

Learning Explanation-Based Search Control Rules For Partial Order Planning

Suresh Katukam & Subbarao Kambhampati*

Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406
email: suresh@enuxsa.eas.asu.edu, rao@asu.edu

Abstract

This paper presents SNLP+EBL, the first implementation of explanation based learning techniques for a partial order planner. We describe the basic learning framework of SNLP+EBL, including regression, explanation propagation and rule generation. We then concentrate on SNLP+EBL's ability to learn from failures and present a novel approach that uses stronger domain and planner specific consistency checks to detect, explain and learn from the failures of plans at depth limits. We will end with an empirical evaluation of the efficacy of this approach in improving planning performance.

1 Introduction

One way of coping with the computational complexity of domain-independent planning involves application of learning techniques to speed up planning. Accordingly, there has been a considerable amount of research directed towards applying explanation-based learning (EBL) techniques to planning [2, 10]. Much of this work has been concentrated on the state-based planning. Motivated by the known advantages of partial order (PO) planning over state based planning in plan generation [1] and reuse [5, 6], in this paper we address the problem of adapting EBL techniques to speed up partial order planning.

The EBL frameworks for state-based planning, such as PRODIGY/EBL [10] and FailSafe [2] typically construct search control rules that aim to steer the planner away from unpromising paths. The search control rules are generated by analyzing the search space explored by the planner to locate failures, constructing explanations for those failures, and regressing the failure explanations over the planning decisions.

Given that partial order and state-based planners search in very different search (decision) spaces, adapting these EBL frameworks to partial order (PO) planning offers two important challenges. First, since the space of decisions in PO planning is different, the process of regressing and

generalizing the explanations needs to be extended significantly. Secondly, since the types of failures encountered in PO planning are different from those encountered in state-based planning, we need to investigate effective learning opportunities for PO planners.

In this paper, we address both these issues. Specifically, we describe SNLP+EBL, a system that learns search control rules for SNLP, a causal link based PO planner [1, 9]. We will start by describing the basic learning framework in SNLP+EBL, including the details of regression, explanation propagation and search-control rule learning (Section 2). We will then concentrate on SNLP+EBL's ability to learn from failures. We will show that the failures detected by SNLP (analytical failures) alone do not by themselves provide effective learning opportunities for SNLP+EBL in many domains. This is because many futile lines of reasoning either never end in analytical failures or cross depth limits much before they do. Since depth limit failures are not analytical, it is not possible to learn from them.

To deal with this impasse, we adopt a novel approach of strategically applying stronger consistency checks to the plans crossing depth limits, to detect and explain the implicit failures in those plans. These explanations are then used to generate search control rules. In Section 3, we will describe a specific realization of this strategy that utilizes the *domain axioms* (or readily available physical laws of the domain) to detect and explain inconsistencies (failures) at some depth limit failures. In Section 3.1, we describe the results of an empirical study which demonstrate the effectiveness of the search control rules learned by this method.

2 The SNLP+EBL system

2.1 The base level planner

As mentioned earlier, our base level planner is SNLP, a causal link based PO planner described in [1, 9]. SNLP searches in the space of partial plans. Each partial plan can be seen as a 5 tuple: $\langle S, O, B, L, G \rangle$ where: S is the set of actions (also called steps) in the plan. The actions are described in the STRIPS representation, with *add*, *delete* and *precondition* lists. S contains two distinguished steps *start* and *fin*. The effects of *start* and the preconditions of *fin* correspond, respectively, to the initial state and the desired goals of the planning problem. O describes the ordering constraints over

*This research is supported in part by National Science Foundation under grant IRI-9210997, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039. Thanks to Bulusu Gopi Kumar, Steve Minton, Prasad Tadepalli and Dan Weld for helpful comments.

