In Lorenza Saitta, ed., *Machine Learning: Proceedings of the Thirteenth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA. 1996.

Representing and Learning Quality-Improving Search Control Knowledge

M. Alicia Pérez
Computer Science Department
Boston College
Chestnut Hill, MA 02167
perezab@cs.bc.edu

Abstract

Generating good, production-quality plans is an essential element in transforming planners from research tools into real-world applications, but one that has been frequently overlooked in research on machine learning for planning. This paper describes QUALITY, an architecture that automatically acquires operational quality-improving control knowledge given a domain theory, a domain-specific metric of plan quality, and problems which provide planning experience. The framework includes two distinct domain-independent learning mechanisms which differ in the language used to represent the learned knowledge, namely control rules and control knowledge trees, and in the kinds of quality metrics for which they are best suited. QUALITY is fully implemented on top of the PRODIGY4.0 nonlinear planner and its empirical evaluation has shown that the learned knowledge is able to substantially improve plan quality. Although the learning mechanisms have been developed for PRODIGY4.0, the framework is general and addresses a problem confronted by any planner that treats planning as a constructive decision-making process.

1 INTRODUCTION

Controlling search is a central issue in many AI systems and in particular in classical planners. Search-control knowledge serves a double purpose: to increase the efficiency of the planner's search, so that plans are found faster, and to improve the quality of the plans that are found. There is usually more than one plan for a problem, but only the first one that is found will be returned. By directing the problem solver's attention along a particular path, control knowledge can express preferences for plans that are qualitatively better (e.g., more reliable, less costly to execute, etc.). Previous machine learning research has focused on *planning efficiency* (or "speed-up learning" (Mitchell et al., 1986; Minton et al., 1989; Tadepalli, 1989; Etzioni, 1990; Pérez and Etzioni, 1992; Knoblock, 1994; Veloso, 1994; Gratch et al., 1993)). We concentrate instead on *learning quality-enhancing search control knowledge* from a combination of planning experience and interaction with a human expert. Sections 2 and 3 describe the problem of learning plan quality and our architecture to address it, which includes two distinct learning mechanisms (Sections 4 and 5). The final sections describe their empirical evaluation and discuss their properties.

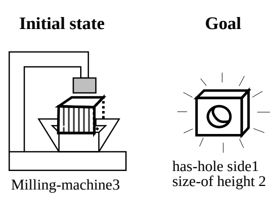
2 THE PROBLEM

Knowledge about plan quality in a domain D comes in two forms: (a) a post-facto quality metric $Q_D(P)$ that computes the quality (e.g. the execution cost) of plan P, and (b) planning-time decision-control knowledge used to guide the planner's towards producing higher-quality plans (e.g. search-control rules to choose an operator to achieve a goal). The first kind of knowledge, $Q_D(P)$, is non-operational; it cannot be brought into play until *after* a plan is produced. Yet, cost functions is exactly the kind of quality knowledge typically available, in contrast to the far more complex operational decision-time knowledge. Hence, automatically acquiring the second kind of knowledge from the first is a very useful, if quite difficult endeavor. In essence, learning operational (planning-time) quality control knowledge can be seen as a *translation* problem of domain knowledge D and quality metric Q_D into runtime decision control guidance aided by planning experience E in the form of actual planning episodes and planning decisions, and human expert suggestions that bias the learning

$$\textit{Quality: } D \times Q_D \times E \rightarrow \textit{Decision-Control}_{D,Q_D}$$

And the *full automation of the quality mapping problem* is the ultimate objective of this research.

Figure 1 summarizes the problem of finding good plans with an example in a process planning domain (Figure 1(a)). Three distinct metrics of plan quality in this domain are displayed in the table of Figure 1(b). In this example plan quality corresponds to plan execution cost and the metrics assign a fixed cost to each operator; lower values are higher quality. However those costs in general may depend on the particular operator instantiation (bindings). The total quality (cost) of a plan P is computed by adding the cost of the plan operators. Throughout the paper I focus on quality metrics of that kind. Figure 1(c) presents two plans for the problem in (a). Figure 1(d) shows the quality values for each of the plans and quality metrics. Values in bold face indicate the cost of the better plan in each case: Plan 1 is better under metrics Q_1 and Q_3. Plan 2 is better under metric Q_2.



