

Derivational Replay in an Universal Classical Planning Framework

Héctor Muñoz-Avila,¹ Dana S. Nau,² and Tsz-Chiu Au²

¹Department of Computer Science and Engineering
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015, USA
munoz@cse.lehigh.edu

²Department of Computer Science and Institute for System Research
University of Maryland
College Park, MD 20742, USA
{nau, chiu}@cs.umd.edu

Abstract: The status of plan adaptation has been somewhat controversial in the AI-planning literature, due to a conflict between worst-case complexity analyses (which have led to pessimistic conclusions about the utility of plan adaptation) and empirical results (in which plan adaptation has performed significantly better than planning from scratch). This paper provides a step toward the resolution of this conflict, for the case of a prominent plan-adaptation technique called derivational replay. First, we provide a general model of derivational replay. Next, we use this model to show that derivational replay does not satisfy a key assumption in the previous complexity analyses, and thus those analyses do not apply to it. Finally, we show that the search space for a planner using derivational replay is no bigger than it would be without derivational replay, and that derivational replay can reduce the size of the search space by n exponential amount.

Key words: derivational analogy, case-based planning, classical planning

1. Introduction

Plan adaptation—the technique of generating plans for new problems by adapting solution plans for previous problems—has a somewhat controversial status. On one hand, a well-known theoretical study concluded that in the worst case, plan adaptation takes exponentially more time than planning from scratch (Nebel & Koehler, 1995). On the other hand, empirical studies have shown plan adaptation running significantly faster than planning from scratch (Veloso, 1994; Ihrig & Kambhampati, 1996, Muñoz-Avila & Weberskirch, 1996). The purpose of this paper is to help resolve the apparent conflict. Our contributions are as follows.¹

¹ This paper expands and extends the work presented in (Au et. al., 2002), which won the award for the best research paper at *ECCBR-02*.

First, we provide a general model of derivational replay, a prominent plan-adaptation technique. Our model, DerUCP (for Derivational UCP), is based on UCP (Universal Classical Planner), which is a general model for classical AI planning (Kambhampati and Srivastava, 1995).

Next, we use our model to show that derivational replay does not satisfy a key assumption, called minimal plan modification, needed for the complexity analysis in (Nebel & Koehler, 1995). Roughly, a plan adaptation algorithm satisfies the minimal plan modification condition if it is guaranteed to reuse the largest possible number of steps of a given plan when solving a new problem. Our main results are as follows:

- Nebel & Koehler's minimal plan modification assumption does not apply to derivational replay, so their complexity results do not apply. The reason for this is that derivational analogy, rather than looking for the replay that uses largest possible number of steps, just uses the first replay that it happens to find—and if this replay turns out to be minimal, this is purely coincidental. Thus, for each case in which derivational replay produces a minimal plan modification, there are many others where it does not.
- The search space for a planner using derivational replay is no bigger than it would be without derivational replay. The reason for this is that derivational replay does not change the structure of the search space: it merely guides the planner through a certain part of this space by replaying a polynomial number of decisions from a previous planning episode. We show that this can reduce the size of the search space by an exponential amount, even if the planner later needs to backtrack over a portion of the replay.

Section 2 provides background material on derivational replay and UCP. Section 3 describes the DerUCP algorithm. Section 4 discusses various instances of DerUCP. Section 5 presents the complexity results. Section 6 analyzes the search space traversed by instances of DerUCP. Section 7 discusses related work, and Sections 8 presents the conclusions.

2. Background

2.1. Derivational Replay

Derivational Replay (DR) was first introduced in (Carbonell, 1986) and first operationalized into a system in (Velooso & Carbonell, 1993).

In DR the so-called derivational traces, sequences of planning decisions that led to previous solution plans, are reused when solving a new problem (Velooso & Carbonell, 1993). These decisions indicate choices that an underlying first-principles planner must make to generate the solution plans. Since reusing derivational traces may not

entirely solve the new problem, DR presupposes a first-principles planner to enhance and revise pieces of the plan obtained by reusing derivational traces.

Studies of DR have included its application for partial-order planning (Ihrig & Kambhampati, 1994; Muñoz-Avila et al, 1994), integrating user interactions (Cox & Veloso, 1999), synthesizing UNIX script shells (Bhansali & Harandi, 1994), design applications (Blumenthal, 1990), and studies about its knowledge engineering requirements (Cunningham et al, 1994). As reported in several experiments, case-based planners using derivational replay have consistently outperformed their underlying first-principles planner (Veloso, 1993; Ihrig & Kambhampati, 1994, Muñoz-Avila, 1996). These experiments were performed independently of one another, using a random problem generator, and were performed in 3 different domains. Veloso (1993) reported on the logistics transportation domain, Ihrig & Kambhampati (1994) on a variant of the logistics transportation domain, and Muñoz-Avila (1998; 2001) on the process planning domain.

2.2 Universal Classical Planning

Analyses of DR have been impeded by the lack of a general theoretical model for DR. The lack of such a model has made it impossible to analyze the efficiency of DR except in certain specific cases, e.g., DR using plan-space planners (Ihrig & Kambhampati, 1997). In this section, we formulate a general model of derivational replay for classical (i.e., STRIPS-style) planning domains. Our model is based on Universal Classical Planning (UCP), Kambhampati & Srivastava's (1996) general model of classical planners (Fikes & Nilsson, 1972).

UCP (Kambhampati & Srivastava, 1996; Kambhampati, 1997) is an abstract algorithm in which planning is done by making repeated refinements to partial plans, stopping when a complete plan is obtained. All known classical planning algorithms are special cases of UCP; for example, SNLP and PRODIGY can be described as instances of UCP.

In UCP, a *partial plan* consists of a set of steps to be performed, and a set of constraints that must be satisfied. Formally, a partial plan is a 5-tuple (T, O, B, L) , where T is the set of steps in the plan, O is the partial ordering relation over T , B is a set of codesignation and non-codesignation constraints² on the variables in the preconditions and post-conditions of the operators, and L is a set of auxiliary constraints. There are three kinds of auxiliary constraints:

- An interval preservation constraint (IPC) is denoted as $(t \stackrel{p}{-} t')$, which means the condition p must be preserved between the operators corresponding to steps t and t' .

² A codesignation constraint between two variables indicates that they must be instantiated to the same value. A non-codesignation constraint indicates that they cannot take the same value.

- A point truth constraint (PTC) is a 2-tuple $(p@t)$, which ensures condition p must be true before step t .
- A contiguity constraint $(t * t')$ does not allow any step between t and t' . These constraints restrict the ground linearization of the steps in the set of candidate solution plans represented by a partial plan.

A partial plan is a *solution* if every ground linearization of its steps achieves the goal state from the initial state.

UCP starts out with a single partial plan: a “null plan” consisting of two steps t_0 and t_∞ that represent the initial state and the final state, and a single constraint $(t_0 < t_\infty)$ indicating that t_0 must occur before t_∞ . Within UCP, each partial plan P can be seen as representing a set of candidate solution plans, namely the ones that can be produced by refinements to P . Thus, the null plan represents all plans that are solutions to the planning problem.

UCP has several *refinement strategies* that it can use to produce refinements of a partial plan. These refinement strategies can be seen as ways to constrain the conditions that any solution plan must meet; thus with each refinement that UCP makes to P , the set of solution plans is reduced. Each refinement strategy adds new steps or constraints to P , thus eliminating all candidate solution plans that do not contain the new steps or satisfy the new constraints. There are three kinds of plan refinements:

- *Forward state-space (FSS)* refinements add constraints or new steps only after the head fringe of the plan. The **head fringe** is the last step, t_n , in the sequence of contiguous steps $t_0 * t_1 * \dots * t_n$ starting at t_0 . In the initial plan, the head fringe is the step t_0 . The effect of t_0 is the initial state. The **head state** is referred to the state obtained by transforming forwards the initial state with the sequence of steps $t_1 * \dots * t_n$.
- *Backward state-space (BSS)* refinements add constraints or new steps before the **tail fringe**. The tail fringe is the first step, t_n , in the sequence of contiguous steps $t_n * \dots * t_1 * t_\infty$ ending at t_∞ . In the initial plan, the tail fringe is the step t_∞ . The preconditions of t_∞ are the final state. The **tail state** is the state obtained by transforming backwards the final state with the sequence of steps $t_n * \dots * t_1$.
- *Plan-space (PS)* refinements resemble the establishment of open prerequisites in partial-order planners by introducing causal links and ordering constraints between steps.

