Learning Rationales to Improve Plan Quality for Partial Order Planners

Muhammad Afzal Upal **Renée Elio**

Department of Computing Science
University of Alberta, Edmonton
Canada, T6G 2H1
{upal,ree}@cs.ualberta.ca

Abstract

Plan rationale has been variously defined as "why the plan is the way it is", and as "the reason as to why the planning decisions were taken" (PT98). The usefulness of storing plan rationale to help future planning has been demonstrated by several types of case-based planners. However, the existing techniques are unable to distinguish between planning decisions that, while leading to successful plans, may produce plans that differ in overall quality, as defined by some quality metric. We outline a planning and learning system, PIPP, that applies analytic techniques to learn plan-refinement control rules that partial-order planners can use to produce better quality plans. Quality metrics are assumed to be variant: whether a particular plan refinement decision contributes to a better plan is a function of contextual factors that PIPP identifies. Preliminary evaluation results indicate that the overhead for applying these techniques to store and then use planning-refinement rules is not very large, and the outcome is the ability to produce better quality plans within a domain. Techniques like these are useful in those domains in which knowledge about how to measure quality is available and the quality of the final plan is more important than the time taken to produce it.

According to Hammond (Ham90), case-based planning involves remembering past planning solutions so that they can be reused, remembering past planning failures so that they can be avoided, and remembering repairs to plans that once failed but got "fixed" so that they can be re-applied. Early work on derivational analogy (Car83a; Car83b) proposed that each planning decision within a plan be annotated with the *rationale* for making that planning decision. An example of a rationale for taking a particular action A might be "it achieves goal g." Both state-space planners (Vel94) and partial-order planners (IK97) make use of such annotations in building a case library. State-space planners move through a search space of possible worlds, where each operator is a possible action to take in the world. Partial-order planners move through a space of possible plans, where each operator is a plan-refinement decision that typically adds additional steps or constraints to an evolving plan description. In the partial-order planning paradigm, a refinement decision such as *"add-step A to partial plan P"* might be annotated with the rationale "A's effects match an open (unsatisfied) condition of partial plan p." The idea behind storing rationales is that a previously-made and retrieved planning decision will only be applied in context of the current planning problem if the rationale for it also holds in the current problem.

DerSNLP+EBL (IK97) is a partial-order planner that uses explanation-based learning to avoid decisions that lead to planning failures as well as rationales for good planning decisions. DerSNLP+EBL, however, is unable to distinguish among planning decisions that lead to plans that are qualitatively different from one another. Thus, if two actions A and B are applicable at some choice point in refining a plan, then the justification for both decisions would be the same (namely, each action supplies an effect that satisfies a currently unsatisfied condition). While both plans might ultimately succeed, one may be *better* than the other by some criterion, such as plan quality.

The concern with plan quality is the focus of this work. Most conventional planning systems, including derivational analogy systems, use plan length as a measure of plan quality. This metric can suffice, if each possible action has the same unit cost and there is no sense in which the plan's execution uses or impacts other domain resources. However, it has been widely acknowledged in both the theoretical and practical planning camps that plan-quality for most real-world problems depends on a number of (possibly competing) factors (KR93; Wil96).

What makes reasoning about plan quality difficult is that two different plans can both succeed in achieving a goal, but they can have different overall qualities. Because they both succeed, there is no sense in which a "planning failure" can be reasoned about and remembered. Instead, the challenges are (a) to analyze how plan refinement decisions that yielded two successful plans lead to differences in overall quality and (b) store that analysis in a form that is usable to produce better quality plans for new problems.

