Automatic Storage and Indexing of Plan Derivations based on Replay Failures

Laurie Ihrig & Subbarao Kambhampati*

Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406
laurie.ihrig,rao@asu.edu

Abstract

When a case-based planner is retrieving a previous case in preparation for solving a new similar problem, it is often not aware of all of the implicit features of the new problem situation which determine if a particular case may be successfully applied. This means that some cases may fail to improve the planner's performance. By detecting and explaining these case failures as they occur, retrieval may be improved incrementally. In this paper we provide a definition of case failure for the case-based planner, DERSNLP (derivation replay in SNLP), which solves new problems by replaying its previous plan derivations. We provide explanation-based learning (EBL) techniques for detecting and constructing the reasons for the case failure. We also describe how the case library is organized so as to incorporate this failure information as it is produced. Finally we present an empirical study which demonstrates the effectiveness of this approach in improving the performance of DERSNLP.

1 Introduction

Case-based planning provides significant performance improvements over generative planning when the planner is solving a series of similar problems, and when it has an adequate theory of problem similarity [3; 4; 5; 8; 9]. One approach to case-based planning is to store plan derivations which are then used as guidance when solving new similar problems [2; 9]. Recently we adapted this approach, called *derivational replay*, to improve the performance of the partial-order planner, SNLP [4]. Although we found that replay tends to improve overall performance, its effectiveness on any one planning problem depends on retrieving an appropriate case for that problem. Often the case-based planner is not aware of the implicit features of the new problem situation which determine if a certain case is applicable.

Earlier work in case-based planning has retrieved cases through the use of a static similarity metric which considers the goals that were achieved in the previous case as well as the features of the initial state which contributed to their achievement [6; 9; 4]. If these goals are also input goals in the new problem situation and the initial state conditions are also present then the case is retrieved. Since the new problem usually contains extra goals, the planner must extend the previous case by adding further constraints to the plan (including plan steps and step orderings). Sometimes straightforward extension is not possible and the planner may have to backtrack over the old plan and retract some constraints from it. In the current work we treat such instances as indicative of a case failure. We provide a framework by which a planner may learn from the case failures that it encounters and thereby improve its case retrieval.

We present the derivation replay framework, DERSNLP+EBL, which extends DERSNLP, a replay system for a partial-order planner, by incorporating explanation-based learning (EBL) techniques for detecting and explaining analytical failures in the planner's search space. These include a method for forming explanations of plan failures in terms of their inconsistent constraints, and regressing these explanations over the planning decisions in the failing search paths [7]. The failure reasons are then used to annotate the case to constrain its future retrieval. We evaluate the effectiveness of using case failure information in an empirical study which compares the performance of DERSNLP on replay of cases both *with and without* learning from case failure.

The rest of the paper is organized as follows. In Section 2, we first describe DERSNLP which implements our approach to derivation replay in the partial-order planner, SNLP, and discuss a definition of case failure for DERSNLP. Then, in Section 3, we describe the explanation-based learning techniques that were developed in [7], including the construction of failure explanations, and their regression up the failing search paths. We then show how reasons for case failure are formed using these techniques, and how case failure reasons are used to refine the labeling of cases in the library. Section 4 provides an example of the construction of a case failure reason. Finally, in Section 5, we describe an empirical study demonstrating the relative effectiveness of case retrieval when failure information is utilized.

2 A Definition of Case Failure for DERSNLP

DERSNLP is a case-based planning system that replays previous plan derivations [4]. Figure 1 illustrates the replay of a derivation trace. Whenever a new problem is attempted,

*This research is supported in part by NSF research initiation award (RIA) IRI-9210997, NSF young investigator award (NYI) IRI-9457634, and ARPA/Rome Laboratory planning initiative grant F30602-93-C-0039. Thanks to Suresh Katukam and Biplav Srivastava for helpful comments.

