Learning Rewrite rules versus search control rules to improve plan quality

M. Afzal Upal[1] and Renee Elio[2]

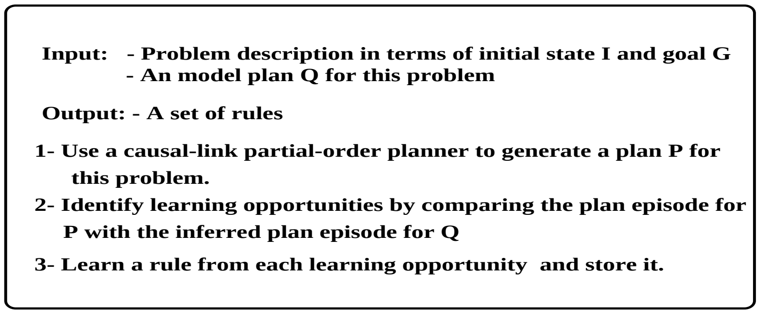
[1] Faculty of Computer Science
Dalhousie University, Halifax
`afzal.upal@dal.ca`
[2] Department of Computing Science
University of Alberta, Edmonton
`ree@cs.ualberta.ca`

Abstract. Domain independent planners can produce better-quality plans through the use of domain-specific knowledge, typically encoded as search control rules. The planning-by-rewriting approach has been proposed as an alternative technique for improving plan quality. We present a system that automatically learns plan rewriting rules and compare it with a system that automatically learns search control rules for partial order planners. Our results indicate that learning search control rules is a better choice than learning rewrite rules.

1 Introduction

AI planners must be able to produce high quality plans, and do so efficiently, if they are to be widely deployed in the real-world planning situations. Various approaches have shown that incorporating domain knowledge into domain-independent planners can improve both the efficiency of those planners [6, 4, ?] and as well as quality of the plans they produce [12, 5]. Traditionally, this knowledge is encoded as search control rules to limit the search for generation of the first viable plan. Recently, Ambite and Knoblock have suggested an alternative approach called *planning by rewriting* [1]. Under this approach, a partial-order planner generates an initial plan, and then a set of rewrite rules are used to transform this plan into a higher-quality plan. Unlike the search control rules for partial order planners (such as those learned by UCPOP+EBL [6] and PIPP [14]) that are defined on the space of partial plans, rewrite rules are defined on the space of complete plans. In addition, it has been argued that plan-rewrite rules are easier to state than search control rules, because they do not require any knowledge of the inner workings of the planning algorithm [1]. That may partially explain why most of the search-control systems have been designed to automatically acquire search-control rules, whereas existing planning by rewriting systems use manually generated rewrite-rules. To date, there has been no comparison of these two techniques to study their strengths and weaknesses. This paper presents an empirical comparison of how the two techniques (search control rules vs rewrite rules) improve plan quality within a partial-order planning framework. Our focus, however, assumes that both rewrite rules as well as search control rules are to be learned as a function of planning experience.

We designed two systems, Sys-REWRITE and Sys-SEARCH-CONTROL, that automatically learn to improve quality of the plans produced by the partial order planners. Both systems have the same overall structure, shown in Figure 1, and only differ in their implementation of the last step. For Step 1, both systems use a partial order planning algorithm, POP, of the sort described in [9]. The learning algorithm, ISL (Intra-Solution Learning algorithm), that the two systems use for Step 2 is similar to that used in [14] and is described in the section that follows. In Step 3, Sys-REWRITE uses the output of Step 2 to create plan-rewrite rules, while Sys-SEARCH-CONTROL uses that information to create search-control rules. The performance component of the two systems necessarily differs, by definition: Sys-SEARCH-CONTROL uses its rules during its planning process, whereas Sys-REWRITE uses the rules after it has completed what we might think of as its draft partial plan.



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

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 plan episode for P with the inferred plan episode for Q

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

Fig. 1. High level Algorithm

2 Overview of Approach

Our approach to plan quality representation and the underlying learning algorithm may be briefly described as follows[1]. We assume that complex quality tradeoffs among a number of competing factors can be mapped to a quantitative statement. Methodological work in operations research indicates that a large 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 [3]. We also assume that a quality function defined on resources consumed in a plan exists for a given domain and use a modified version of R-STRIPS [16] to represent resource attributes and the effects of actions on those resources.

