Learning Search Control rules for Plan-space Planners: Factors affecting the performance

Yong Qu and **Subbarao Kambhampati***
Department of Computer Science and Engineering,
Arizona State University, Tempe, AZ 85287-5406
Email: {yqu, rao}@asu.edu
WWW: http://rakaposhi.eas.asu.edu:8001/yochan.html

Appears in Current Trends in AI Planning: EWSP '95, IOS Press

Abstract

Given the intractability of domain-independent planning, learning effective search control knowledge is vitally important. One way of learning search control knowledge is to use explanation based learning (EBL) methods. This paper aims to analyze and understand the factors influencing the effectiveness of EBL. We present an EBL framework for UCPOP, a partial order planner, and use it to systematically analyze the effect of (i) expressive action representations (ii) domain specific failure theories and (iii) sophisticated backtracking strategies on the utility of EBL. Through empirical studies, we demonstrate that expressive action representations allow for more explicit domain representations which in turn increase the ability of EBL to learn from analytical failures, and obviate the need for domain specific failure theories. We also explore the strong affinity between dependency directed backtracking and EBL in planning.

1 Introduction

Given the exponential worst case complexity of domain independent planning, there is a significant interest in improving the search efficiency of a planner over a given problem population. One way of doing this involves learning search control rules to customize the planner's search to the expected problem population. Control rule learning has originally been developed in the context of state space planning [8, 1]. More recently, we extended it to plan-space planners [6], and developed SNLP+EBL, which learns control rules for the partial order planner SNLP.

Although our work with SNLP+EBL [6] showed that control rule learning is an effective way of improving the performance of a plan space planner, it also brought up the critical dependencies between the effectiveness of control rule learning, and a variety of other

*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. Special thanks to Laurie Ihrig for many critical comments on drafts of the paper, and Suresh Katukam for his help in implementing UCPOP+EBL.

factors, including the expressiveness of the action representation used by the planner, the types of backtracking strategies being employed by the planner as well as the type of goal selection heuristics used by the planner. For example, we found that in the propositional planning domains that we experimented with, SNLP+EBL was not able to learn effective control rules unless there is an accompanying domain specific theory of failure (usually in the form of domain axioms). This brings up the importance of domain representation, and poses the question as to whether a more expressive action description language might allow richer domain representations and thereby reduce the dependence on outside failure theories. Similarly, we noted that the analysis done in learning control rules is very similar to the analysis required to support dependency directed backtracking (ddb). Since ddb itself can improve the performance of planning to some extent, it is important to understand how it interacts with the use of learned control rules. The current research is aimed at understanding the influence of these factors on the effectiveness of EBL.

To facilitate the analysis, we start by extending our control rule learning framework to UCPOP, a partial order planner that is powerful enough to handle a larger class of planning domains including those that contain actions with conditional and quantified effects, as well as quantified and disjunctive preconditions. The resulting system, UCPOP+EBL, is used as the basis for a systematic empirical investigation of the effect of (i) expressive action representations (ii) domain specific failure theories and (iii) sophisticated backtracking strategies on the utility of EBL. We will show that expressive representations allow us to make the relations between the preconditions and effects of the actions more explicit, thereby increasing the effectiveness of learning control rules from analytical failures alone. This in turn reduces the need for domain specific failure theories to guide EBL. We will also demonstrate the strong affinity between dependency directed backtracking and control rule learning, and clarify as to when EBL can provide savings over and above those offered by dependency directed backtracking.

The rest of this paper is organized as follows. Section 2 contains an overview of the learning framework used in UCPOP+EBL. Apart from reviewing the details of the base level planner, and the way control rules are synthesized and generalized from failures detected during search (which is a straightforward extension of SNLP+EBL algorithm described in [6]), the overview will also bring out the connections between EBL, dependency directed backtracking and domain representation. Section 3 provides an evaluation of UCPOP+EBL and also describes the results of a focussed empirical study to analyze the factors affecting the performance of UCPOP+EBL. Finally, Section 4 presents our conclusions. A more detailed description of the design and analysis of UCPOP+EBL can be found in [7]. That paper also explains in detail the differences between SNLP+EBL [6] and UCPOP+EBL.

2 Overview of UCPOP+EBL

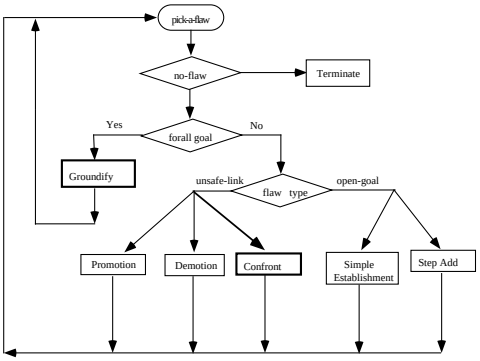
2.1 Base level Planner: UCPOP

Like SNLP [4], UCPOP [14] searches in a space of partial plans, refining (adding constraints to) a partial plan until it becomes a complete solution to the planning problem. Figure 1(a) shows the description of a simple example domain for UCPOP called the briefcase domain [14, 13], which involves moving objects from one location to another with the help of a briefcase. Note that the actions contain conditional and quantified effects.