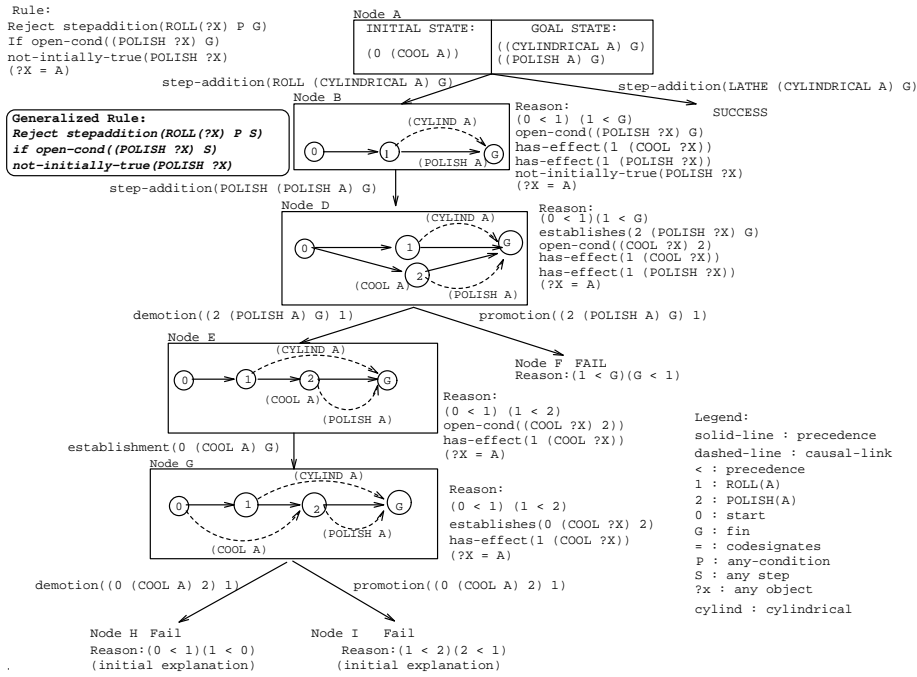


Figure 1: Search Tree illustrating learning from analytical failures

the steps in \mathcal{S} . \mathcal{B} is a set of codesignation (binding) and non-codesignation (prohibited bindings) constraints on the variables appearing in the preconditions and post-conditions of the operators.

\mathcal{G} is the set of open conditions of the partial plan, i.e, tuples $\langle c, s \rangle$ such that c is a precondition of step $s \in \mathcal{S}$. The planning process consists of establishing the open conditions with the help of the effects of either an existing step or a new step. Whenever an open condition $\langle c, s \rangle$ is established with the help of the effects of some step s' , it is removed from \mathcal{G} , and a causal link $s' \xrightarrow{c} s$ is added to \mathcal{L} . If s is a new step, its preconditions are also added to \mathcal{G} .

A causal link should be seen as a commitment by the planner to protect c in the range between s' and s . Whenever new steps are introduced into the plan, the existing causal links are checked to see if any of their conditions are violated. A step t of the plan is said to be a *threat* to a causal link $s \xrightarrow{p} w \in \mathcal{L}$, if t has an add or delete list literal q such that q possibly codesignates with p , and t can possibly come in between s and w . The threat is resolved by either *promoting* t to come after w , or *demoting* it to come before s (in both cases, appropriately updating \mathcal{O}), or adding non-codesignation constraints to ensure that q does not codesignate with p . A threat for a causal link is said to be *unresolvable* if all of these possibilities make either \mathcal{O} or \mathcal{B} inconsistent. SNLP backtracks when it encounters an unresolvable threat, or an unestablishable open condition.

The search tree in Figure 1 illustrates SNLP's planning process in terms of an example from a simple job-shop scheduling domain with the operators shown below:

Action	Precond	Add	Delete
Roll(o)	-	Cylind(o)	Polish(o) \wedge Cool(o)
Lathe(o)	-	Cylind(o)	Polish(o)
Polish(o)	Cool(o)	Polish(o)	-

The initial planning problem is to polish an object A and make its surface cylindrical. The object's temperature is cool in the initial state. The figure shows a failing branch of the search tree. In this branch, SNLP establishes the open condition $\langle Cylindrical(A), G \rangle$ with the help of the new step 1:Roll(A). It then establishes the other open condition $\langle Polished(A), G \rangle$ with the operator 2:Polish(A).