In this article, we describe a partial-order planning system, PIPP, that employs a more complex representation of plan quality than plan length, and uses that

representation to learn to discriminate between plan-refinement decisions that lead to better or worse quality plans. The assumption underlying this work is that complex quality tradeoffs can be mapped to a quantitative statement. There is a long history of methodological work in operations research that guarantees that a set of quality-tradeoffs (of the form "prefer to maximize X rather than minimize Y") can be encoded into a value function, as long as certain rationality criteria are met (Fis70; KR93). We assume that a quality function defined on the resource levels for a plan exists. PIPP uses a modified version of the R-STRIPS (Wil96) that allows it to represent resource attributes and the effects of actions on those resources. The learning problem then is that of translating this global quality knowledge into knowledge that allows the planner to discriminate between different refinement decisions at a local level that impact the final overall plan quality.

Architecture and Algorithm Overview

Architecture

PIPP has three main components. The first is a partial-order planner (POP) of the sort described in (MR91). The second component does the analytic work of identifying plan refinement decisions that may affect the overall plan quality. The third component is a case library of plan-refinement rules.

The Planning Component

POP is a causal-link partial-order planner that, given an initial state and a goal state, produces a linearized plan that is consistent with the partial ordering constraints on steps that it identified during its planning process. Partial-order planners move through a search space of partially-specified plan descriptions. These specifications include the steps to be included in the plan, partial-ordering constraints on these steps, causal link relationships between the steps, and variable binding decisions. Causal link relationships record the interdependencies between steps and themselves are a kind of rationale. The operators that move the planner from one plan description to another by adding these sorts of constraints or additional specifications are called *planning refinement operators*.

Partial-order planners begin with a null plan, that includes a *start-step* constrained to occur before a *final-step*. The solution is a set of steps, variable bindings for the steps, and partial-ordering constraints. The final plan can be any linearization of those steps that are consistent with the partial ordering constraints.

The Analytical Learning Component

The input to PIPP's analytic component is (a) a problem described as an initial state and a goal state, (b) the plan and planning trace produced by the partial order planner for this problem, and (c) a *better plan* for the same problem that serves as a kind of model. The better plan is one that has a higher quality rating than the one produced by the underlying partial-order planner, as per the quality function that assesses how resources are impacted by each plan. This model plan might be provided by some oracle, by a user, or by some other planner. PIPP's algorithm is summarized in Figure 1.

Input: - Problem description in terms of initial state I and goal G
- An optimal quality plan for this problem Q

Output: - A set of rules

1- Use a causal-link partial-order planner to generate a plan P for this problem.

2- Identify learning opportunities by comparing the two plannings episodes.

3- Learn a rule from each learning opportunity and store it.

Figure 1: Algorithm 1

Identifying the learning opportunities We assume that the planning trace that *produced* the model plan is not available to PIPP. Therefore, PIPP's analytic component first reconstructs a set of causal link relationships between the steps in the better plan and a set of required ordering constraints. The second step is to retrace its own planning-trace, looking for plan-refinement decisions that added a constraint that is *not present* in the better-plan's constraint set. We call such a decision point a *conflicting choice point*. Each conflicting choice point indicates a gap in PIPP's control knowledge and hence a possible opportunity to learn something about producing a better quality plan. Identification of the conflicting choice point is Step 2 in Algorithm 1 which is further detailed in Figure 2.

Having found a conflicting choice point, PIPP replaces its plan refinement decision with a decision that adds the relevant constraint from the higher-quality plan. Figure 3 illustrates the idea of a conflicting choice point, using a problem from the transportation logistics domain. At Node 1 in the plan-tree shown in Figure 3, the default POP algorithm removes the open-condition flaw *at-obj(o1,ap2)* by performing *add-action unload-truck(o1,X2,ap2)*, which adds the causal-link $unload\text{-}tr(o1, X2, ap2) \xrightarrow{at\text{-}obj(o1,ap2)} final\text{-}step$ to the partial plan. But this causal-link is not in the causal-link set that PIPP inferred from the higher-quality model plan. In that constraint set, the precondition *at-obj(o1,ap2)* of the *final-step* is supplied by the action *unload-pl(o1, pl1, ap2)*. This sort of conflict in how a refinement decision is made offers PIPP a learning opportunity.

