On the Complexity of Plan Adaptation by Derivational Analogy in a Universal Classical Planning Framework

Tsz-Chiu Au[1], Héctor Muñoz-Avila[2], Dana S. Nau[1]

[1] Department of Computer Science and Institute for System Research
University of Maryland, College Park, MD 20742, USA
{chiu, nau}@cs.umd.edu
[2] Department of Computer Science and Engineering
Lehigh University, Bethlehem, PA 18015, USA
munoz@cse.lehigh.edu

Abstract. In this paper we present an algorithm called DerUCP, which can be regarded as a general model for plan adaptation using Derivational Analogy. Using DerUCP, we show that previous results on the complexity of plan adaptation do not apply to Derivational Analogy. We also show that Derivational Analogy can potentially produce exponential reductions in the size of the search space generated by a planning system.

1 Introduction

As reported in several independent experiments, case-based planners using Derivational Analogy have consistently outperformed the base-level, first-principles planner on which these case-based planners were constructed [1–3]. On the other hand, formal studies on the complexity of adaptation versus the complexity of first-principles planning seem to suggest that in the worst case, adaptation can be harder than planning from scratch if certain conditions on the adaptation strategy are satisfied [4]. These complexity results raise questions about the effectiveness of adaptation and case-based planning in general and Derivational Analogy in particular. In this paper we intend to clarify this apparent contradiction.

We take advantage of an algorithm called Universal Classical Planning (UCP) [5] that can be regarded as a general model of STRIPS-Style planners [6]. In this paper we formulate a general algorithm for Derivational Analogy called DerUCP that can be regarded as a general model of Derivational Analogy built on top of STRIPS-Style planners.

We analyze the results of [4] and examine the assumption that led to that paper's main result about plan modification being harder than planning from scratch. We show that adaptation by Derivational Analogy does not fall under the assumption, and thus the worst case analysis in [4] does not apply to adaptation with Derivational Analogy.

Furthermore, we show that Derivational Analogy can never make planning harder, and can potentially make planning much easier. More specifically, we show that if s_1 is the search space that can potentially be explored by an instance of DerUCP and s_2 is the search space that can potentially be explored by the corresponding instance of UCP, then s_1 is never any larger than s_2, and s_1 can be exponentially smaller than s_2.

This paper is organised as follows. Section 2 reviews the definition of UCP. Section 3 describes the DerUCP algorithm. Section 4 presents the complexity results. Finally, Section 5 analyzes the efficiency of DerUCP.

2 Universal Classical Planning

UCP is a generalized algorithm for classical planning that encompasses traditional planning approaches into a single framework [5]. UCP can be instantiated into a variety of planners, such as STRIPS-Style planners, SNLP and PRODIGY [5].

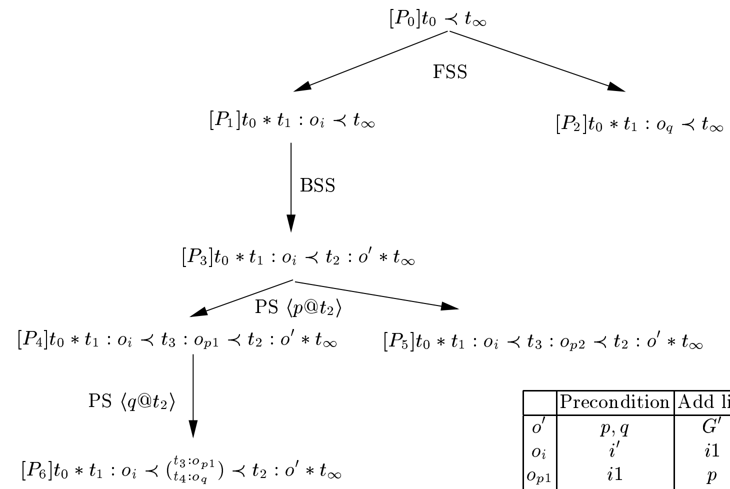
UCP operates on a *partial plan* P that represents some set of candidate solution plans $\mathcal{P}$. Formally, P is a 5-tuple $\langle \mathcal{T}, \mathcal{O}, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$, where $\mathcal{T}$ is the set of steps in the plan, $\mathcal{ST}$ is the symbol table which maps step names to actions, $\mathcal{O}$ is the partial ordering relation over $\mathcal{T}$ (the corresponding relational operator is $\prec$), $\mathcal{B}$ is a set of codesignation and non-codesignation constraints on the variables in the preconditions and post-conditions of the operators, and $\mathcal{L}$ is a set of auxiliary constraints. There are three kinds of auxiliary constraints. (1) An interval preservation constraint (IPC) is denoted as $(t \overset{p}{-} t')$, which means the condition p must be preserved between the operators corresponding to steps t and t'. (2) A point truth constraint (PTC) is a 2-tuple $\langle p@t \rangle$, which ensures condition p must be true before step t. (3) A contiguity constraint $t_i * t_j$ does not allow any step between t_i and t_j. These constraints restrict the ground linearization of the steps in the set $\mathcal{P}$ of candidate solution plans represented by P.