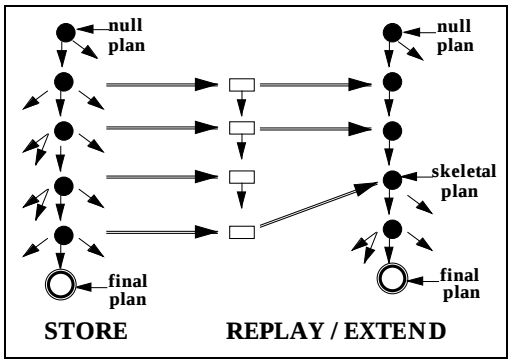

Figure 1: *Schematic characterization of derivation storage and replay. Each time that a plan is derived, the decisions contained in the plan derivation (shown as filled circles to the left of the figure) are stored as a sequence of instructions (shown as open rectangles) which are used to guide the new search process.*

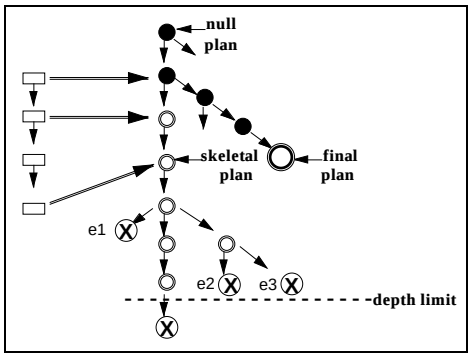

Figure 2: *A replay failure is indicated when the skeletal plan has to be backtracked over in order to reach a solution. In the figure, a solution to the new problem could be reached only by backtracking over the skeletal plan, which now lies outside the new plan derivation (shown as filled circles).*

and a solution is achieved, a trace of the decisions that fall on the derivation path leading from the root of the search tree to the final plan in the leaf node is stored in the case library. Later, when a similar problem is attempted this trace is used as a sequence of instructions to guide the new search process. DERSNLP employs an *eager* replay strategy. With this strategy, search control is shifted to the series of instructions provided by the previous derivation, and is returned to from-scratch planning only after all of the valid instructions in the trace have been replayed [4]. This means that the plan which is produced through replay, called the *skeletal plan*, contains all of the constraints that were added on the guidance of the previous trace. When the skeletal plan contains open conditions relating to extra goals not covered by the earlier case, further planning effort is required to extend this plan into a solution for the new problem.

In the current work *we define replay success and failure in terms of the skeletal plan. Replay is considered to fail if the skeletal plan which contains all of the constraints prescribed by the old case cannot be extended by the addition of further constraints into a solution for the new problem* (see Figure 2). In such instances, the planner must first explore the failing subtree underneath the skeletal plan, then backtrack over the replayed portion of the search path to find a solution. If a solution is found, the final derivation path which leads from the root of the search tree to the final plan in the leaf node does not contain the skeletal plan. The new derivation path thus avoids (or *repairs*) the failure encountered in replaying the old case.

3 Learning from Case Failures using EBL

Having defined case failure, we are now in a position to describe how the planner learns the reasons underlying a case failure. In this section, we show how the EBL techniques developed in [7] are employed to construct these reasons.

DERSNLP+EBL constructs case failure reasons incrementally as the skeletal plan is extended by the addition of plan constraints and plan failures are encountered. As Figure 2 illustrates, DERSNLP+EBL forms explanations for the failures that occur in the subtree rooted at the skeletal plan. [1] Path failure explanations identify a minimal set of conflicting constraints in the plan which are together inconsistent. These are then regressed up the tree to construct reasons for case failure.

Since a plan failure is explained by a subset of plan constraints, failure explanations are represented in the same manner as a partial plan. DERSNLP represents its partial plans as a 6-tuple, $\langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}, \mathcal{E}, \mathcal{C} \rangle$, where [1]: $\mathcal{S}$ is the set of actions (step-names) in the plan, each of which is mapped onto an operator in the domain theory. $\mathcal{S}$ contains two dummy steps: t_I whose effects are the initial state conditions, and t_G whose preconditions are the input goals, G. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and post-conditions of the operators which are represented in the plan steps, $\mathcal{S}$. $\mathcal{O}$ is a partial ordering relation on S, representing the ordering constraints over the steps in $\mathcal{S}$. $\mathcal{L}$ is a set of causal links of the form $\langle s, p, s' \rangle$ where $s, s' \in \mathcal{S}$. A causal link contains the information that s causes (contributes) p which unifies with a precondition of s'. $\mathcal{E}$ contains step effects, represented as $\langle s, e \rangle$, where $s \in \mathcal{S}$. $\mathcal{C}$ is a set of open conditions of the partial plan, each of which is a tuple $\langle p, s \rangle$ such that p is a precondition of step s and there is no link supporting p at s in $\mathcal{L}$.