(a)

Three distinct quality metrics

	q_1	q_2	q_3
Milling machine operators	5	7	7
Drill Press operators	5	1	1
Part set-up operators	1	1	2
Tool set-up operators	1	1	1

Plan quality $Q_i(P) = \sum_{op_j \in P} q_i(op_j)$

(b)

Two plans for the problem

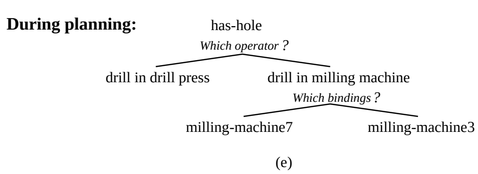
Plan 1	Plan 2
face-mill switch tool drill in milling machine	face-mill release part move holding device to drill press move part to drill press hold part put-tool drill in drill press

(c)

The quality of the plans

	Q_1	Q_2	Q_3
Plan 1	**11**	15	**15**
Plan 2	15	**13**	17

(d)



(e)

Figure 1: The Problem of Finding Good Plans

Figure 1 (e) shows the decisions that the planner confronts at problem solving time when given the example problem. Those decisions are different depending on the quality metric chosen. The example shown is a simple one. There may be other goals to achieve, and thus decisions to make about which goal to work on next, and also interactions among goals. There may be other alternative operations to achieve the problem goals, and many resources (tools, machines, different types of raw materials) that must be selected by the planner.

Thus, the problem we address is how to automatically acquire search-control knowledge that will guide the planner to make the decisions leading to the better plans, for each particular domain-specific quality metric. Note that such control knowledge is orthogonal to planning efficiency control knowledge for early pruning of choices that are guaranteed to lead to failure paths. Such knowledge can be learned by building explanations or proofs of the failure and has been the target of previous research, for example (Minton, 1988; Etzioni, 1990; Pérez and Etzioni, 1992; Katukam and Kambhampati, 1994). In the case of quality control knowledge the alternatives decided upon may all lead to success, that is, to plans for the problem, but those plans have different quality. Proving that an alternative will always generate a plan of higher quality than other alternative turns out to be a much more complex problem which the cited previous research does not address.

3 THE ARCHITECTURE

Figure 2 shows the architecture of QUALITY, which addresses the quality mapping problem. QUALITY is *given* a domain theory D (operators and inference rules) and a domain-dependent metric that evaluates the quality of the plans produced $Q_D(P)$. It is also given problems to solve in that domain. QUALITY analyzes the planning episodes by comparing the search trace[1] for the plan obtained given the current control knowledge, and another search trace corresponding to a better plan (*better* according to the quality metric). The latter search trace is obtained by asking a human expert for a better plan and then producing a search trace that leads to that plan.[2] Alternatively, the system can function autonomously by letting the planner search further until a better plan is found.[3] QUALITY then analyzes the differences between the sequence of decisions that the planner made initially and the ones

[1] By *search trace* we mean the sequence of decisions made during planning.

[2] We have developed an interactive plan checker to facilitate the interaction (Pérez, 1995).

[3] Large search spaces in complex domains can make this computationally very expensive.

that should have been made to generate the plan of better quality. The learner interprets these differences as learning opportunities and identifies the conditions under which the individual planning choices will lead to the desired final plan. In this way QUALITY compiles knowledge to control the decision making process in new similar planning situations to generate plans of better quality.

We have developed learning algorithms for two different formalisms of representing quality-enhancing search control knowledge, namely search-control rules and control knowledge trees. The next sections describe them.