UCP begins with a *null plan* $t_0 \prec t_\infty$, where t_0 is the initial step which has the initial state as its postcondition, and t_∞ is the final step which has the goal state as its precondition. UCP's objective is to find a *solution*, i.e, a partial plan that contains a ground linearization of steps which achieves the goal state from the initial state. UCP tries to find a solution by performing *refinements*, i.e., by adding new steps and constraints to the partial plan. A refinement may eliminate some of the candidate solution plans, in which case the refinement is *progressive*, or it may not eliminate any candidate solution plans, in which case the refinement is *nonprogressive*.

The *header* of P consists of the step t_0 plus all steps t_i such that $i > 0$ and $t_{i-1} * t_i$. The *head fringe* is the set of all steps that can come immediately after the header. The *trailer* and the *tail fringe* are defined analogously, but with respect to the final step of the plan rather than the initial step.

In UCP, the progressive refinements are classified into three types: *forward state-space (FSS) refinements* add constraints or new steps at the head fringe of the plan, *backward state-space (BSS) refinements* add constraints or new steps at the tail fringe of the plan, and *plan-space (PS) refinements* add constraints or new steps somewhere between the head fringe and the tail fringe. The non-progressive refinements are also classified into three types: *refine-plan-pre-order* adds ordering constraints between steps, *refine-plan-pre-position* adds contiguous constraints between steps, and *refine-plan-pre-satisfy* resolves conflicts in the partial plan.

The following example, which is taken from page 19 of [5], shows how various refinements work in UCP. UCP begins with a null plan $t_0 \prec t_\infty$, where t_0 is the initial step which has the initial state $\{i'\}$ as its postcondition, and t_∞ is the final step which has the goal state $\{G'\}$ as its precondition. In the figure, each node represents a partial plan, the label below the node represents the particular refinement strategy that UCP is using at this node, and the set of branches emanating from the node represents the set of alternative refinements produced by the refinement strategy. UCP chooses among these refinements nondeterministically. In this particular example, the algorithm introduces an FSS refinement, a BSS refinement, and two PS refinements, and finally produces a partial plan which contains a ground linearization of steps achieving the goal. We will generate a derivational trace for this example in Section 3.1.



	Precondition	Add list	Delete List
o'	p, q	G'	$-$
o_i	i'	$i1$	$-$
o_{p1}	$i1$	p	$-$
o_{p2}	$i2$	p	$-$
o_q	$-$	q	p

Fig. 1. An example of plan generation with UCP in [5]

3 DerUCP: Derivational Analogy in UCP

Derivational Analogy is a widely used adaptation method that has been the subject of frequent studies [1, 2, 7–12]. In Derivational Analogy cases contain the *derivational trace*, the sequence of decisions made to obtain a plan, rather than the plan itself. Typically in a problem-solving session, part of a solution plan is obtained through case replay of the derivational traces stored in retrieved cases, and part through first-principles planning.