Since these three refinement strategies elongate the ground operator sequences a partial plan covers, they are called *progressive refinements*. UCP also has three *non-progressive refinements*, which do not elongate the ground operator sequences but reduce the set of solutions that a partial plan represents. These are Refine-plan-pre-order, which adds ordering constraints between steps, Refine-plan-pre-position, which adds contiguous constraints between steps, and Refine-plan-conflict-resolve, which resolves conflicts in the partial plan.

Table 1 shows the pseudocode of the UCP algorithm. It maintains a list PL of currently active partial plans. It refines these plans by adding new steps and constraints; it does this repeatedly until either a solution is found or no partial plans are left.

Table 1. The UCP Algorithm

```

procedure UCP( $PL$ )
  Inputs:  $PL$  - a list of partial plans
begin
  0. Plan Selection
    If  $PL$  is empty then exit with failure. Otherwise, select a partial
    plan  $P$  in  $PL$ , and remove  $P$  from  $PL$ .
  1. Termination Check
    If  $P$  contains a ground operator sequence that solves the problem,
    returns it and terminates.
  2. Progressive Refinement
    Using pick-refinement strategy, select any one of
    1. Refine-plan-forward-state-space( $P$ )
    2. Refine-plan-backward-state-space( $P$ )
    3. Refine-plan-plan-space( $P$ )
    Let  $L'$  be all of the returned plans.  $PL \leftarrow PL \cup L'$ 
  3. (optional) Non-progressive Refinement
    For each  $P'$  in  $L'$ , select zero or more of:
    1. Refine-plan-conflict-resolve( $P'$ )
    2. Refine-plan-pre-ordering( $P'$ )
    3. Refine-plan-pre-positioning( $P'$ )
    Let  $L''$  be all of the returned plans.  $PL \leftarrow PL \cup L''$ 
  4. (optional) Consistency Check
    For each  $P''$  in  $L''$ , select zero or more of:
    if the partial plan  $P''$  is inconsistent or non-minimal then
      remove  $P''$  from  $L''$ .
  5. Recursive Invocation:
    Call UCP( $PL$ )
end

```

In Step 0, the algorithm selects a partial plan to be refined. In the version of UCP presented in (Kambhampati & Srivastava, 1996; Kambhampati, 1997) this selection was done nondeterministically—but any practical implementation would be deterministic, and we need a deterministic algorithm in order to do our complexity analyses. Thus, we have used the standard technique for turning nondeterministic choices into deterministic ones: we keep a list, PL , of all of the currently active partial plans, and select one of them using an implementation-dependent selection function. The different search strategies used in classical planning algorithms (depth-first, breadth-first, heuristically guided, etc.) can be expressed as different selection functions. Also in Step 0, UCP chooses one of a partial plan P in PL . If the partial

plan contains a solution, the algorithm returns it and terminates (Step 1). Otherwise, UCP performs progressive refinements (Step 2) and non-progressive refinements (Step 3) to P . It inserts the partial plans generated by these refinement operations into PL , performs a consistency check (Step 4) to remove plans that cannot lead to solutions or cannot lead to minimal solutions, and invokes itself recursively on PL . All of UCP's basic operations (how it chooses a plan, how it refines the plan, and how does the consistency check) will vary from one instance of UCP to another. Furthermore, Steps 3 and 4 are optional: some instances of UCP will not do them at all.

3. DerUCP: Derivational Replay in UCP

We will now formulate the DerUCP algorithm for performing derivational replay on top of UCP. DerUCP serves as a general model for derivational replay in classical planning. For example, as we will show later, derivational-analogy case-based planners such as Prodigy/Analogy (Velo & Carbonel, 1993) CAPlan/CbC Muñoz-Avila & Weberskirch, 1996) and DerSNLP (Ihrig & Kambhampati, 1994) are special cases of DerUCP.

3.1 Derivational Trace

A derivational trace for DerUCP consists of a sequence of decisions made at decision points during an execution of an instantiation of UCP. The UCP algorithm contains search choice points where choices are made, and each choice is captured in a form of a **decision record** with three components: the match-point, the decision and the refinement. The **match-point** is the part of the plan for which the decision is applicable. In derivational replay algorithms such as Prodigy/Analogy, the match-points are the goals. The reason is that these algorithms perform backward refinements starting with the goals, and generating subgoals at every step. In DerUCP this is one of the possibilities for match-points and are represented by the head state. Other possibilities for match-points are the head fringe and constraints. The **decision** that was made, and the corresponding actions that were undertaken. **Refinements** mimic UCP's notation. A **derivational trace** is an ordered list of decision records. The match-point associated with the first decision record is called the **top match-point** of the derivational trace. Informally, the derivational trace corresponds to a branch of the search tree.

There are two kinds of refinements in UCP: the progressive refinements and non-progressive refinements. All search choice points are located in the refinement operations in the UCP algorithm. For example, each time UCP needs to make a progressive refinement, it makes an arbitrary choice. Also, after the generation of partial plans by any refinement, one of the partial plans is arbitrarily selected. According to which refinement a choice points serves, we can roughly group the

choice points into six categories. Each category is the set of choice points for a refinement.

There are three types of progressive refinements: forward state-space refinement, backward state-space refinement and plan-space refinement. Here are the decision records for the progressive refinements:

1. Decision record of forward state-space refinements
 - a. Match-point: The head state.
 - b. Decision: Select a step t , either a new or existing one from the head-fringe, such that all preconditions of its operator are satisfied in the head-state.
 - c. Refinement: If t is a new step, create it. Add a contiguity constraint between the head state and the step t .
2. Decision record of backward state-space refinements
 - a. Match-point: The tail state.
 - b. Decision: Select a step t , either a new or existing one from the tail-fringe that does not delete any precondition in the tail state, and achieves at least one precondition in the tail state.
 - c. Refinement: If t is a new step, create it. Add a contiguity constraint between the step t and the tail state.
3. Decision record of plan-space refinements
 - a. Match-point: a point truth constraint ($p@t'$).
 - b. Decision: Select a step t , either a new or existing one, that establishes the point truth constraint ($p@t'$).
 - c. Refinement: If t is a new step, create it. Add enough auxiliary constraints to the plan such that ($t < t'$) and no steps that come between t and t' delete p . Optionally, add IPCs to indicate the establishment decision.

Non-progressive refinements reduce the number of candidate solutions by imposing additional constraints on the plan. For example, the introduction of an ordering constraint between two steps of a partial plan will reduce the number of possible linearizations of the plan. Here are the decision records for the non-progressive refinements:

1. Decision record of refine-plan-pre-order
 - a. Match-point: A pair of unordered steps t_1 and t_2 .
 - b. Decision: Reduce the number of candidate solutions by imposing the following constraints.
 - c. Refinement: Generate two refinements of P : $P + (t_1 < t_2)$ and $P + \neg(t_1 < t_2)$.
2. Decision record of refine-plan-pre-position
 - a. Match-point: A pair of non-contiguous steps t_1 and t_2 .

- b. Decision: Reduce the number of candidate solutions by imposing the following constraints.
 - c. Refinement: Generate two refinements of P : $P + (t_1 * t_2)$ and $P + \neg(t_1 < t_2)$.
3. Decision record of refine-plan-conflict-resolve
 - a. Match-point: An auxiliary constraint C and a step t_3 in conflict with C .
 - b. Decision: Reduce the number of candidate solutions by imposing the following constraints.
 - c. Refinement: Refine P such that C will hold in all the ground linearizations of P . If C is an IPC $(t_1 \stackrel{p}{\prec} t_2)$, the refinement is called a Conflict Resolution refinement. In this case, a step t_3 that has an effect that unifies with $\neg p$ is selected. The refinements consist of $P + (t_3 < t_1) \vee (t_2 < t_3)$ and $P + II(t_3, p)@t_3$, where $II(t_3, p)$ denotes the preconditions of t_3 with respect to p .

The refine-plan-conflict-resolve refinements are typical of plan-space planners. The step t_3 threatens the IPC $(t_1 \stackrel{p}{\prec} t_2)$ and thus, the decision is to promote or demote t_3 . That is, ordering t_3 after t_2 or before t_1 .

These decision records take into account all of the search choice points in the UCP algorithm. Therefore, during the recording of a case, one of the six decision records is created at each search choice point.