Given the knowledge about how to measure the quality of a complete plan, the learning problem then is how to translate this global quality knowledge into

[1] For details the reader is referred to [14].

knowledge that allows the planner to discriminate between different refinement decisions at a "local" level, i.e., to learn search control knowledge. It is this general approach that we will contrast with learning rewrite rules.

As Figure 1 indicates, the training data for both Sys-REWRITE and Sys-SEARCH-CONTROL consists of the description of a problem in terms of the initial state and goals, and a completed high-quality plan (a set of totally ordered steps) that serves as a kind of model. The higher quality model plan can be generated by the planner itself through a more exhaustive search of the plan space or supplied by an external agent (as is done in apprenticeship learning systems [8]). The learning step (Step 2) is triggered if the model plan is of higher quality (as per the quality metric) than the the system's default plan for the same problem. Learning occurs in the context of considering differences between higher-quality model plan with the lower-quality default plan produced by the system. But because the planner is a partial-order planner, what it must learn is how to make better plan-refinement decisions, i.e., the form of knowledge to be acquired must affect the partial-order planning process, at least when the rules to be learned are search control rules. Thus, learning occurs by considering differences in the planning refinement trace that produced the partial order plan and elements of an *inferred* planning refinement trace that is consistent with the model plan, Q. This is the heart of the ISL algorithm that is described in more detail in Section 3.2, and that identifies the knowledge that will be turned into either search control rules or rewrite rules.

3 System Architecture

We describe the architecture in terms of the three steps outlined in Figure 1.

3.1 Step 1: The Planning Component

The planning element is a causal-link partial-order planner (POP) that, given an initial state and some goals, produces a linearized plan that is consistent with the partial ordering constraints on steps that it identified during its planning process.

3.2 Step 2: Learning from Plan Refinement Traces

We will use the transportation problem shown in Figure 3 to illustrate the workings of the ISL algorithm. The ISL algorithm, shown in Figure 2 looks for differences between two *solutions* to a planning problem that differ in overall quality. It has both the default plan and the default planning trace produced from Step 1, plus the model plan Q. We do not assume that the planning trace that *produced* the model higher-quality plan is available–just the model plan itself. Therefore, ISL's first step is to reconstruct the causal-link and ordering constraints that are consistent with the step sequence that defines the model plan. The model

```
Input:     - trace for system's plan Ptr={d1, d2,..., dn}
           - the better plan Q
Output: - A set of conflicting choice points C
        - for each conflicting choice point
                - the plan obtained by making thebetter-choice
                  at this point and then letting the system refine
                  it completely and the trace of this plan
                - the trace of the better plan

2.1- Analyse Q to determine the set of better-plan-constraints QC
2.2- di <- d1
2.3- While not empty(Ptr) do
       - If the constraint C added by the decision di is in QC then
            - add C to the current partial plan P0
            - i <- i+1
       -else
         2.3.1- mark this decision point as a conflicting choice point
         2.3.2- examine QC to compute the constraint BC that
                  resolves the current flaw
         2.3.3 - add BC to P0
         2.3.4- invoke POP to refine P0 and produce
                 a plan Pc and its trace Trc.
         2.3.5- Ptr <- Trc
               - i <- i+1
```

Fig. 2. The Intra-Solution Learning (ISL) Algorithm (Step 2 of Algorithm 1).

constraint set inferred by ISL from the model plan presented earlier in Figure 3 is shown in Figure 5.

The next step is to retrace the default planning-trace (from step 1), looking for plan-refinement decisions that added a constraint that is *absent* in the model plan's constraint set. We call such a decision point a *conflicting choice point*. Each conflicting choice point indicates a possible opportunity to learn a plan-refinement decision that contributes to producing a better quality plan.

Given the default planning trace and the model constraint-set shown in Figure 5, ISL retraces the default planning trace (shown in Figure 4) looking for a planning decision that adds a constraint not present in the model constraint-set. Node 1 in Figure 4 is one such node where the default planner resolves the open-condition flaw *at-object(o1,ap2)*$_{end}$ by performing *add-action: unload-truck(o1,TR,ap2)*, which adds the causal-link $unload\text{-}truck(o1, TR, ap2) \xrightarrow{at\text{-}obj(o1,ap2)}$ *end* to the partial plan. But this causal-link is not in the model constraint-set for this problem shown in Figure 5. The model constraint-set contains a causal link $unload\text{-}plane(o1, pl1, ap2) \xrightarrow{at\text{-}object(o1,ap2)}$ *end*. In other words, the model planner resolved the precondition *at-object(o1,ap2)*$_{end}$ by *add-action: unload-plane(o1, pl1, ap2)*. Hence, Node 1 is labeled as a conflicting choice point. Simply put, the two plans differed in the way they achieved an open condition.