4 QUALITY-ENHANCING SEARCH-CONTROL RULES

Previous research has evaluated the use of control rules to improve the efficiency of planning and proposed mechanisms to automatically acquire them from experience. Part of our contribution has been to demonstrate that learned search control rules also improve plan quality by developing a domain-independent learning algorithm specifically targeted to the plan quality problem. Here we only sketch it. The learned rules are domain specific and quality metric specific, that is, if the quality metric change the rules may render invalid and must be relearned. The algorithm is explanation-based and receives as input the two search traces corresponding respectively to the planner's initial plan and the expert-suggested improved plan. The traces are compared in order to generate an explanation of why one plan is better that the other according to the given quality metric. From the explanation control rules are generated to guide operator, bindings, and goal ordering choices. Figure 3 shows a very simple learned rule for an example similar to the one in Figure 1. The rule is relevant when the goal is to drill a hole in a part, for which two operators are available: drill in the drill press and drill in the milling machine. If the part needs in addition to be set on a milling machine, for example because another problem goal is to have the part cut, then prefer the operator to drill in the milling machine, thus reducing set-up cost (the quality metric in this example). Note that this rule ignores the relative cost of each drill operator and only optimizes the cost of setting up the work. QUALITY also learns bindings (instantiation) and goal ordering preference rules. QUALITY builds incomplete explanations since it would be computationally too expensive to consider all the possible scenarios (goal statements and initial states) to build the complete explanation of when and why the alternative suggested by the rule is going to lead to a better plan. QUALITY refines the learned knowledge upon failure by adding new rules and setting priorities among the rules. Details of the learning algorithm can be found in (Pérez, 1995).

```
(control-rule pref-drill-with-spot-drill-in-milling-machine30
  (if (and (current-goal
             (has-spot <part> <hole> <side> <loc-x> <loc-y>))
           (pending-goal
             (holding <mach> <holding-dev> <part> <side> <side-pair>))
           (type-of-object <mach> milling-machine)))
  (then prefer operator drill-with-spot-drill-in-milling-machine
                        drill-with-spot-drill))
```

Figure 3: An Automatically Learned Operator-Preference Control Rule. < > Indicate Variables

5 CONTROL KNOWLEDGE TREES

Search control rules apply to individual decision points and make local decisions. However more complex quality metrics require reasoning about tradeoffs. In those cases it is desirable to take a global view of the plan in order to make a set of globally optimal choices. In order to capture tradeoffs the control rules have to be quite specific to describe the conditions that break the tradeoff towards the better plan. For a control rule to fire all its preconditions must be matched. In addition such specific rules do not transfer well across problems. For the learner to be effective it is desirable that the learned knowledge can be reused in similar but different problems from the ones it was learned, even if it matches only partially the current situation.[4] When the goal of the learned knowledge is to improve plan quality, partially matched control knowledge can provide an estimate of the quality of the available alternatives that guides the planner to choose the most promising one. Finally, the learned control rules implicitly capture the quality metric as the choice of their preconditions was guided by differences in quality between two plans. If the metric changes, the rules may become invalid and must be relearned. Thus a desirable property of quality-enhancing control knowledge would be its robustness to changes in the quality metric. A way to achieve such robustness is by expressing explicitly and parameterizing the quality metric in the control knowledge.

[4] Partial matching and adaptation is a key feature in case-based reasoners (Veloso, 1994).