The following example shows various refinements with UCP and the corresponding decision records for a problem in the logistics transportation domain (Velo, 1994). In this domain, packages must be relocated using some transportation methods. There are some restrictions imposed on the transportation methods. For example, trucks can only deliver packages within a city. Suppose that we have the problem depicted in Figure 1. There are four locations A , B , C and D . A vehicle $V1$ and a package $p1$ are in location A and another vehicle $V2$ and another package $p2$ are in location D . Suppose that the goals are to relocate both packages at location C . Here are the operators that we use in the example:

- $L(pack, V, loc)$: loads the package $pack$ into the vehicle V at the location loc . The vehicle V and the package $pack$ must be at location loc .
- $UL(pack, V, loc)$: unloads the package $pack$ from the vehicle V at the location loc . The vehicle V must be at location loc and $pack$ must be in V .
- $MV(V, loc1, loc2)$: moves the vehicle V from $loc1$ to $loc2$. The vehicle V must be at location $loc1$.

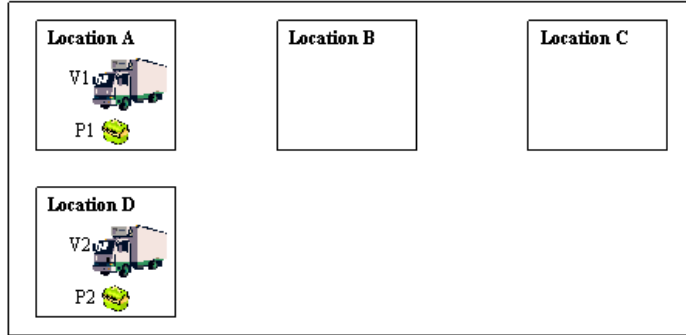


Figure 1. A simple planning problem from the Logistics domain

UCP begins with a null plan ($t_0 < t_\infty$), where t_0 is the initial step, which represents the initial state depicted in Figure 1, and t_∞ is the final step, which represents the two goals that must be achieved. UCP starts the refining process of the null plan by performing a PS refinement unloading $p2$ from $V1$ at C (Step t_1 of Figure 2). An alternative step, unloading $p1$ from $V2$ at location C (Step t_1') is also shown for illustration purposes. The UCP algorithm continues refining the partial plan with two successive FSS refinements: first, loading $p1$ to $V1$ at A (Step t_2), and then, loading $p2$ to $V2$ at D (Step t_3). The bottom of Figure 2 shows a final plan after several refinements (only the first three refinements are shown). Any linearization of the final plan is a solution. For example, a possible solution is to: (1) load the package $p1$ into $V1$, (2) Load the package $p2$ into $V2$, (3) move $V1$ to location B , (4) move $V2$ to location B , (5) unload the package $p2$ from $V2$, (6) load $p2$ into $V1$, (7) move $V1$ to location C , (8) unload $p1$ from $V1$, and (9) unload $p2$ from $V1$.

Table 2 shows the derivational trace for the first three refinements of Figure 2. The first decision record indicates that the operator $UL(p2, V1, C)$ was selected to achieve the goal $p2$ must be located at C using a plan-space refinement. The next decision record transforms the head state by using forward state-space refinement with the operator $L(p1, V1, A)$. The third decision record transforms the head state once more by using the operator $L(p2, V2, D)$. The next section describes how the DerUCP algorithm uses these decision records.

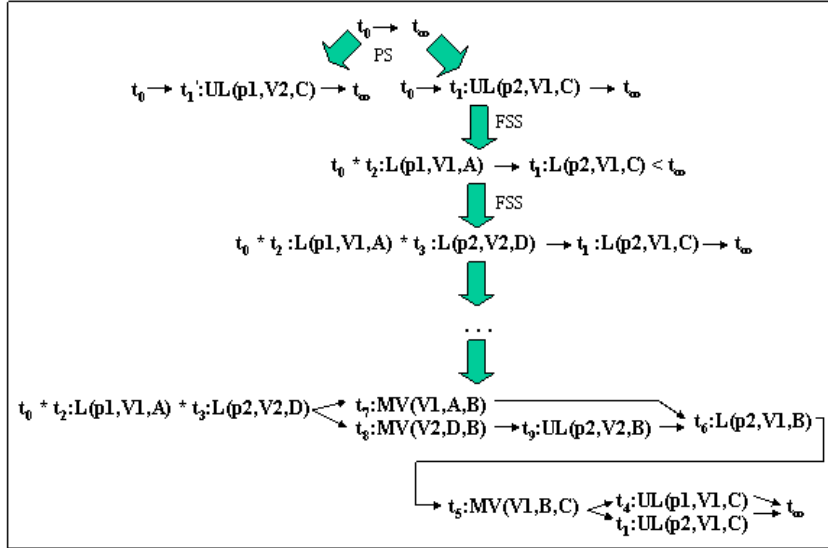


Figure 2. An example of plan generation in UCP. The arrows between steps indicate non-contiguous ordering relations between them. The asterisks (*) indicate contiguity constraints between them. The block arrows indicate UCP's refinement operations.

Table 2. The derivational trace for the example in Section 2

<p>Step: 1 Type: Plan-space refinement Match-point: The point truth constraint ($g @ t_\infty$), where g is the goal: “p2 must be located at C” Decision: Select operator $UL(p2, V1, C)$. Refinement: Create a new step t_1 and add a constraints ($t_0 < t_1$), ($t_1 < t_\infty$), and the IPC ($t_1 \stackrel{g}{\prec} t_\infty$)</p>
<p>Step: 2 Type: Forward state-space refinement Match-point: The head-state. Decision: Select operator $L(p1, V1, A)$. Refinement: Create a new step t_2 and add a contiguity constraint ($t_0 * t_2$)</p>
<p>Step: 3 Type: Forward State-space refinement Match-point: The head-state Decision: Select operator $L(p2, V2, D)$ Refinement: Create a new step t_3, add ordering constraints ($t_1 * t_2$)</p>

The set of choice points and decisions confronted by an execution of DerUCP depends on the particular instance of DerUCP. Therefore, a derivational trace

recorded from an execution of instance of UCP cannot be used for a different instance of UCP. For example, if we construct a derivational trace for a partial-order planner, some of the decision records in the derivational trace would not make sense for a total-order planner. A total-order planner can only add new steps at the head fringe or at the tail fringe of the partial plan whereas a partial-order planner can add steps anywhere in the partial plan.

3.2 The DerUCP Algorithm

The DerUCP algorithm extends the UCP planning algorithm by interleaving derivational replay with UCP's refinement operations. This section describes only the derivational replay mechanism, which relies on the replay step and the procedure *Replay*. For a precise description of other parts of the UCP algorithm, please refer to (Kambhampati & Srivastava, 1996).

Table 3 shows the pseudocode of the DerUCP algorithm. The algorithm is basically the same as the one shown in Table 1, except that it now receives as input the case library and includes the replay step (Step 2). Each **select** instruction represents a backtracking point.

The replay step selects a case and replays it. For case replay, any match-point g_c of the case can be selected as long as it matches part of the current plan. The details of *Replay* procedure are discussed below. After the replay step, the partial plan, P , is refined with the standard UCP process. Finally, all the partial plans generated by the refinement operations are inserted into the priority list and DerUCP is invoked recursively (Step 6). Notice that Step 3 (progressive refinement) is now optional. The reason is that we want to allow replay of multiple cases as done with systems like Prodigy/Analogy and DerSNLP.

In the replay step, the set of refinement points of the current partial plan is computed (Table 4). The possible refinement points are the match-points of all six decision records described in Section 3.1. The replay of the current case continues only if the next match-point g_c in the case matches the match-point g (Steps 2-3). Each decision record in the derivational trace guides the selection of the appropriate refinements. The *Replay* procedure recursively iterates over the decision records of the case, and in each iteration it checks if the refinement suggested by the decision of the current decision record is a valid refinement in the current partial plan (Steps 4-6). If so, the refinement is selected and the alternative refinements are put on the priority list for possible backtracking (Steps 7-10). After every iteration, DerUCP makes an implementation-dependent choice of whether to continue with the next decision record or stop the replay process (Step 11).