Initial-state: {at-object(o1, ap1), at-object(o2, ap2), at-truck(tr1, ap1), at-truck(tr2, ap2), at-plane(pl1, ap1), same-city(ap1, po1), same-city(po1,ap1), same-city(ap2,po2),same-city(po2,ap2), position(ap1, 10), position(po1, 15), position(ap2, 100), position(po2, 110), money(1000), time(0)}

Goals: {at-object(o1, ap2), at-object(o2, po2)}

System's Plan	Model Plan
load-truck(o1, tr1, ap1)	load-plane(o1, pl1, ap1)
drive-truck-acities(tr1, ap1, ap2)	fly-plane(pl1, ap1, ap2)
unload-truck(o1, tr1, ap2)	unload-plane(o1, pl1, ap2)
load-truck(o2, tr2, ap2)	load-truck(o2, tr2, ap2)
drive-truck(tr2, ap2, po2)	drive-truck(tr2, ap2, po2)
unload-truck(o2, tr2, po2)	unload-truck(o2, tr2, po2)

Fig. 3. Problem 1: A Transportation planning problem.

Learning a single search control rule that ensures the application of the model planning decision at this point may turn a low-quality plan into a higher-quality plan, but it is rather unlikely that this was the only reason for the difference in quality between the default plan and the model plan. There may be more opportunities to learn what other decisions lead to a better quality plan for the same problem. To identify the other planning decisions whose rationale the existing planer lacks, ISL adds the constraint added by the model plan at this point to the partial plan being refined. Once the higher-quality plan's planning decision has been applied to the partial plan being refined, ISL calls the existing planner *again to re-plan from that point on* (Step 2.3.4 of ISL). 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 analysis may lead to more conflicting choice points (as indeed is the case with the example scenario shown in Figure 4: at Node 10 the system's new plan makes a different choice than the model plan). Eventually, the default planner will generate a planning trace that is consistent with the constraint set inferred for the higher-quality model plan. That ends the learning about plan quality that can be accomplished from that single training problem.

For any conflicting choice point, there are two different planning decision sequences that can be applied to a partial plan: the one added by the existing planner, and the other added by the model planner. The application of one set of planning 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 planning decision associated with the better-quality plan should be taken if that same 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 planning decisions that are "downstream" in the refinement process, i.e., further along the search path. Thus, some effort must be expended to identify the dependencies between a particular planning decision and other planning decisions that follow it.

To identify what downstream planning 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 planning decisions (i.e., the ones associated with the high quality model plan and the lower quality