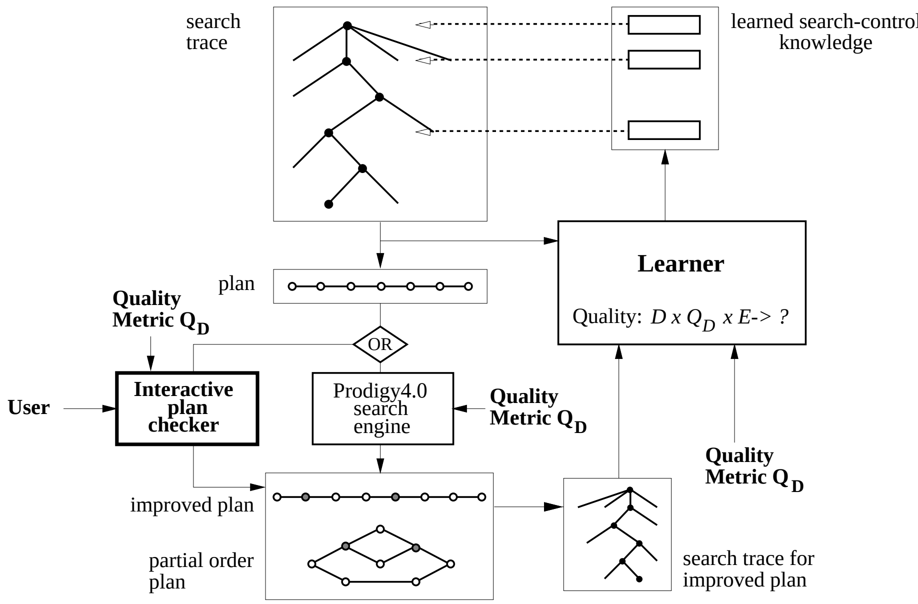


Figure 2: Architecture of QUALITY

This analysis of the characteristics of the learned control rules lead us to develop control knowledge trees, a new formalism for control knowledge representation. A control knowledge tree (*cktree*) has goal, operator, and parameterized operator binding nodes.[5] For our purposes consider that an operator node and its child binding node form a single node. Figure 4 shows two cktrees in the process planning domain. The root of a cktree is a parameterized goal node that corresponds to a goal of a problem in which learning was invoked. The children of a goal node are operator nodes corresponding to the operators that have been used in past planning experience to achieve that goal. In the example, the `has-spot` goal node has two children: two alternative ways to achieve the goal (cf. Figure 1), and it was learned from a problem in which the planner's initial plan and the improved plan achieved the goal in two different ways. When the cktree is built, constants are generalized to variables. The children of an operator node are the operator preconditions. The cktree contains additional links that capture achievement and operator side-effect information. For example, the link marked *achieves* in Figure 4 indicates that achieving the `holding` goal with the appropriate instantiation fulfills a precondition of both the face-mill and drill in milling machine operators. The link was added to the cktree because such achievement occurred in a past planning episode.

[5]These three types of nodes correspond to the goal, operator, and bindings choice points in PRODIGY4.0's planning algorithm.

Cktrees are automatically learned from a planning episode by translating the search trace into a cktree and parameterizing its constants. The learner checks whether a cktree for the problem goal exists already. If not, a new one is built. Otherwise the learner updates the existing cktree detecting which new information is relevant to a decision that should have been made differently in order to obtain the higher quality plan, and adding it in the form of subtrees to the existing cktrees. The cktrees in the figure were learned from two planning episodes. The area in lighter font shows the subtree learned from the second episode. In the first episode, the `holding` subgoal was true in the initial state and needed not to be achieved. In the second episode that was not the case and further work (represented by the subtree below `holding`) was needed. The cktree learning algorithm is described in detail in (Pérez, 1995).

The cktrees are used to suggest operators and their instantiations for the goals and subgoals that appear in the cktree by *estimating* the quality of each alternative for the current situation and quality metric. When the planner is solving a new problem and confronts a control decision, if a cktree rooted at the problem top-level goal is available, it is used to provide an estimate of the quality of the final plan for each of the alternatives. Figure 5 shows the basic cktree traversal procedure. The cktree matcher traverses the cktree starting at its root which is instantiated with the current goal and recursively computing the estimate

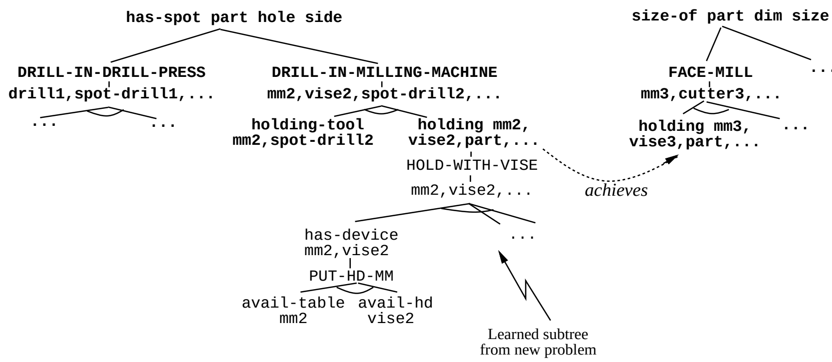


Figure 4: Example Cktrees Automatically Learned from a Planning Episode in the Process Planning Domain Similar to the One Described in Figure 1