Table 3. The DerUCP Algorithm

```
procedure DerUCP( $PL, CL$ )
  Inputs:  $PL$  - a priority list
          $CL$  - a case library
begin
  0. Plan Selection
    If  $PL$  is empty then exit with failure. Otherwise, select a partial
    plan  $P$  in  $PL$ , and remove  $P$  from  $PL$ .
  1. Termination Check
    If  $P$  contains a ground operator sequence that solves the
    problem, return it and terminate.
  2. (optional) Replay
    Construct a list  $G$  of refinement points for  $P$ .
    Let  $S \subseteq CL$  be the set of cases that match refinement points in  $G$ 
    if  $S$  is not empty then
      From  $S$ , select a case  $C$  having a match-point  $g_c$  that matches
      a refinement point  $g$  in  $G$ .
       $P \leftarrow \mathbf{Replay}(g_c, g, P, G, C, PL)$ 
       $PL \leftarrow PL \cup \{P\}$ 
  3. (optional) Progressive Refinement
    Using pick-refinement strategy, select any one of
    1. Refine-plan-forward-state-space( $P$ )
    2. Refine-plan-backward-state-space( $P$ )
    3. Refine-plan-plan-space( $P$ )
    Let  $L'$  be all of the returned plans.  $PL \leftarrow PL \cup L'$ 
  4. (optional) Non-progressive Refinement
    For each  $P'$  in  $L'$ , select zero or more of:
    1. Refine-plan-conflict-resolve( $P'$ )
    2. Refine-plan-pre-ordering( $P'$ )
    3. Refine-plan-pre-positioning( $P'$ )
    Let  $L''$  be all of the returned plans.  $PL \leftarrow PL \cup L''$ 
  5. (optional) Consistency Check
    For each  $P''$  in  $L''$ , select zero or more of:
    if the partial plan  $P''$  is inconsistent or non-minimal then
      remove  $P''$  from  $L''$ .
  6. Recursive Invocation:
    Call DerUCP( $PL, CL$ )
end
```

Table 4. The Replay Procedure

```
procedure Replay( $g_c, g, P, G, C, PL$ )
begin
  1. if  $C$  is null then
    return  $P$ 
  2. Let  $d_c$  be the decision for  $g_c$  in  $C$ 
  3. Let  $cs$  be the set of applicable refinements for  $g$ 
  4. if  $\exists r \in cs$  that matches(decision( $r$ ),  $d_c$ ) then
  5.   Apply the refinement  $r$  to  $P$  to obtain  $P'$ 
  6.   Apply the refinements in  $cs - \{r\}$  to  $P$  to obtain a set of partial
      plans  $PS$ 
  7.   Append  $PS$  to  $PL$  with a lower priority
  8.   Construct a list of refinement points  $G'$  for  $P'$ 
      ( $G'$  can be obtained by modifying  $G$ )
  9.   Let  $g_c$  be the match-point of  $C$ 
      if there is no  $g \in G'$  that match  $g_c$  then
        return  $P'$ 
  10.  else
  11.   Select one of the following:
        return  $P'$  or
        if there is a  $g \in G'$  that that match  $g_c$  then
          return Replay( $g_c, g, P', G', C, PL$ ),

  12. else return  $P$ 
End
```

The following proposition follows immediately from Table 4:

Proposition 0. The running time for **Replay** is polynomial in the size of the case C .

A more complicated issue is the amount of time needed to retrieve C in Step 2 of the DerUCP algorithm. One way to do case matching would be to use AC unification (Siekman, 1989). But since AC unification can take exponential time in the worst case, most derivational replay systems do not use it. For example, derivational replay systems such as CAPlan/CbC use domain information to restrict the number of combinations when matching the goals during retrieval. Prodigy/Analogy's retrieval mechanism avoids spending too much time in case retrieval by performing a partial match between the new problem and the cases instead of pursuing to find the best matching case. DerSNLP start by solving 1-goal problems and storing the cases generated. Then it proceeds to solve 2-goal problems and so forth. All these strategies resulted in polynomial time retrieval in the empirical evaluations performed by these systems.

3.3 Example of Case Replay in DerUCP

As an example of problem solving with DerUCP, consider the problem in the logistics transportation domain whose initial situation is depicted in Figure 1 and whose two goals are to relocate packages $p1$ and $p2$ to C . A solution trace is shown in Figure 2. The first three steps of the derivational trace for this solution trace are shown in Table 2. Now suppose that a new problem is given with the same goals as before and almost identical initial state to the one in Figure 1. The only difference is that $V2$ is not available. Initially DerUCP is called with PL containing the null plan and CL containing the derivational trace partly shown in Table 2 as its only case. A resulting trace obtained by DerUCP is shown in Figure 3. In Step 0 of the DerUCP procedure, the null plan is selected as the partial plan P to refine. Since P has no solutions (Step 1), the derivational trace is selected for replay (Step 2). The match-point of the first goal $p2$ must be located at C, in the derivational trace matches a goal of P (Step 3 of the Replay procedure). Since the decision *Select operator* $UL(p2, V1, C)$ is applicable in the partial plan, it is replayed and P is expanded (Steps 4-10). Suppose that in Step 11, the recursive call is selected. The next decision, *Select operator* $L(p1, V1, A)$, is also applicable in the new problem, and, thus, it is also replayed. Suppose that one more time, the recursive call to the Replay procedure is made. The next decision, *Select operator* $L(p2, V2, D)$, is not applicable in P since we assume that the vehicle $V2$ is not available. Thus, Step 6 of the Replay procedure fails. Assuming that this time no recursive call is made, the Replay procedure terminates. At this point, standard UCP procedures continue refining the plan and recursive calls to DerUCP are made (Steps 3-6 of the DerUCP procedure). At the bottom of Figure 3, a possible solution plan is shown, in which vehicle $V1$ is used first to relocate package $p1$ (Steps t_2, t_3 and t_4) and then to relocate package $p2$ (Steps t_5, t_6, t_7 and t_l). In this example, only t_1, t_2 , and t_4 were obtained from case replay. The remaining steps were obtained with standard UCP. Other runs of DerUCP with the same input may result in solutions containing more steps replayed from the case. We will come back to this issue later on.

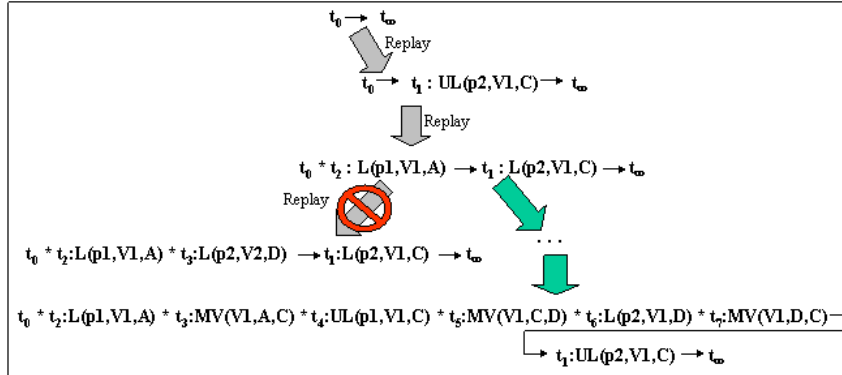


Figure 3. An example of plan generation in DerUCP. The arrows between steps indicate non-contiguous ordering relations between

them. The asterisks (*) indicate contiguity constraints between them. The block arrows indicate refinement operations.

4. Instances of DerUCP

We now describe several well-known derivational-analogy planners as instances of DerUCP.

4.1 Prodigy/Analogy

Prodigy/Analogy (Veloso & Carbonell, 1993) was the first system to operationalize and implement DR on top of a first principles planner. We now discuss the crucial characteristics of Prodigy/Analogy and how they are represented in our framework:

- **Forward search, means-end analysis.** The first-principles planner used by Prodigy/Analogy is Prodigy, which performs a forward state-space search whereby goal regression is used to guide the selection of operators that transform the state (Veloso et al, 1995). Thus, the progressive refinement done by Prodigy is Refine-plan-forward-state-space in the DerUCP algorithm (Step 3.1 in Table 3), and, correspondingly, the decision records of the derivational trace contain forward state-space refinements. These decision records indicate how the plan was refined and, therefore, the operator that was used for refinement of the plan.
- **Case interleaving.** One of the crucial characteristics of Prodigy/Analogy is that it *interleaves* case replay and first-principles planning. Prodigy/Analogy can switch from case replay to first-principles planning and vice versa. In the DerUCP algorithm interleaving occurs with successive calls to case replay (Step 2 in Table 3) and the recursive call to case replay (Step 6 in Table 3). In Prodigy/Analogy a case need not to be replayed fully, this is covered in Step 11 of Table 4, where a decision is made to continue replaying the case or not.
- **Skipping Decision Records.** Prodigy/Analogy may replay decision records from a case up to a certain point in the derivational trace T , skip some steps in T (possibly interleaving case replay from other cases or performing first-principles planning), and then continue with case replay with the next steps in T . In the DerUCP algorithm, such situation is covered when the decision is made whether to continue replaying a case C or not (Step 11 in Table 4), and then in Step 2 of Table 3 when DerUCP “select a case C having a match-point g_c that matches a refinement point g in G ”. The match point g_c can be any point in the derivational trace of the same case C including a later point after the case replay was stopped.
- **Justifications replay.** At decision points, a planning system may make decisions that turn out to be wrong forcing the planner to make a different decision. In Prodigy/Analogy, the derivational trace not only contains the sequence of decisions that led to successfully solving the problem, but

also annotations, called justifications, of the reasons why decisions made by the planner were wrong. During case replay, both the sequence of decisions that led to a solution and the justifications for invalid decisions are replayed. The advantage of replaying these justifications is that if backtracking occurs on decisions made during case replay, the planner does not need to explore those invalid decisions whose justifications were replayed. Therefore, the search space that needs to be explored is reduced. In the DerUCP framework we do not include justifications of invalid decisions. Therefore, our complexity and search space analysis can be considered as a worst-case scenario for Prodigy/Analogy where no justifications could be replayed.