The explanation for the failure of the partial plan contains the constraints which contribute to an inconsistency in the plan. These inconsistencies appear when new constraints are added which conflict with existing constraints. DERSNLP makes two types of planning decisions, *establishment* and *resolution*. Each type of decision may result in a plan failure. For example, an *establishment* decision makes a choice as to a method of achieving an open condition, either through a new/existing plan step, or by adding a causal link from the initial state. When an attempt is made to achieve a condition by linking to an initial state effect, and this condition is not satisfied in the initial state, the plan then

[1] Depth limit failures are ignored. This means that the failure explanations that are formed are not sound in the case of a depth limit failure. However, soundness is not crucial for the current purpose, since explanations are used only for case retrieval and not for pruning paths in the search tree.

Type : ESTABLISHMENT Kind : NEW STEP Preconditions : $\langle p', s' \rangle \in \mathcal{C}$ Effects : $\mathcal{S}' = \mathcal{S} \cup \{s\}$ $\mathcal{O}' = \mathcal{O} \cup \{s \prec s'\}$ $\mathcal{B}' = \mathcal{B} \cup unify(p, p')$ $\mathcal{L}' = \mathcal{L} \cup \{\langle s, p, s' \rangle\}$ $\mathcal{E} = \mathcal{E} \cup effects(s)$ $\mathcal{C}' = \mathcal{C} - \{\langle p', s' \rangle\}$ $\cup\ preconditions(s)$	Type : ESTABLISHMENT Kind : NEW LINK Preconditions : $\langle p', s' \rangle \in \mathcal{C}$ Effects : $\mathcal{O}' = \mathcal{O} \cup \{s \prec s'\}$ $\mathcal{B}' = \mathcal{B} \cup unify(p, p')$ $\mathcal{L}' = \mathcal{L} \cup \{\langle s, p, s' \rangle\}$ $\mathcal{C}' = \mathcal{C} - \{\langle p', s' \rangle\}$

Figure 3: Planning decisions are based on the current active plan $\langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}, \mathcal{E}, \mathcal{C} \rangle$ and have effects which alter the constraints so as to produce the new current active plan $\langle \mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}', \mathcal{E}', \mathcal{C}' \rangle$.

contains a contradiction. An explanation for the failure is constructed which identifies the two conflicting constraints: $\langle \emptyset, \emptyset, \emptyset, \langle t_I, p, s \rangle, \langle t_I, \neg p \rangle, \emptyset \rangle$.

As soon as a path failure is detected and an explanation is constructed, the explanation is regressed through the decisions in the failing path up to the root of the search tree. In order to understand the regression process, it is useful to think of planning decisions as STRIPS-style operators acting on partial plans (see Figure 3). The preconditions of these operators are specified in terms of the plan constraints that make up a plan flaw, which is either an open condition or a threat to a causal link. The effects are the constraints that are *added* to the partial plan to eliminate the flaw.

Each of the conflicting constraints in the failure explanation is regressed through the planning decision following the procedure shown in Figure 4, and the results are sorted according to type to form the new regressed explanation. As an example, consider that a new link from the initial state results in a failure. The explanation, e_1 is: $\langle \emptyset, \emptyset, \emptyset, \langle t_I, p, s \rangle, \langle t_I, \neg p \rangle, \emptyset \rangle$ When e_1 is regressed through the final decision, d_f to obtain a new explanation, e_1^f, the initial state effect regresses to itself. However, since the link in the explanation was added by the decision, d_f, this link regresses to the open condition which was a precondition of adding the link. The new explanation, e_1^f, is therefore $\langle \emptyset, \emptyset, \emptyset, \emptyset, \langle t_I, \neg p \rangle, \langle p, s \rangle \rangle$. The regression process continues up the failing path until it reaches the root of the search tree. This process is illustrated graphically in Figure 5. The open condition in the explanation is regressed over new-step establishment decisions in the failing path until it is eventually replaced by one of the input goals. When all of the paths in the subtree underneath the skeletal plan have failed, the failure reason at the root of the search tree provides the reason for the failure of the case. It represents a combined explanation for all of the path failures. The case failure reason contains only the aspects of the new problem