of achieving each subgoal as the sum of the instantiated operator's cost, according to the quality metric Q_D, plus the estimates of achieving its preconditions. At any time if the estimate exceeds a bound, corresponding to the cost of the best alternative found so far, the exploration of the current alternative is abandoned. When exploring a goal node, if an estimate is available and can be reused, the subtree below the goal node needs not be explored. If the achievement links indicate that the goal is achieved as a side effect of other node visited, its cost is 0. Otherwise if there is no subtree below the node (because in past planning experience it as never achieved by subgoaling on it) a default approximate value is returned as an estimate. Thus the cktree provides *partial match* of previous experience to the current situation. When a node is traversed its achievement links are followed to mark as achieved other nodes it adds or deletes as a side effect (Figure 4).

At each goal node there may be several alternative operators that in the past were used to achieve that goal. Similarly at each operator node there may be several possible instantiations of the operator. When several alternatives are available, the first one is initially explored and the rest are stored to be explored later. All the alternatives must be explored to find the one with the better quality estimate, but our algorithm includes techniques to limit the effort (Pérez, 1995).

The basic traversal procedure just described is invoked by the top-level procedure in Figure 6. This procedure is used by the planner when it must choose operator and bindings to achieve a goal G. First the relevant cktree is found and its root is instantiated for the goal. Then the routine of Figure 5 is invoked for each alternative. The relevant parts of other cktrees are explored as well to compute the estimates more accurately in the case of goal interactions. In Steps 6-8 the previous and current alternatives are compared to determine which parts of the cktree will not change their estimates therefore allowing the reuse of the cached values. This reuse is similar to the state-saving incremental techniques for minimal recomputation in network programs in (Perlin, 1988). Finally the choice of operator and bindings with the better estimate is returned and planning continues after comitting to that choice. The cktree matcher may also provide guidance for other subgoals that appear in the traversed cktree that were part of the best alternative and that the planner will confront when subgoaling (Step 9). For example, if the current goal is to have a hole in `part5` and the milling machine is suggested as the better choice, the cktree could also suggest operators and instantiations for drilling the hole (which instances of machine and tool to use), for holding the part, and even for face-milling it (cf. Figure 4).

Discussion. This section started with some characteristics lacking in control rules as a formalism to obtain good plans in the case of quality metrics that require reasoning about tradeoffs. Cktrees have those desired properties that we summarize now. Cktrees provide global guidance by making operator and bindings suggestions for the top goal and subgoals that appear in the explored cktree and possibly other cktrees corresponding to interacting goals. Cktrees allow partial match of the learned knowledge because when no knowledge is available to estimate the cost of an alternative a default value is used (Step 4 of Figure 5) and parts of the cktree that are irrelevant to the current problem are not explored (Step 3). Finally cktrees are robust to changes in the quality metric because the metric itself is used to generate estimates when the cktrees are traversed (Step 11). Thus the cost of learning can be amortized along quality metrics. Ad-

match_cktree$(q, \beta_q, alt, c_{best})$
Input: cktree root q, alternative being explored *alt*, instantiation β_q (for current goal), best estimate so far c_{best}
Output: alternative found *alt* and quality estimate c
case **type**(q)
goal node: if **reusable_value?**(q) then $c \leftarrow$ **cached_value**(q)
else if **achieved?**(q, β_q)then $c \leftarrow 0$
else if **children**$(q) = \emptyset$ then $c \leftarrow$ **default_cost**(q)
else **store_other_ops**(rest(**children**(q)))
$c \leftarrow$ **match_cktree**(first(**children**(q)),β_q)
mark_goal_achieved(q, β_0)
operator node: $\beta_0 \leftarrow$ first(**all_bnds_for_ckop**(q, β_q)); $alt \leftarrow alt \cup \{beta_0\}$
store_other_instantiations
mark_side_effects(q, β_0)
$c \leftarrow \boxed{Q_D(q, \beta_0)} + \Sigma_{p \in children(q)}$ **match_cktree**$(p, \beta_0, alt, c_{best})$
if $c > c_{best}$ abandon *alt* and return
cached_value$(q) \leftarrow c$; return($<c, alt>$)