We will now formulate an algorithm that we will call DerUCP, for performing Derivational Analogy on top of UCP. DerUCP serves as a general model for Derivational Analogy in STRIPS style planning since its instantiations include derivational analogy case-based planners such as Prodigy/Analogy [1], CAPlan/CbC [13] and DerSNLP [11].

3.1 Derivational Trace

Suppose we run an instantiation of UCP on a planning problem, and we keep a record of the sequence of choices it made at each of the nondeterministic choice points. This record, which called a *derivational trace*, consists of a sequence of *decision records*, each of which tells the particular choice that was made at one of UCP's nondeterministic choice points. The decision record gives the refinement strategy (such as FSS or PS), the *refinement goal* i.e., what portion of the partial plan is relevant for applying the refinement strategy, and the *decision* i.e., which particular refinement was chosen from among the alternative refinements produced by the refinement strategy.

More specifically, here is the information that needs to go into the decision records for the different kinds of refinements that UCP can make:

1. Decision record for a forward state-space refinement:
   - Refinement goal: the head-state s at the time the refinement was applied.
   - Decision: what step t was chosen (out of the set of all steps whose preconditions are satisfied by s), and whether t was a new step or a previously existing step.

   If the decision record says that t was a new step, then this means that UCP added t to the partial plan, and added a contiguity constraint between the head step and t. If the decision record says that t was an existing step, then this means that UCP simply added a contiguity constraint between the head step and t.
2. Decision record for a backward state-space refinement:
   - Refinement goal: the tail-state s at the time the refinement was applied.
   - Decision: what step t was chosen (out of the set of all steps that do not delete any condition in s and achieve at least one condition in s), and whether it was a new step or a previously existing step.

   If the decision record says that t was a new step, then this means that UCP added t to the partial plan, and added a contiguity constraint between t and the tail step. If the decision record says that t was an existing step, then this means that UCP simply added a contiguity constraint between t and the tail step.

3. Decision record for a plan-space refinements
   - Refinement goal: a point truth constraint $\langle p@t' \rangle$.
   - Decision: what step t was chosen (out of the set of all steps that can establish the point truth constraint $\langle p@t' \rangle$), whether it was a new or existing step, and whether an IPC was added to the plan.

   If the decision record says t was a new step, it means that UCP added it to the plan, and in any case it means that UCP added constraints to require that $t \prec t'$ and to prevent any step from coming between t and t' and deleting p.
4. Decision record for a refine-plan-pre-order refinement:
   - Refinement goal: a pair of unordered steps t_1 and t_2.
   - Decision: Whichever of the following two plans was chosen: $P + (t_1 \prec t_2)$ or $P + (t_1 \not\prec t_2)$.
5. Decision record for a refine-plan-pre-position refinement:
   - Refinement goal: a pair of non-contiguous steps t_1 and t_2.
   - Decision: Whichever of the following two plans was chosen: $P + (t_1 * t_2)$ or $P + (t_1 \not* t_2)$.
6. Decision record for a refine-plan-pre-satisfy refinement:
   - Refinement goal: an auxiliary constraint C and a step t_3 in conflict with C.
   - Decision: the way in which P was refined in order to make C hold in every possible ground linearization of P. For example, if C is an IPC $(t_1 \overset{p}{-} t_2)$, then the possible refinements of P may be any plans of the form $P + (t_3 \prec t_1) \vee (t_2 \prec t_3)$ or $P + \pi^p_{t_3}@t_3$, where t_3 is a step having an effect that unifies with $\neg p$, and $\pi^p_{t_3}$ is as described in [5].

These decision records take all search choice points in the UCP algorithm into account. As an example of these decision records, Table 1 shows a derivational trace for the example in Section 2.

Note that 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 DerUCP Algorithm

The DerUCP algorithm extends the UCP planning algorithm by adding the *derivational replay* before the refinement steps. This section describes only the derivational replay mechanism, which relies on the replay step and the procedure

Table 1. A derivational trace for the example in Section 2.