PROCEDURE **RegressConstraint** (c , d)
  IF $c \in$ effects(d) RETURN preconditions (d)
    ELSE RETURN $\{c\}$

Figure 4: *Regression in* DERSNLP+EBL

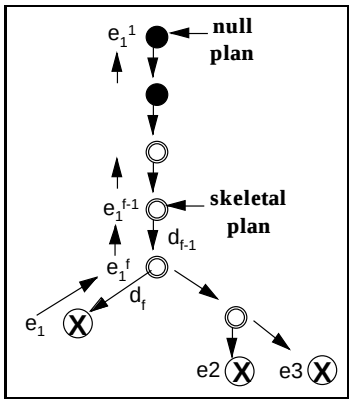

Figure 5: *Explanations are constructed for the failing plans in the leaf nodes of the subtree rooted at the skeletal plan, and are regressed up the tree. The explanation at the root of the tree is computed as* $e_i^1 = d_1^{-1}(d_2^{-1} \cdots (d_f^{-1}(e_i)) \cdots)$.

which were responsible for the failure. It may contain only a subset of the problem goals. Also, any of the initial state effects that are present in a leaf node explanation, are also present in the reason for case failure.

Each time a case fails the failure reason is used to annotate the case in the library. The library fragment depicted in Figure 6 stores in a discrimination network all of the cases which solve a single set of input goals. Individual cases which solve these goals are represented one level lower in the network. Each case is labeled by the relevant initial state conditions. When one of these cases is retrieved for replay and the case fails, it is then annotated with the reason for the case failure. When the case is retrieved again this failure reason is tested in the new problem situation, and, if satisfied, the retrieval process returns the alternative case that repairs the failure. The case failure reason is thus used to direct retrieval away from the case which will repeat a known failure, and towards the case that avoids it.

4 An Example of Case Failure

Figure 7 provides a trace of DERSNLP's decision process in arriving at a solution to a simple problem taken from the logistics transportation domain of [9]. We will use this trace