A partial plan in UCPOP is best seen as a set of step, ordering, binding and causal link

mov-b(l, m) ;; Move B from **l** to **m**
precond: $m \neq l \wedge at(B, l)$
eff: $at(B, m) \wedge \neg at(B, l)$
$\forall_{obj(x)} in(x) \Rightarrow at(x, m) \wedge \neg at(x, l)$
take-out(o) ;;Take out **o** from **B**
precond: $in(o)$
eff: $\neg in(o) \wedge \neg closed(B)$
close-b() ;; Close the briefcase
precond: $\neg closed(B)$
eff: $closed(B)$

(a) The Briefcase Domain



(b) Flowchart of refinement process in UCPOP

Figure 1: *Example domain and Flowchart of UCPOP*

constraints. The constraint set corresponding to a partial plan can be represented compactly as a 6-tuple: $\langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}, \mathcal{E}, \mathcal{C} \rangle$, where $\mathcal{S}$ is a set of **s**teps, $\mathcal{O}$ is a set of **o**rdering constraints over $\mathcal{S}$ (denoted by ''$\prec$''), $\mathcal{B}$ is a set of codesignation (**b**inding) and noncodesignation (prohibited bindings) constraints on the variables appearing in the preconditions and post-condition (effects) of the operators (denoted respectively by ''$\approx$'' and ''$\not\approx$''), $\mathcal{L}$ is a set of causal **l**inks, $\mathcal{E}$ is the set of effects of the steps (of the form `has–effect`(s, c)), and $\mathcal{C}$ is the set of preconditions of steps in the plan (of the form $c@s$). Each plan is also associated with a datastructure $\mathcal{A}$, which is an agenda of ''flaws'' in the partial plan to be worked on. There are two types of flaws: preconditions that need to be established (called **open condition** flaws), and causal links that need to be protected (called **unsafe link** flaws).

Given a planning problem consisting of the initial state $\mathcal{I}$, and the set of goals $\mathcal{G}$ to be achieved, UCPOP starts with a null plan with two dummy steps 0 and G, corresponding to the initial state and the goal state. The effects of 0 correspond to the initial state conditions (the notation `init–true`(c) is used as a shorthand for `has–effect`$(0, c)$), while the preconditions of G correspond to the set of goals to be achieved. For every goal condition g, an open condition flaw $g@G$ is inserted into the preconditions list $\mathcal{C}$ and the flaw list $\mathcal{A}$. Planning consists of repeatedly selecting a flaw and resolving it, until a partial plan without any flaws is produced. Figure 1(b) shows a flow chart of the plan-refinement process in UCPOP.

Resolving Open Condition flaws: A partial plan P is said to have an open condition flaw $p@s$, if a precondition p is needed at step s, and the plan does not contain a causal link supporting this condition. UCPOP resolves the open condition flaw $p@s$ by selecting some contributor step s' (either new or existing), and constraining the plan to make an effect e of s' *establish* p at s. The constraints added to ensure this establishment include ordering s' to come before s, and adding a sufficient number of bindings to ensure that the effect e of s' necessarily unifies with p. Additionally, if the effect e is a conditional one of the form ''$r \Rightarrow e$'' (signifying that if the condition r is true before the execution of the action, then e will be produced after the execution), then the condition r is made a (secondary) precondition of s', to be considered for establishment later. The different ways of establishing an open condition serve as backtrack points for the search. Universally quantified open condition flaws are handled by converting the quantified formula into a conjunction over the objects that are spanned by the quantification.