Step: 1 Type: forward state-space refinement Refinement goal: the head-state $\{i'\}$ Decision: a new step with operator o_i.
Step: 2 Type: backward state-space refinement Refinement goal: the tail-state. $\{G'\}$ Decision: a new step with operator o'.
Step: 3 Type: plan-space refinement Refinement goal: the point truth constraint $\langle p@t_2 \rangle$. Decision: a new step with operator o_{p1}
Step: 4 Type: plan-space refinement Refinement goal: the point truth constraint $\langle q@t_2 \rangle$. Decision: a new step with operator o_q
Step: 5 Type: refine-plan-pre-satisfy Refinement goal: IPC $(t_3 \stackrel{p}{-} t_2)$ and t_4 Decision: the plan $P + t_4 \prec t_3$.

`Replay`. For a precise description of other parts of the UCP algorithm, please refer to [5].

Figure 2 shows the pseudocode of the DerUCP algorithm. The DerUCP algorithm is a recursive search procedure in which partial plans are refined by additions of new steps and constraints in the refinement steps in each iteration until a solution is found. Instead of allowing nondeterministic choice in the algorithm description of UCP, DerUCP explicitly maintains a priority list which stores the pending partial plans. The priority list plays a role similar to the open list in the $A*$ search algorithm. At the beginning, the priority list containing an empty partial plan is provided, together with a case library. The DerUCP algorithm first picks up one of the partial plans from the priority list according to a choice function (Step 0), which depends on the particular UCP instance. If the partial plan contains a solution, the algorithm returns it and terminates (Step 1). Otherwise, the algorithm proceeds to the replay step, which selects a case achieving any refinement goal and replays it (Step 2). The details of `Replay` procedure are discussed below. After the replay step, the partial plan, P, is refined. This refined plan is used in the remaining steps. Note that the steps after the replay step, i.e., the progressive refinement, the non-progressive refinement, and consistency check, are optional, and therefore they can be skipped (Steps 3-5). Finally, all the partial plans generated by the refinement steps are inserted into the priority list and DerUCP is invoked recursively (Step 6).

In the replay step, the set of refinement goals of the current partial plan is computed (Figure 3). The possible refinement goals are the refinement goals of

procedure `DerUCP`(PL, CL)

Inputs: PL - a priority list

CL - a case library

begin

0. Plan Selection

Select one partial plan P from PL (by a choice function).
Remove P from PL.

1. Termination Check

If P contains a ground operator sequence that solves the problem, returns it and terminates.

2. Replay

Construct a list of refinement goals G for P.
Let $S \subseteq CL$ be the set of cases which achieve some refinement goals in G

if S is not empty **then**

Select one case C from S according to a metric
$P :=$ `Replay`(P, G, C, PL)
$PL := PL \cup \{P\}$

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. $PL := PL \cup L'$

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. $PL := PL \cup L''$

5. (optional) Consistency Check

For each P'' in L'', select zero or more of:

If the partial plan P'' is inconsistent or non-minimal, remove P'' from L''.

6. Recursive Invocation:

Call `DerUCP`(PL, CL)

end

Fig. 2. The DerUCP algorithm

procedure `Replay`(P, G, C, PL)
begin
1. **if** C is null **then** return P
2. Let g_c be the refinement goal of $first(C)$
3. **if** $\exists g \in G$ that match g_c **then**
4.   Let d_c be the decision for g_c in C
5.   Let cs be the set of applicable refinements for g
6.   **if** $\exists r \in cs$ that $matches(decision(r), d_c)$ **then**
7.   Apply the refinement r to P to obtain P'
8.   Apply the refinements in $cs - \{r\}$ to P to obtain a set of partial plans PS
9.   Append PS to PL with a lower priority
10.   Construct a list of refinement goals G' for P' (G' can be obtained by modifying G)
11.   Select one of the following:
            return `Replay`$(P', G', next(C), PL)$, or
            return P'

else

12.   Select one of the following:
        return `Replay`$(P, G, next(C), PL)$, or
        return P'

end

Fig. 3. The `Replay` Procedure

all six decision records described in Section 3.1. The replay of the current case continues only if the next goal in the case g_c matches the refinement goal g (Steps 2-3). Each decision record in the derivational trace guides the selection of 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 each iteration DerUCP makes a nondeterministic choice to continue with the next decision record or stop the replay process (Step 11).