Learning a single search control rule that would ensure the addition of this alternative action at this point may turn the low-quality plan into a higher-quality plan, but it is rather unlikely that this was the only planning decision accounting for the difference in quality between the POP produced plan and the model

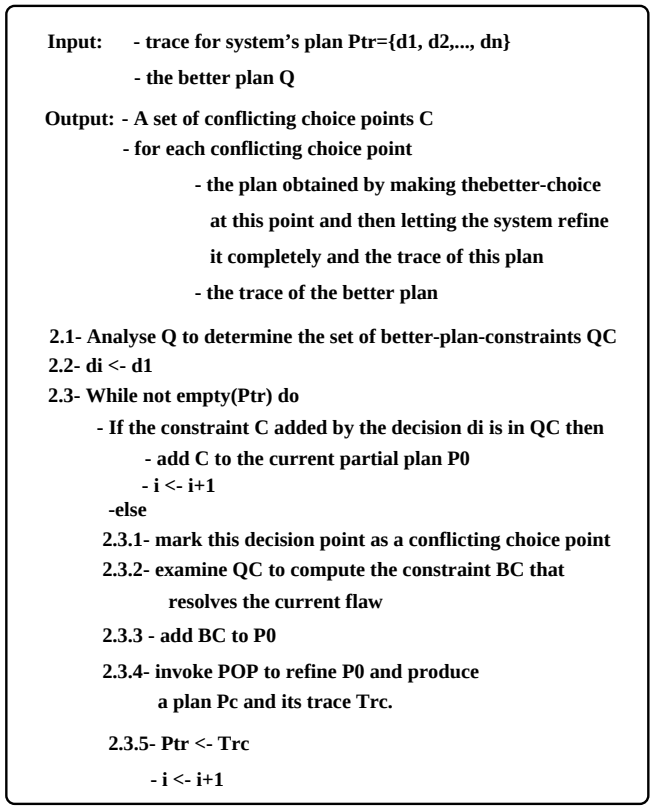

Figure 2: Identification of learning opportunities (Step 2 of Algorithm 1).

plan. There may be more opportunities to learn what other decisions contributed to the generation of a better quality plan. Once the higher-quality plan's refinement decision has been spliced into the plan produced by the default planner, PIPP calls the default planner again to re-plan from that point on. A new plan and a new trace (that is the same as the initial trace up to the now-replaced conflicting choice point, and possibly different thereafter) is returned for this same problem, and the process of analyzing this new trace against the constraints of the higher-quality model plan is done again. This process repeats until the POP algorithm has produced a set of plan-refinement decisions that are consistent with the inferred plan refinement decisions associated with the better plan.

We now describe what is learned from the analysis of any given conflicting choice point. For any conflicting choice point, there are two different plan-refinement decision sequences that can be applied to a partial plan: the one added by the default POP algorithm, and the other inferred from the better-quality plan. The application of one set of plan-refinement decisions leads to a higher quality plan and the other to a lower quality plan. It would be possible to construct a rule that indicates that the refinement decision associated with the better-quality plan should be taken if that same plan flaw is ever encountered again. However, this would ensure a higher-quality plan *only if* that decision's impact on quality was not contingent on other refinement decisions that are "downstream" in the refinement process, i.e., further along the refinement path. Thus, some effort must be expended to identify the dependencies between a particular refinement decision and other refinement decisions that follow it.

To identify what downstream refinement decisions are relevant to the decision at a given conflicting choice point, the following method is used. The open-conditions at the conflicting choice point and the two different refinement decisions (i.e., the ones associated with the high quality model plan and the lower quality plan produced by the default planner) are labelled as *relevant*. The rest of the better-plan's trace and the rest of the worse-plan's trace are then examined, with the goal of labeling a subsequent plan-refinement decision q relevant if

- there exists a causal-link $q \xrightarrow{c} p$ such that p is a relevant action, or
- q binds an uninstantiated variable of a relevant open-condition.