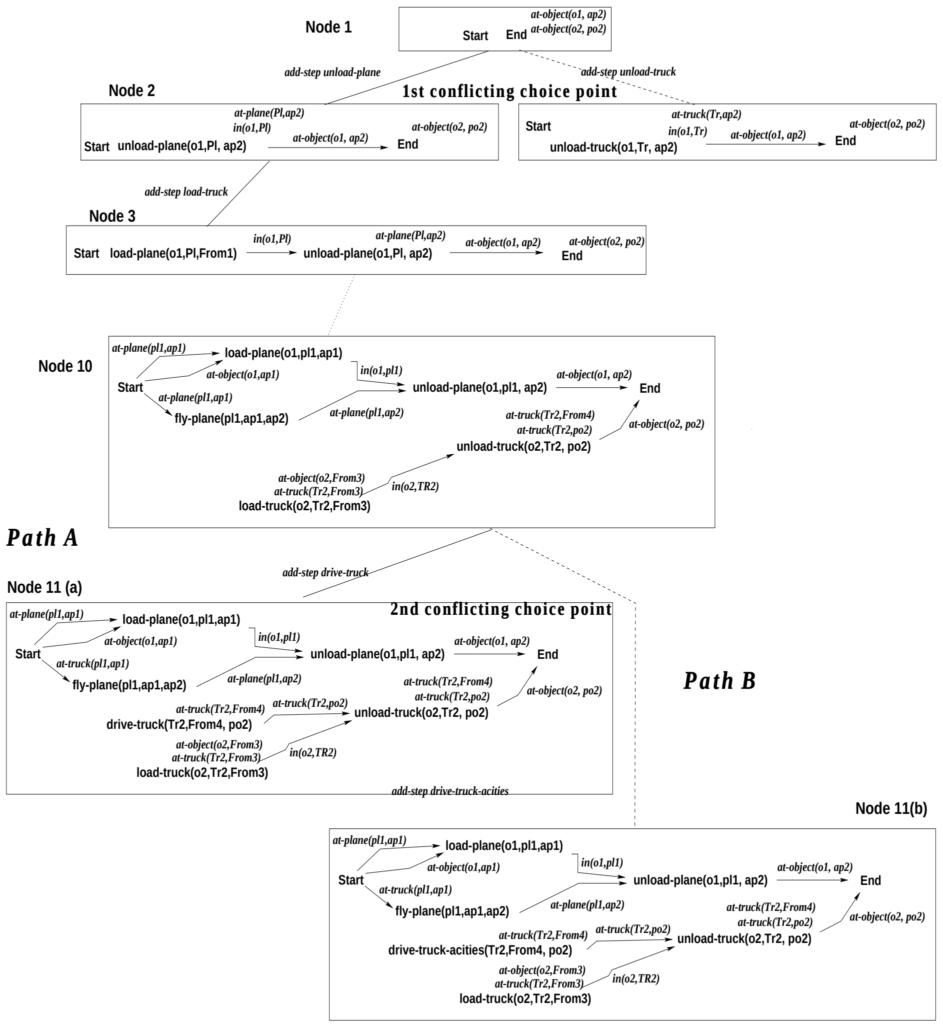


Fig. 4. Conflicting choice point that leads to Path A (left), from the higher-quality plan, and to Path B (right), the lower-quality plan. We use italics to represent open preconditions which are treated as subgoals. When these preconditions are still open (i.e., have not been satisfied), they are displayed next to the action that requires them. Arrows between actions denote causal-links showing which subgoals of an action have been satisfied. The arrow direction is from producer to the consumer of a condition.

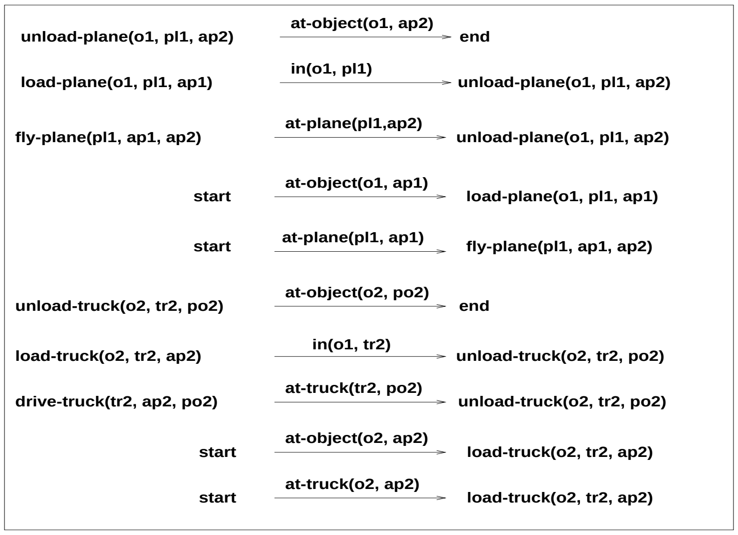

unload-plane(o1, pl1, ap2) —at-object(o1, ap2)→ end
load-plane(o1, pl1, ap1) —in(o1, pl1)→ unload-plane(o1, pl1, ap2)
fly-plane(pl1, ap1, ap2) —at-plane(pl1,ap2)→ unload-plane(o1, pl1, ap2)
start —at-object(o1, ap1)→ load-plane(o1, pl1, ap1)
start —at-plane(pl1, ap1)→ fly-plane(pl1, ap1, ap2)
unload-truck(o2, tr2, po2) —at-object(o2, po2)→ end
load-truck(o2, tr2, ap2) —in(o1, tr2)→ unload-truck(o2, tr2, po2)
drive-truck(tr2, ap2, po2) —at-truck(tr2, po2)→ unload-truck(o2, tr2, po2)
start —at-object(o2, ap2)→ load-truck(o2, tr2, ap2)
start —at-truck(o2, ap2)→ load-truck(o2, tr2, ap2)