Eager and Interleaved Replay. DerUCP supports the two variants of Derivational Replay: eager and interleaved replay. In *eager replay*, case replay is done first and then the partial plan obtained is completed by a first principles planner. It is called "eager" since each of the selected cases is replayed as long as possible. That is, in steps 11 and 12 of the `Replay` procedure, the recursive call is always made. An example of a system performing eager replay is DerSNLP [11]. In *Interleaved Replay*, case replay may be interleaved with first-principles planning and it is not eager. An example of a system implementing the interleaved approach is Prodigy/Analogy. Interleaving is covered in DerUCP in two places: first, in the recursive call of Step 6 of the DerUCP algorithm, interleaving between first-principles planning and case replay is secured. Second, the nondeterministic choices of steps 11 and 12 in the `Replay` procedure ensure that the cases need not be eagerly replayed. In Prodigy/Analogy heuristics are used to decide whether to continue replay of the current case or not.

4 Complexity Analysis of Plan Adaptation

We now analyze the main result in [4] that sometimes has been quoted as proof that plan adaptation is harder than planning from scratch. First the standard definitions of a planning problem, i.e., "planning from scratch", is given. [14]

Definition 1. *An instance of propositional STRIPS planning is denoted by a tuple* $\langle P, O, I, G \rangle$*, where:*

- *P is a finite set of ground atoms. Let L be the corresponding set of literals:* $L = P \cup \{\neg p : p \in P\}$.
- *O is a finite set of operators of the form $Pre \Rightarrow Post$, where $Pre \subseteq L$ is 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.*
- *$I \subseteq P$ is the initial state.*
- *$G \subseteq L$ is the goal.*

A state S is a subset of P, indicating that $p \in P$ 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. *PLAN-EXISTENCE (Π) Given an instance of the planning problem $\Pi = \langle P, O, I, G \rangle$, does there exist a plan Δ that solves Π?*

The following is the definition of plan adaptation used in [4]:

Definition 3. *MODSAT* (Π_1, Δ, Π, k)
Given the instance of the planning problem $\Pi_1 = \langle P, O, I_1, G_1 \rangle$, and a plan Δ that solves the instance $\Pi = \langle P, O, I, G \rangle$, does there exist a plan Δ_1 that solves Π_1 and contains a subplan of Δ of at least length k *?*

This definition states what the authors of [4] call a *conservative* plan modification strategy in the sense that it is trying to reuse as much of the starting plan Δ to solve the new plan. The main conclusion from [4] is that in the worst case MODSAT is harder than PLAN-EXISTENCE. That is, plan adaptation is harder than planning from scratch with a conservative strategy plan modification strategy .

In [4], Priar [15] and SPA [16] are cited as examples of systems performing this conservative adaptation strategy. Interestingly, neither of these systems performs plan adaptation by Derivational Analogy; cases in these systems contains plans and provably correct plan transformation rules are used to perform adaptation.

So the question remains: does adaptation by derivational analogy perform a conservative plan modification strategy in the sense of the MODSAT definition or not? If it does, then the worst case analysis applies and thus the empirical results reported about case-based planners such as Prodigy/Analogy, DerSNLP and CAPlan/CbC outperforming their respective underlying first-principles planners would have been mere accidents. As it turns out, *adaptation by derivational analogy does not perform a conservative adaptation strategy in the sense of MODSAT*.

The key to our analysis lies in Step 11 of the `Replay` procedure. Clearly, if the algorithm stops replaying the derivational trace rather than executing the recursive call, the adaptation is not conservative as the remaining parts of the derivational trace are not taken into account. This typically happens in Interleaved Replay. But what would happen if the recursive call always occurs, as happens in Eager Replay? The answer to this question can be seen from step 6 of the `Replay` procedure. Step 6 stops the replay process of a derivational trace if the current decision record is not applicable. At this point all of the remaining decision records are ignored. This is not a conservative strategy; in this situation, a conservative strategy would try to fix the impasse, for example by adding a few plan steps or revising the previous decision record, and then continuing with the remaining decision records.