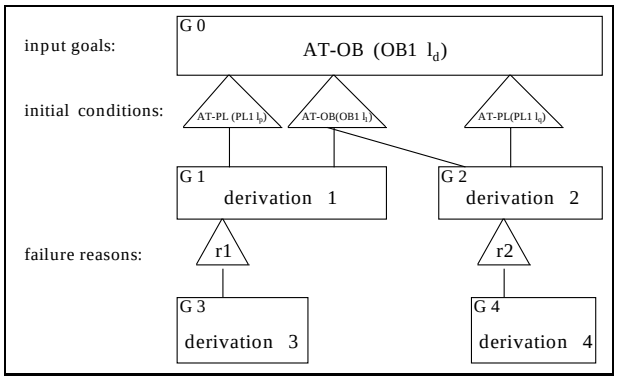

Figure 6: *Local organization of the case library.*

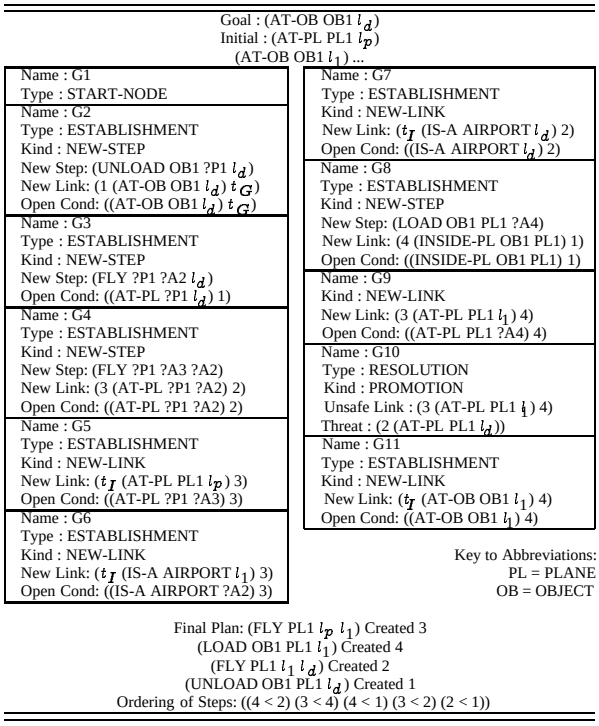

Figure 7: *An Example Solution Trace for* DERSNLP

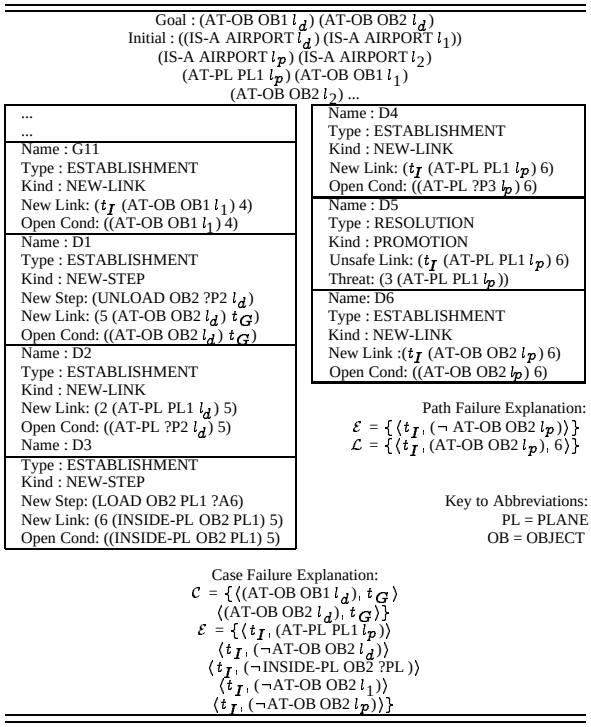

Figure 8: *An Example Trace of a Failing Path for* DERSNLP

as an example and assume that this derivation is replayed in solving a new problem that contains an extra goal. In the logistics transportation domain, there are various transport devices which move packages between different locations. The trace in Figure 7 derives a plan to solve a problem which has a single goal, that OB1 be at the destination airport, location l_d.

Starting with the null plan DERSNLP first chooses a method of achieving this open condition. The first step to be added to the plan unloads the package, OB1, at its destination, l_d. Planning continues by adding a second step to fly the plane to the package destination. This step accomplishes a precondition of the `unload` action, which is that the plane has to be at l_d in order to unload the package. As planning continues more steps are added, and some of the open conditions are linked to existing steps, or to the initial world state. Eventually a threat is detected. The flight to the final destination location threatens a precondition of the `load` action, since the plane has to be at the package location in order to load the package. This threat is resolved by promoting the flight to the package destination to follow the loading of the package. The final plan is shown at the bottom of Figure 7.

If the derivation in Figure 7 is replayed in solving a new problem where there is an extra package to be transported to the same destination, and this package lies outside the plane's route, replay will fail. Figure 8 illustrates how the decisions that DERSNLP makes in extending the derivation result in a path failure. Although the subtree underneath the skeletal plan fails, for illustration purposes only one failing path is traced in Figure 8. The figure shows only the latter portion of the path, starting with the skeletal plan. DERSNLP first attempts to extend the replayed path by adding the `unload` and then the `load` actions. A precondition of a `load` is that the plane is at some location. DERSNLP links the plane's location to the initial world state. The plane originates at l_p. A decision is then followed by a decision to link the object's location to the initial state. This results in an inconsistency in the plan since the object is not at l_p initially (see path failure explanation in Figure 8). The case failure reason is shown at the bottom of Figure 8. It gives the conditions under which a future replay of the same case will result in failure. A summary of the information content of the explanation is: *There is an extra package to transport to the same location, and that package is not inside the plane initially, and it is not located on the plane's route.* This explanation identifies implicit features of the problem description which are predictive of failure and are used to censor retrieval.

5 Empirical Evaluation of the Retrieval Strategy

In this section we describe an empirical study which was conducted to evaluate the advantage of retrieving cases on the basis of previous replay failures. We chose domains in which randomly generated problems contained interacting goals, and tested planning performance when DERSNLP was solving n-goal problems from scratch and through replay of a similar (n-1)-goal problem. We compared the performance improvements provided by replay in *static mode* when information about the previous case failure was not used in case retrieval vs when it was used (*learning mode*).