Since Roll(A) deletes Polish(A), it is now a threat to the link $2 \xrightarrow{Polish(A)} G$. SNLP resolves this threat by demoting 1:Roll(A) to come before 2:Polish(A). Polish(A) also introduces a new open condition $\langle Cool(A), 2 \rangle$. SNLP establishes it using the effects of the start state. Since Roll(A) also deletes Cool(A), it also threatens this last establishment. When SNLP tries to deal with the threat by demoting 1:Roll(A) to come before step 0, it fails (since 0 already precedes 1).¹ Such failures represent learning opportunities for the SNLP+EBL system, as discussed in the next section.

2.2 Interaction between the learner and the planner

Search control rules attempt to provide guidance to the underlying problem solver at critical decision points. As we have seen above, for SNLP these decision points are selection of open conditions; establishment, including simple-establishment and step-addition (operator selection); threat

¹To simplify the exposition clear, we omitted the failing separation branch from the figure.

Decision: The new step s_1 is added to establish the condition p at step s_2 in the current partial plan. The preconditions of this decision are simply that s_2 requires a condition p .

- (1) Result of regressing the ordering constraint $s' \prec s''$
 - True*, If $s' = s_1$ and $s'' = s_2$
 - True* \wedge start-special, If $s' = \text{start}$ and $s'' = s_1$
 - (see *rule generalization* section for explanation of start-special flag)
 - $s_2 \prec s''$, if $s' = s_1$ and $s_2 \prec s''$
 - $s' \prec s''$ otherwise
- (2) Result of regressing the causal link $s' \xrightarrow{p'} s''$
 - True* If $s' = s_1$ and $s'' = s_2$ and $p = p'$
 - $s' \xrightarrow{p'} s''$ otherwise

Figure 2: *Partial procedure for regressing explanations over step establishments*

selection; and threat resolution, including promotion, demotion and separation. Of these, it is not feasible to learn goal-selection and threat-selection rules using the standard EBL analysis since SNLP never backtracks over these decisions. SNLP+EBL system learns search control rules for all the other decisions. A search control rule may either be in the form of a selection rule or a rejection rule. In our current work, we have concentrated on learning rejection rules (although the basic framework can be extended to include selection rules).

Unlike systems such as PRODIGY/EBL, which commence learning only after the planning is completed, SNLP+EBL does adaptive (intra-trial) learning (c.f. [2]), which combines a form of dependency directed backtracking with generation of search-control rules. The planner does depth first search both in the learning and non-learning phases. During the learning phase, SNLP+EBL invokes the learning component whenever the planner encounters a failure.

There are two types of failures that are recognized by SNLP: the first is the analytical failure (where the planner reaches an impasse and declares that the current partial plan cannot be refined further). As explained earlier, this happens when the partial plan contains a causal link with an unresolvable threat, or an unestablishable open condition. The second type of failure occurs when the problem solver crosses a pre-set depth limit. The purpose of this limit is to prevent runaway search down fruitless alleys.

If the learner is able to explain the failure, it constructs an initial explanation and then regresses that explanation over the decisions in that branch to generate search control rules. From our discussion above, it is clear that analytical failures can be explained in terms of the inconsistency of the ordering and binding constraints of the partial plan, or in terms of the unestablishable open condition. For example, the initial explanation of failure for the partial plan at node H in Figure 1 is simply that $(0 \prec 1) \wedge (1 \prec 0)$ (causing an ordering cycle). We defer the treatment of depth limit failures to Section 3.