For instance, consider again the conflicting choice point shown in Figure 3. There are two open-conditions flaws in the partial plan, but the flaw selected to be removed at this point is the open-condition *at-obj(o1,ap2)*. Clearly, the decision *add-step unload-pl(o1,X1,ap2)* on path A (left path) is relevant. Similarly, the decision to add steps *load-pl(o1,X2,Y1)* and *fly-pl(X1,Y1,ap2)* are relevant because they supply preconditions to the relevant action *unload-pl(o1,X1,ap2)*. Further along path A, the decision *establish at-obj(o1, Y1)* is relevant because it supplies a precondition to the relevant step *fly-pl(X1,Y1,ap2)*.

In sum, the PIPP algorithm identifies the subsequent refinement decisions which have a dependency relation with the refinement decision at the conflicting choice point, for both the path associated with the higher-quality model plan and the path associated with the (lower quality) plan produced by the default POP algorithm.

Learning rationales Once PIPP identifies the relevant refinement decisions associated with the way in which a given choice point was resolved differently for the higher-quality plan and the worse plan, a search control rule can be created. To do this, PIPP computes:

- the open-condition flaws present in its partial plan that the relevant decision sequence removes
- the effects present in its partial plan that are required by any establishment decisions present in the relevant decision sequence
- the quality value of the new subplan produced by the relevant decision sequence.

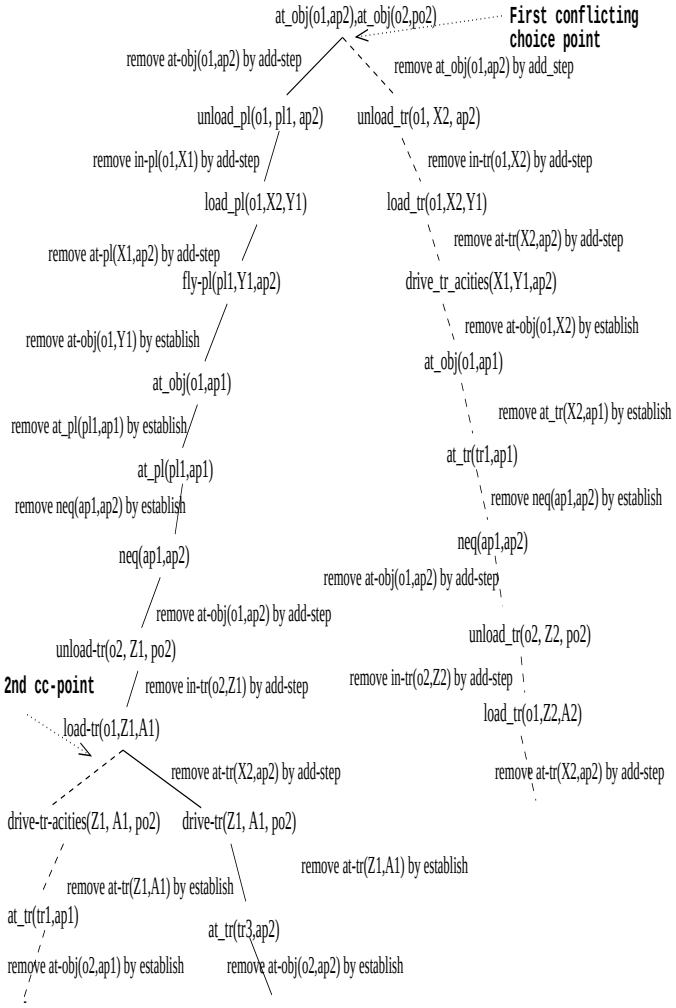

Figure 3: Conflicting choice point that leads to Path A (left), from the higher-quality plan, and to Path B (right), the lower-quality plan produced by POP.

