for the individual conjuncts. For example, consider the problem that contains the conjunctive goal $G_1 \wedge G_2$. The subplan for achieving this goal would be: $A_1^1 \rightarrow A_2^1 \rightarrow A_1^2 \rightarrow A_2^2$. This plan has to be interleaved with steps to solve the additional goal G_3. The plan for the new conjunctive goal $G_1 \wedge G_2 \wedge G_3$ is $A_1^1 \rightarrow A_2^1 \rightarrow \underline{A_3^1} \rightarrow A_1^2 \rightarrow A_2^2 \rightarrow \underline{A_3^2}$.

Logistics Transportation Domain: The logistics transportation domain of (Veloso 1992) was adopted for the second set of experiments. Initial conditions of each problem represented the location of various transport devices (one airplane and three trucks) over three 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.

Whereas each problem in ART-MD-NS has a unique solution, in the logistics 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 logistics domain, they can sometimes avoid backtracking by continuing in the replayed path, but will find inoptimal plans in such cases.

Testing

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 logistics 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 logistics domain, the planners used a best-first strategy (with a heuristic that biases the planner towards the replayed path).

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* (see below). 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

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	-	-

Table 1: Performance statistics in ART-MD-NS and Logistics Transportation Domain (Average solution length is shown in parentheses next to %Solved for the logistics domain only)

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. Cases were chosen so as to have all but one of the goals matching. 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 eager replay on PO vs. state-space planning to be apparent in these phases.

Results

The results of testing are shown in Tables 1 and 2. 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 1 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 logistics domain (Solution length was omitted in ART-MD-NS since all the problems have unique solutions.) The subsequent rows of Table 1 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 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 logistics domain within the time limit, and this column is therefore omitted from the table.

In ART-MD-NS domain, replay resulted in performance improvements for DerSNLP which increased with problem size (See Table 1). 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). These results are consistent with our hypothesis that state-space planners will be misled by eager replay when step order is critical.

Table 2 reports three different measures that indicate the effectiveness of replay. The first is the percentage of *sequenced* replay. Replay of a trace is judged to be *sequenced* with the new search process if the search path that is obtained through guidance from the previous trace is extended by further planning effort to solve the new problem. Sequenced replay is indicated when all of the plan-refinements created through replay of a previous trace appear on the search path that leads to the final solution. The percentage of sequenced replay therefore indicates the percentage of problems for which replay guides the new search directly down a path to the solution. When replay is not sequenced, search is directed down the wrong path and replay may actually increase search time.

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 2). In the logistics domain, the PO planner also experienced some nonsequenced replay, but less than the state-space planner in the multi-goal phase. There are two reasons for the non-sequenced replay shown by DerSNLP in the logistics domain. First, unlike ART-MD-NS, where all the problems had the same initial state, in the logistics 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 logistics 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.

The efficiency of replay for the PO planner is also indicated by the two other measures contained in the final two rows of Table 2. These are the percentage of plan-refinements on the final solution path that were formed through guidance

Phase	ART-MD-NS			Logistics	
	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 2: Measures of Effectiveness of Replay

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.

Related Work

Previous research in DA which is rooted in state-space planning has demonstrated significant performance improvements with replay (Veloso 1992). In contrast, our experiments show rather poor replay performance on the part of the state-space planners for the more complex problems. We believe the main reason for this may be the strong presence of interacting goals in our multi-goal problems, coupled with the fact that we used vanilla planning algorithms without any sophisticated backtracking strategies or pruning techniques (Veloso 1992). While our experiments used an eager-replay strategy, work by Blumenthal (Blumenthal and Porter 1994) provides heuristic strategies aimed at interleaving planning and replay effort. However, it is our contention that the shift to PO planning obviates to a large extent the need for interleaving. Finally, our conclusions regarding the advantages of PO planning in replay also complement the recent results regarding its advantages in reuse (Kambhampati and Chen 1993).

Conclusion

In this paper, we described DerSNLP, a framework for doing derivation replay within SNLP, a partial-order planner. We then compared DerSNLP with replay systems for state-space planners. We started by arguing that since there are no domain-independent grounds for deciding when and how to interleave replay with from-scratch planning, eager replay is the most straightforward way of combining replay with from-scratch effort. We then showed that eager replay tends to mislead state-space planners in situations where step order is critical to the success of a plan. In contrast, DerSNLP, which is based on a partial-order planning framework, and thus decouples derivation (planning) order of plan steps from their execution order, does not suffer from this problem. Thus, DerSNLP is more likely to be able to exploit previous cases using eager replay than replay systems based on state-space planners. We supported this hypothesis with the help of empirical comparisons of DerSNLP with two different state-space replay systems, and across two different domains. Our future work will aim to explore the impact of possible variations within plan-space planning algorithms on the efficiency of replay.

References

Barrett, A. and Weld, D. 1994. Partial order planning: evaluating possible efficience gains. *Artificial Intelligence* 67(1).

Bhansali, S. and Harandi, M. 1991. Synthesizing unix shell scripts using derivational analogy: an empirical assessment. In *Proceedings AAAI-91*.

Blumenthal, B. and Porter, B. 1994. Analysis and empirical studies of derivational analogy. *Artificial Intelligence*. Forthcoming.

Carbonell, J. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, Ryszard; Carbonell, Jaime; and Mitchell, Tom M., editors 1986, *Machine Learning: an Artificial Intelligence approach: Volume 2*. Morgan-Kaufman.

Kambhampati, S. and Chen, J. 1993. Relative utility of ebg based plan reuse in partial ordering vs total ordering planning. In *Proceedings AAAI-93*. 514--519. Washington, D.C.

Kambhampati, S. 1994. Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence Journal* 10(2).

McAllester, D. and Rosenblitt, D 1991. Systematic nonlinear planning. In *Proceedings AAAI-91*. 634--639.

Minton, S.; Drummond, M.; Bresina, J.; and Philips, A 1992. Total order vs partial order planning: factors influencing performance. In *Proceedings KR-92*.

Mostow, J. 1989. Automated replay of design plans: Some issues in derivational analogy. *Artificial Intelligence* 40:119--184.

Veloso, M. 1992. *Learning by analogical reasoning in general problem solving*. Ph.D. Dissertation, Carnegie-Mellon University.