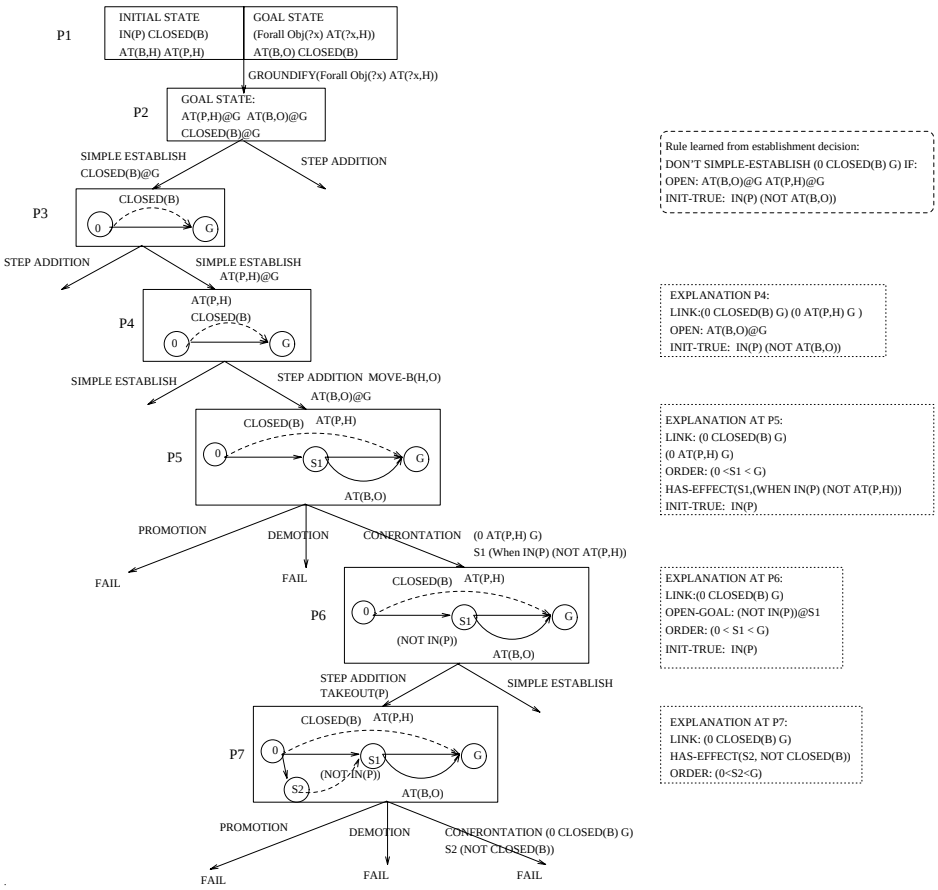


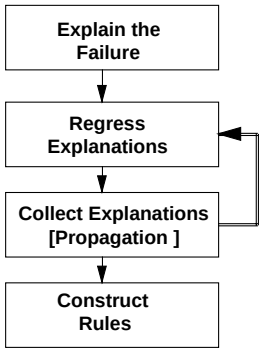
Figure 2: *Trace of UCPOP+EBL solving a problem in the briefcase domain: A failing path. (used as the Running Example)*

Resolving unsafe link flaws: A partial plan is said to have an *unsafe link* flaw, if it has a causal link $s' \xrightarrow{p} s$, and a step s'' such that s'' can possibly come between s' and s and has an effect $\neg e$ such that e necessarily unifies with p. The unsafe link is denoted by $s' \xrightarrow{p} s \otimes s''$, and s'' is referred to as a **threat** to the link $s' \xrightarrow{p} s$. UCPOP resolves the unsafe link flaw by either making s'' come before s' (**demotion**) or making s'' come after s (**promotion**). Additionally, if the threatening effect is a conditional effect of the form $r \Rightarrow \neg e$, then the threat can also be resolved by making $\neg r$ a precondition of s'' (**confrontation**), such that $\neg e$ will not be true after s'', and inserts an open condition $\neg r@s''$ into the flaw list.

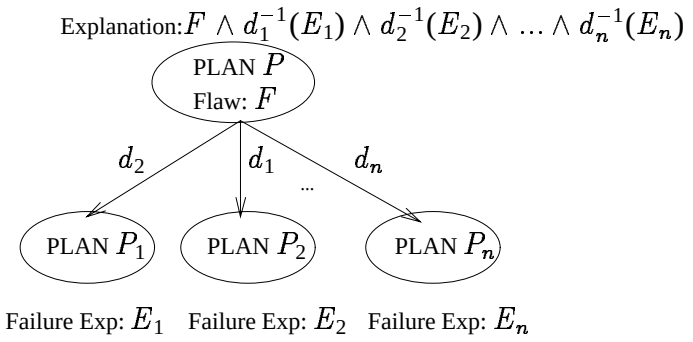
Example: Consider the problem of getting an empty briefcase to the office, while leaving everything else at home. Suppose the pay check P is in the briefcase, and the briefcase is at home in the initial state. The goal for this problem is specified by the formula $\forall_{obj(x)} at(x, H) \wedge at(B, O) \wedge closed(B)$. Figure 2 shows a trace of UCPOP solving this problem (for now ignore the explanation boxes on the right).

2.2 Explanation Based Learning in UCPOP

In response to search failures, UCPOP+EBL learns control rules that will steer the planner away from the failing branches of the search space. These rules can be used to avoid search failures in similar circumstances in the future. In this section, we will illustrate the process of control rule learning in UCPOP+EBL. Figure 3 gives an overview of this process. Any



(a) Schematic of UCPOP+EBL



(b) Computing failure explanation of a plan given the failure explanations of its children

Figure 3: *Schematic overview of control rule learning process in UCPOP+EBL*

time a failing partial plan P is encountered, the learning process starts by constructing an initial explanation E for the failure of P. The explanation is the minimal set of constraints in P that are together inconsistent. The regression and propagation processes are then used to abstract failure information from the leaves of the search tree to higher level partial plans in the tree. Specifically, given a search node corresponding to a partial plan P such that all the search branches below P terminate in failing plans, regression and propagation help in computing a minimal explanation of failure for P. In the following, we provide the details of the regression and propagation process.