This can be easily illustrated with an example from the *logistics* domain [1], in which 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. Figure 4 shows a possible configuration. There are four locations A, B, C and D. A truck T1 and a package P1 are in location A and another truck T2 and another package P2 are in location D. Suppose that our goal is to relocate both packages in location C. The arrows in Figure 4 show the paths followed by the trucks to achieve this problem. The first truck, T1, loads P1 and drives to B. Meanwhile, the second truck, T2, loads P2 and also drives to B. Once in B, P2 is unloaded from T2 and loaded into T1. T1 continues to C where both packages are dropped. Now, suppose that the derivational trace of this plan is stored in a case and that our particular instance of UCP performs only forward state-space refinement (i.e., steps can only be added to the head fringe of partial plan).

Let us suppose that a new problem is given which is almost identical to our previous example. The only difference is that T2 is not available. The corresponding instance of DerUCP will start replaying the first part of the plan in which P1 is loaded into T1 and T1 drives from A to B. At this point the replay fails since P2 is not in B (to be precise, the condition at Step 6 of the `Replay` procedure fails). DerUCP either continues by planning from scratch or by using a case that can continue solving this problem. This is not a conservative strategy in the sense of the MODSAT definition; a conservative strategy will drive T1 to D pick the second package, drive back to B and continue reusing the rest of the case. DerUCP never "continues" replay from a point at which a failure occurs because the replay step is always done from the beginning of the deriva-

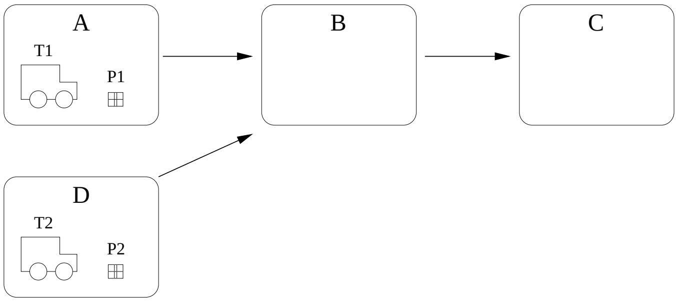


Fig. 4. A simple planning problem from the Logistics domain

tional trace. Thus, DerUCP would ignore the rest of the derivational trace even though most of it could have been used. One could constructs a similar example for partial order planning.

5 Efficiency of DerUCP

The analysis of the efficiency of DerUCP is divided into two parts. We first analyze the efficiency of the replay of one case, and then extend the analysis to the replay of multiple cases.

5.1 Replay of a Single Case

We consider the efficiency of DerUCP with exactly one case being replayed during the whole planning process. In other words, the `Replay` procedure is invoked once only. This analysis is an extension of a similar analysis for eager replay in Section 5 of [2].

Theorem 1. *Suppose exactly one case is replayed in DerUCP. Furthermore:*

- *Let the branching factor b be the average number of partial plans that can be generated from a given plan by the application of a single refinement.*
- *Let the depth of the solution d be the average length of the solution path from the initial (null) plan to a solution.*
- *Let l_s be the number of nodes before the replay begins.*
- *Let l_c be the number of nodes visited during the replay.*
- *Let l_b be the number of nodes on the replay path that are backtracked.*

Then the search space size is $O(b^{d-l_1})$, where $l_1 = l_c - l_b$

Proof. When only one case is replayed in the whole planning process, we can divide the execution of DerUCP into four phases: planning from scratch before the call to `Replay` procedure, the replay of the case, backtracking over various decisions suggested by the case, and finally using first-principles planning to extend the resulting partial plan to produce the final solution.