Fig. 5. Constraints inferred from the model plan of Problem 1.

default plan) are labeled 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 planning 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 first conflicting choice point at Node 1 shown in Figure 4. 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-object(o1, ap2)*. Clearly, the decision *add-action: unload-plane(o1,Pl,ap2)* on Path A (left path) is relevant. Similarly, the decisions to *add-action: load-plane(o1,pl1,ap1)* and *add-action: fly-plane(pl1,ap1,ap2)* are relevant because they supply preconditions to the relevant action *unload-plane(o1,Pl,ap2)*. Further along Path A, the decision *establish: at-object(o1, ap1)* is relevant because it supplies an precondition to the relevant action *fly-plane(pl1,ap1,ap2)*. However, the planning decisions *add-action: unload-truck(o2, Tr2, po2)*, and *add-action: drive-truck(Tr2, From4, po2)* are not relevant because the open conditions they resolve are not relevant. The labeling process stops on reaching the leaf nodes and the two relevant planning decision sequences (for each conflicting choice point) are out put. ISL outputs the two planning decision sequences shown in Figure 3.2 for the first conflicting choice point.

3.3 Step 3: Learning search-control rules

Once ISL identifies the relevant refinement decisions associated with the way in which a given choice point was resolved differently for the the higher-quality

Lower quality sequence

add-action: *unload-truck(o1,Tr,ap2)* to resolve *at-object(O, Y)*$_{End}$

add-action: *load-truck(o1,Tr,From2)* to resolve *in(o1,Tr)*$_{unload\ mbox-truck}$

add-action: *drive-truck-acities(Tr,From2,ap2)* to resolve *at-truck(Tr,ap2)*$_{unload-truck}$

establish: *at-object(o1, From2)*$_{load-truck}$ with *at-object t(O, X)*^{Start}

establish: *at-truck(Tr,From2)*$_{drive-truck-acities}$ with *at-truck(Tr, X)*^{0}

establish: *neq(ap1,ap2)*$_{drive-truck-acities}$ with *neq(X, Y)*^{0}

Better quality sequence:

add-action: *unload-plane(o1,Pl,ap2)* to resolve *at-object(o1, ap2)*$_{End}$

add-action: *load-plane(o1,Pl,From1)* to resolve *in(o1, Pl)*$_{unload-plane}$

add-action: *fly-plane(Pl,From1,ap2)* to resolve *at-plane(Pl,From1))*$_{unload-plane}$

establish: *at-object(o1, From)*$_{load-plane}$ with *at-object (O, X)*^{Start}

establish: *at-plane(Pl,X))*$_{fly-plane}$ with *at-plane(Pl, X)*^{Start}

establish: *neq(ap1,ap2)*$_{fly-plane}$ with *neq(X, Y)*^{Start}

Fig. 6. Two planning decision sequences identified by ISL for the first conflicting choice point shown in Figure 4. The notation Pre_{Act} indicates that Pre is a precondition of Action Act and the notation Eff^{Act} indicates that Eff is an effect supplied by the action Act.

plan and the worse plan, a search control rule can be created. To do this, Sys-SEARCH-CONTROL computes (a) the open-condition flaws present in its partial plan that the relevant decision sequence removes, (b) the effects present in its partial plan that are required by the relevant decision sequence, and (c) the quality value of the new subplan produced by the relevant decision sequence. Sys-SEARCH-CONTROL then use this information to store the rationale (the pre-conditions) 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:[2]

open-conditions: { *at-object(O, Y)*$_{Act1}$ }

effects: { *at-object(O, X)*^{Act2}, *at-plane(Pl, X)*^{Act3}, *neq(X, Y)*^{Act4} }.

