Thesis Summary: Learning Search-Control Knowledge to Improve Plan Quality

M. Alicia Pérez

June 29, 1995

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 systems. Most work has been aimed at improving the *efficiency of planning* ("speed-up learning") or at acquiring or refining domain knowledge. This thesis focuses on learning search-control knowledge to improve the *quality of the plans* produced by the planner.

Knowledge about plan quality in a domain comes in two forms: (a) a post-facto quality metric that computes the quality (e.g. the execution cost) of a plan, and (b) planning-time decision-control knowledge used to guide the planner towards producing higher-quality plans. The first kind of knowledge is not operational until *after* a plan is produced, but is exactly the kind typically available, in contrast to the far more complex operational decision-time knowledge. Learning operational quality control knowledge can be seen as a *translation* of domain knowledge and quality metrics into runtime decision guidance. The full automation of this mapping based on planning experience is the ultimate objective of this thesis.

Given a domain theory, a domain-specific metric of plan quality, and problems which provide planning experience, the QUALITY architecture developed in this thesis automatically acquires operational control knowledge that effectively improves the quality of the plans generated. QUALITY can (optionally) learn from human experts who suggest improvements to the plans at the operator (plan step) level. We have designed two distinct domain-independent learning mechanisms to efficiently acquire quality control knowledge. They 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. Its empirical evaluation has shown that the learned knowledge produces near-optimal plans (reducing before-learning plan execution costs up to 96%). Although the learning mechanisms and learned knowledge representations have been developed for PRODIGY4.0, the framework is general and addresses a problem that any planner that treats planning as a constructive decision-making process must confront.

1. Introduction

The title of this thesis, *Learning Search Control Knowledge to Improve Plan Quality*, captures three basic points that have served as motivation for my line of research. The first one is the *importance of plan quality* in planning systems. The second one is the use of *search control knowledge to generate good quality plans*. The third one is the use of *machine learning* as the vehicle to automatically acquire quality-enhancing search control knowledge.

Much research on artificial intelligence (AI) planning so far has concentrated on methods for constructing sound and complete planners that find a *satisficing* solution, and on how to find such solution in an efficient way. The definition of the *quality* of the planner's solution subjacent in this work was a narrow one: a plan is *good* if it achieves the specified goal(s). As planners are applied to more interesting, realistic tasks, plan quality becomes a crucial factor. Chien *et al* [4] report that one of the issues hindering the efforts to field planning applications is the ability to represent and reason about plan quality. Hayes [11] points that if a "knowledge-based system produces solutions of lower sophistication and quality than the user can produce on his or her own, the user may consider the system to be a hindrance and try to work around it instead of with it". Generating production-quality plans is an essential element in transforming planners from research tools into real-world applications. This thesis addresses the problem of *producing good quality plans*.

Controlling search is a central issue in many AI systems and in particular in classical planners. Unguided search can be prohibitely slow, even in toy application domains. Domain-specific search-control knowledge is required to capture heuristics or strategies. This search-control knowledge serves a double purpose: to increase the efficiency of the planner's search, so that solutions are found faster (i.e. more efficiently, by pruning the search space), and to improve the quality of the solutions that are found. There is usually more than one solution to 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 solutions that are qualitatively better (e.g., more reliable, less costly to execute, etc.). This thesis explores the *use of search-control knowledge to generate better plans*. This is different from other efforts that improve the quality of the solutions by post-facto modification.

In spite of advances in knowledge acquisition techniques and tools, acquiring knowledge, and in particular acquiring control knowledge, is still a major bottleneck in building complex planning domains. Previous research has shown the effectiveness of a variety of machine learning methods to capture problem solving heuristics expressed in a number of representation formalisms. In particular, most of the research to date in the application of machine learning to planning systems has focused on *planning efficiency*, that is, on acquiring problem solving strategies that control search in order to make problem solving more efficient. This area of research has been termed "speed-up learning". Several techniques have been used in this framework, including learning search control knowledge [20, 19, 29, 7, 24, 17, 1, 13], macro-operators [8, 15, 3, 28], chunking [16], abstraction hierarchies [14, 5], and problem-solving cases [31]. This thesis looks instead at the application of machine learning to acquire strategies that guide a planner at plan generation time towards improving *plan quality*. This aspect of planning, namely the need to generate good quality plans, has been frequently overlooked by previous learning for planning systems (exceptions are [27, 12, 1]). This thesis describes an architecture to *learn quality-enhancing search control knowledge* from a combination of planning experience and interaction with a human expert.

This thesis concentrates on quality metrics related to plan execution cost, and expressed as an evaluation function additive on the cost of the individual operators. These functions are linear and do not capture the existence of tradeoffs between different quality factors. Several factors that influence a plan's execution cost are the execution time, the material resources, or the agent skill requirements. Other quality metrics capture such as plan robustness or ability to respond well under changing or uncertain conditions, and other factors that capture the satisfaction of the client with the solution itself (for example the accuracy of the result, the comfort it provides to the user, or marketing concerns) which in some cases are harder to quantify as linear functions. In future work we want to explore how the learning mechanisms developed apply to a wider range of metrics.