Figure 5: Cktree Traversal Procedure

cktree_guidance_for_goal$(G, cktrees)$
Input: current goal G, learned cktrees *cktrees*
Output: *operator* and *bindings* choice for G (and possibly for its subgoals)
$ckroot \leftarrow$ **relevant_cktree**$(G, cktrees)$
$\beta_0 \leftarrow$ **instantiate**$(G, ckroot)$
$alt \leftarrow \{\beta_0\}$; $c_{best} \leftarrow \infty$
Loop: $<c_{best}, alt_{best}> \leftarrow$ **match_cktree**$(ckroot, \beta_0, alt, c_{best})$
$<c_{best}, alt_{best}> \leftarrow$ **update_with_other_cktrees**$(alt, cktrees, c_{best})$
if *stored_alts* then $alt_{prev} \leftarrow alt$; $alt \leftarrow$ **next_alt**(*stored_alts*)
mark_not_reusable_values$(ckroot, alt_{prev}, alt)$
go to *Loop*
store_other_guidance(alt_{best}); return(alt_{best})

Figure 6: Using Cktrees to Guide Planning Decisions

ditionally a set of cktrees learned for one metric can suggest alternative plans for a problem depending on different quality criteria. A limitation of the cktree algorithms described is that they do not provide guidance on which goal to work on next.

6 EXPERIMENTAL RESULTS

The algorithms described to learn control rules and to learn and use cktrees are fully implemented. This section describes some experiments in a complex process planning domain (Gil, 1991) to evaluate the quality gained by using the learned control knowledge and to compare the performance of both representation formalisms. The problems given to the planner in the experiments described focus on cutting parts along different dimensions and on drilling and finishing different kinds of holes on one or more parts. Multiple tools are available in the shop to perform those operations. The problems have up to four goals, which frequently interact since they may apply to the same part or can be achieved by sharing the shop resources (machines and/or tools). The average length of the plans for the problems in these experiments is around 100 steps, with some plans taking up to 260 steps. The number of steps includes the plan operators and the inference rules fired as both are equally used for subgoaling by the PRODIGY4.0 planner and are alternatives at decision points in the planning algorithm. Table 1 shows two different quality metrics for this domain[6]. Q_1 focuses on execution cost, in particular on the cost of setting up the work before machining a part. A good plan is one that reduces the set-up cost. Set-ups are commonly used by machinists as rough heuristics to

[6] Our implementation of the process planning domain has 73 operators, of which 38 correspond to machining operations and 35 to set-ups, and 44 inference rules. Table 1 refers only to the operators the planner used for the problems in these experiments.

Table 1: Two Distinct Quality Metrics in the Process Planning Domain

Type	Q_1	Q_2	Operators	
Drill press operators	1	3	drill-in-drill-press, ...	(4 ops.)
Milling machine operators	1	5	face-mill, drill-in-milling-machine, ...	(6 ops.)
Machine and holding device set-up operators	8	2	put-holding-device-in-drill, put-holding-device-in-milling-machine, ...	(6 ops.)
Tool operators	1	1	put-tool-on-milling-machine, ...	(4 ops.)
Cleaning operators	6	2	clean, remove-burrs	
Oil operators	3	1	add-soluble-oil, ...	(3 ops.)

construct efficient plans (Hayes, 1995). QUALITY also allows metrics that depend on operator bindings.

The training phase. QUALITY was given 60 randomly-generated problems in order to learn search control knowledge and in 13 of those an improved plan was suggested by the human expert. QUALITY learning is cheap compared to planning: In those 13 problems the average planning time was 25.9 seconds (in a Sun Sparcstation ELC running Allegro Common Lisp version 4.1 under Mach/Unix) while the average learning time was 3.5 seconds. (Similar values were obtained when learning rules and cktrees.)

The test phase. To test the performance of the knowledge learned PRODIGY4.0 was given 180 randomly-generated problems, different from the training problems. Table 2 compares the quality (execution cost) of the plans obtained with and without the learned control rules. The improvement obtained for this quality metric using cktrees was similar. Each set of 30 problems corresponds to different parameters such as the number and type of goals. The problem sets are roughly ordered in increasing complexity. Not in all problems quality could actually be improved because the planner was able to find the better solution without any control knowledge. The table shows improvements over the base planner ranging between 20 and 60% in average and upto 96% in individual problems. By sampling the problem sets we found that most plans were *virtually optimal*, meaning that they could not be improved by expert, without systematically exploring the complete space of plans. The experiments also showed that the learned knowledge does not degrade considerably planning efficiency; in fact it improves it in many cases due to shorter solutions. (Pérez, 1995) reports detailed results of these experiments.