4.2 DerSNLP

DerSNLP (Ihrig & Kambhampati, 1997) is built on top of the plan-space planner SNLP. SNLP performs Steps 3.3 (plan-space refinement) and 4.1 (Refine-plan-conflict-resolve) of the DerUCP algorithm (Table 3). Thus, decisions in the derivational traces contain one of these two kinds of refinements.

A key characteristic of DerSNLP is that the case is replayed first and then the resulting partial plan (called the *skeletal* plan) is completed by first-principles planning. Thus, no interleaving as in Prodigy/Analogy takes place. The rationale for this form of replay is that plan-space planners can interleave steps as needed (Ihrig & Kambhampati, 1994). This form of replay is covered in DerUCP as follows:

1. In the first iteration of DerUCP the replay procedure is called (Step 2 of Table 3). The replay procedure always performs the recursive call (Step 11 of Table 4) and terminates only when no goal in the partial plan matches a goal in the case (Step 3). This form of replay is called *eager replay* since the replay procedure continues until no decision from the case can be replayed.
2. In subsequent recursive calls of DerUCP other cases may be selected and the same process described in 1 is repeated
3. The partial plan obtained from Steps 1 and 2 is completed by first-principles planning. This is achieved by performing plan-space refinements and conflicts resolution steps in subsequent recursive calls to DerUCP.

4.3 CAPlan/CbC

CAPlan/CbC (Muñoz-Avila & Weberskirch, 1996; Muñoz-Avila & Weberskirch, 2001) also implements DR on top of SNLP. CAPlan/CbC was motivated by the domain of process planning for mechanical pieces that are symmetrical with respect to an axis. For these kinds of domains, it is crucial to guide SNLP on how to solve conflicts between parts of the partial plan. These conflicts reflect interactions between

components of the mechanical piece being machined. CAPlan/CbC also introduces justifications for invalid decisions as in prodigy/Analogy but in the context of plan-space planners.

5. Complexity Analysis of DerUCP

We now analyze the main result in (Nebel & Koehler, 1995) to see whether or not it applies to DerUCP. Definitions 1–4 are the same ones used in (Nebel & Koehler, 1995).

Definition 1. An instance of propositional STRIPS planning is denoted by a tuple (Pr, O, I, G) , where:

- Pr is a finite set of ground atoms. Let L be the corresponding set of literals: $L = Pr \cup \{\neg p : p \in Pr\}$.
- O is a finite set of operators of the form $Pre \rightarrow Post$, where $Pre \subseteq L$ are the preconditions and $Post \subseteq L$ is the postconditions or effects. The positive postcondition is the add list, while the negative postcondition is the delete list.
- $I \subseteq Pr$ is the initial state.
- $G \subseteq L$ is the goal.

A state S is a subset of Pr , indicating that $p \in Pr$ is true in that state if $p \in S$, and false otherwise. A state S is a goal state if S satisfies G , i.e., if all positive literals in G are in S and none of the negative literals in G is in S .

Definition 2. PLANSAT is the following decision problem: given an instance of the planning problem $\Pi = (Pr, O, I, G)$, does there exist a plan Δ that solves Π ?

Definition 3. A **conservative strategy** to plan modification is one that solves the following **plan modification problem**: given a planning-problem instance $\Pi_I = (Pr, O, I, G_I)$ and a plan Δ that solves another instance $\Pi = (Pr, O, I, G)$, produce a plan Δ_I that solves Π_I by minimally modifying Δ .

Definition 4. MODSAT is the following decision problem: Given a planning-problem instance $\Pi_I = (Pr, O, I, G_I)$, a plan Δ that solves another instance $\Pi = (Pr, O, I, G)$, and an integer k , does there exist a plan Δ_I that solves Π_I and contains a subplan of Δ of at least length k ?

As pointed out by Nebel and Koehler, MODSAT is the definition produced by turning the plan modification problem into a search problem. Given a plan-modification strategy M , we can write a **decision-theoretic version** of M that uses M to produce a plan, measures the plan's length l , and returns “yes” iff $l \leq k$.

Obviously, if M is a conservative strategy then the decision-theoretic version of M solves MODSAT. The main result of Nebel and Koehler is the following.

Proposition 1. If PLANSAT is PSPACE-complete or NP-complete, then MODSAT is a PSPACE-complete or NP-complete problem, respectively.

Furthermore, they show that the converse is not true; in some situations for which plan generation is a polynomial time problem while plan modification is NP-complete. This result provides strong evidence that plan adaptation can be computationally worse than plan generation and, therefore, calls into question problem-solving by plan adaptation.

We now consider two different ways in which DerUCP's replay can be characterized:

- *Interleaved versus non-interleaved replay.* Interleaved replay occurs when case replay and first-principles planning alternate at different stages of planning process. The recursive call of DerUCP ensures that interleaved replay can take place (Step 6 of Table 3). Non-interleaved replay occurs when case replay is followed by first-principles planning or vice versa.
- *Eager versus non-eager replay.* Eager replay occurs when case replay continues as long as there are decisions in the case that are applicable in the partial plan being generated. Step 11 of the replay procedure (Table 4) determines if the replay procedure is eager or not. If the recursive call is always made, the replay is eager. Non-eager replay occurs if some of the applicable decisions from the case are replayed (e.g., if the recursive call in Step 11 is not always made).

Since these two characterizations are independent of each other, there are four possible cases for the type of replay used in DerUCP:

- Case 1: interleaved and non-eager replay.
- Case 2: non-interleaved and eager replay.
- Case 3: interleaved and eager replay.
- Case 4: non-interleaved and non-eager replay.

Proposition 2. In all four of the cases described above, DerUCP does not follow a conservative strategy.

Proof. We will construct a counter-example using the example of plan adaptation discussed in Section 3.3. In this example, a plan Δ is available that relocates two packages to location C (see Figure 1). As part of this plan (see bottom of Figure 2), the package originally located in D is moved to B by using the vehicle $V2$, which originally is also located in D . Figure 4 (a) sketches the plan Δ . Arguments from the actions have been removed for simplification purposes. A new problem is given with an almost identical situation as the one before. The only difference is that the only vehicle available is $V1$ at location A . Figure 4 (b) illustrates the steps of Δ that lead to

a minimal plan modification. The resulting plan is the following: (L₁) load the package *p1* into *V1*, (M₁) move *V1* to location *B*, (1) move *V1* to location *D*, (2) load *p2* into *V1*, (3) move *V1* to location *B*, (4) unload *p2* from *V1*, (L₃) load *p2* to *V1*, (M₃) move *V1* to location *C*, (UL₂) unload *p1* from *V1*, and (UL₃) unloading *p2* from *V1*. Steps (1)-(4) were generated by first-principles planning.

Case 1. If the instance of DerUCP uses FSS, BSS and PS refinement strategies and the form of replay is non-interleaved and non-eager, then a possible outcome is the plan at the bottom of Figure 3, in which several decisions from the original plan were not replayed even though they could have been used to obtain a solution. Therefore, the adaptation is not conservative. Figure 4 (c) illustrates the steps of Δ used to generate the solution. The minimal plan modification could potentially be obtained if interleaved and non-eager replay is used. The problem is that there is no guarantee that such an outcome will always be obtained. For example, using first-principles planning when the vehicle is at *B*, may lead to move to *D*, load *p2* and move directly to *C* to drop *p2*.