2.2.1 Failures & Explanations

UCPOP+EBL flags a partial plan to be a failing plan in a variety of circumstances. In each case, the explanation of failure is the minimal set of constraints that are together inconsistent. Some of the failing plans are detected by the consistency checks by the planner as a matter of course. They include inconsistent orderings (e.g. $(s_1 \prec s_2), (s_2 \prec s_1) \in O$) , inconsistent bindings (e.g. $(x \approx y), (x \not\approx y) \in \mathcal{B}$), inconsistent preconditions (e.g. $p@s_1, \neg p@s_1 \in \mathcal{A}$), and inconsistent effects (e.g. $\texttt{init\_true}(x), \texttt{init\_true}(\neg x)$). As discussed in [6, 1], these analytical failures alone are not enough to detect and terminate every failing branch before they cross search depth limits. In such cases, UCPOP+EBL can utilize any available domain specific failure theories to analyze implicit failures in the plans crossing depth limits. For example, in the briefcase domain, domain axioms such as $\forall_{x,l,l'} at(x,l) \wedge l \neq l' \supset \neg at(x,l')$, that states that an object cannot be at two places at the same time, can be used to prove that any plan that contains the constraints $s_1 \overset{at(P,H))}{\rightarrow} s_2 \wedge (s_1 \prec s_3 \prec s_2) \wedge at(P,O)@s_3$ is a failing plan.

2.2.2 Regression and Propagation

Regression and Rule Learning: After an initial failure explanation E is constructed for the failing plan, it is regressed up through decision d leading to the failing plan to compute the weakest conditions E' that need to be true in the plan before d such that the failure will result after d. Given E', we can construct a control rule that rejects d whenever E' is true

Confront$((s_1 \overset{r_1}{\rightarrow} s_2 \otimes s_3)(p) \Rightarrow \neg r_2)$
precond:
$s_1 \overset{r_1}{\rightarrow} s_2 \wedge r_1 \approx r_2$
`has-effect`$(s_3, (p) \Rightarrow \neg r_2)$
Eff: $\neg p@s_3$

(a) Confrontation decision as a STRIPS-style operator

Regress(c, d) = True if $c \in effects(d)$ (i)
$= c'$ if $c'' \in effects(d)$ and $(c'' \wedge c') \vdash c$ (ii)
$= c$ Otherwise (iii)
Regress$(c_1 \wedge c_2 \cdots \wedge c_n, d)$
$= Regress(c_1, d) \wedge Regress(c_2, d) \cdots \wedge Regress(c_n, d)$

(b) Regressing constraints over decisions

Figure 4: *Regression in UCPOP+EBL*

in the current plan. For regression purposes, it is useful to think of UCPOP decisions as STRIPS-style operators operating on partial plans. The preconditions of these operators are specified in terms of the constraints that need to be present in a partial plan for the decision d to be applicable, and the effects are specified in terms of the constraints that are *added* to the partial plan as a result of adding d. Figure 4(a) shows the confrontation decision in this format, and Figure 4(b) summarizes the process of regressing an explanation of failure over a decision. Regression of a constraint c over a decision d is $true$, if d is added by c, and c itself if c is not added by d. If c is a transitive constraint (such as an ordering or a binding constraint), then regression of c over d is some set of constraints c' such that c', together with the constraints added by d will entail c.

Example: In our briefcase domain example, when the planner tries to resolve the unsafe link $(0 \overset{close(B)}{\rightarrow} G \otimes s_2)$, the demotion option fails because of the ordering inconsistency $0 \prec s_2 \wedge s_2 \prec 0$. When this explanation is regressed over the demotion decision, the constraint $s_2 \prec 0$ regresses to $true$ (since the demotion decision adds this ordering), while the constraint $0 \prec s_2$ regresses to itself. Thus the combined regression of the failure of the demotion branch is $0 \prec s_2$. We can learn a (trivial) control rule stating that demotion should be avoided whenever the threat s_2 precedes the producer step. Similarly, the failure explanations for promotion and confrontation branches can be regressed over the respective decisions.

Propagation: When all the branches under a partial plan P are failing, we can construct an explanation of failure for P itself. Figure 3(b) illustrates this process. Specifically, suppose F is the description of the flaw in P that is being worked on, and $d_1, d_2 \cdots d_n$ are the decisions corresponding to the various ways of removing F, and $E_1, E_2, \cdots E_n$ are the explanations of failure of the plans resulting from each of those decisions. Then the explanation of failure of P is given by$F \wedge d_1^{-1}(E_1) \wedge d_2^{-1}(E_2) \cdots \wedge d_n^{-1}(E_n)$ (where ''d^{-1}'' denotes regression over d). When each of P's siblings have also failed and the explanations of these failures have been constructed, the explanation of failure for P's parent plan can be constructed. In this manner, information about failures is propagated up the failing subtree.