This thesis builds on the PRODIGY4.0 problem solver [30, 2]. PRODIGY4.0 is a means-ends analysis nonlinear planner. PRODIGY4.0's precursor, PRODIGY2.0, was designed as a testbed for learning in the context of planning. The PRODIGY architecture provides an expressive language for representing independently both domain knowledge and search-control knowledge. The planner is given a specification of the domain and can become proficient in it by forming (*learning*) its own control knowledge and/or refining its domain knowledge through the analysis of the domain and of its problem-solving experience. The PRODIGY architecture is well suited as a vehicle for this investigation because it has clear explicit decision points that permit the infusion of automatically or manually acquired control knowledge to improve plan quality.

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 towards producing higher-quality plans. 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

$$\textit{Quality: } D \times Q_D \times \textit{Experience} \rightarrow \textit{Decision-Control}_D$$

And the *full automation of the quality mapping problem* is the ultimate objective of this thesis. In practice, the mapping is learned incrementally with experience.

2.1. Planning Decisions and Plan Quality

Planning is a constructive, incremental decision-making process. Each particular planner faces multiple decisions when generating a complete plan to solve a given problem. The types of those decisions vary from planner to planner, but they generally include choosing actions (operators) to achieve goals, choosing objects in the world with which to instantiate those operators, and choosing orderings on how the problem goals are achieved. The decisions made at those points have an effect on both planning *efficiency* and plan *quality*. A few planners, in particular PRODIGY allow planning-time decision-control knowledge to guide those decisions. Such control knowledge can be *hand-coded* or automatically acquired (*learned*) from planning experience. The problem addressed by this thesis is how to *learn* control knowledge to guide the planner to generate high-quality plans.

In this section I present two examples to illustrate how control knowledge is relevant to plan quality and how it is distinct from other kinds of control knowledge relevant to planning efficiency. Figure 1 summarizes the problem of finding good plans with an example in a process planning domain. Figure 1(a) describes the example problem as it is posed to the planner: in the initial state a square block of raw material is set on the table of a milling machine (*milling-machine3*); the goal is to have a part of height 2 with a hole in one of its sides (side1). 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. The metrics assign a fixed cost to each operator (lower values are higher quality) although 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. Figure 1(c) presents two solutions to the problem in (a). **Plan 1** uses a single machine, the milling machine on which the part is set, to both reduce the part height (using the *face-mill* operator) and drill a hole (using a *drill* operator). **Plan 2** starts by cutting the part on the milling machine and then chooses the drill press to make the hole. In order to do that the part and the only holding device available must be moved to the drill press. Figure 1(d) shows the quality values for each of the plans and the 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.

Figure 1 (e) shows the decisions that the planner confronts at problem solving time when given the example problem. It must make the appropriate decisions regarding operator (drill in the milling machine or drill in the drill press) and operator instantiation, or bindings,[1] (e.g. which of the available milling machines) in order to obtain the better plan. Those decisions are different depending on the chosen quality metric (since the quality values of the plans obtained by making the decisions are different). 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 I solve in this thesis 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 that may be available to the planner for early pruning choices that lead to failure paths. (Such control knowledge has been the target of previous research). In the case of quality control knowledge the alternatives decided upon may all lead to success, that is, to solutions to the problem, but those solutions have different quality.

[1] I use the terms *bindings* and operator *instantiation* indistinctly, unless otherwise indicated by the context. I also use *planning* and *problem solving* interchangeably.

