Design and Implementation of a Replay Framework based on a Partial Order Planner*

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

Abstract

In this paper we describe the design and implementation of the derivation replay framework, DERSNLP+EBL (Derivational SNLP+EBL), which is based within a partial order planner. DERSNLP+EBL replays previous plan derivations by first repeating its earlier decisions in the context of the new problem situation, then extending the replayed path to obtain a complete solution for the new problem. When the replayed path cannot be extended into a new solution, explanation-based learning (EBL) techniques are employed to identify the features of the new problem which prevent this extension. These features are then added as censors on the retrieval of the stored case. To keep retrieval costs low, DERSNLP+EBL normally stores plan derivations for individual goals, and replays one or more of these derivations in solving multi-goal problems. Cases covering multiple goals are stored only when subplans for individual goals cannot be successfully merged. The aim in constructing the case library is to predict these goal interactions and to store a multi-goal case for each set of negatively interacting goals. We provide empirical results demonstrating the effectiveness of DERSNLP+EBL in improving planning performance on randomly-generated problems drawn from a complex domain.

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 (Hammond 1990; Ihrig 1996; Ihrig & Kambhampati 1994; Veloso & Carbonell 1993). One approach to case-based planning is to store plan derivations which are then used as guidance when solving new similar problems (Veloso & Carbonell 1993). Recently we adapted this approach, called *derivational replay*, to improve the performance of the partial-order planner, SNLP (Ihrig & Kambhampati 1994). Although it was found that replay tends to improve overall performance, its effectiveness depends on retrieving an appropriate case. Often the 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 previous cases on the basis of a static similarity metric which considers the previous problem goals as well as the features of the initial state which are relevant to the achievement of those goals (Kambhampati & Hendler 1992; Ihrig & Kambhampati 1994; Veloso & Carbonell 1993). If these are again elements of the problem description then the case is retrieved and reused in solving the new problem. Usually the new problem will contain extra goal conditions not covered by the case. This means that the planner must engage in further planning effort to add constraints (including plan steps and step orderings) which achieve the conditions that are left open. Sometimes an extra goal will interact with the covered goals and the planner will not be able to find a solution to the new problem without backtracking and retracting some of the replayed decisions. 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 improve its case retrieval.

In this paper, 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 methods for forming explanations of search path failures and regressing these explanations through the planning decisions in the failing paths (Kambhampati, Katukam, & Qu 1996). Here we employ these techniques to construct reasons for case failure, which are then used to annotate the failing cases to constrain their future retrieval. Furthermore, each failure results in the storage of a new case which repairs the failure. DERSNLP+EBL normally stores plan derivations solving single input goals. When a case fails in that it cannot be extended to solve extra goals, a new multi-goal case is stored covering the set of *negatively interacting* goals. DERSNLP+EBL thus builds its case library incrementally in response to case failure, as goal interactions are discovered through the course of problem solving.

In (Ihrig & Kambhampati 1995), the potential effective-

*This research is supported in part by an NSF Research Initiation Award IRI-9210997, NSF Young Investigator Award IRI-9457634, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039 and F30602-95-C-0247. I would like to thank Biplav Srivastava for his helpful comments.

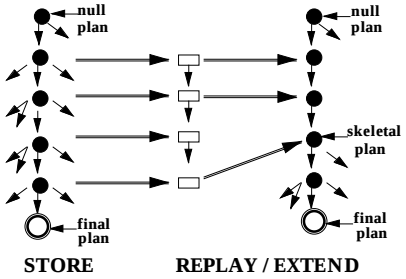


Figure 1: *Schematic characterization of derivation storage and replay. Each time that a plan is derived, the decisions contained in the plan derivation (shown as the filled circles to the left of the figure) are stored as a sequence of instructions (shown as open rectangles) which are used to guide the new search process. The guidance provided by replay is considered successful if the skeletal plan that replay produces can be extended (by the addition of constraints) into a solution to the new problem. If replay has been successful, the skeletal plan lies on the new derivation path leading to the solution.*