Figure 5 shows a possible exploration of the search space by DerUCP. The solid arrows represent the paths traversed by means of first-principles planning. The dashed arrow refers to the path due to derivational replay. The major difference between the solid arrows and the dashed arrow is that the searching on the solid arrows is possibly over the whole search tree which has size exponential to the length of the path. The searching on the dashed arrows is a traversal of a single branch of the search tree, which is guided by the decisions encoded in the derivational trace of the retrieved case. Therefore, the dashed arrow can be seen as a shortcut that enables the searching process to jump from one point in the search space to the other. Only the regions of the search space bounded by the two bold triangles are explored by first principles. Thus, a total of $l_c - l_b$ node expansions of the search tree are skipped, and the total search space size is $O(b^{d-(l_c-l_b)})$. □

If the planning is done by merely planning from scratch without derivational replay, then the search space size is $O(b^d)$ (see [17]). In the worst case, this is exponentially larger than the $O(b^{d-(l_c-l_b)})$ search space size of DerUCP as stated in Theorem 1, since $l_c - l_b \geq 0$.

5.2 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. Since the paths may interleave with each other in a complicated way, it is hard to identify which part of the search space is skipped. To simplify the analysis, consider only replays in which some refinements are retained 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 $O(b^{d-\sum l_i})$, where each l_i is the number of refinements that occur between the $(i-1)$'th and i'th bold triangles. By generalizing this argument, we get:

Theorem 2. *If two or more cases are replayed in DerUCP, then the search space size is $O(b^{d-\sum l_i})$, where each l_i is the number of refinements during the i'th replay that are retained for use in the final plan.*

Theorem 2 states that whenever some refinements suggested by any case are retained in the final plan, the search space size can be reduced.

These results show that Derivational Analogy 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. The worst-case scenario occurs when all decisions taken during replay need to be revised. In such a situation the size of the search

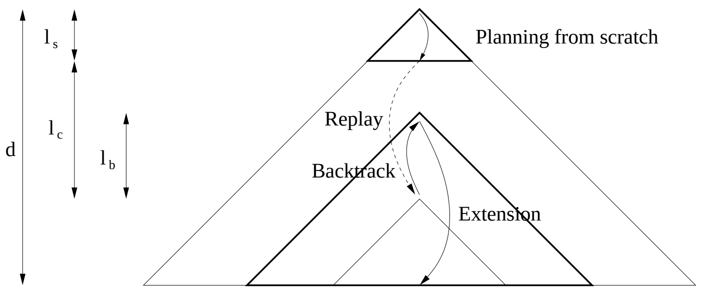


Fig. 5. The search-space size for DerUCP with one call to `Replay` is $O(b^{d-l_1})$, where b is the branching factor, d is the depth of the solution, l_s is the number of nodes before the replay begins, l_c is the number of nodes visited during the replay, l_b is the number of nodes on the replay path that are backtracked, and $l_1 = l_c - l_b$.

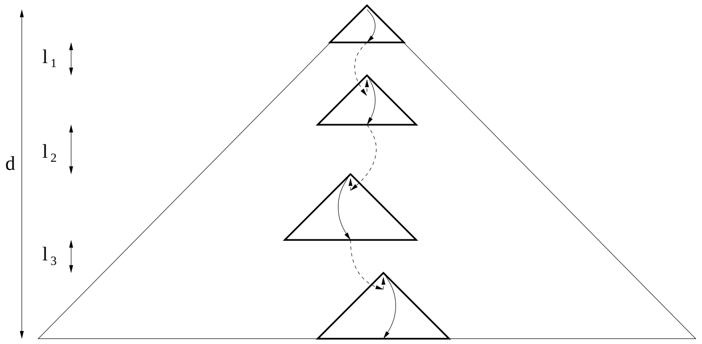


Fig. 6. The search-space size for DerUCP with multiple calls to `Replay` is $O(b^{d-\sum l_i})$, where each l_i is the number of refinements during the i'th replay that are retained for use in the final plan.

space is $O(b^d)$ since $l_b = l_d$ in Theorem 1. This worst-case search-space size is the same as the search-space size generated by the underlying planner. This proves the following theorem:

Theorem 3. *In the worst case, the size of the search space potentially generated by DerUCP is $O(b^d)$, i.e., the same as the size of the search space potentially generated by UCP.*

6 Conclusion

This paper describes the DerUCP algorithm, which is a general model of plan adaptation using Derivational Analogy. DerUCP is an extension of the well known UCP algorithm [5], which is a general model for classical AI planning. DerUCP covers all existing forms of Derivational Analogy that we are aware of, including Interleaved and Eager Replay.