Regression: Once an initial explanation for a failure is constructed, SNLP+EBL regresses this explanation over the decisions leading to the failing partial plan. For state-based planners, the planning decisions correspond closely to opera-

Procedure Propagate(E, d_i)

- (d_i : failing partial plan; E : initial explanation of failure at d_i).
0. Set $d \leftarrow d_i$
 1. $E' \leftarrow \text{Regress}((E, \text{decision}(d)))$
 2. If $E' = E$, then set $d \leftarrow \text{parent}(d)$; Goto Step 1. (*a form of DDB*)
 3. If $E' \neq E$, then
 - 3.1. If there are unexplored siblings of d
 - 3.1.1 Make a rejection rule rejecting the decision of d , with E' as the antecedent generalize it and store it in the rule set
 - 3.1.2. $\text{fexp}(\text{parent}(d)) \leftarrow E' \wedge \text{precond}(\text{decision}(d)) + \text{fexp}(\text{parent}(d))$
(store E' as one the failure explanations under $\text{parent}(d)$)
 - 3.1.3. Restart search at the first unexplored sibling of d
 - 3.2. If there are no unexplored siblings of d ,
 - 3.2.1. Set $E \leftarrow [E' \wedge \text{precond}(\text{decision}(d))] + \text{fexp}(\text{parent}(d))$
 - 3.2.2. If all the siblings of d are establishing an open condition $\langle c, s \rangle$, and none of them establish it from start,
Set $E \leftarrow E + \neg \text{initially-true}(c)$
 - 3.2.3. Set $d \leftarrow \text{parent}(d)$; Goto Step 1.

Figure 3: *Propagating Failure Explanations*

tor applications, and thus regression over planning decisions is very close to regression over operators [12]. In contrast, decisions in the PO planners correspond to addition of generalized constraints (steps, orderings, bindings, causal links) to the partial plan. SNLP+EBL provides a sound and complete framework for regressing explanations over these decisions. Figure 2 contains a partial outline of the procedure for regressing arbitrary constraints of an explanation over an establishment decision involving step addition. A complete description of the regression rules for this and other planning decisions is beyond the scope of this paper, and can be found in [8].

Propagating Failure Explanations: Once an initial explanation of the failure has been identified, it is propagated up the failure branch to learn search control rules, as well as to do a form of dependency directed backtracking. Figure 3 provides the outline of this procedure. We will illustrate this process with the help of the example in Figure 1.

As discussed at the end of Section 2.1, the first failure in this example is noticed at node H . Here, the demotion caused order inconsistency in the plan. The explanation for this failure is simply that $(0 \prec 1) \wedge (1 \prec 0)$ (causing a cycle in the ordering). When this explanation is regressed over the demotion decision to make step 1 precede 0, we get $0 \prec 1$. Since the regressed explanation is different from the original one (step 3 in Figure 3)², it is then conjoined with the preconditions of the demotion decision (which in this case is that 1 threatens the link $0 \xrightarrow{C_{ool}(A)} 2$) to get the weakest preconditions for this branch of failure under G . These are then stored as one of the failure explanations at node G (step

²Had the explanation not changed during regression over H , the propagation process would have continued to the parent of this decision (step 2 in Figure 3). The rationale being that at least as far as this failure is concerned the choice taken at H 's parent node didn't contribute to the failure. Thus, as long as the decisions at the upper levels remain same, exploring the other siblings of H is guaranteed to keep this failure intact. This process constitutes a simple form of dependency directed backtracking.

3.1.2 in Figure 3).

Since the explanation changed after regression, and since there are unexplored siblings of H , technically, we can learn a rejection rule here (step 3.1.1 in Figure 3). However, its utility is going to be very low since the consistency check can find out the failure in the next level any way. To avoid generation of such low-utility rules, we currently use a preset constant l and ignore any rules generated within l levels of the failure.