Case 2. In the same counterexample, if eager replay is used, more decisions are replayed. In particular, the following plan steps will be obtained from case replay: loading *p1* into *V1*, covering the route $A \rightarrow B \rightarrow C$ with *V1*, and dropping *p1* at *C*. First-principles planning can be used to complete this partial plan by moving *V1* to *D*, loading *p2* into *V1*, moving back to *C* and dropping *p2*. Figure 4 (d) illustrates the steps of Δ used to generate the solution.

Cases 3 and 4. Counterexamples can easily be constructed for these cases by modifying the one described above. We omit the details. \square

Discussion. In the above proof, we referred to some specific (and very simple) counter-examples. However, the same argument applies to any planning problem in which (1) more than one replay is possible and (2) the first replay (i.e., the one tried by DerUCP) doesn't produce the smallest possible modification of Δ . Since such situations are possible in most nontrivial planning domains, it is possible to construct an infinite number of counter-examples.

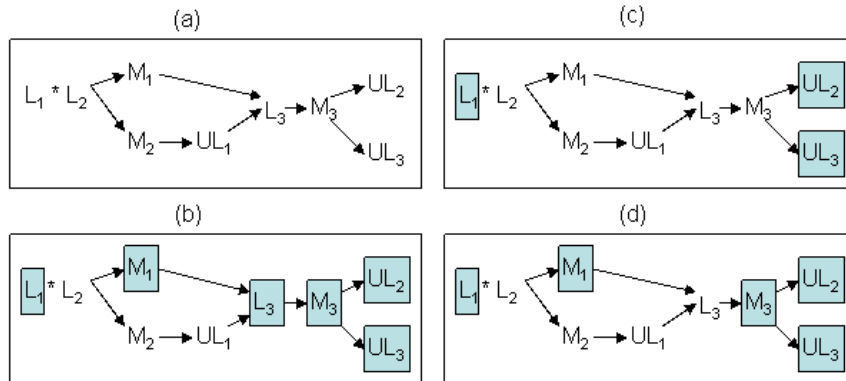


Figure 4: (a) Sketch of a plan Δ from Figure 2, generated by UCP. (b) Steps of Δ that lead to a minimal plan modification (boxed steps indicate steps that were

reused). (c) Steps of Δ used in the solution discussed in Case 1 of the proof. (d) Steps of Δ used in the solution discussed in Case 2 of the proof

The DerUCP algorithm presented in this paper covers all implementations of derivational replay that we know of. From the theorem and corollary, it follows that DerUCP does not satisfy a key assumption of Nebel and Koehler’s complexity results for plan reuse. Thus, their complexity results do not apply to it.

We can show that an infinite number of counter-examples as in Proposition 2 can be constructed. First, we define a **plan adaptation problem** as a pair (Π, Π') , where Π is a problem for which the solution is known and Π' is the problem we want to solve. For an arbitrary number n , let Π_n be the problem constructed as follows. Π_n is obtained by mimicking the problem depicted in Figure 1. Π_n has n locations D_1, \dots, D_n and two more locations B and C. In addition a package u_i and a truck t_i are located in each D_i . As with the problem in Figure 1, the goal is to relocate all packages into C. Let Δ_n be the plan that uses each truck t_i to relocate each package u_i in location B. Then t_1 is used to load all packages in B and relocate them into C. Now let a new problem Π_n' be given which is almost identical to Π_n , but in one of the locations D_i has no truck, only the package u_i . In this situation, DerUCP can produce non-minimal modifications for the plan adaptation problem (Π_n, Π_n') .

We conclude this section by demonstrating that for every plan adaptation problem, there is another plan adaptation problem for which DerUCP does not necessarily result in a minimal plan modification.

Proposition 3. For every adaptation problem (Π, Π') , there are one or more adaptation problems (Π_q, Π_q') for which DerUCP, using non-interleaved and eager replay and backward state-space refinement (BSS), does not follow a conservative adaptation strategy.

Proof. Let $\Pi = (Pr, O, I, G)$, $\Pi' = (Pr, O, I', G')$, and Δ be the plan solving Π . Let q be a new atom in Pr that is not mentioned in I, G, Δ, I' , or G' . Let p be a precondition of a plan step s_p in Δ . We define $\Pi_q = (Pr, O_q, I, G)$ and $\Pi_q' = (Pr, O_q, I', G_q')$ as follows: O_q has the same operators at O plus one, called op_q , without preconditions and having as effects q and $\neg p$. G_q' has the same goals as G' plus q . Under these conditions, s_p will need to be revised because in eager replay the derivational trace of Δ is replayed first. Thus, s_p will be added to the partial plan before applying op_q . Since DerUCP is performing backward state-space refinement, it will add a step s_q , applying op_q , before s_p , which makes it impossible to enhance the partial plan to obtain a solution. Therefore, all decisions made during replay from the point where s_p was added will need to be revised. But these decisions could have been preserved if s_q would have been added first before adding s_p to the plan. \square

In the above proof, note that it would be easy to construct additional counterexamples by choosing different values for p and q . We constructed the

adaptation for BSS refinement and non-interleaved and eager replay. Constructions can be made for the other variants as well.

6. Efficiency of DerUCP

In this section, we examine the efficiency of DerUCP by considering the size of its search space. This size is the same as the number of nodes that the planner would visit in the worst case.

To start out, we observe that for nearly every classical planning algorithm, it is easy to insert a simple test into the algorithm that will guarantee termination while preserving the algorithm's soundness and completeness.³ Thus, we can assume that the search space is finite. Let b be the maximum branching factor in the search space, and d be the maximum depth. If plan generation is done solely by first-principles planning (i.e., without case replay), then (see (Korf, 1987)) the number of nodes in the search space is

$$n = O(b^d). \tag{1}$$

In Figure 4, this search space is represented by the outermost triangle.

We divide the rest of the analysis into two sections. In the first one, we examine how the size of the search space is affected by replaying a single case. In the second one, we look at how the replay of multiple cases affects the size of the search space. In both situations, we examine best case, worst case and intermediate cases.

³ The argument is as follows. Every classical planning problem has finite state space, so let L be the size of that state space. In most classical planning algorithms, there is some notion of a partial plan. If a partial plan Δ contains more than L actions, then some of those actions are redundant, because any executable completion of Δ will generate at least one state more than once. Whenever such a plan is reached, most classical planners (STRIPS is an exception, because it is not complete) can safely backtrack because they will be guaranteed to find better plans along other paths in their search spaces.

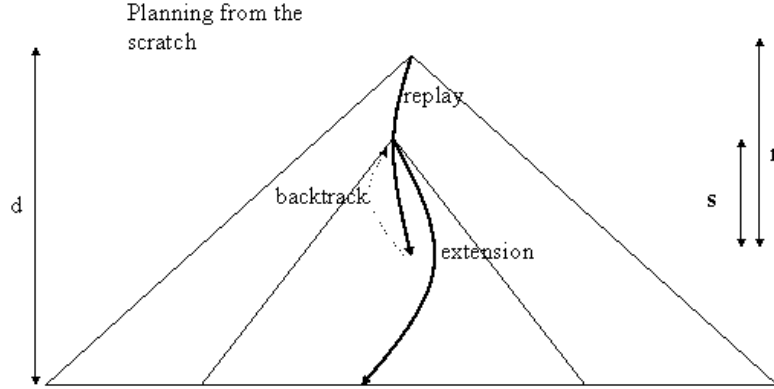


Figure 5. The search-space size for DerUCP with replay of a single case is $O(b^{d-r+s})$, where b is the branching factor, d is the depth of the solution, r is the number of nodes obtained from replay, and s is the number of nodes on the replay path that are backtracked.

6.1 Single Replay

In this section, we analyze the search space for DerUCP in the case where a single replay occurs. We will consider three cases for single replay. In each of them, we will consider the size of the planner’s search space (which is the same as the number of nodes that the planner would visit in the worst case).

Case I. The first case for single replay is where the replay occurs at the very beginning, before any first-principles planning is done. This analysis is an extension of a similar analysis for eager replay in Section 5 of (Ihrig & Kambhampati, 1996), which was made for plan-space planners. In this case, we can divide the execution of DerUCP into two phases. The first phase is the replay of one or more cases; in Figure 4, this corresponds to the solid arrow emanating from the apex of the outermost triangle. We let r be the number of nodes visited during the replay. The second phase is the use of first-principles planning after the replay has completed; this phase may include some backtracking over various decisions made during the first phase. In Figure 4, the dashed arrow represents this backtracking; we let s be the number of nodes on the replay path that are backtracked during the second phase. The search space during this phase, which is represented by the inner triangle in Figure 4, contains the following number of nodes:

$$n = O(b^{d-(r-s)}) = O(b^{d-r+s}). \quad (2)$$

The best case for Case I occurs when replay leads directly to the solution without any backtracking. This situation arises when the case solves the current problem. In this situation, $r = d$ and $s = 0$. This would imply $n = O(1)$. The worst case arises when all decisions made during case replay need to be revised. Since $r-s \geq 0$, this search space

is no larger than the original one (see Eq. (1)). The intermediate case occurs when $r-s > 0$ then the search space is exponentially smaller.