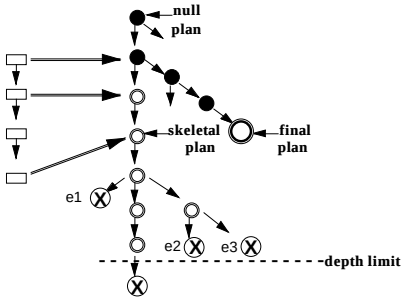


Figure 2: *A replay failure is indicated when a solution to the new problem can be reached only by backtracking over the skeletal plan, which now lies outside the new plan derivation (shown as filled circles). Explanations are constructed for the failing plans in the leaf nodes of the subtree directly beneath the skeletal plan, and are regressed up the search tree and collected at the root to become the reason for case failure.*

ness of this approach was evaluated in an empirical study which compared the replay performance of DERSNLP+EBL both *with and without* failure information. In this paper, we demonstrate overall performance improvements provided by multi-case replay when a case library is constructed on the basis of replay failures. In the next section, we describe DERSNLP+EBL which implements derivation replay within the partial-order planner, SNLP.

Derivation Replay in Partial-Order Planning

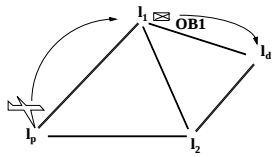
Whenever DERSNLP+EBL attempts a new problem, and achieves a solution, a trace of the decisions that fall on the derivation path leading from the root of the search tree to the final plan in the leaf node is stored in the case library. Then, when a similar problem is encountered, this trace is replayed as guidance to the new search process. Figure 1 illustrates the replay of a derivation trace. DERSNLP+EBL employs an *eager* replay strategy. With this strategy, 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. 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 case, further planning effort is required to extend this plan into a solution for the new problem.

In the current work replay success and failure is defined in terms of the skeletal plan. Replay is considered to fail if the skeletal plan cannot be extended by the addition of further constraints into a solution for the new problem (See Figure 2). In such instances, the planner first explores the failing subtree underneath the skeletal plan, then recovers by backtracking over the replayed portion of the search path.

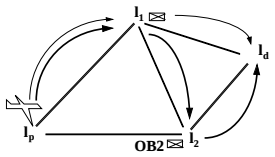
Replay failure usually results in poor planning performance since, over and above the cost of the search effort, it entails the additional cost of retrieving a trace from the library, as well as the cost of validating each of the decisions in the trace. This means that when replay fails and the planner has to backtrack over the skeletal plan performance may be worse than in from-scratch planning.

When a case fails, and the planner goes on to find a new solution, the final plan that it reaches does not contain some of the constraints that are present in the skeletal plan. The new derivation path which leads from the root of the search tree to the final plan in the leaf node thus avoids (or *repairs*) the failure encountered in replaying the old case. Consider a simple example taken from the logistics transportation domain of (Veloso & Carbonell 1993). Figure 3a illustrates the solution to a simple problem drawn from this domain. The goal is to have package OB1 located at the destination location l_d. The package is initially at location l_1. There is a plane located at l_p which can be used to transport the package. A previous plan which solves this problem will contain steps (shown by the curved arrows in Figure 3a) that determine the plane's route to the destination airport as well as steps which accomplish the loading of the package at the right place along this route. This plan may be readily extended to load and unload extra packages which lie along the same route. However, if the new problem involves the additional transport of a package which is off the old route, the planner may not be able reach a solution without backtracking over some of the previous step additions. The new plan shown in Figure 3b contains some alternative steps that achieve the goal covered by the previous case. The plane takes a longer route which means that the plan may be readily extended to solve the extra goal.