quality: `170 - 3 * distance(Y, X)/200.`

trace: *add-action:* *unload-plane(O,Pl,Y)* to resolve *at-object(O, Y)*$_{Act1}$

add-action: *load-plane(O,Pl,X)* to resolve *in(O, L)*$_{unload-plane}$

add-action: *fly-plane(Pl,X,Y)* to resolve *at-plane(Pl,Y))*$_{unload-plane}$

establish: *at-object(O, X)* with *at-object(O, X)*^{Act2}

establish: *at-plane(Pl,X)* with *at-plane(Pl, X)*^{Act3}

establish: *neq(X,Y)* with *neq(X, Y)*^{Act4}

Rules such as these are then consulted by the default planner in Step 1. When refining a partial plan P, Sys-SEARCH-CONTROL's planner checks to

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

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 Sys-SEARCH-CONTROL's planner uses the rule whose quality-formula has the highest value when evaluated in context of P. A rule is guaranteed to guide the planner towards applying refinements that result in a higher-quality plan unless the partial plan has some yet unseen open-conditions that *negatively interact* with the preconditions in the antecedent of the rule. A negative interaction occurs if the application of a rule leads to a qualitatively lower plan. Sys-SEARCH-CONTROL detects these cases and learns a more specific rule. The reader is directed to previous publications [14] that provide the details of this algorithm.

3.4 Step 3: Learning rewrite rules

As noted earlier, Sys-SEARCH-CONTROL and Sys-REWRITE differ in the use they make of the output of Step 2. Rather than making a search-control rule that will be applied during the partial-order planning process, Sys-REWRITE computes (a) the actions that are added by the worse plan's relevant decision sequence. These become the action sequence to-be-replaced, (b) The actions that are added by the better plan's relevant decision sequence. These become the replacing action sequence, and (c) The preconditions and effects of the replacing and the to-be-replaced action sequence. Sys-REWRITE then stores this information as a rewrite rule. For instance, the rule learned for the example shown in Figure 3 is:

```
:replace
   :actions  {load-tr(O,T,X),drive-tr-acities(T,X,Y),unload-tr(O,T,Y)}
   :causal-links {

             load-tr(O,T,X) $\xrightarrow{in-tr(O,T)}$ unload-tr(O,T,Y),
             drive-tr-acities(T,X,Y) $\xrightarrow{at-tr(T,Y)}$ unload-tr(O,T,Y) }


:with
      :actions  {load-pl(O,L,X),fly-plane(L,X,Y),unload-pl(O,L,Y)}
```

Sys-REWRITE uses a POP algorithm to generate an initial plan P_i, the set of casual links Cl_p, ordering constraints O_p and the set of effects E_p. It then checks to see if a rule exists whose to-be-replaced sequence S_1 is a subset of P and whose causal-link constraints Cl_{s1} are a subset of P's causal-links set. If any such rule $R = (S_1, Pre_{s1}, Eff_{s1}, Cl_{s1}, S_2, Pre_{s2}, Eff_{s2})$ is retrieved, then all ordering constraints from O_p that involve an action from S_1 are deleted. It also deletes all causal links from Cl_p whose producer is a members of S. All those conditions in the casual-links that have a producer in S_1 and a consumer in $P - S_1$ are added to the set of open conditions. The replacing action sequence is appended to the set of actions. The new partial plan is then refined. If all its flaws can be removed without adding any actions and the resulting plan P_N has a higher quality value than P_i then it is returned, otherwise P_i is returned.

As Ambite and Knoblock [1] point out, the performance of a rule-rewriting system depends on a number of factors: (1) the algorithm used to produce the initial plan, (2) the search algorithm used for plan-rewriting. Two search strategies are (a) *first improvement* generates the neighborhood incrementally and selects the first solution of better quality than the current one, and (b) *Best improvement* generates the complete neighborhood and selects the best solution within this neighborhood.

To provide a fair comparison of the two approaches, we used the derivational-analogy algorithm of [4] to speed-up the generation of the initial plans for Sys-REWRITE. Therefore, Sys-REWRITE not only learns rewrite rules on Step 3, but also caches entire planning episodes.

4 Experiments and Results

Three domains were used for the experiments reported here: Softbot [16], modified transportation logistics domain [14], and Minton's process planning domain [10]. For the Softbot domain, plan quality is a function of the sum of all the resources consumed [16]. For the logistics domain the quality function depends on the resources of time and money and is described in [14]. Each action had a fixed cost associated with it in the process planning domain.