Our analysis of DerUCP resolves the difference between theoretical results [4] suggesting that plan adaptation is worse than planning from scratch and empirical results [1–3] suggesting that Derivational Analogy does better than planning from scratch. In particular, we have shown the following:

– The conservative plan adaptation strategy as defined in [4] does not hold for Derivational Analogy. Thus, the worst-case complexity result in [4] about plan adaptation being harder than planning from scratch does not apply to Derivational Analogy.
– If we compare the size of the search space for any instance of DerUCP with the size of the search space for the corresponding instance of UCP, the DerUCP search space is no larger—and can potentially be exponentially smaller—than the UCP search space.

The amount of reduction is directly proportional to the number of steps obtained through case replay that remain after the solution plan is completed. Thus, performance improvements depend on the retrieval method being able to select cases that require fewer revisions. This is precisely what instances of DerUCP such as Prodigy/Analogy, DerSNLP and CAPlan/CbC do; these systems implement sophisticated retrieval and indexing techniques to improve the accuracy of the retrieval and reduce backtracking on the replayed steps.

Acknowledgments

This work was supported in part by the following grants, contracts, and awards: Air Force Research Laboratory F306029910013 and F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, and the Office of Research at Lehigh University. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

1. Veloso, M., Carbonell, J.: Derivational analogy in PRODIGY: Automating case acquisition, Storage and Utilization. Machine Learning (1993) 249–278
2. Ihrig, L., Kambhampati, S.: Plan-space vs. State-space planning in reuse and replay. Technical report, Arizona State University (1996)
3. Muñoz-Avila, H.: Case-Base Maintenance by Integrating Case Index Revision and Case Retention Policies in a Derivational Replay Framework. Computational Intelligence **17** (2001)
4. Nebel, B., Koehler, J.: Plan reuse versus plan generation: a theoretical and empirical analysis. Artificial Intelligence **76** (1995) 427–454
5. Kambhampati, S., Srivastava, B.: Unifying Classical Planning Approaches. Technical report, Arizona State University (1996)
6. Fikes, R., Hart, P., Nilsson, N.: Learning and executing generalized robot plans. Artificial Intelligence **3** (1972) 251–288
7. Carbonell, J.G.: Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. Machine Learning (1986)

8. Bhansali, S., IIarandi, M.T.: When (not) to Use Derivational Analogy: Lessons Learned Using APU. In Aha, D., ed.: Proceeding of AAAI-94 Workshop: Case-based Reasoning. (1994)
9. Blumenthal, B., Porter, B.: Analysis and Empirical Studies of Derivational Analogy. Artificial Intelligence **67** (1994) 287–328
10. Finn, D., Slattery, S., Cunningham, P.: Modelling of Engineering Thermal Problems - An implementation using CBR with Derivational Analogy. In: Proceedings of EWCBR'93, Springer-Verlag (1993)
11. Ihrig, L., Kambhampati, S.: Derivation Replay for Partial-Order Planning. AAAI-1994 (1994)
12. Muñoz-Avila, H., Weberskirch, F.: Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions. Proc. 3rd Int. Conference on AI Planning Systems (AIPS-96) (1996)
13. Muñoz-Avila, H., Paulokat, J., Wess, S.: Controlling Nonlinear Hierarchical Planning by Case Replay. In: Proceedings the 2nd European Workshop on Case-Based Reasoning (EWCBR-94). (1994) 195–203
14. Bylander, T.: The Computational Complexity of Propositional STRIPS Planning. Artificial Intelligence **69** (1994) 165–204
15. Kambhampati, S.: Exploiting causal structure to control retrieval and refitting during plan reuse. Computational Intelligence **10** (1994) 213–244
16. Hanks, S., Weld, D.S.: A Domain-Independent Algorithm for Plan Adaptation. Journal of Artificial Intelligence Research **2** (1995) 319–360
17. Korf, R.: Planning as Search: A Quantitative Approach. Artificial Intelligence **33** (1987) 65–88