Metric Q_2 of Table 1 requires reasoning about tradeoffs between the cost of setting up the work on different machines and the cost of using those machines to perform the operation. For example, although drilling

Table 2: Quality (With Metric Q_1) Improvements Obtained by the Learned Control Rules

Problem set (30 probs per set)	1	2	3	4	5	6
Number of probs with improvement	9	24	14	25	30	20
Cost decrease (improved probs)	49%	55%	40%	28%	57%	20%
Max cost decrease (in a single prob)	63%	71%	87%	88%	96%	34%

holes is more expensive in the milling machine, using it may lead to cheaper plans if the part and tool are set on it already. The choice changes if the number of holes is large enough. Section 5 discussed how control rules do not perform well in these kinds of metrics. Table 3 shows the quality improvement obtained by using control knowledge trees. Note that due to the tradeoffs between machining operators cost and set-up cost the differences in quality between the plans, and thus the possible quality increases, are smaller than for the previous quality metric. Still small increases in quality in this domain lead to large economical savings, as when a plan will be executed multiple times. Cktrees also achieved significant quality improvements in a small transportation domain (Pérez, 1995).

Table 3: Quality (With Metric Q_2) Improvements Obtained Using Cktrees

Problem set (30 probs per set)	1	2	3	4	5	6
Number of probs with improvement	8	25	18	22	27	21
Cost decrease (improved probs)	31%	30%	25%	18%	32%	8%
Max cost decrease (in a single prob)	40%	44%	40%	46%	55%	23%

A useful property of cktrees is their robustness to changes in the quality metric. Table 4 shows the impact in quality of reusing the learned cktrees across quality metrics. It shows the total plan quality per problem set over 60 test problems, given quality metric Q_2, using two different sets of cktrees. The second row shows the results when the cktrees learned for metric Q_1 (Table 1) are used. Thus those cktrees were learned for a metric different from the one used for testing in this table. The third row shows the results when the cktrees learned for metric Q_2 are used. The quality values obtained in both cases are comparable, denoting transfer of the learned knowledge across quality metrics.

Table 4: Robustness of Learned Cktrees to Changes in Quality Metric

Problem set (10 problems per set)	1	2	3	4	5	6
Cktrees learned for Q_1	309	451	622	936	929	1431
Cktrees learned for Q_2	310	446	622	945	898	1448

7 RELATED WORK

Iwamoto (Iwamoto, 1994) has developed an extension to PRODIGY to solve optimization problems and an EBL method to learn control rules to find near-optimal solutions in LSI design. The learning method is similar to our control rule learner (but very different from the cktree learner) in that it compares two solutions of different quality, builds an explanation, and learns operator preferences. Our method however makes use of the quality evaluation function to build the explanation, and learns in addition bindings and goal preference rules. Iwamoto's method does not allow user guidance as it uses exhaustive search until the quality goal is satisfied to find the best solution. HAMLET (Veloso et al., 1995) learns control rules that improve both planning efficiency and also the quality (length) of the plans generated, by a combination of bounded explanation and induction. Instead of relying on a comparison of pairs of complex plans as in QUALITY, HAMLET assumes that the learner is trained with simple problems for which the planner can explore the space of all possible plans. Since both HAMLET and Iwamoto's system learn control rules, they may suffer of the limitations described of using control rules when quality metrics grow in complexity. Cktrees address those limitations and can be reused across different quality metrics. ADEPT (Cohen, 1988) also learns control knowledge in the form of trees. The trees combine knowledge learned from multiple examples and they guide a theorem prover by providing guidance for a sequence of decisions, as cktrees do, rather than for a single decision as in more conventional EBL approaches. ADEPT's trees guide the prover much as a finite state machine does and they increase the theorem prover's efficiency, but may decrease its coverage as no backtracking is possible. On the other hand QUALITY focuses on plan quality performance and uses cktrees differently to provide quality estimates of different alternatives. Alternatives not chosen remain open for backtracking in case of failure or for further search if the plan's quality can be improved.