DERSNLP+EBL detects that a previous case has failed when all attempts to refine the skeletal plan have been tried, and the planner is forced to backtrack over this plan. At this point, the planner has already constructed an explanation



(a) Previous Case



(b) New Problem with Extra Goal

Figure 3: *An example of plan failure. The plan derived in an earlier problem-solving episode is shown in (a). This plan accomplishes the transport of a single package, OB1, to the destination airport l_d. Replay fails for a new problem, whose solution is illustrated in Figure (b). The new problem contains an extra goal which involves the additional transport to l_d of a second package, OB2, which is initially located off the previous route.*

for the skeletal plan's failure (which becomes the reason for case failure). This explanation is incrementally formed with each path failure experienced in the subtree rooted at the skeletal plan. Each analytical failure that is encountered is regressed through the decisions in the failing path and the regressed path failure explanations are collected at the root of the search tree to form the reason for case failure. An example of a case failure reason is shown in Figure 4. It gives the conditions under which a future replay of the case will again result in failure. These conditions refer to the presence in the new problem of a set, $\mathcal{C}$, of negatively interacting goals, as well as some initial state conditions, contained in $\mathcal{E}$. A summary of the information content of the failure reason is: *There is an extra package to transport to the same destination location, and that package is not at the destination location, is not inside the plane, and is not located on the plane's route*.

Since replay merely guides the search process (without pruning the search tree), a replay failure does not affect the soundness or completeness of the planning strategy. After backtracking over the skeletal plan, the planner continues its search, and will go on to find a correct solution to the full problem if one exists. This new solution achieves all of the negatively interacting goals identified in the case failure reason. Moreover, since the interacting goals represent a subset of the new problem goals, the new derivation may be used to construct a new repairing case covering only these goals. The repairing case is indexed directly beneath the failing case so as to censor its retrieval. In the future, whenever the failure reason holds, the retriever is directed away from the case that experiences a failure and toward the case that repairs the failure.

We are now in a position to describe how the planner learns the reasons underlying a case failure. Specifically, we use EBL techniques to accomplish this learning. In the next section, we show how the techniques developed in (Kambhampati, Katukam, & Qu 1996) are employed to construct these reasons.

Case Failure Explanation:

$\mathcal{C}$ = {⟨(AT-OB OB1 l_d), t_G⟩
⟨(AT-OB OB2 l_d), t_G⟩}
$\mathcal{E}$ = {⟨t_I, (¬AT-OB OB2 l_d)⟩
⟨t_I, (¬INSIDE-PL OB2 ?PL)⟩
⟨t_I, (¬AT-OB OB2 l_1)⟩
⟨t_I, (¬AT-OB OB2 l_p)⟩}

Figure 4: *An example of a case failure reason*

Learning from Case Failure

DERSNLP+EBL constructs reasons for case failure through the use of explanation-based learning techniques which allow it to explain the failures of individual paths in the planner's search space. A search path experiences an analytical failure when it arrives at a plan which, because it contains a set of inconsistent constraints, cannot be further refined into a solution. EBL techniques are used to form explanations of plan failures in terms of these conflicting constraints (Kambhampati, Katukam, & Qu 1996). DERSNLP+EBL constructs explanations for each of the analytical failures that occur in the subtree beneath the skeletal plan[1].

Since a plan failure explanation is a subset of plan constraints, these explanations are represented in the same manner as a partial plan. DERSNLP+EBL represents its partial plans as a 6-tuple, $\langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}, \mathcal{E}, \mathcal{C} \rangle$, where (Barrett & Weld 1994): $\mathcal{S}$ 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 a minimal set of plan constraints which represent 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*

[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 5: *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 $\langle \mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}', \mathcal{E}', \mathcal{C}' \rangle$.*

decision makes a choice as to a method of achieving an open condition, either through a new plan step, or by adding a causal link from an existing step (See Figure 5). 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 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 plan 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. 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, in the case of a resolution decision, is 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, and the results are sorted according to type to form the new regressed explanation. As an example, consider that a new decision, d_f, adds a link from the initial state which 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, $d_f^{-1}(e_1)$, 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, $d_f^{-1}(e_1)$, 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. When all of the paths in the subtree underneath the skeletal plan have failed, the failure reason at the root of the 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 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. The next section describes how case failure reasons are used to build the case library.