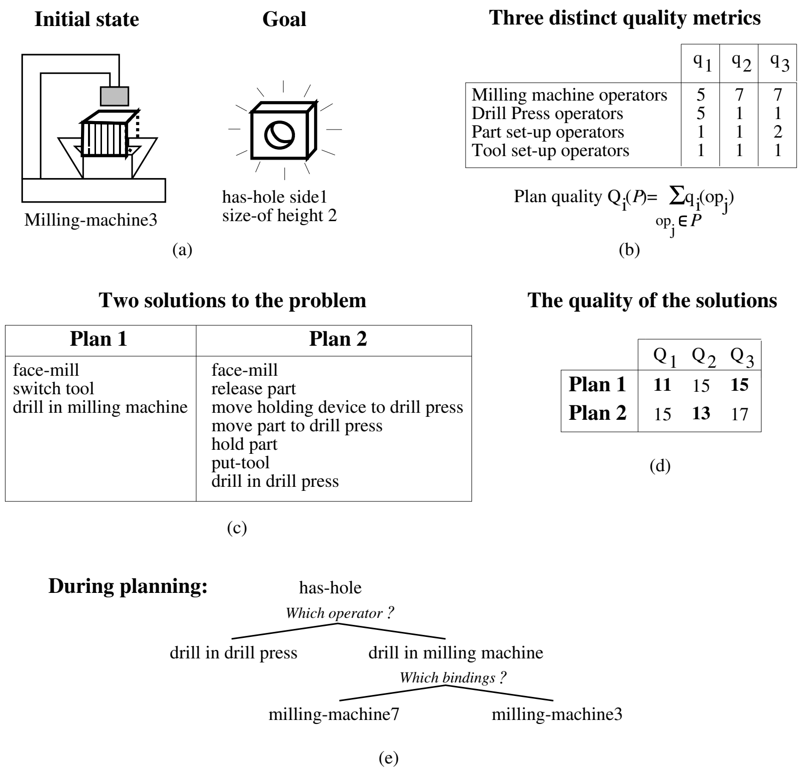


Figure 1: The problem of finding good plans addressed by this thesis.

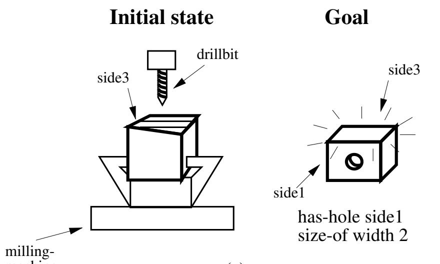
In addition to operator and operator instantiation choices, the decision about which goal to work on next in the context of the PRODIGY4.0 planner is relevant to plan quality.

Planning goals rarely occur in isolation and the interactions between conjunctive goals have an effect in the quality of the plans that solve them [32, 26, 22]. In [25] we argued for a distinction between *explicit goal interactions* and *quality goal interactions*. Explicit goal interactions are represented as part of the domain knowledge in terms of preconditions and effects of the operators. In the simpler case, given two goals g_1 and g_2 achieved respectively by operators op_1 and op_2, if op_1 deletes g_2 then goals g_1 and g_2 interact because there is a strict ordering between op_1 and op_2. Goal interactions of this type have been extensively analyzed in the planning literature. In least-commitment planners threat- solving mechanisms or critics take care of these goal interactions by establishing ordering constraints among the conflicting goals. In the case of PRODIGY goal preference control knowledge is automatically acquired to deal effectively (in the sense of problem solving effort) with this kind of goal interactions [19, 7, 24, 31, 1, 6].

On the other hand, quality goal interactions are not directly related to successes and failures. As a particular problem may have many different solutions, quality goal interactions may arise as the result of the particular problem solving search path explored. Figure 2 illustrates another case of goal interactions related to plan quality. The goal of the problem shown in (a) is to have a part of width 2 with a hole on side1. Assume that in order to reduce a part's width the part must be face-milled with its side3 facing up (i.e. facing the tool). In the initial state the drill bit to make the hole is ready (a precondition of drilling a hole), but the part is set with its side 3 facing up (a precondition of face-milling the part's width). Both goals (part's width and hole) interact because in order to achieve each one, the planner must delete a precondition of the other one already present in the initial state. For example, in order to face-mill the part, the milling cutter must be in the machine's tool holder, and achieving that requires removing the drill bit. Figure 2(c) shows two solutions to the problem, corresponding to two different orders of achieving the top-level goals. Figure 2(b) shows a quality metric for the domain. As in the first example, quality refers to plan execution cost (lower values indicate higher quality). With this metric, switching the machine tool is cheaper, as it is done automatically in semi-automated machining centers, than turning the part around, which requires a human operator. Although the two solutions shown for this problem have the same length, they have different quality according to such metric (Figure 2(d)), because the cost of reachieving the preconditions deleted from the initial state in each plan is different.

This example illustrates several points. First, plan length is not necessarily a good measure of plan quality. Second, goal achievement ordering is another planning decision relevant to plan quality, in addition to operator and operator instantiation decisions (shown in the first example). Third, how the goal interactions should be solved to obtain a good plan may depend on the metric of plan quality for the domain. The control knowledge to guide the planner to solve these quality-related interactions is harder to learn automatically than in the case of explicit goal interactions, because the domain theory, i.e. the set of operators, does not encode these quality criteria.

These two examples have shown how plan quality is affected by operator, operator instantiation, and goal achievement ordering decisions. Any system that treats planning as search is faced with decisions, or commitments, to make during search. The type of decisions varies between planning algorithms, in particular between state-space based planners (such as PRODIGY4.0) and plan-space based planners.



(a)

A quality metric

	q_1
drill, face-mill	2
put-tool, remove-tool	1
hold-part, release-part	4

Plan quality $Q_1(P) = \sum_{op_j \in P} q_1(op_j)$

(b)

Two solutions to the problem

Plan 1	Plan 2
release-part side3 hold-part side1 up drill hole remove-tool drillbit put-tool milling-cutter release-part side1 hold-part side3 up face-mill	remove-tool drillbit put-tool milling-cutter face-mill remove-tool milling-cutter put-tool drillbit release-part side3 hold-part side1 drill hole

(c)

The quality of the solutions

	Q_1
Plan 1	22
Plan 2	**16**

(d)

Figure 2: An example of the effect of goal ordering in plan quality.