Propagation and Dependency Directed Backtracking: Sometimes, we do not need to wait for all the branches under P to fail before computing the failure explanation of P. Suppose, in the situation shown in Figure 3(b), one of the explanations E_i is such that regressing it over the corresponding decision does not change it; i.e., $d_i^{-1}(E_i) = E_i$. Since E_i is a set of inconsistent constraints (recall that E_i is the explanation of failure of the i^{th}

child of P), and E_i is present in P, E_i a sufficient explanation of the failure of P. Given that we already have an explanation of the failure of P, *we do not need to continue exploring any as yet unexplored branches under* P. This is because decisions in partial order planning can only *add*, but not *remove* any constraints. Since P already contains a set of inconsistent constraints E_i, any refinement of P is also doomed to be a failing plan, and consequently no further refinements of P need be considered. In other words, *the regression and propagation processes that drive the control rule learning process also provide a powerful form of dependency directed backtracking*.

Example: Continuing the briefcase example shown in Figure 2, since all the branches under the plan P_7 fail, the partial plan P_7 itself fails. The failure explanation is constructed by conjoining the regressed failure explanations of the three branches with the description of the flaw in P_7 that is being removed. Specifically, we get the failure explanation of P_7 as

$$E_7 : \underbrace{0 \overset{closed(B)}{\rightarrow} G \wedge (s_2 \not\prec 0) \wedge (G \not\prec s_2) \wedge \texttt{has-effect}(s_2, \neg closed(B))}_{\text{unsafe link flaw}} \wedge \underbrace{(0 \prec s_2) \wedge (s_2 \prec G)}_{\text{regressed fromchildren}}$$

(The ordering constraints simplify to $0 \prec s_2 \prec G$.) This explanation is then regressed over the step addition decision $STEP\text{–}ADD(\texttt{take–out}(P))$, and the process continues as shown in Figure 2, eventually computing the failure explanation of P_3 as

$$E_3 : 0 \overset{closed(B}{\rightarrow} G \wedge at(B, O)@G \wedge at(P, H)@G \wedge \texttt{init–true}(in(P)) \wedge \texttt{init–true}(\neg at(B, O))$$

When E_3 is regressed over the simple establishment decision under P_2, we get $at(B, O)@G \wedge at(P, H)@G \wedge \texttt{init–true}(in(P)) \wedge \texttt{init–true}(\neg at(B, O)) \wedge \texttt{init–true}(closed(B))$. This leads to a useful control rule, shown at the top right corner of Figure 2, which states that simple establishment of $closed(B)@G$ should be avoided when the paycheck is in the briefcase, briefcase is not at office, and we want the briefcase to be at the office and paycheck to be left at home.

2.2.3 Generalization

Once the search control rule is made, it is generalized using the standard EBL process. This process aims to replace any problem-specific constants in the search control rule with variables, without affecting the rule correctness. There are two types of constants that need to be generalized -- the step names and the object names. In UCPOP+EBL both generalizations are accomplished by doing the original regression process in terms of variables and their bindings; the UCPOP code 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, or by the decisions themselves.

Example: In the brief case example, the generalized form of the rule learned at P_2 will be to reject simple establishment of $closed(?b)@?s$ from initial state, **if** $at(B, ?o)@?s \wedge at(?p, ?h)@?s \wedge (?o \not\approx ?h) \wedge \texttt{init–true}(in(?p)) \wedge \texttt{init–true}(\neg at(B, ?o))$.

Domain	From-scratch		Online Learning		Using learned rules	
	% Solv	cpu	% Solv	cpu	% Solv	cpu
Brief Case Dom	49%	6568	92%	1174 (*5.6X*)	98%	1146 (*5.6X*)
Blocks World	53%	7205	100%	191(*38X*)	100%	190 (*38x*)

Table 1: Performance of UCPOP+EBL in Blocks world and Briefcase Domain

3 Experimental Evaluation

In this section we will describe empirical studies with our implementation of UCPOP+EBL. The objectives of our experiments were two fold. First, to show that UCPOP+EBL provides an effective means of improving planning performance, and second, to understand the factors that influence the effectiveness of EBL in planning.