Case II. The second case for single replay is where the replay occurs after some of the first-principles planning has been done, and where the planner is doing either a breadth-first search or an iterative-deepening search. Let v be the node at which the replay begins, and e be v 's depth. The planner has already visited $O(b^e)$ nodes, and the number of nodes below v is $O(b^{d-e})$. By replacing d with $d-e$ in the analysis in the previous paragraph, we can conclude that the size of the search space searched by the planner *after* node v is $O(b^{d-e-r+s})$. Thus the total number of nodes in the planner's search space is

$$n = O(b^e + b^{d-e-r+s}). \quad (3)$$

As before, this is no larger than the size of the original search space (worst case), and may be exponentially smaller (intermediate case). The best case occurs when replay leads directly to the solution without any backtracking. In this situation, $r = d-e$ and $s = 0$. This would imply $n = O(b^e)$.

Case III. The third case for single replay is where the replay occurs after some of the first-principles planning has been done, and where the planner is doing a depth-first search. As before, Let v be the node at which the replay begins, and e be v 's depth. If the replay occurs on the leftmost branch, then the size of the search space is $O(b^{d-e-r+s})$. If the replay occurs on a branch other than the leftmost branch, then the planner has already visited $O(b^d)$ nodes, so the replay does not provide any significant help. In this case, the search-space size is

$$n = O(b^d). \quad (4)$$

6.2 Replay of Multiple Cases

We now consider the situation in which more than one case is replayed. When multiple cases are replayed, the search process contains more than one path that is guided by the derivational trace of the cases, Case₁, ..., Case_m. Let l_i be the number of refinements of Case_i that are retained for use in the final plan, as these guided refinements replace the node expansion of the search tree—the effect is similar to pruning the search space. Figure 6 shows a typical search space for DerUCP. The bold triangles are the regions of the search spaces explored by first principles. Other regions are essentially skipped by replay. Therefore, the search space size is described by:

$$O(b^{d - \sum l_i})$$

The best case occurs if the cases combined lead to a solution. In this situation: $\sum l_i = d$ and, therefore, the search space remaining is $O(0)$. In the worst case, $l_1 = l_2 = \dots = l_n = 0$. That is, all decisions taken during replay need to be revised. In such a situation the size of the search space is $O(b^d)$. In the intermediate case, Derivational Replay can potentially make an exponential reduction in the size of the search space visited

during adaptation. The reduction in the size of the exponent is proportional to the number of steps obtained during replay that remain when the solution plan is completed, i.e., it is proportional to the number of steps taken during replay that were not revised to obtain the solution plan.

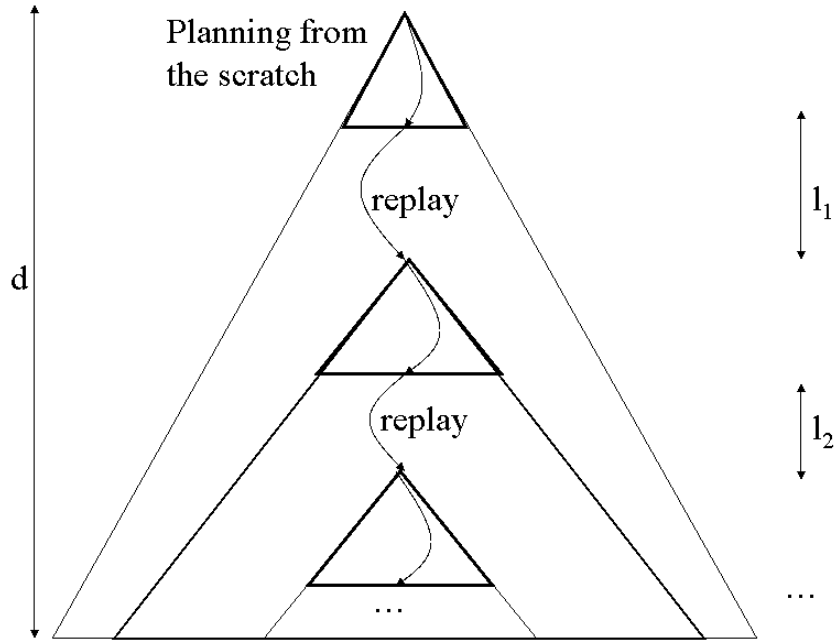


Figure 6. The search-space size for DerUCP with replay of multiple cases

6.3 Average-Case Performance

Sections 6.1 and 6.2 analyzed the size of the search space searched by the planning algorithm. This is a good measure of the planning algorithm’s worst-case performance, i.e., the case in which the planner explores the entire search space before finding a solution. But the results in Sections 6.1 and 6.2 do not show how much of the search space would actually be searched in the average case.

We know of no good way to analyze the average-case performance theoretically, because there is no clear definition for the “average case”; it depends largely on the problem being solved and on how the planning algorithm decides what path to search next. For example, even if derivational replay reduces the size of a planning algorithm’s search space, it is conceivable that for some planning problems and some

planning algorithms, the algorithm might search a larger part of this reduced search space than it would have searched if derivational replay had not been used.

For reasons such as the ones discussed above, average-case analyses of planning algorithms are typically done experimentally rather than theoretically. In all of the experimental studies that we know of, derivational replay has run significantly on the average faster than planning from scratch (Veloso, 1994; Ihrig & Kambhampati, 1996, Muñoz-Avila & Weberskirch, 1996). These studies were done in three different domains, were developed independently from one another, and all use some form of a random problem generation procedure.

7. Related Work

Failure Annotations Replay. The derivational traces in DerUCP indicate the path in the search tree that led to solution plans. However, it is feasible that when the solution plan was generated, decisions were made that didn't lead to a solution forcing the planner to backtrack on these decisions. Veloso observed that annotations about failed decisions could be added to the derivational traces (Veloso, 1994). During case replay, these annotations can be also replayed in the context of the new problem. If backtracking occurs on decisions made during case replay, the annotations can be used to avoid unnecessary backtracking. Prodigy/Analogy introduced these annotations for state-space planners (Veloso, 1994) and CAPlan/CbC introduced them for plan-space planners (Muñoz-Avila & Weberskirch, 1996).

Whether failure annotations are replayed or not is not related to the issue of the instances of DerUCP being non-conservative. The instances of DerUCP, including Prodigy/Analogy and CAPlan/CbC, are non-conservative because there is no guarantee that the largest possible number of steps of the selected plan will form part of the solution of the new problem.

Transformational Analogy. In transformational analogy, the solution plans, rather than the derivational traces, are reused to solve new problems. Transformational adaptation defines a collection of rules to transform a solution plan into a new problem. It does not require a first-principles planner to adjust the plan. Early case-based planners such as CHEF used domain-specific rules to perform the transformation of the solution plans (Hammond, 1989). However, Priar (Kambhampati and Hendler, 1992), SPA (Hanks & Weld, 1995), and Adjust-plan (Gerevini and Serina, 2000) specify domain-independent rules to transform the plan. Priar transforms hierarchical plans and SPA partial-order plans. Both transformation procedures are provably correct. Adjust-plan uses Graphplan as part of the transformation process and shows performance gains compared to Graphplan. Introspective reasoning has been used to adjust the adaptation rules (Fox and Leake, 1995). Despite these successes, no general framework encompassing different approaches for plan transformation has been developed. It is not known if such a

framework would fall under the worst-case scenario for plan adaptation from (Nebel & Koehler, 1995).