For example, SNLP's descendants [18, 23] must choose how to solve threats and open conditions, by adding new causal links (ordering and binding constraints) to the existing plan steps, or by adding new steps. Making the correct decisions that will lead the planner to good plans is an issue in any planner, and one that has not received much attention in the literature.

One approach to generating good plans is to incorporate heuristics to guide the planner towards better solutions into the planner itself. Examples are removing unnecessary steps at the end of planning, or using breadth-first search when plan length is used as the evaluation function. However in some domains plan length is not an accurate metric of plan quality, as different operators have different costs. The difficulty of best-first search methods when applied to planning in knowledge-intensive domains is finding good heuristic functions that apply to incomplete solutions in the presence of conjunctive goals. The goal of learning plan quality is to *automatically acquire* heuristics that guide the planner *at generation time* to produce plans of *good quality*.

3. The Thesis

This thesis addresses the quality mapping problem. Given a domain theory, a domain-specific metric of the quality of the plans, and problems in that domain which provide planning experience, the techniques described here automatically acquire operational, planning-time search-control knowledge that effectively improves the quality of the plans generated by a planning system. In addition, the system can (optionally) benefit from the interaction with a human expert in the application domain who suggests improvements to the plans at the operator (plan step) level. The learning techniques I have developed are independent of the application domain. I have fully implemented them in the QUALITY architecture. This section gives an overview of QUALITY.

3.1. Overview of the Approach

Figure 3 shows the architecture of QUALITY. QUALITY is implemented on top of PRODIGY4.0, the most recent nonlinear planner of the PRODIGY architecture. QUALITY is given a domain theory D (operators, inference rules, and a hierarchy of the types of objects in the domain) and a domain-specific function Q_D that evaluates the quality of the plans produced. It is also given problems to solve in that domain; solving those problems provides QUALITY with planning experience in the form of a search trace, i.e. the sequence of decisions that the planner made during a problem-solving episode.[2] QUALITY analyzes problem-solving experience by comparing the search trace obtained by solving a problem given the current control knowledge, and another search trace corresponding to a better solution to the same problem (*better* according to the quality metric). QUALITY analyzes the differences between the sequence of decisions that the planner made initially and the ones that should have been made to generate the better quality plan. The learner interprets these differences as learning opportunities and identifies and generalizes the conditions under which the individual planning choices will lead to the

[2] A design principle of the PRODIGY architecture is to make that problem solving information available at any time to allow introspection by a variety of learning methods.

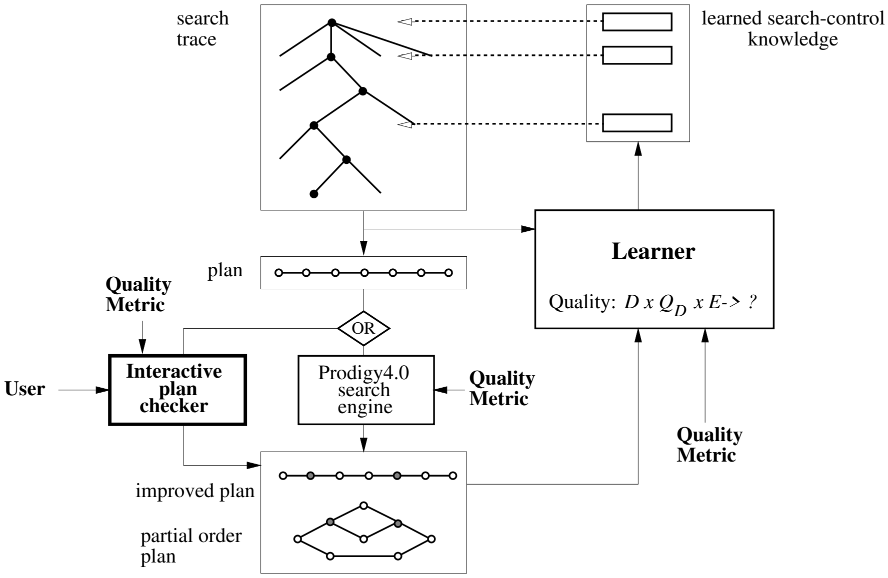


Figure 3: QUALITY: a domain-independent architecture for learning search-control knowledge to improve the quality of plans. QUALITY automates the quality mapping problem of translating the non-operational quality metric Q_D into operational quality control knowledge that can be used at planning time.

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.

In order to obtain the search trace corresponding to the better solution QUALITY can either function autonomously or interact with a human expert. In its autonomous mode, once PRODIGY4.0 has come up with the initial solution, QUALITY asks it to further explore the search space finding solutions of increasing quality, by using the available quality metric Q_D, until the space is exhausted or some resource bound is met. The best solution found within that bound and its corresponding search trace are passed to the learner.