Dependent measures were planning effort, as a function of the number of partial-plans searched, and plan quality. One hundred and twenty 2-goal problems were randomly generated for logistics and Softbot domain. For the process planning problems, the number of goals for each problem was randomly ranged between 2 and 5. The process planning domain had two objects and the goal was to shape them. For the logistics domain, each problem had two objects to deliver, three cities, three trucks and two planes. Softbot problems contained two persons about whom some information was sought.

Training sets of 20, 30, 40, and 60 were randomly selected from the 120-problem corpus, and for each training set, the remaining problems served as the corresponding testing set. To identify the high quality model plan for each training problem, POP was run in a depth-first search mode with a depth limit of 15. The first 20 plan (or all possible solutions for a problem if this number was less than 20) were generated and the highest quality plan from these was used as a model plan for that problem. These were also the plans from which the distance was measured to compute the plan quality metric. Planning effort was measured by the number of new nodes expanded by each planner and plan quality was measured by computing the average between the quality value of the optimal quality plan and the quality of the plan produce by the planner on the test problems. Rewrite module of Sys-REWRITE-first uses the first-improvement search strategy and the rewrite module of Sys-REWRITE-best uses the best-improvement search strategy.

number of training examples		0	20	30	40	60
number of new nodes expanded	Sys-REWRITE-first	24+0	9.8 + 6.7	8.3+ 6	7+ 7	7.8 + 5
	Sys-REWRITE-best	24+0	9.8 + 36	8.9+ 44	8.5+ 75	7.9 + 95
	Sys-SEARCH-CONTROL	24	18.3	17.45	17.3	16.8
average difference from optimal quality plans	Sys-REWRITE-first	1	0.82	0.84	0.78	0.80
	Sys-REWRITE-best	1	0.01	0	0	0
	Sys-SEARCH-CONTROL	1	0.05	0.04	0.03	0

Table 1. Performance data for the process planning domain.

Tables 1, 2 and 3 show the performance of Sys-REWRITE and Sys-SEARCH-CONTROL on Softbot, process-planning and transportation domains, respectively. The new nodes expanded by Sys-REWRITE are shown as $N + M$, where N is the number of nodes expanded by the default planner and M is the number of nodes expanded by the rewrite-module (i.e., the number of nodes required to refine the flaws introduced by applying rewrite rules to the initial plan). The two counts are represented separately because the rewrite nodes are slightly less costly than the planning nodes.

number of training examples		0	20	30	40	60
number of new nodes expanded	Sys-REWRITE-first	36+0	14+132	14+156	13+124	12+127
	Sys-REWRITE-best	36+0	14+14212	14+21518	13+22020	12+22788
	Sys-SEARCH-CONTROL	36	12.5	13	12	11
average difference from optimal quality plans	Sys-REWRITE-first	1	0.95	0.96	0.94	0.92
	Sys-REWRITE-best	1	0.85	0.74	0.72	0.70
	Sys-SEARCH-CONTROL	1	0.03	0.02	0.01	0

Table 2. Performance data for the transportation domain.

For all three domains, both rewrite and the search-control rules lead to significant improvements in plan quality. As expected, the quality of the plans produced by Sys-REWRITE-best is higher than those produced by Sys-REWRITE-first. It is interesting to note, however, that for all three domains quality improvements obtained by using search-control rules are comparable or better than those obtained by rewrite rules. Recall that Sys-Rewrite-best does an exhaus-

number of training examples		0	20	30	40	60
number of new nodes expanded	Sys-REWRITE-first	10.4+0	3.4 + 21	3.0+ 22	2.5+25	2.1 + 24
	Sys-REWRITE-best	10.4+0	3.4 + 86	3.0 +96	2.5+108	2.1 + 126
	Sys-SEARCH-CONTROL	10.4	3.03	3.0	2.44	2.1
average difference from optimal quality plans	Sys-REWRITE-first	1	0.67	0.65	0.59	0.60
	Sys-REWRITE-best	1	0.22	0.18	0.14	0.13
	Sys-SEARCH-CONTROL	1	0.55	0.47	0.14	0.12

Table 3. Performance data for the softbot domain.

tive consideration of the complete neighborhood, and that is reflected in the $2 - 2000$-fold increase in node expansion over Sys-SEARCH-CONTROL. For Softbot, Sys-REWRITE-best expands about 30 times times more nodes than Sys-SEARCH-CONTROL. For transportation domain, which is the most complex one, planning by rewriting system need to search thousands of extra nodes without finding better quality plans than those produced by the search control system. For all that work, no improvement in quality!