At this point, search continues with the other sibling I of H , which uses the promotion alternative to resolve the threat (step 3.1.3). This plan also fails, and the explanation of this failure is $(1 \prec 2) \wedge (2 \prec 1)$. When regressed over the promotion decision, this becomes $1 \prec 2$. The preconditions for promotion, which are the same as those for the demotion, are conjoined with $(1 \prec 2)$, and added to the failure explanations at G . Finally, since there are no more alternatives at G , the existing explanations are conjoined to give the combined explanation $(0 \prec 1) \wedge (1 \prec 2) \wedge 0 \xrightarrow{Cool(A)} 2 \wedge \text{has-effect}(1, \text{cool}(A))$ (step 3.2.1). This combined explanation is now regressed over the establishment decision at node G , and the resultant explanation is regressed once again over E (since E has no more unexplored alternatives, and it already considered the establishment from initial state). The result is stored as the explanation of failure for the branch through E at node D . The search continues on the promotion branch through node F and eventually fails. This then allows SNLP+EBL to conjoin the explanations at D and pass the conjunction over to B .

Since D is the only alternative at B , we can continue the regression process. But, before doing so, we note that in the current planning episode none of the establishment branches at B have considered start as an establisher (because start did not have an effect unifying with $\text{Polish}(A)$). However, since the effects of the start step change from problem to problem (while those of all other steps, which correspond to domain operators, remain same), in a new problem situation it may well be the case that start step would be giving $\text{Polish}(A)$, and thus the failure of node B may no longer hold in that situation. To ensure the soundness of the learned rules, we must explicitly account for this possibility in explaining the failure of B . We do this by conjoining the condition $\neg \text{initially-true}(\text{Polish}(A))$ to the explanation failure at B (step 3.2.2).³

The explanation regressed over the establishment decision at B can be used to learn a useful step establishment rejection rule at A (since A still has unexplored alternatives). This rule is shown to the left of node A . It says that Roll should be rejected as a choice for establishing any condition at goal step G , if $\text{Polish}(A)$ is also a goal at the same step. Notice that the rule does not mention the specific establishment $\text{Cylindrical}(A)$, that lead to the introduction of Roll . This is correct because the failure explanation at node B does not involve $\text{Cylindrical}(A)$.⁴

³A more eager learning possibility would be to extend additional planning effort and see if B will have failed even if initial state were giving the open condition (as it would have, in the current case).

⁴It is interesting to note that in a similar situation, Prodigy [11] seems to learn a more specific rule which depends on establishing

Rule Generalization: Once a search control rule is made, it is generalized using the standard EBL process (c.f. [5, 10]). This process aims to replace any constants in the search control rule with variables, without affecting the rule correctness. In SNLP+EBL this is accomplished by doing the original regression process in terms of variables and their bindings (SNLP already provides support for this). During generalization, any bindings that are forced by the initial and goal state specifications of the original problem are removed from the explanation, leaving only those binding constraints that were forced by the initial explanation of the failure[5]. In the example in Figure 1, the binding $?x \approx A$ in the failure explanation of node B is stripped when making the generalized rule.

The generalization process also needs to generalize step names occurring in the failure explanation. Since the explanations qualify the steps in terms of their effects and conditions and their relations to other steps, most step names including $\text{fin}(G)$ can be generalized. The only exception to this rule is the status of the start step, which may or may not be generalizable based on the specifics of the explanation. To help in this decision, our regression rules explicitly flag start step as special when it must not be generalized. An example of this can be seen in the regression rules for step establishment decision in Figure 2. When an ordering of the form $\text{start} \prec s_1$ is regressed over the addition of step s_1 , the start step is flagged special since this ordering is automatically introduced as a result of step addition only with respect to the start step. When start is not flagged as special, it is generalized just as any other step. In the example in Figure 1, the rule learned after step and variable generalization is shown in a box to the left of node B .

Rule Storage: Once a rule is generalized, it is entered into the corpus of control rules available to the planner. These rules thus become available to the planner in guiding its search in the other branches during the learning phase, as well as subsequent planning episodes. In storing rules in the rule corpus, SNLP+EBL makes some bounded checks to see if an isomorphic rule is already present in the stored rules.