PIPP then use this information to store the rationale for applying each refinement decision sequence. For the example shown in Figure 3, the rationale learned for the refinement sequence associated with the higher-quality plan is:[1]

```
:preconditions [(at-obj(O,Y), A1)]
:effects [(at-obj(O,X),A2), (at-pl(L,X),A2)]
:quality-formula [distance(X,Y)*2]
:trace [add-step(unload-pl(O,L,Y)),
        add-step(load-pl(O,L,X)),
        add-step(fly-pl(L,X,Y)),
        establish(at-obj(O, X)),
        establish(at-pl(L,X)),
        establish(neq(X,Y))].
```

This rule captures the rationale for applying the decision sequence specified by the trace field of the rule to resolve the open-condition flaws specified by the preconditions field.

Rules such as these are then consulted by the default planner in Step 1. When refining a partial plan P, PIPP's planner checks to see if a rule exists whose preconditions and effects are subsets of P's preconditions and effects respectively. If more than one such rule is available, then the rule that has the largest precondition set (i.e., it resolves the largest number of preconditions) is selected. If more than one such rule is available, then PIPP's planner uses the rule whose quality-formula has the highest value when evaluated in context of P.

Evaluation

There are two main issues we address in the empirical evaluations reported here. First, we can ask whether the local refinement rules that PIPP acquires do lead it to produce better quality plans. Second, we can evaluate some of the overhead for using these control rules to improve plan quality. There are a number of domains in the planning literature that are often used to evaluate planning systems. Previously, we used a modified version of Veloso's logistics-transportation domain (Vel94). However, for the experiments reported here, we followed Barett and Weld (BW94) and devised artificial domains in which we could vary various features.

We were particularly interested in evaluating PIPP's performance in domains in which it is difficult to capture the global knowledge about plan quality in local rules. Notice that the difficulty of learning quality-improving local rules is orthogonal to the complexity of the value function as well as the planning complexity. We call the target function that PIPP must learn (and capture in its local rules) to discriminate between qualitatively different local planning decisions as the *discriminant function*. The complexity of the discriminant function depends (among other things) on (a) the number of conflicting choice points and (b) the number of local rules per conflicting choice point required to capture the discriminant function. For instance, a discriminant function at a conflicting choice point could be trivially captured by one rule if all applicable operators have the same preconditions and effects but different costs. This rule would say (essentially) "When given a choice between two operators, choose the one with the lower cost." A discriminant function is complex, however, if a single rule learned from one episode guides the planner to a lower quality path for another problem. In such domains, PIPP is forced to learn more rules to capture this complexity. We designed two domains, as described below, to evaluate PIPP's acquisition of more complex discriminant functions.

General Methodology

We devised two domains that differed in the complexity of the plan space and therefore in the complexity of discriminant function that characterized what contextual features map to what sorts of refinement decisions that impact plan quality. For both domains,the quality function was $Q(X, Y, Z) = X + Y - 2 \times Z$.

[1] In the Prolog tradition, we use capital letters to show variables throughout the paper.

Domain I Domain I had three types of operators:

```
for i=1,...,10.
(defoperator :action Ai :params {X,Y,Z}
        :preconds {Pj|j<i}
        :add {gi}
        :delete{}
        :metric-effects {(X, add(2)),
                         (Y, add(2)), (Z, add(1))})

for i=1,...,10.
(defoperator :action Bi :params {X,Y,Z}
        :preconds {I}
        :add {Pi}
        :delete{I1}
        :metric-effects {even(i) -> (X, add(4)),
                         (Y, add(4)), (Z, add(1))
        odd(i) -> (X, add(1)),
                         (Y, add(1)), (Z, add(4))})

(defoperator :action C1 :params {X,Y,Z}
        :preconds {Qi|0< i <6}
        :add {Pj| 0< j <11}
        :delete{I}
        :metric-effects {(X, add(1)),
                         (Y, add(2)), (Z, add(1))})

for i=2,...,5.
(defoperator :action Ci :params {X,Y,Z}
        :preconds {Q(i-1)}
        :add {Qi}
        :delete{I1}
        :metric-effects {(X, add(1)),
                         (Y, add(2)), (Z, add(1))}
```