3.1 Performance of UCPOP+EBL

To evaluate the performance of UCPOP+EBL we conducted experiments in two different domains -- the first one is a variant of the briefcase domain (Figure 1(a)) that has multiple locations, and multiple objects to be transported among those locations using the briefcase. This domain is similar in character to the logistics transportation domain described in [16], except with conditional and quantified effects. We generated 100 random problems containing between 3 to 5 objects, 3 to 5 locations and between 3 to 5 goal conjuncts. The second domain is the blocks world domain called `BW-quant` described in Figure 5. We generated 100 random problems using the procedure described in [9]. The problems contained between 3 to 6 blocks, and 3 to 4 goals. In each domain, we compared the performance of the from-scratch planner with that of the planner using the search control rules generated by UCPOP+EBL. Table 1 shows the results of these experiments. As can be seen, UCPOP+EBL achieves significant savings in performance in both the domains, both in terms of the number of problems solved, and the speedup obtained. To gauge the cost of learning itself, we also ran UCPOP+EBL in an ''*online learning*'' mode, where it continually learns control rules and uses them in the future problems. The statistics in the *online learning* column show that the cost of learning does not outweigh its benefits.

3.2 Factors influencing the effectiveness of UCPOP+EBL

Although learning search control rules is an attractive way of improving planning performance, there are a variety of factors that can affect the utility of control rule learning in a given domain. In particular, the nature of the domain theory, and the nature of the base level search strategy used by the planner can have a significant impact on the effectiveness of learning control rules from analytical failures. Furthermore, as discussed elsewhere, the ability of EBL to learn control rules crucially depends on detecting and explaining failures in the partial plans before they cross depth limits [6]. This in turn depends on the nature of the domain theory (viz, how much information is left implicit and how much is represented explicitly), and the availability of domain specific theories of failure (c.f. [6, 1]). Finally, availability of sophisticated dependency directed backtracking strategies can directly compete with the performance improvements produced through learned control rules.

We have used our implementation of UCPOP+EBL to investigate the effect of these factors. In this section, we will describe the results from these studies, and analyze them.

```
(defun   BW-prop ()
(define (operator newtower)
:precondition (and (on ?x ?z) (clear ?x) (neq ?x ?z)
                   (block ?x) (block ?z) (Tab Table))
:effect (and (on ?x Table) (clear ?z) (not (on ?x ?z)))))
(define (operator puton)
:precondition (and (on ?x ?z) (clear ?x)  (clear ?y) (neq ?x ?y) (neq ?x ?z)
                   (neq ?y ?z) (block ?x) (block ?y) (block z))
:effect  (and (on ?x ?y) (not (on ?x ?z)) (clear ?z) (:not (clear ?y)))))
```

```
(defun BW-cond ()
(define (operator puton)
:precondition (and (on ?x ?z) (clear ?x) (clear ?y)
                   (neq ?x ?z) (neq ?x ?z)
                   (neq ?x ?y) (block ?x))
:effect (and (on ?x ?y) (not (on ?x ?z))
             (when (block ?z) (clear ?z))
             (when (block ?y) (not (clear ?y)))))
```

```
(defun univ-bw ()
(define (operator puton)
:precondition
      (and (on ?x ?z) (neq ?x ?y) (neq ?x ?z) (neq ?y ?z)
      (or (tab ?y)
          (:forall (block ?b) (:not (on ?b ?y))))
      (:forall (block ?c) (:not (on ?c ?x))))
:effect (:and (on ?x ?y) (:not (on ?z ?x))))
```

Figure 5: *Three different Descriptions of Blocksworld*

Domains: To evaluate the effect of expressive representations on the performance of EBL systems, we tried three different domain theories of the blocks world domain, as shown in Figure 5. The first, called `BW-prop`, contains two types of predicates, On and $Clear$, and two actions, $puton$ and $newtower$ with no conditional effects. The second, called `BW-cond`, contains the same two predicates, but with a single $puton$ action with conditional effects. The third domain, called `BW-quant`, contains a single predicate On, with the condition $Clear(x)$ replaced by the quantified formula $Table(x) \vee \forall_y \neg On(y, x)$. Note that `BW-prop` is forced to make a choice between its two actions, while `BW-cond` and `BW-quant` don't have to make such premature commitment. Similarly, because of their restricted language, `BW-cond` and `BW-prop` are forced to hide the relation between $Clear$ and On, while the expressive language of `BW-quant` allows it to make the relation explicit.

Experimental setup: The experiments consisted of three *phases*, each corresponding to the use of one of the three domain descriptions above. In each phase, the same set of 100 randomly generated blocksworld problems were used to test the from-scratch and learning performance of UCPOP+EBL, and statistics regarding the number of problems solved and the cumulative cpu time were collected. To understand the effect of domain specific failure theories on the performance, we ran UCPOP+EBL in three different *modes*. In the first mode, UCPOP+EBL's learning component was turned off, and the problems were solved from scratch by the base level planner, UCPOP. In the second mode, UCPOP+EBL was trained over the problems, and the control rules it learned from the analytical failures alone were used in solving the test set. The third mode was similar to the second mode except that UCPOP+EBL was also provided domain specific theories of failure in the form of domain axioms (such as the one stating that $\forall_{x,y} clear(x) \supset \neg On(y, x)$), which could be used to detect and explain failures that would otherwise be not detected by the base level planner.