Library Organization

A large complex domain means a great variety in the problems encountered. When problem size (measured in terms of the number of goals, n) is large, it is unlikely that a similar n-goal problem will have been seen before. It is therefore an advantage to store cases covering smaller subsets of goals, and to retrieve and replay multiple cases in solving a single large problem. In implementing this storage strategy, decisions have to be made as to which goal combinations to store. Previous work (Veloso & Carbonell 1993) has reduced the size of the library by separating out connected components of a plan, and storing their derivations individually. Since DERSNLP+EBL is based on a partial order planner, it can replay cases in sequence and later add step orderings to accomplish the merging of their subplans. It therefore has a greater capability of reducing the size of the library, since it may store smaller problems. In the current work, we store multi-goal cases only when subplans for individual goals cannot be merged to reach a full solution.

With this aim in mind, we have implemented the following deliberative storage strategy. When a problem is attempted which contains n goals, a single goal problem containing the first goal in the set is attempted and, if solved, the case covering this goal alone is stored in the library. Multi-goal problems to be stored are solved incrementally by increasing the problem size by one goal at a time. For example, if the problem just attempted solved goals $G = \langle g_1, g_2, ..., g_i \rangle$ through a decision sequence D_i then a second decision sequence, D_{i+1}, is stored whenever D_i cannot be successfully extended to achieve the next goal g_{i+1}. When this occurs, the explanation of replay failure is used to identify a subset of input goals that are responsible for the failure. A new derivation is produced which solves only these negatively interacting goals. This derivation is then stored in the library. Whenever the next goal in the set is solved through simple extension of the previous decision sequence, no case is stored which includes that goal. This means that each new case that is stored corresponds to either a single-goal problem or to a multi-goal problem containing negatively interacting goals. Moreover, all of the plan derivations stored from a single problem-solving episode are different in that no decision sequence stored in the library is a prefix of another stored case.

This strategy drastically reduces the size of the library. It means that goals that interact *positively* in that they can be solved through one or more common steps are stored individually in single cases. Goals that are *negatively interacting* (in that solving one means having to alter the solution to the other) are stored together as multi-goal cases. The more experience that the planner has in problem-solving, the more of these multi-goal cases are discovered and stored, and the less likely it is that the

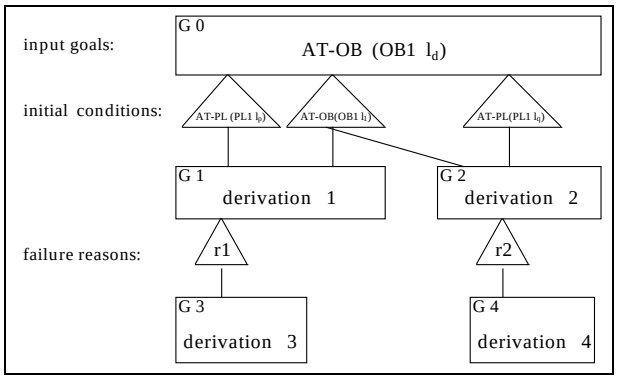


Figure 6: *Local organization of the case library.*

planner has to backtrack over its replayed paths. The aim is to store a minimum number of cases such that all of the problems encountered in the future may be achieved through sequential replay of multiple stored cases.

Multi-goal cases are indexed in the library so as to censor the retrieval of their corresponding single-goal subproblems. The discrimination net depicted in Figure 6 indexes one fragment of the case library. This fragment includes all of the cases which solve a single input goal. Individual cases which solve this goal are represented one level lower in the net. Each case is indexed by its relevant initial state conditions. When one of these cases is retrieved for replay and the case fails, the alternative derivation corresponding to the additional interacting goal is added to the library and indexed directly under the failing case so as to censor its future retrieval. Before the case that experienced a failure is retrieved again, the retriever checks whether the extra goals responsible for the failure are present under the same initial conditions. If so, the retrieval process returns the alternative case containing these extra goals. 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.