8 CONCLUSION

This paper has presented a general framework for generating good plans and two distinct domain-independent learning mechanisms which differ in the language used to represent the learned knowledge, namely control rules and control knowledge trees, and in the kinds of quality metrics for which they are best suited.

The learned quality-enhancing control rules provide effective operator, bindings, and goal ordering guidance when the quality metric does not require reasoning about complex global tradeoffs. They lead to near-optimal plans in many cases, are highly operational and very efficiently used at planning time. The performance in improving plan quality of the learned control knowledge trees is equivalent to that of control rules for simpler non-interacting situations, and superior for more complex metrics that require reasoning about tradeoffs and taking a global view of the plan to make a set of globally optimal choices. However using control knowledge trees is computationally more expensive. Control knowledge trees do not provide goal ordering control knowledge. In our experiments we have used them together with the goal preference rules learned by the first method and achieved a synergistic effect. An important advantage of control knowledge trees is their robustness to changes in the quality metric, which is parameterized. Integrating the two learning mechanisms to take advantage of their complementary characteristics remains an open issue.

Acknowledgements

This research was conducted while the author was at Carnegie Mellon University and was sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U. S. Government. The author held previously a scholarship from the Ministerio de Educación y Ciencia of Spain. Thanks to Jaime Carbonell, Manuela Veloso, Reid Simmons, Tom Mitchell, Martha Pollack, and the members of the PRODIGY project for helpful discussions and comments on this research, and to the anonymous reviewers for comments on this paper.

References

Cohen, W. W. (1988). Generalizing number and learning from multiple examples in explanation-based learning. In Laird, J., editor, *Proceedings of the Fifth International Conference on Machine Learning*, pages 256–269. Morgan Kaufmann, San Mateo, CA.

Etzioni, O. (1990). *A Structural Theory of Explanation-Based Learning*. PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA. Also appeared as Technical Report CMU-CS-90-185.

Gil, Y. (1991). A specification of process planning for PRODIGY. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Gratch, J., Chien, S., and DeJong, G. (1993). Learning search control knowledge for deep space network scheduling. In *Machine Learning. Proceedings of the Tenth International Conference*, pages 135–142, Amherst, MA. Morgan Kaufmann.

Hayes, C. C. (1995). QUEM: A method for measuring the solution quality and experience level of knowledge-based systems. *IEEE Transactions on Data and Knowledge Engineering*, forthcoming.

Iwamoto, M. (1994). A planner with quality goal and its speedup learning for optimization problem. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 281–286, Chicago, IL.

Katukam, S. and Kambhampati, S. (1994). Learning explanation-based search control rules for partial order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 582–587, Seattle, WA. AAAI Press/The MIT Press.

Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, 68.

Minton, S. (1988). *Learning Effective Search Control Knowledge: An Explanation-based Approach*. Kluwer Academic Publishers, Boston, MA. PhD thesis available as Technical Report CMU-CS-88-133, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., and Gil, Y. (1989). Explanation-based learning: A problem-solving perspective. *Artificial Intelligence*, 40:63–118.

Mitchell, T. M., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1).

Pérez, M. A. (1995). Learning search control knowledge to improve plan quality. Technical Report CMU-CS-95-175, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. PhD thesis.

Pérez, M. A. and Etzioni, O. (1992). DYNAMIC: A new role for training problems in EBL. In Sleeman, D. and Edwards, P., editors, *Machine Learning: Proceedings of the Ninth International Conference, ML92*, pages 367–372. Morgan Kaufmann, San Mateo, CA.

Perlin, M. W. (1988). Transforming programs into networks: Call-Graph Caching, applications, and examples. Technical Report CMU-CS-88-202, School of Computer Science, Carnegie Mellon University.

Tadepalli, P. (1989). Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 694–700, Detroit, MI.

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

Veloso, M. M. (1994). *Planning and Learning by Analogical Reasoning*. Springer Verlag, Berlin, Germany. PhD thesis available as technical report CMU-CS-92-174, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.