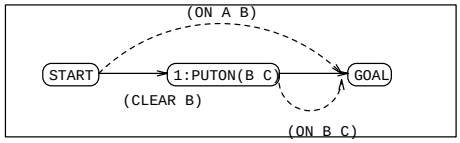
Since it is well-known that the performance of a plan-space planner depends critically on the order in which open condition flaws are handled (goal selection order) [3], we experimented with two goal-selection strategies -- one which corresponds to a LIFO strategy and one that works on goals with the least number of variable left uninstantiated (mimicking the least-cost flaw refinement strategy, [3]).

Results: Table 2 shows the statistics from these experiments. The performance of the base level planner varies considerably across the three domains and the two goal selection strategies. What is more interesting, the effectiveness of the learned control rules in improving the planning performance also varies significantly, giving a speedup anywhere between 0.95x and 38x. We will now analyze this variation with respect to several factors.

Domain	I. Scratch		II. EBL with analytical failures			III. EBL with dom. spec. fail. theories		
	% Solv	cpu	% Solv	cpu	# rules.	% Solv	cpu	# rules.
Achieving most instantiated goals first								
BW-prop	51%	7872	69%	5350 (*1.5x*)	24	68%	5410 (*1.5x*)	28
BW-cond	89%	2821	88%	2933 (*0.96x*)	15	91%	2567 (*1.1x*)	37
BW-quant	53%	7205	100%	210(*34x*)	4	100%	210 (*34x*)	4
Achieving goals in a LIFO order								
BW-prop	10%	13509	10%	13505 (*1x*)	30	10%	13509 (*1x*)	30
BW-cond	42%	9439	60%	6954 (*1.4x*)	14	75%	4544 (*2.1x*)	36
BW-quant	81%	3126	89%	2136 (*1.5x*)	32	94%	1699 (*1.8x*)	37

Table 2: *Performance of UCPOP+EBL in the three blocks world domains*

Expressive Domain Theories and Analytical Failures: The results show that the magnitude of improvement provided by UCPOP+EBL when learning from analytical failures alone (mode II) depends on the nature of the domain theory. For example, we note that search control rules learned from analytical failures improve performance more significantly in BW-quant than they do in BW-prop and BW-cond. This can be explained by the fact that in the latter two domains, in many cases, the base level planner is not able to detect any analytical failures in various branches before the planner crosses the depth-limit. In contrast, the explicitness of the domain description in BW-quant enables the planner to detect and explain analytical failures in many more situations. To illustrate this point, consider the blocks world partial plan:



Given the relation between $Clear$ and On predicates, it is clear that this plan cannot be refined into a solution (since it is impossible to protect the condition $On(A, B)$ while still ensuring the precondition $Clear(B)$ of step 1). However, since the relation between $Clear$ and On is not explicit in BW-prop and BW-cond, the planner may not recognize this failure in those domains before crossing the depth limit (unless some domain specific theories of failure are provided). In contrast, in BW-quant, the precondition $Clear(B)$ will be expressed as the quantified condition $\forall_y \neg On(y, B)$, and the planner will immediately notice an analytical failure, when trying to add a step to achieve $\neg On(A, B)$ at step 1.

Expressive domain theories and domain specific failure theories: We note that the availability of domain specific theories of failure does not uniformly improve performance of EBL. In particular, we see a bigger improvement in BW-prop than we do in the other two domains (Table 2). This can be explained by the fact that the information in the domain axioms (which constitute our domain-specific theories of failure), is subsumed to a large extent by the information in the quantified preconditions and effects of the actions in BW-quant. The situation is opposite in the case of BW-cond, and it benefits from the availability of domain specific theories of failure.

Importance of Explainable Failures: Another interesting point, brought about by the results above is the correlation between the performance of the base level planner and the performance of the planner in the presence of learned control rules. We note that the

Domain	I. Scratch		II. EBL with analytical failures			III. EBL with dom. spec. fail. theories		
	% Solv	cpu	% Solv	cpu	# rules.	% Solv	cpu	# rules.
Achieving most instantiated goals first								
BW-prop	71%	5093	61%	6613 (*0.8x*)	24	70%	5193 (*1.0x*)	28
BW-cond	89%	2837	88%	2983 (*0.8x*)	15	95%	1835 (*1.6x*)	37
BW-quant	100%	197	100%	190(*1.03x*)	4	100%	190 (*1.03x*)	4
Achieving goals in a LIFO order								
BW-prop	22%	12001	21%	12054 (*0.97x*)	30	21%	12080 (*0.98x*)	36
BW-cond	42%	9439	60%	7666 (*1.2x*)	14	75%	4544 (*2.1x*)	29
BW-quant	90%	1640	96%	1175 (*1.4x*)	32	98%	1146 (*1.4x*)	37

Table 3: *Performance of UCPOP+EBL in the three blocks world domains, when the base level planner uses a dependency directed backtracking strategy*

planner performs poorly in the BW-quant domain, compared to BW-cond domain in the from-scratch mode, but out-performs it with learning. At first glance, this might suggest the hypothesis that the planner that makes more mistakes in the from-scratch phase has more *opportunities* to learn from. This hypothesis is not strictly true -- in particular, it is not the number of mistakes, but rather the number of *explainable mistakes* that provide learning opportunities. As an example, BW-prop, which also does worse than BW-cond in from-scratch mode, continues to do worse with learning.