In this domain, for every goal g_i there exist exactly two viable plans: ${P^1}_i = \{C_1, C_2, C_3, C_4, C_5, A_i\}$ and ${P^2}_i = \{B_1, \ldots, B_i, A_i\}$. If i is odd then ${P^1}_i$ has a higher quality, otherwise ${P^2}_i$ has a higher quality.

There is only one conflicting choice point between the lower and higher quality plan during plan refinement, namely, the choice of operator C or B_1. If an unseen problem with a goal $g_i, i > j$ is presented to PIPP when the highest goal value it has learned so far is g_j then the rule learned form the jth episode would provide PIPP with wrong guidance and another more specific rule would have to be learned for the ith problem. Hence, we expected PIPP to sometimes produce lower quality plans than POP, but we wanted to see how PIPP's performance was affected in such a domain.

We performed a 15-, 30-, and 45-fold cross-validations, using a 90 2-goal problem set that was randomly-generated and guaranteed to have no repeated problems. Thus, for the 15-fold cross-validation, each of 6 distinct sets of 15 problems served as the training set, and the remaining 5 sets were used for testing, with the results averaged. The dependent measures were (a) planning effort, defined as the number of nodes expanded by the planner, (b) plan quality, defined as the percentage of optimal quality plans generated by PIPP for the test problems, (c) number of rules learned, (d) number of rules retrieved during testing, and (e) number of rules that were both retrieved and ultimately lead to the optimal quality plan.

number of training examples	0	15	30	45
number of nodes expanded	33	14.5	11.08	11.5
%age plans of optimal quality	0.50	0.54	0.58	0.63
number of rules learned	0	24.1	40.3	51.5
number of rules retrieved	0	9	14.3	12.5
number of rules used	0	9	14.3	12.5

Table 1: Performance data for Domain I.

Table 1 presents these results. The first column, zero training items, corresponds to the base planner operating with no learning component. The remaining columns correspond to the base planner operating with PIPP's learning algorithm.

There are a few things to observe at this point. First, the probability of the base planner (without PIPP) selecting the optimal quality plan is 0.5. This is because there is only one conflicting choice point i.e., two possible ways of generating a plan to solve any given problem. Secondly, there is not much improvement by PIPP over this level of performance except in 45 training item scenario. Finally, we note that PIPP, having learned rules for how to refine a plan, must engage in less planning effort than the base planner.

These results are not surprising and the small size of the plan space presents something of a ceiling effect: the base planner has a 50% chance at generating the optimal quality plan. Still, Domain 1 provides a good baseline for considering PIPP's performance when we increase the complexity of the refinement space, and thereby increase the complexity of the discriminant function that PIPP must learn.

Domain II To increase the number of conflicting choice points, we replaced the action *B2* in Domain I with the following two actions:

```
(defoperator :action B2a :params {X,Y,Z}
:preconds {i},
:add {p2,p6,p9},
:del {},
:metric-effects {(X, add(4)),
                         (Y, add(4)), (Z, add(1))})

(defoperator :action B2b :params {X,Y,Z}
:preconds {i},
:add {p2,p4,p7},
:del {i1},
:metric-effects {(X, add(4)),
                         (Y, add(3)), (Z, add(1))})
```

number of training examples	0	15	30	45
number of nodes expanded	45.0	45.4	37.9	37.6
%age plans of optimal quality	0.10	0.37	0.54	0.66
number of rules learned	0	47.8	73.0	114.0
number of rules retrieved	0	17.3	23.6	28.5
number of rules used	0	13.0	20.3	24.5

Table 2: Performance data for Domain II.