Case Retrieval/Case Library Indexing. There is a trade-off between the amount of time spent searching for adequate solution plans (cases) and the effort required to adapt these plans (Veloso, 1994). More generally, the utility problem also arises for case-based reasoning (Francis & Ram, 1995): as more cases are learned, the overall performance of the case-based reasoning system is expected to degrade. Indexing the case libraries is a crucial issue to improve the performance of case-based reasoning systems. In the context of case-based planning different approaches has been explored to index case libraries. Prodigy/Analogy performs goal-regression techniques to determine the features of the problem that are relevant with respect to a particular solution (Veloso, 1994). Priar ranks features from the initial state based on their contribution to the plan (Kambhampati, 1994). DerSNLP+EBL uses explanation-based learning techniques (Minton, 1988) when backtracking occurs over the partial plan obtained by case replay. These rules are used to avoid retrieving the same case in situations where such backtracking is known to occur (Ihrig & Kambhampati, 1997). CAPlan/CbC uses feature weighting techniques to learn the importance of the features in the cases over a period of several problem-solving episodes (Munoz-Avila & Weberskirch, 1996; Munoz-Avila, 2001).

8. Conclusions

In this paper, we have extended UCP, a well-known general model of classical AI planning, to provide DerUCP, a general model for derivational replay in classical planning. DerUCP covers all existing forms of derivational replay that we are aware of, including Interleaved and Eager Replay.

Our analysis of DerUCP resolves the difference between the previous theoretical and empirical studies. The former (Nebel & Koehler, 1995) suggested that plan adaptation is worse than planning from scratch, whereas the latter (Veloso, 1993; Ihrig & Kambhampati, 1996, Muñoz-Avila, 2001) suggested that derivational replay does better than planning from scratch. The definition of a conservative plan adaptation strategy used in the theoretical studies (Nebel & Koehler, 1995) does not hold for derivational replay, so their complexity results do not apply to derivational replay. The reason is that derivational replay, rather than looking for the replay that uses the largest possible number of steps, just uses the first replay that it happens to find—and if this replay turns out be minimal, this is purely coincidental. Thus, for each case in which derivational replay produces a minimal plan modification, there are many others where it does not.

Our analysis also provides information about the performance benefits that derivational replay can provide. We analyze three cases. First, when replay occurs before any first-principles planning is done. Second, when the replay occurs after some of the first-principles planning has been done, and where the planner is doing

either a breadth-first search or an iterative-deepening search. Third, when the replay occurs after some of the first-principles planning has been done, and where the planner is doing a depth-first search. For each case, we conclude that derivational replay will never make the search space larger than it would be without derivational replay, and that derivational replay can reduce the size of the search space by an exponential amount, even if the planner later backtracks over a portion of the replay.

These results suggest that what really matters is not derivational replay's worst-case performance, but instead its average-case performance. A theoretical analysis of average-case performance does not seem feasible because there is no clear notion of what an "average case" should be. But in experimental studies, derivational replay has run significantly on the average faster than planning from scratch (Veloso, 1994; Ihrig & Kambhampati, 1996, Muñoz-Avila & Weberskirch, 1996).

A topic for future work is to analyze similarity measures such as the foot-printing similarity metric (Veloso, 1994), the weighted foot-printing similarity metric (Muñoz-Avila, 2001) and use of EBL techniques to improve the retrieval process (Ihrig & Kambhampati, 1997). The goal is to obtain a model for analyzing the various retrieval techniques and study their effects on derivational replay. Another topic that we will study is adaptation with transformational analogy. Most domain-specific case-based planners such as CHEF (Hammond, 1987) use this adaptation technique. The only domain-independent implementation of transformational analogy is the system SPA (Hanks & Weld, 1994). It is not known, how transformational analogy compares to first-principles planning, nor how does derivational replay compares to transformational analogy from a complexity standpoint.

Acknowledgments

This work was supported in part by ISLE subcontract 0508268818 and NRL subcontract N00173-05-1-G034 to DARPA's Transfer Learning program, and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

References

- Au, T.C., Muñoz-Avila, H., & Nau, D.S. On the Complexity of Plan Adaptation by Derivational Analogy in a Universal Classical Planning Framework. In *Proceedings of the Sixth European Conference on Case-Based Reasoning (ECCBR-02)*. Springer. 2002.
- Bacchus, F. The AIPS-2000 Planning Competition. In *AI Magazine*. Vol. 3, n. 1. AAAI Press. 2001.
- Bergmann, R., and Wilke, W. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research* 3:53–118. 1995.

- Bhansali, S., and Harandi, M. T. Synthesis of UNIX programs using derivational analogy. *Machine Learning* 10. 1993.
- Blumenthal, B.. Empirical comparisons of some design replay algorithms. In *Proceedings Eighth National Conference on Artificial Intelligence*, pages 902–907, Boston, MA., MIT Press. 1990.
- Bylander, T. The Computational Complexity of Propositional STRIPS Planning, *Artificial Intelligence*, 69:165-204, 1994.
- Carbonell, J. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 2, pages 371–392. Morgan Kaufmann, 1986.
- Fox, S. & Leake, D. Using introspective reasoning to refine indexing. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, AAAI Press, 1995.
- Fikes, R., and Nilsson, N. J. 1972. Learning and executing generalized robots. *Artificial Intelligence* 3(4):251–288.
- Francis, A.G. Jr., and Rahm, A. A comparative utility analysis of case-based reasoning and control-rule learning systems. In *Proceedings of the European Conference on Machine Learning (ECML-95)*. Lecture Notes in Artificial Intelligence, Springer, 1995.
- Gerevini, A. and Serina, I. Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, AAAI Press, 2000.
- Ghallab, M. , Nau, D., and Traverso, P. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
- Hanks, S. and Weld, D. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2, 1995.
- Hammond, K. *Case-based planning: Viewing planning as a memory task*. Boston, MA: Academic Press, 1989.
- Ihrig, L. & Kambhampati, S. Design and implementation of a replay framework based on a partial order planner. In Weld, D., editor, *Proceedings of AAAI-96*. IOS Press, 1996.
- Ihrig, L. and Kambhampati, S. Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Journal of Artificial Intelligence Research*, 7:161–198, 1997.
- Kambhampati, S. and Srivastava B. Universal Classical Planner: An algorithm for unifying state-space and plan-space planning. In *Proceedings of the Third European Workshop on Planning (EWSP-95)*. 1995.
- Kambhampati, S., Ihrig, L., and Srivastava B. A Candidate Set based analysis of subgoal interactions in conjunctive goal planning. In *Proceedings of the Third International Conference on AI Planning Systems (AIPS-96)*. 1996.
- Kambhampati, S. Refinement planning as a unifying framework for plan synthesis *AI Magazine*, Vol 18. No. 2, Summer, 1997.
- Kambhampati, S. and Hendler, J. A Validation Structure Based Theory of Plan Modification and Reuse, *Artificial Intelligence*, 55(23) :193-258, 1992.
- Korf, R. E. Planning as search, a quantitative approach. *Artificial Intelligence*, 33:65-88, 1987.

Minton, S. *Learning effective search control knowledge: An explanation-based approach*. Technical report, Carnegie-Mellon University Department of Computer Science, 1988.

Muñoz-Avila, H. *Integrating Twofold Case Retrieval and Complete Decision Replay in CAPlan/CbC*. PhD Thesis, University of Kaiserslautern, 1998.

Muñoz-Avila, H. Case-Base Maintenance by Integrating Case Index Revision and Case Retention Policies in a Derivational Replay Framework. *Computational Intelligence*. Vol. 17, n. 2. Blackwell Publishers Inc. 2001

Muñoz-Avila, H. & Weberskirch F.: Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions. In: Proceedings of the Third International Conference on AI Planning Systems (AIPS-96), AAAI-Press, 1996.

Muñoz-Avila, H. & Weberskirch F.: *A Case Study on the Mergeability of cases with a Partial-Order Planner*. In S. Steel & R. Alami (Eds.): Recent Advances in AI Planning. Proceedings of ECP-97, Springer, 1997.

Long, D. and Fox, M. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20, 1–59, 2003.

Nebel, N. and Koehler, J. Plan reuse versus plan generation: a theoretical and empirical analysis, *Artificial Intelligence*, 76, 427–454, 1995.

Siekmann, J. H.. Unification Theory. *Journal of Symbolic Computation*, 7:207--274, 1989.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., and Blyte, J. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Theoretical and Experimental AI*, 7(1), 1995

Veloso, M. *Planning and learning by analogical reasoning*. Berlin: Springer-Verlag, 1994.

Veloso, M. Flexible strategy learning: Analogical replay of problem solving episodes, In *Proceedings of AAAI-94, the Twelfth National Conference on Artificial Intelligence*, pages 595-600, Seattle, WA. AAAI Press. 1994.

Veloso, M. M., and Carbonell, J. G. Towards scaling up machine learning: A case study with derivational analogy in PRODIGY. In Minton, S., ed., *Machine Learning Methods for Planning*. Morgan Kaufmann. 233–272. 1993.