Because of the large search spaces in complex domains, finding good enough plans from which to learn can be computationally expensive. On the other hand, human expertise is available in many domains, and can be advantageously used to help the system find useful strategies to obtain good plans. In its interactive mode, QUALITY asks a human for a better solution and then calls PRODIGY4.0 to produce a search trace that leads to that solution. QUALITY assumes that the expert's model of plan quality and the quality metric Q_D are consistent. In particular if the expert's solution is worse than the initial one, it is rejected. The interaction with the expert is the task of the *Interactive Plan Checker* in the figure. This interaction occurs at the level of plan steps, i.e. concrete actions in the plan, which correspond to instantiated operators, and not at the level of the full range of problem-solving time decisions. QUALITY's assumption (and design goal) is that the human is an expert in the application domain and can remain oblivious to the planner's algorithm and representation language. This relieves him/her of understanding PRODIGY4.0's search procedure, a very important feature to interact productively with domain experts who are not knowledge engineers.

When we allow a system to learn through interaction with a human, we must consider the gap between the expert's view of the world and efficient search control knowledge that captures such expertise. Ideally the expert should be able to provide advice using terms about the specific application domain and to ignore the details of the internal representation and the planning algorithm. On the other hand, the planner's control knowledge typically refers to problem solving states (e.g. operators that have been expanded, goals pending of exploration) in which the control decision applies. This kind of control knowledge is hard to give by such an expert. However s/he can easily provide a solution to the problem and, even better, critique and improve the solution obtained by the planner by suggesting additions, deletions, or modifications of plan steps. This advice, which corresponds to the quality metric, is then *operationalized* by the learning system [21], that is, translated into knowledge usable efficiently during problem solving. Thus the goal of learning problem solving expertise can be seen as translating a non-operational domain theory into an operational one. In the previous section I referred to it as the *quality mapping problem*. In this case the domain theory consists of the description of the domain (planning operators, inference rules, and type hierarchy) and the domain-specific quality metric. The goal of learning is to translate it into operational planning-time search-control knowledge based on the planner's experience and the advice of a human expert.

4. Learning Mechanisms to Improve Plan Quality

Two different domain-independent learning methods have been developed within QUALITY. They differ on the mechanism used for learning, on the formalism they use to represent the learned control knowledge, and on the class of quality metrics they are best suited for.

4.1. Learning Quality-Enhancing Search-Control Rules

Chapter 3 of the thesis describes domain-independent algorithms to learn search control rules, in particular PRODIGY's prefer control rules [2]. These are productions (*if-then* rules) that indicate a preferred choice of an operator to achieve the current goal, of bindings or instantiation for the chosen operator, or of the goal next goal to work on among the set of goals still pending. These decisions correspond to some of PRODIGY's decision points and they are all relevant to obtaining good plans. Previous algorithms in the PRODIGY architecture [19, 7, 24] learned control rules for PRODIGY2.0, the initial, linear planner of PRODIGY, with the goal of *planning efficiency*. QUALITY focuses on *plan quality*.

QUALITY starts by building a *plan tree* from each of the search trace for the initial plan and the search trace for the improved plan. The nodes of a plan tree are the goals, operators, and bindings, or instantiated operators, considered by PRODIGY during problem solving in the successful search path that lead to that solution. In this tree, a goal is linked to the operator considered to achieve the goal, the operator is linked in turn to its particular instantiation (bindings) chosen and the bindings are linked to the subgoals corresponding to the instantiated operator's preconditions. Leaf nodes correspond to subgoals that were true when PRODIGY tried to achieve them. The reason for them being true is stored in the goal node: either they were true in the initial state, or were achieved by some operator that was chosen for some other subgoal. This and other additional information are stored in the plan trees in the form of achievement and deletion links. The quality metric is used to recursively compute for each goal node the cost of achieving it as indicated by the subtree below it. The plan trees are the basis for the generation of new control knowledge. The plan trees are compared in order to find the *learning opportunities*, i.e. reasons why the cost of the solution is smaller in the plan tree corresponding to the improved plan. The algorithm blames those differences in operator, bindings, and goal decisions. It *explains* why one solution is better than the other given the quality metric and makes the explanation operational to those decisions. The preconditions thus obtained become part of the left-hand side of control rules that will guide those decisions in future similar situations towards the better plan.

These learning mechanisms are fully independent of the planning domain. Nonetheless there are certain characteristics of the domain and the plan quality metric that make the methods succeed in learning control knowledge that improves the quality of the plans. The method relies on the fact that the improvements in quality, in particular the savings in plan cost, are due to sharing the work among different parts of the plan. At problem solving time this means sharing subgoals among plan operators. If the operator and bindings choice can be guided so that a subgoal in its subgoaling tree is shared, plan steps are saved. These sources of improvement are nicely illustrated in the process planning domain where sharing of subgoals can be mapped to sharing of parts or subparts of a set-up, and the number of set-ups is commonly used by human experts as a rough measure of the quality of the plan. Similar characteristics appear in many other domains.

```
(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))

(control-rule pref-bnds-drill-with-spot-drill-in-milling-machine31
  (if (and (current-goal (has-spot <part> <hole> <side> <loc-x> <loc-y>))
           (current-operator drill-with-spot-drill-in-milling-machine)
           (pending-goal (holding <mach4> <holding-dev5> <part> <side> <side-pair-6>))
           (or (diff <mach4> <mach1>)  (diff <holding-dev5> <holding-dev2>)  (diff <side-pair-6> <side-pair-3>))))
  (then prefer bindings ((<mach> . <mach4>) (<hd> . <holding-dev5>) (<sp> . <side-pair-6>))
                        ((<mach> . <mach1>) (<hd> . <holding-dev2>) (<sp> . <side-pair-3>))))
```