5 Related Work

The basic idea of learning search-control rules to speed-up problem solving can be traced back to the early work on EBL [11, 10]. Minton's [10] PRODIGY/EBL learned control rules by explaining why a search node leads to success or failure. Kambhampati *et al.* [6] propose a technique based on EBL to learn control rules for partial-order planners and apply it to SNLP and UCPOP to learn rejection-rules. Ihrig *et al.* [4] extended SNLP+EBL to learn from planning successes as well as failures. However, these systems only aim to improve planning efficiency and not plan quality. There has been some work on the PRODIGY project for learning control rules to improve plan quality [12, 5]. However, such work has been limited to state-space planners.

Zimmerman *et al.* [17] and Estlin and Mooney [2] present two inductive learning techniques to learn search control rules for partial order planners. SCOPE [2] uses inductive logic programming techniques whereas Zimmerman's system uses a neural network to acquire search control rules for UCPOP.

Ambite and Knoblock [1] coined the term *planning by rewriting*. Their system, PBR, used a small number of hand-coded rewrite rules for the Block's world, the process planning domain and the query planning domain to improve the quality of the plan produced by SAGE [7], a partial-order planner.

6 Conclusion

Much work has been done to improve planning efficiency. The work reported here is concerned with methods to improve the quality of plans that work and specifically. In previous work [14], we demonstrated a learning algorithm that can identify search-control rules that can guide plan-refinement decisions towards producing higher-quality plans. Ambite and Knoblock [1] argue that it may be more practical to generate a low-quality plan efficiently, and then "fix" the quality of the plan with some after-the-fact, hand-crafted rewrite rules. Thus, it was natural for us to ask two questions: (a) can such hand-crafted rules for improving plan quality be learned and (b) if so, how do such learned rules stack up against search control rules in producing high-quality plans. In this paper, we presented a method for learning rewrite rules based on the same framework for learning search control rules. Our data indicates that higher quality plans are produced by using the search control rules than those produced by using the rewrite rules, and at a considerable efficiency savings.

References

1. J. Ambite and C. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *AAAI-97*, Menlo Park, 1997. AAAI Press.
2. T. Estlin and R. Mooney. Learning to improve both efficiency and quality of planning. In *Proceedings of the IJCAI*. Morgan Kaufmann, 1997.
3. P. Fishburn. *Utility Theory for Decision Making*. Wiley, New York, 1970.
4. 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.
5. M. Iwamoto. A planner with quality goal and its speed up learning for optimization problems. In *Proceedings of AIPS*, pages 281–286, 1994.
6. S. Kambhampati, S. Katukam, and Y. Qu. Failure driven dynamic search control for partial order planners. *Artificial Intelligence*, 88:253–316, 1996.
7. C. Knoblock. Building a planner for information gathering: A report from the trenches. In *Proceedings of AIPS*, 1996.
8. S. Mahadevan, T. Mitchel, L Steinberg, and P. Tadepalli. An apprentince-based approach to knowledge acquisition. *Artificial Intelligence*, 64:1–52, 1993.
9. D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *AAAI-91*, pages 634–639, Menlo Park, CA, 1991. AAAI Press.
10. S. Minton. Expalantion-based learning. *Artificial Intelligence*, 40:63–118, 1989.
11. T. Mitchell, R. Keller, and S. Keddar-Cabelli. Explanation based learning: A unifying view. *Machine Learning*, 1:47–80, 1986.
12. A. Perez. Representing and learning quality-improving search control knowledge. In L. Saitta, editor, *Proceedings of ICML*, Altos, CA, 1996. Morgan Kaufmann.
13. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–2666, 1990.
14. M. A. Upal. Learning to improve quality of the plans produced by partial-order planners. PhD thesis, University of Alberta, 2000.
15. M. Veloso. *Learning by Analogical Reasoning*. Springer Verlag, Berlin, 1994.
16. M. Williamson. A value-directed approach to planning. Technical Report TR-96-06-03, PhD thesis, University of Washington, 1996.
17. T. Zimmerman and S. Kambhampati. Neural network guided search control in partial order planning. In *AAAI-96*, Menlo Park, CA, 1996. AAAI Press.