Effect of Sophisticated Backtracking Strategies: One other factor that influences the utility of control rules learned from EBL is the default backtracking strategy used by the planner. In Section 2.2.2, we noticed that the analysis being done by UCPOP+EBL in learning control rules also helps it do a powerful form of dependency directed backtracking (ddb). To understand how much the improvement brought about by dependency directed backtracking affects the utility of control rules learned through the EBL analysis, we repeated our experiments while making the base level planner use dependency directed backtracking. Table 3 shows these results.

We note that while the improved performance of the planner in the from-scratch mode reduces the relative speedup produced by control rules, at least in the case of the second goal selection strategy, control rules do bring out significant additional savings over ddb (see Table 2). This tendency was also confirmed by our experiments in the briefcase domain (not shown in the table). It can be explained by the fact that although ddb captures much of the analysis done while learning control rules, it is effective *only after a failure has been encountered, and backtracking is started*. In contrast, control rules attempt to steer the planner away from the failing branches in the first place. Another factor is that while ddb techniques typically tend to exploit only the analytical failures detected by the planner, the control rules learning may also benefit from domain specific failure theories.

In comparing the statistics in Table 2 and Table 3, we note that there is a strong correlation between the planner's ability to improve its performance through ddb, and its ability to learn useful control rules from analytical failures alone. For example, BW-cond is unable to improve its performance with ddb or with control rules learned from analytical failures. This is not surprising since lack of detectable analytical failures will hurt both the effectiveness of ddb and that of EBL. (Similar relation has been observed in the constraint satisfaction literature between backjumping and learning [2]).

4 Conclusion

Learning search control rules for plan-space planning is an important problem that has received relatively little attention. This paper presents an empirical analysis of the factors that influence the effectiveness of explanation based search control rule learning for partial order planners. To facilitate this analysis, we first presented UCPOP+EBL, that extends our previous work on control rule learning for a propositional partial order planner, SNLP [6] to UCPOP, a planner that is powerful enough to handle actions with conditional and quantified effects, as well as quantified and disjunctive preconditions. We then used UCPOP+EBL as a basis to investigate the effect of expressive action representations, heuristic goal selection strategies and sophisticated backtracking algorithms on the effectiveness of control rule learning. In particular, we showed that expressive action representations facilitate the use of richer domain descriptions, which in turn increase the effectiveness of learning control rules from analytical failures. This reduces the need for domain specific failure theories to guide EBL. We also noted the strong affinity between dependency directed backtracking and control rule learning, and showed that despite the affinity, control rules can still improve the performance of a planner using dependency directed backtracking.

References

[1] N. Bhatnagar and J. Mostow. *On-line Learning From Search Failures Machine Learning*, Vol. 15, pp. 69-117, 1994.

[2] R. Dechter. Enhancement schemes for learning: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, Vol. 41, pp. 273-312, 1990.

[3] D. Joslin and M. Pollack. Least-cost flaw repair: A plan refinement strategy for partial order planning. *Proceedings of AAAI-94*, 1994.

[4] D. McAllester and D. Rosenblitt Systematic Nonliner Planning In *Proceedings of AAAI-91*, 1991.

[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. 1, 1994.

[6] S. Katukam and S. Kambhampati. Learning Explanation-based Search Control Rules For Partial Order Planning In *Proceedings of AAAI-94*, 1994

[7] S. Kambhampati, S. Katukam and Y. Qu. Failure Driven Search Control for Partial Order Planners: An Explanation based approach. ASU CSE TR 95-010, July 1995. Submitted for journal publication.

[8] 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.

[9] S. Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[10] S. Minton. Quantitative Results Concerning the Utility of Explanation Based Learn ing *Artificial Intelligence*, 42:363--391, 1990.

[11] R.J. Mooney and S. Bennett. A domain independent explanation-based generalizer. In *Proc. AAAI-86*, 1986.

[12] J.M. Zelle and R.J. Mooney. Combining FOIL and EBG to speedup logic programs. In *Proceedings of IJCAI-93*, 1993.

[13] E.P.D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context dependent effects. In *Proc. IJCAI-91*, 1991.

[14] J.S. Penberthy and D.S. Weld UCPOP: A Sound, Complete, Partial Order Planner for ADL In *Proceedings of KR-92*, 1992

[15] A. Segre and C. Elkan. A High Performance Explanation-based learning algorithm. *Artificial Intelligence*, Vol. 69, pp. 1-50, 1994.

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