Multi-case replay can result in a lower quality plan if care is not taken to avoid redundancy in step addition. When derivations for positively-interacting goals are stored individually, replaying each case in sequence may result in superfluous steps in the plan. When the first retrieved derivation is replayed, none of its replayed step additions will result in redundancy. However, when subsequent goals are solved through replay of additional cases, some step additions may be unnecessary in that there are opportunities for linking the open conditions they achieve to earlier established steps.

We solved this problem and obtained shorter plans by increasing the *justification* for replaying a step addition decision. In order to take advantage of linking opportunities, before replaying a new step addition, the replay process takes note of any links which are currently available but were not present in the previous case. When new linking opportunities are detected, the decision to add a new step is rejected. After replay, the new links are explored through the normal course of plan refinement. This careful screening of the step addition decisions improves the quality of plans in terms of the number of steps they contain. The next

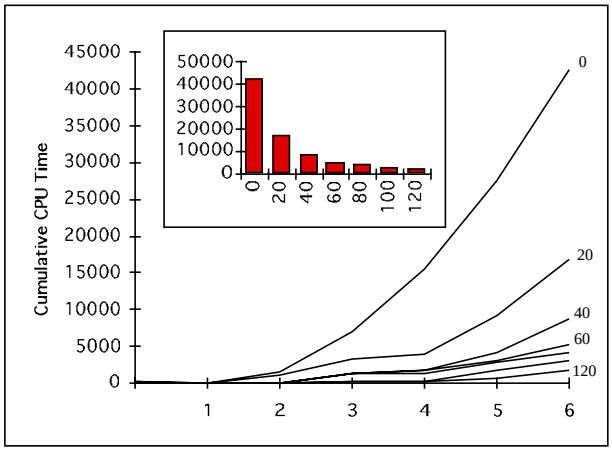


Figure 7: *Replay performance in the logistics transportation domain. The cumulative CPU time (in secs) on problem sets of increasing problem size (1 to 6 goals) is plotted for each level of training (0 to 120 training problems solved). The insert shows total CPU time on all of the 6 test sets after increasing amounts of training*

section describes an empirical study demonstrating the performance improvements provided by multi-case replay.

Empirical Evaluation

Experimental Setup: We tested the improvement in planning performance provided by multi-case replay on problems drawn from the logistics transportation domain (Veloso & Carbonell 1993). Problem test sets increasing in problem size were randomly generated from this domain. The initial state of each problem described the location of 6 packages, and 12 transport devices (6 planes and 6 trucks) within 6 cities, each containing a post office and an airport. See (Ihrig 1996) for similar tests on larger problems in a 15 city domain.

The experiments were run in six phases. At the start of each phase n the library was cleared and thirty test problems, each with n goals, were randomly generated. The planner was then repeatedly tested on these problems after increasing amounts of training on randomly generated problems of the same size. During training, problems were solved and their plan derivations were stored as described above. Multi-goal problems were stored only when retrieved cases failed. In these instances the failure information was used to extract the subset of input goals responsible for the failure, and a case which solved these goals alone was stored in the library.

Experimental Results: The results are shown in Figure 7 and 8. Figure 7 plots replay performance measured as the cumulative CPU time taken in solving the 30-problem sets tested in the 6 phases of the experiment. The figure plots replay performance (including case retrieval time) for the various levels of training prior to testing. For example, level 0 represents planning performance after no training. Since in this instance the case library is empty, level 0 represents from-scratch planning on the problem test set. Level 20 represents testing after training on 20 randomly generated