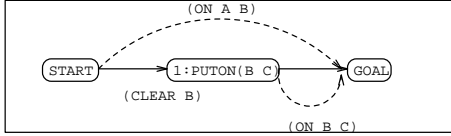
3 Learning from Depth limit Failures

In the previous section, we described the framework for learning search control rules from initial explanations of failed plans. As mentioned in that section, the only failures explainable by SNLP are the order and binding inconsistencies, which it detects during threat resolution (the unestablishable condition failure is rare in practical domains). The rules learned from such failures were successful in improving performance of SNLP in some synthetic domains (such as $D^m S^2^*$ described in [1]).

Unfortunately however, learning from analytical failures alone turns out to be ineffective in other recursive domains such as blocks world or job-shop scheduling. The main reason for this is that many futile lines of reasoning either never end in analytical failures or cross depth limits much before they do. Since depth limit failures are not analytical, no domain independent explanation can be given to these failures.

Cylindrical(A).

However, sometimes it is possible to use strong consistency checks based on the domain theory as well as the meta-theory of the planner to show that the partial plan at the depth limit contains a failure that the planner’s consistency checks have not yet detected. Consider for example a simplified blocks-world partial plan shown below:



Given the blocks world domain axiom that no block can have another block on top of it, and be clear at the same time, and the SNLP meta-theory that a causal link, $s_1 \xrightarrow{c} s_2$, once established, will protect the condition c in every situation between s_1 and s_2 , we can see that the above partial plan can never be refined into a successful plan. To generalize and state this formally, we define the *np-conditions*, or necessarily preservable conditions, of a step s' in a plan \mathcal{P} to be the set of conditions supported by any causal link, such that s' necessarily intercedes the source and destination of the causal link.

$$np\text{-conditions}(s') = \{c | s_1 \xrightarrow{c} s_2 \in \mathcal{L} \wedge s_1 \prec s' \wedge s' \prec s_2\}$$

Given the *np-conditions* of a step, we know that the partial plan containing it can never be refined into a complete plan as long as $precond(s') \cup np\text{-conditions}(s')$ is inconsistent with respect to domain axioms. However, SNLP’s local consistency checks will not recognize this, leading it sometimes into an indefinite looping behavior of repeatedly refining the plan in the hopes of making it complete. In the example above, this could happen if SNLP tries to achieve $Clear(B)$ at step 1 by adding a new step 2 : $Puton(x, y)$, and then plans on making $On(x, B)$ true at 2 by taking A off B , and putting x on B . When such looping makes SNLP cross the depth limit, SNLP+EBL uses the *np-conditions* based consistency check to detect and explain this implicit failure, and learn from that explanation.

To keep the consistency check tractable, SNLP+EBL utilizes a restricted representation for domain axioms (first proposed in [3]): each domain axiom is represented as a conjunction of literals, with a set of binding constraints. The table below lists a set of domain axioms for the blocks world. The first one states that y cannot have x on top of it, and be clear, unless y is the table.

$On(x, y) \wedge clear(y)[y \neq Table]$
$On(x, y) \wedge On(x, z)[y \neq z]$
$On(x, y) \wedge On(z, y)[x \neq z, y \neq Table]$

A partial plan is inconsistent whenever it contains a step s such that the conjunction of literals comprising any domain axiom are unifiable with a subset of conditions in $np\text{-conditions}(s) \cup precond(s)$.

Given this theory, we can now explain and learn from the blocks-world partial plan above. The initial explanation of this failure is: $start \xrightarrow{On(x,y)} G \wedge (start \prec 1) \wedge (1 \prec G) \wedge open\text{-cond}(Clear(y), 1) \wedge y \neq Table$. This explanation can be regressed over the planning decisions to generate rules.