Similarly, the operator C4 in Domain I was replaced by the following two operators to get Domain II:

```
(defoperator :action C4a :params {X,Y,Z}
:preconds {q3},
:add {q4,p0},
:del {i},
:metric-effects {(X, add(1)),
                        (Y, add(2)), (Z, add(1))})

defoperator :action C4b :params {X,Y,Z}
:preconds {q3},
:add {q4,p8},
:del {i},
:metric-effects {(X, add(2)),
                        (Y, add(2)), (Z, add(1))})
```

This created a plan space in which, for any given problem, there was a 10% chance for the base planner to select the optimal quality plan. Table 2 presents the performance data for the base planner (zero training items) and for PIPP on this domain.

Here we see that PIPP's refinement rules generate better-quality plans at a much increased level over chance. With training on 30 items, PIPP generated the optimal quality plan on average over 50% of the time. We also note that the rule library produced is much larger than what is actually applied during testing. Most of the time, the rules retrieved actually did lead to the optimal quality plan (e.g., 13.0 rules used vs. 17.3 rules retrieved for the 15-fold cross validation case). Thus, PIPP's quality performance is perhaps underestimated: we only count the cases in which it generated the optimal quality plan, rather than the cases in which, by applying its rules, it generated a better plan than the base planner would have generated for the same problem.

We observe less of a gain in planning efficiency in Domain II relative to Domain I. However, this is likely due to the increased size of the rule library for Domain II. As the rule library size increases, so too does the possibility of a rule leading to planning failure. When PIPP's application of a rule leads it to fail in generating a plan, it simply calls the base planner from that point.

The larger payoff in plan quality that we see in Domain II, which has greater complexity than domain I, is interesting, because it hints at scale-up potential of such techniques. The results also show that PIPP's rule library quickly grows. However, since few of the rules are actually used during planning, this indicates that we can improve PIPP's rule-library by keeping some utility metrics around and forgetting rules that are not useful.

Related Research

The basic idea of learning the justifications for success or failure of a problem solving episode can be traced back to the early work on EBL. Minton's PRODIGY/EBL (Min89) learned control rules by explaining why a search node lead to success or failure. Veloso implemented the derivational analogy in a state- space planning framework, PRODIGY. Ihrig and Kambhampati (IK97) applied the derivational analogy approach to a partial-order planner, SNLP. DerSNLP+EBL extended DerSNLP by learning from successes as well as failures. All of these systems can be regarded as speed-up learning systems: systems that learn to improve planning efficiency but not plan quality.

PIPP is most closely related to QUALITY (Per96), a learning system that uses analytical approach to learn control rules for a state-space planner PRODIGY (VCP+95). Given a problem, a quality metric and a user's plan, QUALITY assigns a cost value to each node in a user's planning trace and to each node in the system's planning trace. It identifies all those goal-nodes that have a zero cost in the user's trace and non-zero cost in the system's trace. A goal-node's cost can be zero either because it was true in the initial state or because it was added as a side-effect of an operator added to achieve some other goal. The reason for the difference in the cost values of the two nodes is identified by examining both trees. The explanation constructed from these reasons forms the antecedent of the control rule learned by QUALITY. However, QUALITY can learn only from a subset of the PIPP's learning opportunities (i.e., only from those conflicting choice points where one branch has a zero cost) and QUALITY's quality-value assignment procedure can only work if the quality metrics are static, i.e. they assign the same value to an plan step regardless of the context in which it is applied. If the quality metrics are variant (dependent on context), then quality values cannot be assigned to a node which has some action parameters uninstantiated.

Perez (Per96) shows that QUALITY can learn a number of useful rules for the process planning domain (Gil92). However, at this point, PIPP's base planner does not deal with quantified preconditions and effects and therefore we were unable to compare PIPP's rules with those learned by QUALITY.