Figure 4: Examples of simple operator and bindings quality-enhancing control rules learned in a process planning domain, which is described in detail in Chapter 2 of the thesis.

The learning algorithm described suffers of limitations. First of all, if the plan quality metric changes, the learned knowledge make become useless and, what is more important, incorrect. Second, the explanation is built from a single example (although made of two search traces). It is incomplete and leads to overgeneral rules. I have dealt with this problem by refining the learned knowledge upon failure by learning new control rules and priorities among them. Third, I have described above that the class of quality metrics that can be captured is limited. More complex metrics require a deeper reasoning about tradeoffs. In order to capture tradeoffs the control rules need to be more specific. However specific rules transfer little to problems different from the ones they were learned from, as they only fire if there is a complete match of their left-hand side. Additionally control rules apply at individual decision points while good plans may require several globally good choices. Instead, a more globally-scoped method is required. These limitations are not only due to the learning algorithm itself but to the choice of control rules as the formalism to represent quality control knowledge. They motivated the approach described in the next section.

4.2. Learning Control Knowledge Trees

In the second learning mechanism within QUALITY the learned control knowledge is represented using a formalism that I call *control knowledge trees* (cktrees). Control knowledge trees provide a more global of the plan in order to make globally good operator and binding decisions. They are used, together with the quality metric, to estimate the quality of each available alternative at a given planning decision point. These are some of the characteristics of control knowledge trees that make them suitable for capturing quality planning-knowledge: they reuse quality control knowledge learned from a single problem, even when that knowledge is incomplete and there is only a partial match; they represent the quality knowledge, that is, the quality metric, more explicitly than the above control rules, and therefore are useful when the quality metric changes; they produce control knowledge with a reasonable matching cost that depends on the difficulty of making the choice; and they consider more global information and provide more global suggestions than control rules.

When the planner must make a choice of operator or bindings for a given goal, it traverses the cktree rooted at that goal instantiating it for the current problem in order to *estimate* the quality of each alternative (that has been used in the past) for the current problem given the current quality metric. If the plan obtained with that guidance can be improved, learning is triggered. If no relevant cktree was available a new one is automatically built from the plan trees after a planning episode (see the previous section). Cktrees have the same type of nodes and links as the plan trees but differ from them in that the planning episode constants are replaced by variables. If a relevant cktree existed it is updated by adding subtree(s) to it in order to provide more accurate estimates in the future.

Cktrees are used to guide the planner at any decision point in the process of achieving the goal at the root of the cktree. That is, the cktree may provide guidance for a *sequence* of operator and binding decisions. Therefore the cktrees provide global guidance.

I have designed domain-independent algorithms to automatically build control knowledge trees from planning experience and to use them at planning time. In order to increase the efficiency of cktree matching (an exponential process) at planning time the algorithms (a) consider only alternatives that have in the past lead to good plans (instead of exploring the whole space of alternatives); (b) keep bounds on the quality of the best alternative so far; (b) cache and reuse quality estimates using state saving and incremental computation techniques; and (d) store learned constraints that prune bindings that lead to lower quality plans based on past experience. Chapter 4 of the thesis presents in great detail all these algorithms and the motivation for using cktrees to represent quality search-control knowledge.

5. Summary of Empirical Results

All the algorithms described have been fully implemented and their performance evaluated in a complex process planning domain [9, 10], in which they lead to significant plan quality improvements. An additional small transportation domain is also used to test the learned cktrees. Each of the two cited chapters finishes with the results of the empirical evaluation and a discussion of the algorithms characteristics and limitations, in particular the kind of quality metrics for which they are most suitable. Here I summarize some of the results in the process planning domain.