- (1) **Reject establishment** $start \xrightarrow{On(x,y)} s_1$
If $open\text{-cond}(On(y, z), s_1) \wedge$
 $\neg initially\text{-true}(On(y, z)) \wedge \neg binds(y, Table)$
 - (2) **Reject promotion** $s_1 \prec s_3$
If $open\text{-cond}(clear(x_2), s_3) \wedge$
 $establishes(s_1, On(x_1, x_2), s_2) \wedge$
 $precedes(s_3, s_2) \wedge \neg binds(x_2, Table)$
 - (3) **Reject step addition** $puton(x', y) \xrightarrow{Clear(z)} s_1$
If $establishes(start, On(x, y), s_2) \wedge$
 $precedes(s_1, s_2) \wedge \neg binds(y, Table)$

Figure 4: A sampling of rules learned using domain axioms in Blocks world domain

The above theory can be used to learn from some of the depth limit failures. In blocks world, use of this technique enabled SNLP+EBL to produce several useful search control rules. Figure 4 lists a sampling of these rules. The first one is an establishment rejection rule which says that if $On(x, y) \wedge On(y, z)$ is required at some step, then reject the choice of establishing $On(x, y)$ from the initial state, if initial state is not giving $On(y, z)$.

3.1 Empirical Evaluation

To evaluate the effectiveness of the rules learned by SNLP+EBL, we conducted experiments on random problems in blocks world. The problems all had randomly generated initial states consisting of 3 to 8 blocks (using the procedure outlined in Minton’s thesis [10]). The first test set contained 30 problems all of which had random 3-block stacks in the goal state. The second test set contained 100 randomly generated goal states (using the procedure in [10]) with 2 to 6 goals. For each test set, the planner was run on a set of randomly generated problems drawn from the same distribution (20 for the first set and 50 for the second). Any learned search-control rule, which has been used at least once during the learning phase, is stored in the rule-base. This resulted in approximately 10 stored rules for the first set, and 15 stored rules for the second set. (It is interesting to note that *none* of these rules were learned from analytical failures.)

In the testing phase, the two test set problems were run with SNLP, SNLP+EBL (with the saved rules) as well as SNLP+DOMAX, a version of SNLP which uses domain axioms to prune inconsistent plans as soon as they are generated. A cpu time limit of 120 seconds was used in each test set.

Table 1 describes the results of these experiments. Figure 5 shows the cumulative performance graphs for the three methods in the second test set. Our results clearly show that SNLP+EBL was able to outperform SNLP significantly on these problem populations.⁵ SNLP+EBL also outperforms SNLP+DOMAX, showing that learning search-control rules

⁵The experiments reported here were all done on the standard public domain SNLP implementation of Barrett and Weld [1]. In addition, we also experimented with more optimized implementations of SNLP including those that do not resolve positive threats (and hence are not systematic), and avoid separation by defining threats in terms of necessary codesignation [14]. The qualitative relations

Num Prob	SNLP		SNLP+EBL		SNLP+DOMAX	
	% Sol	C. tim	%Sol	C. tim	%Sol	C. tim
I (30)	60%	1767	100%	195	97%	582
II (100)	51%	6063	81%	2503	74%	4623

Table 1: Results from the blocks world experiments

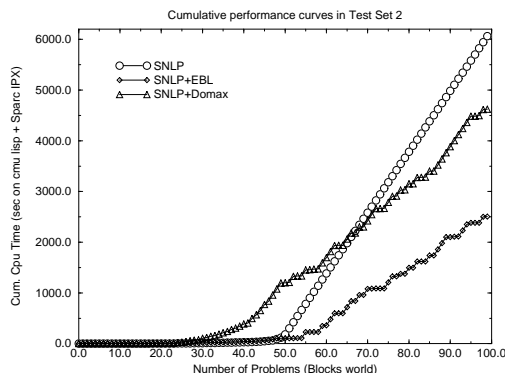


Figure 5: Cumulative performance curves for Test Set 2

is better than using domain axioms directly as a basis for stronger consistency check on every node during planning. This is not surprising since checking consistency of every plan during search can increase the refinement cost unduly. EBL thus provides a way of strategically applying stronger consistency checks.