Knowledge about how plan quality can be measured is required (a) to identify the learning opportunities

(i.e., by identifying a lower and a higher quality plan) and (b) to analytically learn from these learning opportunities. Without such quality knowledge, analytic techniques cannot be used. However, empirical techniques can be used if we assume that the learning opportunities are identified with the help of a user who supplies a better plan. (ZK96) and (EM97) present two inductive learning techniques to learn search control rules for partial order planners. SCOPE (EM97) uses FOIL (Qui90), an inductive concept learner, whereas Zimmerman's system uses a neural network to to acquire search control rules for UCPOP. Given a planning problem to solve and a user's better plan for that problem, SCOPE considers each of user's refinement decisions to be a positive example of the application of that refinement and the system's refinement decision to be a negative example. These positive and negative examples are then passed to FOIL to induce a rule that covers all positive examples and none of the negative examples. Inductive methods are well known to be less efficient than analytic techniques if background knowledge is available (PK92).

Conclusion

Being able to efficiently produce good quality solutions is essential if AI planners are to be widely applied to the real-world situations. However, conventional wisdom in AI has been that "domain independent planning is a hard combinatorial problem. Taking into account plan quality makes the task even more difficult" (AK97). This paper has presented a novel technique for learning local search control rules for partial order planners that improve plan quality without sacrificing much in the way of planning efficiency. We believe that these ideas will contribute towards making AI planning applicable to more practical planning situations where plan quality depends on a number of factors.

Acknowledgement

This work was supported by NSERC research grant A0089 to Renee Elio.

References

J. Ambite and C. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proc. of the Fourteenth National Conf. on Artificial Intelligence*, Menlo Park, CA, 1997. AAAI Press.

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

J. Carbonell. Derivational analogy and its role in problem solving. In *Proc. of the Third National Conf. on Artificial Intelligence*, Los Altos, CA, 1983. Morgan Kaufmann.

J. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R. Michalski, editor, *Machine Learning: An Artificial Intelligence Approach*, pages 131–161. Tioga Pub. Co., Palo Alto, CA, 1983.

T. Estlin and R. Mooney. Learning to improve both efficiency and quality of planning. In *Proc. of the IJCAI*. Morgan Kaufmann, 1997.

P. Fishburn. *Utility Theory for Decision Making*. Wiley, New York, 1970.

Y. Gill. A specification of manufacturing processes for planning. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, 1992.

K. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14(3), 1990.

L. Ihrig and S. Kambhampati. Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Journal of Artificial Intelligence Research*, 7:161–198, 1997.

R. Keeney and H. Raiffa. *Decisions With Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, New York, 2nd edition, 1993.

S. Minton. Expalantion-based learning. *Artificial Intelligence*, 40:63–118, 1989.

D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Ninth National Conf. on Artificial Intelligence*, pages 634–639, Menlo Park, CA, 1991. AAAI Press/MIT Press.

A. Perez. Representing and learning quality-improving search control knowledge. In L. Saitta, editor, *Proc. of the Thirteenth International Conf. on Machine Learning*, Los Altos, CA, 1996. Morgan Kaufmann.

M. Pazzani and D. Kibler. The utility of background knowledge in inductive learning. *Machine Learning*, 9:57–94, 1992.

S. Polyak and A. Tate. Rationale in planning: Causality, dependencies, and decisions. *Knowledge Engineering Review*, 13:1–16, 1998.

R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–2666, 1990.

M. Veloso, J. Carbonell, M. Perez, E. Borrajo, D. amd Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.

M. Veloso. *Learning by Analogical Reasoning*. Springer Verlag, Berlin, 1994.

M. Williamson. A value-directed approach to planning. Technical Report TR-96-06-03, PhD thesis, University of Washington, 1996.

T. Zimmerman and S. Kambhampati. Neural network guided search control in partial order planning. In *Proc. of the Thirteenth National Conf. on Artificial Intelligence*, Menlo Park, CA, 1996. AAAI Press.