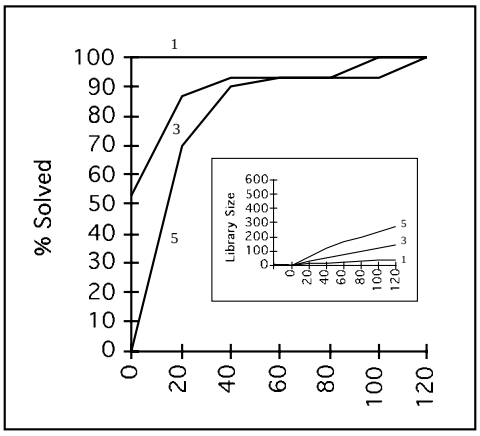


Figure 8: *Percentage of test problems solved with the time limit (500 sec) is plotted for 30-problem test sets containing problems of 1, 3 and 5 goals. This percentage increased with training (0 to 120 training problems solved). The insert shows the corresponding increase in the size of the case library.*

problems of the same size as the test set. The results indicate that this relatively small amount of training provided substantial improvements in planning performance. Moreover, performance improved with increased levels of training. The improvements provided by multi-case replay more than offset the added cost entailed in retrieving and matching stored cases.

Figure 8 reports the percentage of test problems solved within the time limit which was imposed on problem solving. It shows how training raised the problem-solving horizon, particularly in the later phases of the experiment when larger problems were tested. Storing cases on the basis of case failure kept the size of the library low (see insert, Figure 8) and retrieval costs were minimal. In the next section, we discuss the relationship to previous work in case storage and retrieval.

Related Work and Discussion

The current work complements and extends earlier treatments of case retrieval (Kambhampati & Hendler 1992; Veloso & Carbonell 1993). Replay failures are explained and used to avoid the retrieval of a case in situations where replay will mislead the planner. Failures are also used to construct repairing cases which are stored as alternatives to be retrieved when a similar failure is predicted.

CHEF (Hammond 1990) 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 failures encountered in plan generation. We integrate replay with techniques adopted from the planning framework provided by SNLP+EBL (Kambhampati, Katukam, & Qu 1996). This framework includes methods for constructing conditions for predicting analytical failures in its search space.

EBL techniques have been previously used to learn from problem-solving failures (Kambhampati, Katukam, & Qu 1996; Minton 1990; Mostow & Bhatnagar 1987). However, the goal of EBL has been to construct generalized control rules that can be applied to each new planning decision. Here we use the same analysis to generate case-specific rules for case retrieval. Rather than learn from all failures, we concentrate on learning from failures that result in having to backtrack over the replayed portion of the search path. As learned information is used as a censor on retrieval rather than as a pruning rule, soundness and completeness of the EBL framework are not as critical. Furthermore, keeping censors on specific cases avoids the utility problem commonly suffered by EBL systems.

Conclusion

In this paper, we described a framework for a case-based planning system that is able to exploit case failure to improve case retrieval. A case is considered to fail in a new problem context when the skeletal plan produced through replay cannot be extended by further planning effort to reach a solution. EBL techniques are employed to explain plan failures in the subtree directly beneath the skeletal plan. These failure explanations are then propagated up the search tree and collected at the root. The regressed plan failures form the reason for case failure which is used to censor the case and to direct the retriever to a repairing case. Our results provide a convincing demonstration of the effectiveness of this approach.

References

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

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

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

Ihrig, L., and Kambhampati, S. 1995. An explanation-based approach to improve retrieval in case-based planning. In *Current Trends in AI Planning: EWSP '95*. IOS Press.

Ihrig, L. 1996. *The Design and Implementation of a Case-based Planning Framework within a Partial Order Planner*. Ph.D. Dissertation, Arizona State University.

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

Kambhampati, S.; Katukam, S.; and Qu, Y. 1996. Failure driven dynamic search control for partial order planners: An explanation-based approach. *Artificial Intelligence*. To Appear.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. In *Artificial Intelligence*, volume 42, 363--392.

Mostow, J., and Bhatnagar, N. 1987. Failsafe: A floor planner that uses ebg to learn from its failures. In *Proceedings IJCAI-87*, 249--255.

Veloso, M., and Carbonell, J. 1993. Toward scaling up machine learning: A case study with derivational analogy in prodigy. In Minton, S., ed., *Machine Learning methods for planning*. Morgan Kaufmann.