The methodology followed in each case was to train QUALITY with 60 randomly-generated problems. For some of them PRODIGY4.0's initial plan could be improved (either by the domain expert or by the planner itself searching within a 50000-node bound with domain-dependent hand-coded heuristics that guided backtracking and avoided useless paths). In the rest of the problems the planner's solution could not be improved. When an improved plan was available, learning was invoked, and the learned knowledge was incorporated to be used in solving further problems. Then learning was turned off and the learned knowledge was tested in a different set of randomly-generated 180 problems. I used two different quality metrics (execution cost). Here I report only on one of them (called Q_1 from here on) for which both the learned control rules and the learned cktrees suitably represented quality control knowledge.[3]

During the training phase I measured the planning time and the learning time. Planning time here means the time spent by PRODIGY4.0 solving the problem initially and then solving it to obtain the expert-improved plan, by backtracking when PRODIGY4.0's decisions with the currently available control knowledge do not match the desired plan. The learning time includes the time to construct the plan trees from the problem solving traces, and the time to build the new control knowledge. In average per problem in both cases (control rules and cktrees) the learning time (3.2 seconds and 3.5 seconds respectively) is considerably smaller than the planning time (26.5 seconds and 25.9 seconds respectively), that is, learning is cheap compared to planning.

The goal of the test phase was to evaluate the plan quality gained by using the learned control knowledge with respect to the base planner with no quality-enhancing control knowledge. Both the learned control rules and cktrees achieved similar quality performance. Table 1 shows the quality results for the control rules. Each set corresponds to different parameters in the random-problem generator (such as the number and type of goals) which lead to increasing complexity. The learned knowledge is able to reduce the plan execution cost up to 96% in one of the problems and achieves significant improvements in every set.

To further evaluate the quality of the output plans, the human expert analyzed a sample of them and found them optimal in the sense that the expert could not suggest improvements (other than systematically exploring the complete space in the hope of finding some overlooked optimization). Although this is not guarantee of optimality as the expert did not exhaust the plan space, they are considered *virtually optimal*.

Although the goal of the learning algorithms introduced in this thesis is to produce better quality plans, I also analyzed the effect of the learned knowledge in planning efficiency, in particular due to the overhead of matching the rules during planning. Table 2 shows the effects of using the learned control on the planning efficiency (time). The numbers are averaged over 5 runs for each problem. The table shows

[3]The experiments used a Sun Sparcstation ELC running Allegro Common Lisp version 4.1 under the Mach/Unix operating system.

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

Table 1: Improvement on the quality of the plans obtained for 180 randomly-generated problems in the process planning domain. Quality improvement with that metric corresponds to execution cost reduction. Row 2 shows the number of problems in each set in which the solution obtained using the learned knowledge was better than that obtained without any quality-enhancing control knowledge. (Note that for the rest of the problems the planner was able to obtain a good plan without the learned knowledge.) Row 3 shows the rate of improvement considering only those problems. The last row shows the maximum rate of improvement in a single problem of each set.

how the planning time is reduced when the learned rules are used. In spite of the slightly increased matching cost that the planner experiences when the control rules are added to the domain, the planner solves the problems faster due to the shorter length of the solutions obtained (although not in all cases shorter plans corresponded to better quality with the metric used), and therefore the smaller number of nodes searched. Thus quality-rule learning has positive utility in the Minton sense [19] even for speed-up learning.

Problem set (30 probs per set)	1	2	3	4	5	6
Time (secs.) *without* learned control knowledge	84	141	152	272	313	509
Time (secs.) with learned *control rules*	71	99	204	249	214	426
Time (secs.) with learned *cktrees*	97	155	313	410	3630	653

Table 2: Comparison of the planning time to solve the 30 problems in each test set without any quality-enhancing control knowledge (row 2), using the learned control rules (row 3), and with the learned cktrees (row 4).

The experiments also show that cktree matching is more expensive than control rule matching. However cktrees are able to generate good plans for a wider class of metrics (dealing with global tradeoffs) and they are robust to changes in the quality metric. If the quality metric changes, the control rules are invalid and must be relearned since they capture implicitly the metric. In contrast the cktree matcher uses the metric at planning time and the cktrees learned for one metric can be reused if the metric changes. Table 3 presents experimental results of this transfer. In this experiment the 60 problems of the test set were solved using a different quality metric Q_2. Row 2 of Table 3 shows the quality of the plans obtained when using the cktrees learned in the experiment described above, i.e. for metric Q_1. Thus the cktrees were learned for a different metric than that they are being tested on now. Row 3 shows quality values obtained when using the cktrees learned for the current metric Q_2. The quality of the plans generated with those two different control knowledge strategies (cktrees) is comparable. This is a useful feature of the cktree formalism. Reusing the same control knowledge, i.e. the cktrees, when the quality metric changes amortizes the cost of learning along a larger number of problems in which that knowledge is useful.

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

Table 3: Effects in quality of reusing cktrees learned for different metrics. The table shows the quality, given quality metric Q_2, of 60 problems in the test set using two different sets of cktrees. The second row shows the results when the cktrees learned for the metric Q_1 are used. (Those are the cktrees learned during the experiment described above). Thus those cktrees were learned for a metric different from the one used for testing in this table. For comparison, the third row shows the results when the cktrees learned for the metric in Q_2 are used. The quality values obtained in both cases are comparable, denoting transfer of the learned knowledge across quality metrics.