Domains: Experiments were run on problems drawn from two domains. The first was the artificial domain, ($\theta_2 D^m S^1$), originally described in [1] and shown in Figure 9. The

$(\theta_2 D^m S^1)$:
$(A_i^\alpha \ \underline{precond}: \ \{I_i, P_\alpha\} \ \underline{add}: \ \{G_i\} \ \underline{delete}: \ \{I_j | j < i\})$
$(A_i^\beta \ \underline{precond}: \ \{I_i P_\beta\} \ \underline{add}: \ \{G_i\} \ \underline{delete}: \ \{I_j | j < i\})$
$(A_\alpha \ \underline{precond}: \ \{\} \ \underline{add}: \ \{G_\alpha\} \ \underline{delete}: \ \{P_\beta\} \cup \{G_i | \forall i\})$

Figure 9: *The $(\theta_2 D^m S^1)$Domain*

logistics transportation domain of [9] was adopted for the second set of experiments. Eight packages and one airplane were randomly distributed over four cities. Problem goals represent the task of getting one or more packages to a single designated airport.

Retrieval Strategy: In *static* mode, cases were initially retrieved on the basis of the goals that were covered by the case as well as their relevant initial state conditions. In *learning* mode, the failure reason attached to the case was used to censor its retrieval. When a failure reason was found to be satisfied in the new problem situation, then the case that repaired the failure was retrieved. Following retrieval, the problem was solved both by replay of the retrieved case as well as through from-scratch planning.

Experimental Setup: Each experiment consisted of three phases, each phase corresponding to an increase in problem size. In an initial training session that took place at the start of each phase n , 30 n-goal problems were randomly generated and solved from scratch, and each case was stored in the case library. Following training, the testing session consisted of randomly generating problems in the same manner but with an additional goal. Each time that a new $(n + 1)$ goal problem was tried, an attempt was made to retrieve a similar n-goal problem from the library. Cases were chosen so as to have all but one of the goals matching. If during the testing session, a case was retrieved which had previously failed and that case was annotated with a failure reason, then the problem was solved in learning, in static, and in scratch planning modes, and the problem became part of the 30-problem set. With this method, we were able to evaluate the improvements provided by failure-based retrieval when retrieval on initial state conditions alone was ineffective, and when failure reasons were available.

Experimental Results The results of the experiments are shown in Figure 10. Each table entry represents cumulative results obtained from the sequence of 30 problems corresponding to one phase of the experiment. These are respectively the total number of search nodes visited for all of the 30 test problems, and the total CPU time in seconds (including case retrieval time). DERSNLP in learning mode was able to solve the multi-goal problems in substantially less time than in static mode.

$(\theta_2 D^m S^1)$	**Learning**	**Static**	**Scratch**
2 Goal			
nodes	90	240	300
time	1	4	2
3 Goal			
nodes	120	810	990
time	2	15	8
4 Goal			
nodes	150	2340	2533
time	3	41	21

Logistics	**Learning**	**Static**	**Scratch**
2 Goal			
nodes	1773	1773	2735
time	30	34	56
3 Goal			
nodes	6924	13842	20677
time	146	290	402
4 Goal			
nodes	290	38456	127237
time	32	916	2967

Figure 10: *Performance in $(\theta_2 D^m S^1)$ and Logistics Domains*

6 Summary and Conclusion

The current work complements and extends earlier treatments of case retrieval [6; 9]. Replay failures are explained and used to avoid the retrieval of a case in situations where replay of a case will mislead the planner. CHEF [3] learns to avoid execution-time failures by simulating and analyzing plans derived by reusing old cases. In contrast, our approach attempts to improve planning efficiency by concentrating on search path failures encountered in extending a case to solve a new problem. The results from our experiments demonstrate that replay performance can be significantly improved by taking into account these case failures. These results justify the definition of case failure, as well as the method of constructing case failure reasons. The results demonstrate that when case failure information is available, the improvements gained by utilizing this information are such that they more than offset the added cost entailed in testing failure reasons and retrieving on the basis of previous failures.

References

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

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

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

[4] L. Ihrig and S. Kambhampati. Derivation replay for partial-order planning. In *Proceedings AAAI-94*, 1994.

[5] L Ihrig and S. Kambhampati. On the relative utility of plan-space vs state-space planning in a case-based framework. Technical Report 94-006, Department of Computer Science and Engineering, 1994. Arizona State University.

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

[7] S. Katukam and S. Kambhampati. Learning ebl-based search control rules for partial order planning. In *Proceedings AAAI-94*, 1994.

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

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