4 Related Work

As we noted earlier, our work on SNLP+EBL was motivated by the desire to adapt the EBL frameworks developed for state-based planning, such as PRODIGY/EBL [10] and FailSafe [2], to partial order planning. Our use of domain axioms to detect and explain failures at depth limits is related to, and inspired by Bhatnagar's work on FailSafe [2]. Bhatnagar also advocates starting with over-general explanations of failure and relaxing the rules in response to future impasses. The rules learned in SNLP+EBL, in contrast, are always sound in that any path rejected by a rejection rule is guaranteed to fail. Domain axioms have been used by other researchers in the past to control search in PO planning (c.f. [7, 3]). Our use of domain axioms is closest to the work of Kambhampati [7], who uses an idea similar to *np-conditions* to implement a minimal-conflict based heuristic for controlling refitting in plan reuse. The current work shows that EBL provides a way of strategically applying domain axiom based consistency checks. Finally, although we did not explicitly address monitoring the utility of learned rules and filtering bad rules, we believe that utility monitoring models developed for state-based planners [4, 10] will also apply for PO planners.

5 Conclusions and Future Work

In this paper, we presented SNLP+EBL, the first systematic implementation of explanation-based search control rule learning to a PO planner. We have described the details of

between SNLP, SNLP+EBL and SNLP+DOMAX remained same even in the presence of these optimizations.

the regression, explanation propagation and rule generation process in SNLP+EBL. We have then proposed a general methodology for learning from planning failures, viz., using a battery of stronger consistency checks based on the meta-theory of the planner, and the domain theory of the problem, to detect and explain failures at depth limits. We described a specific instantiation of this method, which uses domain axioms to look for inconsistencies in the plans at depth limits, and presented experimental results that demonstrate its effectiveness. Although our EBL framework was developed in the context of SNLP we believe that it can be easily extended to more powerful PO planners such as UCPOP [13].

Learning from domain axiom based failures alone may not be sufficient in domains which do not have any strong implicit domain theory. We are currently working towards identifying other types of stronger consistency checks which can be used to complement the domain axiom based techniques in such domains. One example involves utilizing domain specific theories of loop detection to avoid step-based looping.

References

- [1] A. Barrett and D.S. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, Vol. 67, No.1, 1994.
- [2] N. Bhatnagar. *On-line Learning From Search Failures* PhD thesis, Rutgers University, New Brunswick, NJ, 1992.
- [3] M. Drummond and K. Curry. Exploiting Temporal coherence in nonlinear plan construction. *Computational Intelligence*, 4(2):341-348, 1988.
- [4] J. Gratch and G. DeJong. COMPOSER: A Probabilistic Solution to the Utility problem in Speed-up Learning. In *Proc. AAAI 92*, pp:235--240, 1992
- [5] S. Kambhampati and S. Kedar. A unified framework for explanation based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence*, Vol. 67, No. 2, 1994.
- [6] S. Kambhampati and J. Chen. Relative Utility of EBG based Plan Reuse in Partial Ordering vs. Total Ordering Planning. In *Proc. AAAI-93*, pp:514--519, 1993.
- [7] S. Kambhampati. Exploiting Causal Structure to Control Retrieval and Refitting during Plan reuse. *Computational Intelligence*, 10(2), May 1994.
- [8] S. Katukam. *Learning Explanation-Based Search Control Rules for Partial Order Planning*. Masters Thesis, Arizona State University, Tempe, AZ, 1994. (*forthcoming*).
- [9] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning In *Proc. AAAI-91*, 1991.
- [10] S. Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1988.
- [11] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni and Y. Gil. Explanation-Based Learning: A Problem Solving Perspective. *Artificial Intelligence*, 40:63--118, 1989.
- [12] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, 1980.
- [13] J.S. Penberthy and D.S. Weld. UCPOP: A sound, complete partial order planner for ADL. In *Proc. KRR-92*, 1992.
- [14] M. Peot and D. Smith. Threat removal strategies for Nonlinear Planning. In *Proc. 11th AAAI*, 1993.