6. Scientific Contributions of the Thesis

The scientific contributions of this dissertation include:

- Focus on *plan quality* instead of just planning efficiency, because generating quality plans is an essential element in transforming planners from research tools into real-world applications.
- A demonstration that search-control knowledge can guide a planner's decisions during problem solving towards better plans according to some domain-specific quality metric. The thesis analyzes which planning decisions are relevant to plan quality and how the control knowledge to guide them is represented for a particular planning algorithm.
- The design of a domain-independent algorithm for automatically acquiring domain-dependent quality-enhancing search-control rules from problem solving experience. The learned rules indicate preferred goal, operator, and operator instantiation alternatives and take advantage of the rich control knowledge representation language in the PRODIGY planner.
- A new formalism for representing search control knowledge, control knowledge trees, which allows the explicit use of the quality metric during planning and can be reused across different quality metrics. Control knowledge trees can be automatically built from planning episodes.
- Algorithms to learn control knowledge trees from planning experience and to use them during problem solving in a fully automated, domain-independent fashion.
- A demonstration of how the learning algorithms can additionally benefit from the interaction with a human user who is an expert in the application domain but remains oblivious to the planner's algorithm and representation language.
- Full implementation and empirical demonstration of all the algorithms mentioned. The thesis presents results on their quality improvement performance, as well as in their effect in problem solving time, across different quality metrics. The results show that the learned knowledge provides significant improvements in plan quality.

References

[1] Daniel Borrajo and Manuela Veloso. Incremental learning of control knowledge for improvement of planning efficiency and plan quality. In *Working Notes of the AAAI 1994 Fall Symposium Series, Symposium on Planning and Learning: On to Real Applications*, New Orleans, November 1994.

[2] Jaime G. Carbonell, and the PRODIGY Research Group: Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, (editor), Scott Reilly, Manuela Veloso, and Xuemei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University, June 1992.

[3] Patricia Cheng and Jaime Carbonell. The FERMI system: Inducing iterative macro-operators from experience. In *Proceedings of the National Conference on Artificial Intelligence*, pages 490–495, Philadelphia, PA, 1986.

[4] Steve Chien, Randall W. Hill, and Kristina Fayyad. Why real-world planning is difficult. In *Working Notes of the AAAI 1994 Fall Symposium Series, Symposium on Planning and Learning: On to Real Applications*, pages 28–33, New Orleans, November 1994.

[5] Jens Christensen. A hierarchical planner that creates its own hierarchies. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1004–1009, Boston, MA, 1990.

[6] Rujith de Silva. Reasoning about goal-interactions in non-linear planners. Thesis proposal. School of Computer Science, Carnegie Mellon University, 1995.

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

[8] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), 1972.

[9] Yolanda Gil. A specification of process planning for PRODIGY. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1991.

[10] Caroline C. Hayes. *Machining Planning: A Model of an Expert Level Planning Process*. PhD thesis, The Robotics Institute, Carnegie Mellon University, December 1990.

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

[12] Masahiko Iwamoto. 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, 1994.

[13] Suresh Katukam and Subbarao Kambhampati. 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, July 1994. AAAI Press/The MIT Press.

[14] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68, 1994.

[15] Richard E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[16] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[17] Christopher Leckie and Ingrid Zukerman. An inductive approach to learning search control rules for planning. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI93*, 1993.

[18] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, Anaheim, CA, 1991.

[19] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.

[20] Tom M. Mitchell, Richard Keller, and Smadar Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1), 1986.

[21] D. Jack Mostow. Machine transformation of advice into a heuristic search procedure. In R. S. Michalsky, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach*. Tioga Press, Palo Alto, CA., 1983.

[22] Dana S. Nau. Enabling-condition interactions and finding good plans. In *Working Notes of the AAAI 1993 Spring Symposium Series, Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 93–97, Stanford University, CA, March 1993.

[23] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, KR92*, pages 103–114, San Mateo, CA, October 1992. Morgan Kaufmann.

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

[25] M. Alicia Pérez and Manuela M. Veloso. Goal interactions and plan quality. In *Working Notes of the AAAI 1993 Spring Symposium Series, Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 117–121, Stanford University, CA, March 1993.

[26] Martha E. Pollack. Overloading intentions for efficient practical reasoning. *Noûs*, 25(4):513–536, 1991. Also as Technical Note 497 of SRI International.

[27] David Ruby and Dennis Kibler. Learning steppingstones for problem solving. In *Proceedings of the Darpa Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 366–373, San Diego, CA, November 1990.

[28] Alberto M. Segre, David Sturgill, and Jennifer Turney. Neoclassical planning. In *Preprints of the AAAI 1993 Spring Symposium Series, Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, Stanford University, CA, March 1993.

[29] Prasad Tadepalli. *Tractable Learning and Planning in Games*. PhD thesis, Rutgers, The State University of New Jersey, Department of Computer Science, May 1990. Technical Report ML-TR-31.

[30] Manuela Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), January 1995.

[31] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994. PhD thesis available as technical report CMU-CS-92-174, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[32] Robert Wilensky. *Planning and Understanding*. Addison-Wesley, Reading, MA, 1983.